# PyText Documentation

PyText Contributors

Mar 23, 2021

# Getting Started


1 How To Use 3

2 Indices and tables 775

Python Module Index 777

Index 783

PyText is a deep-learning based NLP modeling framework built on PyTorch. PyText addresses the often-conflicting requirements of enabling rapid experimentation and of serving models at scale. It achieves this by providing simple and extensible interfaces and abstractions for model components, and by using PyTorch's capabilities of exporting models for inference via the optimized Caffe2 execution engine. We use PyText at Facebook to iterate quickly on new modeling ideas and then seamlessly ship them at scale.

**Core PyText Features:**

- Production ready models for various NLP/NLU tasks:
  - Text classifiers
    - Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification
    - Lin et al. (2017): A Structured Self-attentive Sentence Embedding
  - Sequence taggers
    - Lample et al. (2016): Neural Architectures for Named Entity Recognition
  - Joint intent-slot model
    - Zhang et al. (2016): A Joint Model of Intent Determination and Slot Filling for Spoken Language Understanding
  - Contextual intent-slot models
- Extensible components that allow easy creation of new models and tasks
- Ensemble training support
- Distributed-training support (using the new C10d backend in PyTorch 1.0)
- Reference implementation and a pre-trained model for the paper: Gupta et al. (2018): Semantic Parsing for Task Oriented Dialog using Hierarchical Representations

CHAPTER 1

# How To Use

Please follow the tutorial series in **Getting Started** to get a sense of how to train a basic model and deploy to production.

After that, you can explore more options of builtin models and training methods in **Training More Advanced Models**

If you want to use PyText as a library and build your own models, please check the tutorial in **Extending PyText**

**Note:** All the demo configs and test data for the tutorials can be found in source code. You can either install PyText from source or download the files manually from GitHub.

## 1.1 Installation

*PyText requires Python 3.6+*

PyText is available in the Python Package Index via

```
$ pip install pytext-nlp
```

The easiest way to get started on most systems is to create a *virtualenv*

```
$ python3 -m venv pytext_venv
$ source pytext_venv/bin/activate
(pytext_venv) $ pip install pytext-nlp
```

This will install a version of PyTorch depending on your system. See PyTorch for more information. If you are using MacOS or Windows, this likely will not include GPU support by default; if you are using Linux, you should automatically get a version of PyTorch compatible with CUDA 9.0.

If you need a different version of PyTorch, follow the instructions on the PyTorch website to install the appropriate version of PyTorch before installing PyText

### 1.1.1 OS Dependencies

*if you're having issues getting things to run, these guides might help*

#### On MacOS

Install brew, then run the command:

```
$ brew install cmake protobuf
```

#### On Windows

Coming Soon!

#### On Linux

For Ubuntu/Debian distros, you might need to run the following command:

```
$ sudo apt-get install protobuf-compiler libprotoc-dev
```

For rpm-based distros, you might need to run the following command:

```
$ sudo yum install protobuf-devel
```

### 1.1.2 Install From Source

```
$ git clone git@github.com:facebookresearch/pytext.git
$ cd pytext
$ source activation_venv
(pytext_venv) $ pip install torch # go to https://pytorch.org for platform specific␣
↪installs
(pytext_venv) $ ./install_deps
```

Once that is installed, you can run the unit tests. We recommend using pytest as a runner.

```
(pytext_venv) $ pip install -U pytest
(pytext_venv) $ pytest
# If you want to measure test coverage, we recommend `pytest-cov`
(pytext_venv) $ pip install -U pytest-cov
(pytext_venv) $ pytest --cov=pytext
```

To resume development in an already checked-out repo:

```
$ cd pytext
$ source activation_venv
```

To exit the virtual environment:

```
(pytext_venv) $ deactivate
```

### 1.1.3 Cloud VM Setup

This guide will cover all the setup work you have to do in order to be able to easily install PyText on a cloud VM . *Note that while these instructions worked when they were written, they may become incorrect or out of date. If they do, please send us a Pull Request!*

After following these instructions, you should be good to either follow the *Installation* instructions or the *Install From Source* instructions

#### Amazon Web Services

**Coming Soon**

#### Google Cloud Engine

*If you have problems launching your VM, make sure you have a non-zero gpu quota,* click here to learn about quotas

This guide uses Google's Deep Learning VM as a base.

**Setting Up Your VM**

- Click "Launch on Compute Engine"
- Configure the VM:
  - The default 2CPU K80 setup is fine for most tutorials, if you need more, change it here.
  - For Framework, select one of the Base images, rather than one with a framework pre-installed. Note which version of CUDA you choose for later.
  - When you're ready, click "Deploy"
  - When your VM is done loading, you can SSH into it from the GCE Console
- Install Python 3.6 (based on this RoseHosting blog post ):
  - `$ sudo nano /etc/apt/sources.list`
  - add `deb http://ftp.de.debian.org/debian testing main` to the list
  - `$ echo 'APT::Default-Release "stable";' | sudo tee -a /etc/apt/apt.conf.d/00local`
  - `$ sudo apt-get update`
  - `$ sudo apt-get -t testing install python3.6`
  - `$ sudo apt-get install python3.6-venv protobuf-compiler libprotoc-dev`

#### Microsoft Azure

This guide uses the Azure Ubuntu Server 18.04 LTS image as a base

**Setting Up Your VM**

- From the Azure Dashboard, select "Virtual Machines" and then click "add"
- Give your VM a name and select the region you want it in, keeping in mind that GPU servers are not present in all regions
- For this tutorial, you should select "Ubuntu Server 18.04 LTS" as your image

- Click "Change size" in order to select a GPU server.
  - Note that the default filters won't show GPU servers, we recommend clearing all filters except "family" and setting "family" to GPU
  - For this tutorial, we will use the NC6 VM Size, but this should work on the larger and faster VMs as well
- Make sure you set up SSH access, we recommend using a public key rather than a password. * don't forget to "allow selected ports" and select SSH
- install Nvidia driver and CUDA, (based on https://askubuntu.com/a/1036265)
  - `sudo add-apt-repository ppa:graphics-drivers/ppa`
  - `sudo apt update`
  - `sudo apt-get install ubuntu-drivers-common`
  - `sudo ubuntu-drivers autoinstall`
  - reboot: `sudo shutdown -r now`
  - `sudo apt install nvidia-cuda-toolkit gcc-6`
- install OS dependencies: `sudo apt-get install python3-venv protobuf-compiler libprotoc-dev`

## 1.2 Train your first model

Once you've installed PyText you can start training your first model!

This tutorial series is an overview of *using* PyText, and will cover the main concepts PyText uses to interact with the world. It won't deal with modifying the code (e.g. hacking on new model architectures). By the end, you should have a high-quality text classification model that can be used in production.

You can use PyText as a library either in your own scripts or in a Jupyter notebook, but the fastest way to start training is through the PyText command line tool. This tool will automatically be in your path when you install PyText!

```
(pytext) $ pytext

Usage: pytext [OPTIONS] COMMAND [ARGS]...

  Configs can be passed by file or directly from json. If neither --config-
  file or --config-json is passed, attempts to read the file from stdin.

  Example:

    pytext train < demo/configs/docnn.json

Options:
  --config-file TEXT
  --config-json TEXT
  --help              Show this message and exit.

Commands:
  export   Convert a pytext model snapshot to a caffe2 model.
  predict  Start a repl executing examples against a caffe2 model.
  test     Test a trained model snapshot.
  train    Train a model and save the best snapshot.
```

### 1.2.1 Background

Fundamentally, "machine learning" means learning a function automatically. Your training, evaluation, and test datasets are examples of inputs and their corresponding outputs which show how that function behaves. A **model** is an implementation of that function. To **train** a **model** means to make a statistical implementation of that function that uses the training data as a rubric. To **predict** using a **model** means to take a trained implementation and apply it to new inputs, thus predicting what the result of the idealized function would be on those inputs.

More examples to train on usually corresponds to more accurate and better-generalizing models. This can mean thousands to millions or billions of examples depending on the task (function) you're trying to learn.

### 1.2.2 PyText Configs

Training a state-of-the-art PyText model on a dataset is primarily about configuration. Picking your training dataset, your model parameters, your training parameters, and so on, is a central part of building high-quality text models.

Configuration is a central part of every component within PyText, and the config system that we provide allows for all of these configurations to be easily expressible in JSON format. PyText comes in-built with a number of example configurations that can train in-built models, and we have a system for automatically documenting the default configurations and possible configuration values.

### 1.2.3 PyText Modes

- **train** - Using a configuration, initialize a model and train it. Save the best model found as a model snapshot. This snapshot is something that can be loaded back in to PyText and trained further, tested, or exported.
- **test** - Load a trained model snapshot and evaluate its performance against a test set.
- **export** - Save the model as a serialized Caffe2 model, which is a stable model representation that can be loaded in production. (PyTorch model snapshots aren't very durable; if you update parts of your runtime environment, they may be invalidated).
- **predict** - Provide a simple REPL which lets you run inputs through your exported Caffe2 model and get a tangible sense for how your model will behave.

### 1.2.4 Train your first model

To get our feet wet, let's run one of the demo configurations included with PyText.

```
(pytext) $ cat demo/configs/docnn.json
{
  "version": 8,
  "task": {
    "DocumentClassificationTask": {
      "data": {
        "source": {
          "TSVDataSource": {
            "field_names": ["label", "slots", "text"],
            "train_filename": "tests/data/train_data_tiny.tsv",
            "test_filename": "tests/data/test_data_tiny.tsv",
            "eval_filename": "tests/data/test_data_tiny.tsv"
          }
        }
      },
```

(continues on next page)

(continued from previous page)

```
      "model": {
        "DocModel": {
          "representation": {
            "DocNNRepresentation": {}
          }
        }
      }
    }
  }
}
```

This config will train a document classification model (DocNN) to detect the "class" of a series of commands given to a smart assistant. Let's take a quick look at the dataset:

```
(pytext) $ head -2 tests/data/train_data_tiny.tsv
alarm/modify_alarm      16:24:datetime,39:57:datetime   change my alarm tomorrow to␣
↪wake me up 30 minutes earlier
alarm/set_alarm         Turn on all my alarms
(pytext) $ wc -l tests/data/train_data_tiny.tsv
    10 tests/data/train_data_tiny.tsv
```

As you can see, the dataset is quite small, so don't get your hopes up on accuracy! We included this dataset for running unit tests against our models. PyText uses data in a tab separated format, as specified in the config by TSVDataSource. The order of the columns can be configured, but here we use the default. The first column is the "class", the output label that we're trying to predict. The second column is word-level tags, which we're not trying to predict yet, so ignore them for now. The last column here is the input text, which is the command whose class (the first column) the model tries to predict.

Let's train the model!

```
(pytext) $ pytext train < demo/configs/docnn.json
... [snip]

Stage.TEST
Epoch:1
loss: 1.646484
Accuracy: 50.00

Soft Metrics:
+-------------------------+-------------------+---------+
| Label                   | Average precision | ROC AUC |
+-------------------------+-------------------+---------+
|       alarm/modify_alarm |               nan |   0.000 |
|          alarm/set_alarm |             1.000 |   1.000 |
|       alarm/snooze_alarm |               nan |   0.000 |
| alarm/time_left_on_alarm |             0.333 |   0.333 |
|    reminder/set_reminder |             1.000 |   1.000 |
|  reminder/show_reminders |               nan |   0.000 |
|             weather/find |               nan |   0.000 |
+-------------------------+-------------------+---------+

Recall at Precision
+-------------------------+---------+---------+---------+---------+---------+
| Label                   | R@P 0.2 | R@P 0.4 | R@P 0.6 | R@P 0.8 | R@P 0.9 |
+-------------------------+---------+---------+---------+---------+---------+
| alarm/modify_alarm      |   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |
```

(continues on next page)

(continued from previous page)

```
| alarm/set_alarm          |   1.000 |   1.000 |   1.000 |   1.000 |   1.000 |
| alarm/snooze_alarm       |   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |
| alarm/time_left_on_alarm |   1.000 |   0.000 |   0.000 |   0.000 |   0.000 |
| reminder/set_reminder    |   1.000 |   1.000 |   1.000 |   1.000 |   1.000 |
| reminder/show_reminders  |   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |
| weather/find             |   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |
+--------------------------+---------+---------+---------+---------+---------+
saving result to file /tmp/test_out.txt
```

The model ran over the training set 10 times. This output is the result of evaluating the model on the test set, and tracking how well it did. If you're not familiar with these accuracy measurements,

- **Precision** - The number of times the model guessed this label and was right
- **Recall** - The number of times the model correctly identified this label, out of every time it shows up in the test set. If this number is low for a label, the model should be predicting this label more.
- **F1** - A harmonic mean of recall and precision.
- **Support** - The number of times this label shows up in the test set.

As you can see, the training results were pretty bad. We ran over the data 10 times, and in that time managed to learn how to predict only one of the labels in the test set successfully. In fact, many of the labels were never predicted at all! With 10 examples, that's not too surprising. See the next tutorial to run on a real dataset and get more usable results.

## 1.3 Execute your first model

In *Train your first model*, we learnt how to train a small, simple model. We can continue this tutorial with that model here. This procedure can be used for any pytext model by supplying the matching config. For example, the much more powerful model from *Train Intent-Slot model on ATIS Dataset* can be executed using this same procedure.

### 1.3.1 Evaluate the model

We want to run the model on our test dataset and see how well it performs. Some results have been abbreviated for clarity.

```
(pytext) $ pytext test < demo/configs/docnn.json

Stage.TEST
loss: 2.059336
Accuracy: 20.00

Macro P/R/F1 Scores:
    Label                       Precision   Recall      F1          Support

    reminder/set_reminder       25.00       100.00      40.00       1
    alarm/time_left_on_alarm    0.00        0.00        0.00        1
    alarm/show_alarms           0.00        0.00        0.00        1
    alarm/set_alarm             0.00        0.00        0.00        2
    Overall macro scores        6.25        25.00       10.00

Soft Metrics:
    Label       Average precision
    alarm/set_alarm 50.00
```

(continues on next page)

(continued from previous page)

```
    alarm/time_left_on_alarm    20.00
    reminder/set_reminder   25.00
    alarm/show_alarms   20.00
    weather/find    nan
    alarm/modify_alarm  nan
    alarm/snooze_alarm  nan
    reminder/show_reminders nan
    Label       Recall at precision 0.2
    alarm/set_alarm 100.00
    Label       Recall at precision 0.4
    alarm/set_alarm 100.00
    Label       Recall at precision 0.6
    alarm/set_alarm 0.00
    Label       Recall at precision 0.8
    alarm/set_alarm 0.00
    Label       Recall at precision 0.9
    alarm/set_alarm 0.00
    Label       Recall at precision 0.2
    alarm/time_left_on_alarm    100.00
    Label       Recall at precision 0.4
    alarm/time_left_on_alarm    0.00
    Label       Recall at precision 0.6
    alarm/time_left_on_alarm    0.00
... [snip]
    reminder/show_reminders 0.00
    Label       Recall at precision 0.6
    reminder/show_reminders 0.00
    Label       Recall at precision 0.8
    reminder/show_reminders 0.00
    Label       Recall at precision 0.9
    reminder/show_reminders 0.00
```

### 1.3.2 Export the model

When you save a PyTorch model, the snapshot uses *pickle* for serialization. This means that simple code changes (e.g. a word embedding update) can cause backward incompatibilities with your deployed model. To combat this, you can export your model into the Caffe2 format using in-built ONNX integration. The exported Caffe2 model would have the same behavior regardless of changes in PyText or in your development code.

Exporting a model is pretty simple:

```
(pytext) $ pytext export --help
Usage: pytext export [OPTIONS]

  Convert a pytext model snapshot to a caffe2 model.

Options:
  --export-json TEXT  the path to the export options in JSON format
  --model TEXT        the pytext snapshot model file to load
  --output-path TEXT  where to save the exported model
  --help              Show this message and exit.
```

You can also pass in a configuration to infer some of these options. In this case let's do that because depending on how you're following along your snapshot might be in different places!

```
(pytext) $ pytext export --output-path exported_model.c2 < demo/configs/docnn.json
...[snip]
Saving caffe2 model to: exported_model.c2
```

Alternatively you can use the export-json to pass in a json config with only version and export fields populated.

```
(pytext) $ pytext export --output-path exported_model.c2 --export-json demo/configs/
↪export_options.json < demo/configs/docnn.json
...[snip]
Saving caffe2 model to: exported_model.c2
```

This file now contains all of the information needed to run your model.

There's an important distinction between what a model does and what happens before/after the model is called, i.e. the preprocessing and postprocessing steps. PyText strives to do as little preprocessing as possible, but one step that is very often needed is tokenization of the input text. This will happen automatically with our prediction interface, and if this behavior ever changes, we'll make sure that old models are still supported. The model file you export will always work, and you don't necessarily need PyText to use it! Depending on your use case you can implement preprocessing yourself and call the model directly, but that's outside the scope of this tutorial.

### 1.3.3 Make a simple app

Let's put this all into practice! How might we make a simple web app that loads an exported model and does something meaningful with it?

To run the following code, you should

```
(pytext) $ pip install flask
```

Then we implement a minimal Flask web server.

```
import sys
import flask
import pytext

config_file = sys.argv[1]
model_file = sys.argv[2]

config = pytext.load_config(config_file)
predictor = pytext.create_predictor(config, model_file)

app = flask.Flask(__name__)

@app.route('/get_flight_info', methods=['GET', 'POST'])
def get_flight_info():
    text = flask.request.data.decode()

    # Pass the inputs to PyText's prediction API
    result = predictor({"text": text})

    # Results is a list of output blob names and their scores.
    # The blob names are different for joint models vs doc models
    # Since this tutorial is for both, let's check which one we should look at.
    doc_label_scores_prefix = (
        'scores:' if any(r.startswith('scores:') for r in result)
        else 'doc_scores:'
```

(continues on next page)

(continued from previous page)

```
    )

    # For now let's just output the top document label!
    best_doc_label = max(
        (label for label in result if label.startswith(doc_label_scores_prefix)),
        key=lambda label: result[label][0],
    # Strip the doc label prefix here
    )[len(doc_label_scores_prefix):]

    return flask.jsonify({"question": f"Are you asking about {best_doc_label}?"})

app.run(host='0.0.0.0', port='8080', debug=True)
```

Execute the app

```
(pytext) $ python flask_app.py demo/configs/docnn.json exported_model.c2
* Serving Flask app "flask_app" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
```

Then in a separate terminal window

```
$ function ask_about() { curl http://localhost:8080/get_flight_info -H "Content-Type:␣
↪text/plain" -d "$1" }

$ ask_about 'I am looking for flights from San Francisco to Minneapolis'
{
  "question": "Are you asking about flight?"
}

$ ask_about 'How much does a trip to NY cost?'
{
  "question": "Are you asking about airfare?"
}

$ ask_about "Which airport should I go to?"
{
  "question": "Are you asking about airport?"
}
```

## 1.4 Visualize Model Training with TensorBoard

Visualizations can be helpful in allowing you to better understand, debug and optimize your models during training. By default, all models trained using PyText can be visualized using *TensorBoard <https://www.tensorflow.org/guide/summaries_and_tensorboard>*.

Here, we will explore how to visualize the model from the tutorial *Train Intent-Slot model on ATIS Dataset*.

### 1.4.1 1. Install TensorBoard visualization server

The TensorBoard web server is required to host your visualizations. To install it, run

```
$ pip install tensorboard
```

### 1.4.2 2. Verify TensorBoard events in current working directory

Complete the tutorial from *Train Intent-Slot model on ATIS Dataset* if you have not done so. Once that is done, you should be able to see a TensorBoard events file in the working directory where you trained your model. The file path will be something like `<WORKING_DIR>/runs/<DATETIME>_<MACHINE_NAME>/events.out.tfevents.<TIMESTAMP>.<MACHINE_NAME>`.

### 1.4.3 3. Launch the visualization server

To launch the visualization server, run:

```
$ tensorboard --logdir=$EVENTS_FOLDER
```

`$EVENTS_FOLDER` is the folder containing the events file in 2., which is something like `<WORKING_DIR>/runs/<DATETIME>_<MACHINE_NAME>`.

Note: The TensorBoard web server might fail to run might fail to run if TensorFlow is not installed. This dependency is not ideal, but if you see *ModuleNotFoundError: No module named 'tensorflow'* when running the above command, you can install TensorFlow using:

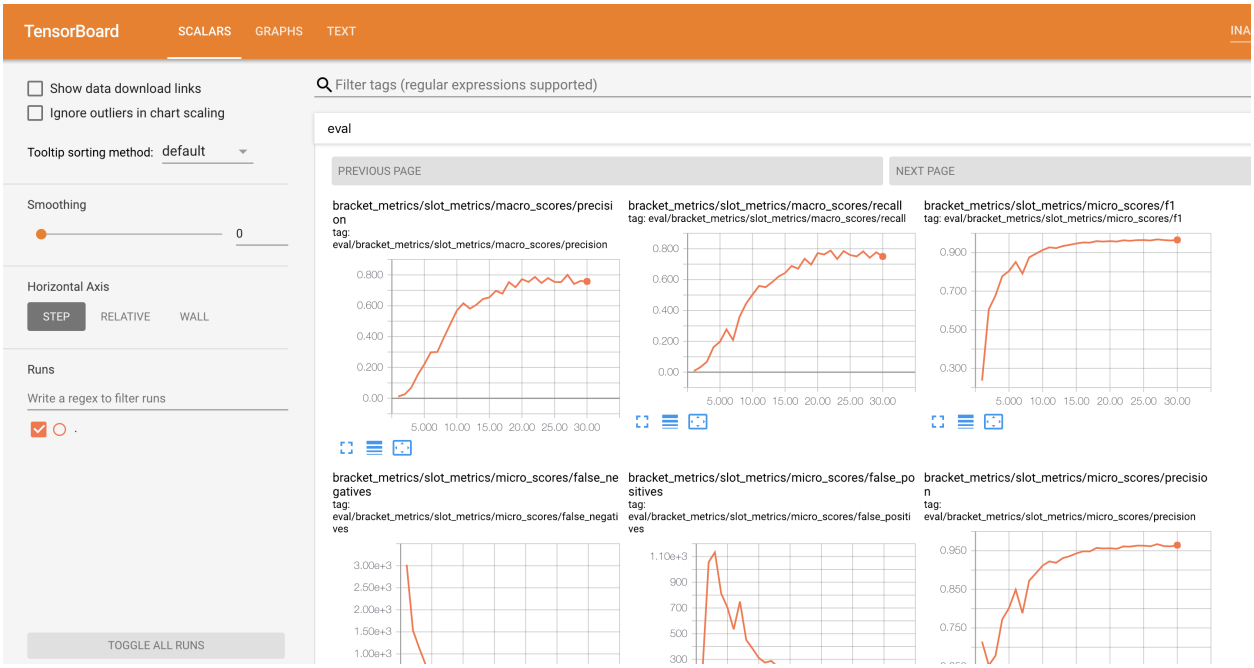
```
$ pip install tensorflow
```

### 1.4.4 4. View your visualizations

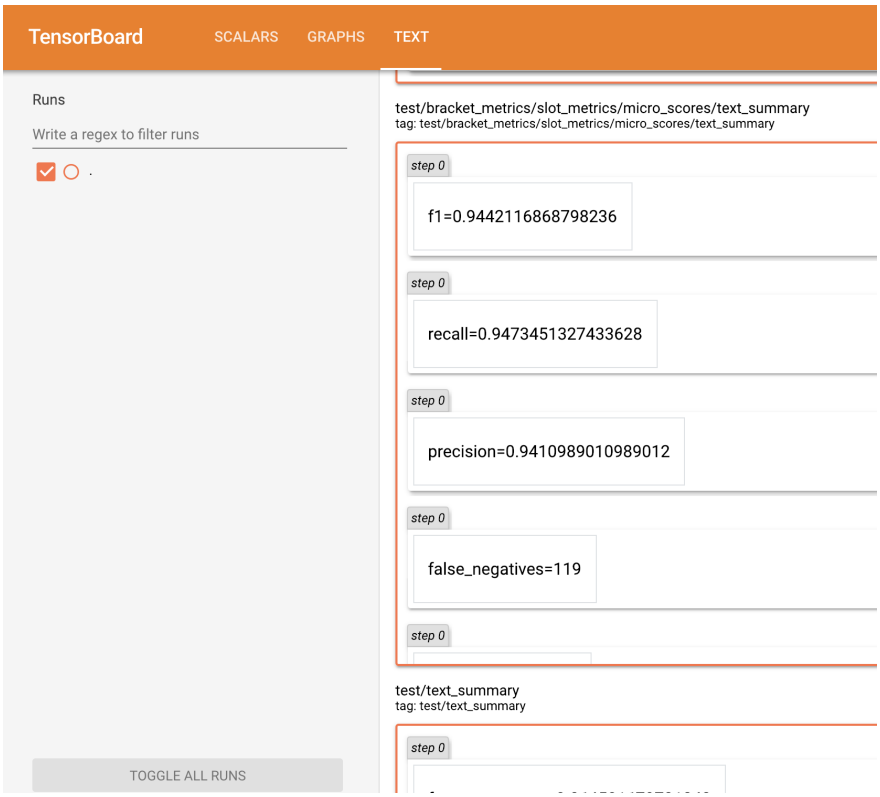
After launching the visualization server, you can view your visualizations in a web browser at *http://localhost:6006*.

PyText visualizes the training metrics as scalars, test metrics as texts, and also the shape of the neural network architecture graph. Below are some screenshots of what you will see:

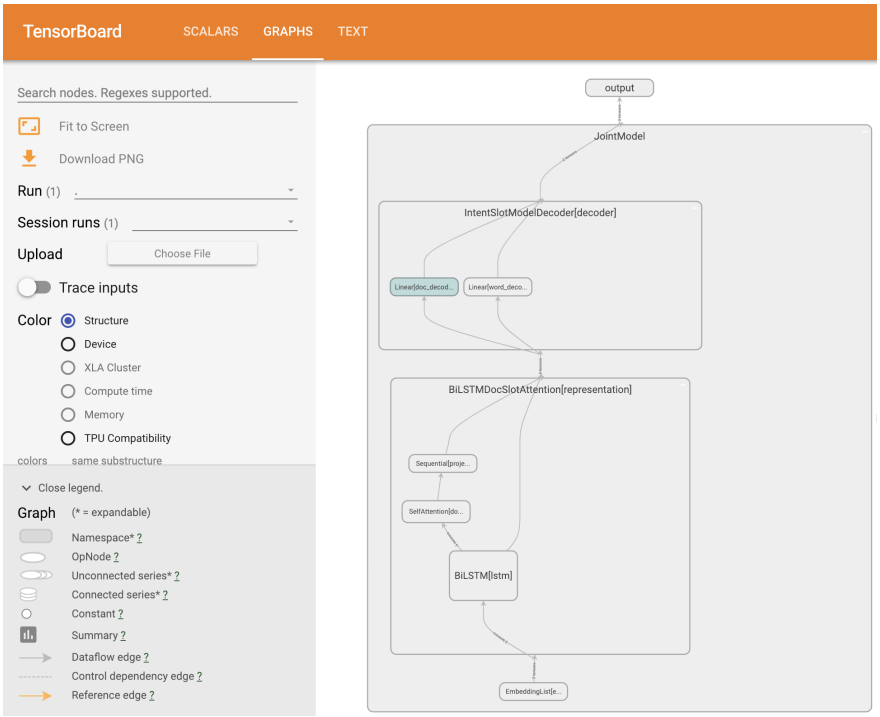
**Training Metrics:**




**Test Metrics:**




**Model Graph:**




## 1.5 Use PyText models in your app

Once you have a PyText model exported to Caffe2, you can host it on a simple web server in the cloud. Then your applications (web/mobile) can make requests to this server and use the returned predictions from the model.

In this tutorial, we'll take the intent-slot model trained in *Train Intent-Slot model on ATIS Dataset*, and host it on a Flask server running on an Amazon EC2 instance. Then we'll write an iOS app which can identify city names in users' messages by querying the server.

### 1.5.1 1. Setup an EC2 instance

Amazon EC2 is a service which lets you host servers in the cloud for any arbitrary purpose. Use the official documentation to sign up, create an IAM profile and a key pair. Sign in into the EC2 Management Console and launch a new instance with the default Amazon Linux 2 AMI. In the *Configure Security Group* step, Add a Rule with type *HTTP* and port *80*.

Connect to your instance using the steps here. Once you're logged in, install the required dependencies -

```
$ cd ~
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O␣
↪miniconda.sh
$ chmod +x miniconda.sh
$ ./miniconda.sh -b -p ~/miniconda
$ rm -f miniconda.sh
$ source ~/miniconda/bin/activate

$ conda install -y protobuf
$ conda install -y boto3 flask future numpy pip
$ conda install -y pytorch -c pytorch
```

(continues on next page)

(continued from previous page)

```
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to 8080
```

We'll make the server listen to (randomly selected) port 8080 and redirect requests coming to port 80 (HTTP), since running a server on latter requires administrative privileges.

### 1.5.2 2. Implement and test the server

Upload your trained model (`models/atis_joint_model.c2`) and the server files (`demo/flask_server/*`) to the instance using *scp*.

The server handles a *GET* request with a *text* field by running it through the model and dumping the output back to a JSON.

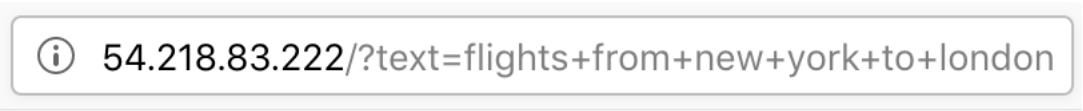
```
@app.route('/')
def predict():
    return json.dumps(atis.predict(request.args.get('text', '')))

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

The code in `demo/flask_server/atis.py` does the pre-processing (tokenization) and post-processing (extract spans of city names) specific to the ATIS model.

Run the server using

```
$ python server.py
```

Test it out by finding your IPv4 Public IP on the EC2 Management Console page and pointing your browser to it. The server will respond with the character spans of the city names e.g.




$$[[13, 21], [25, 31]]$$

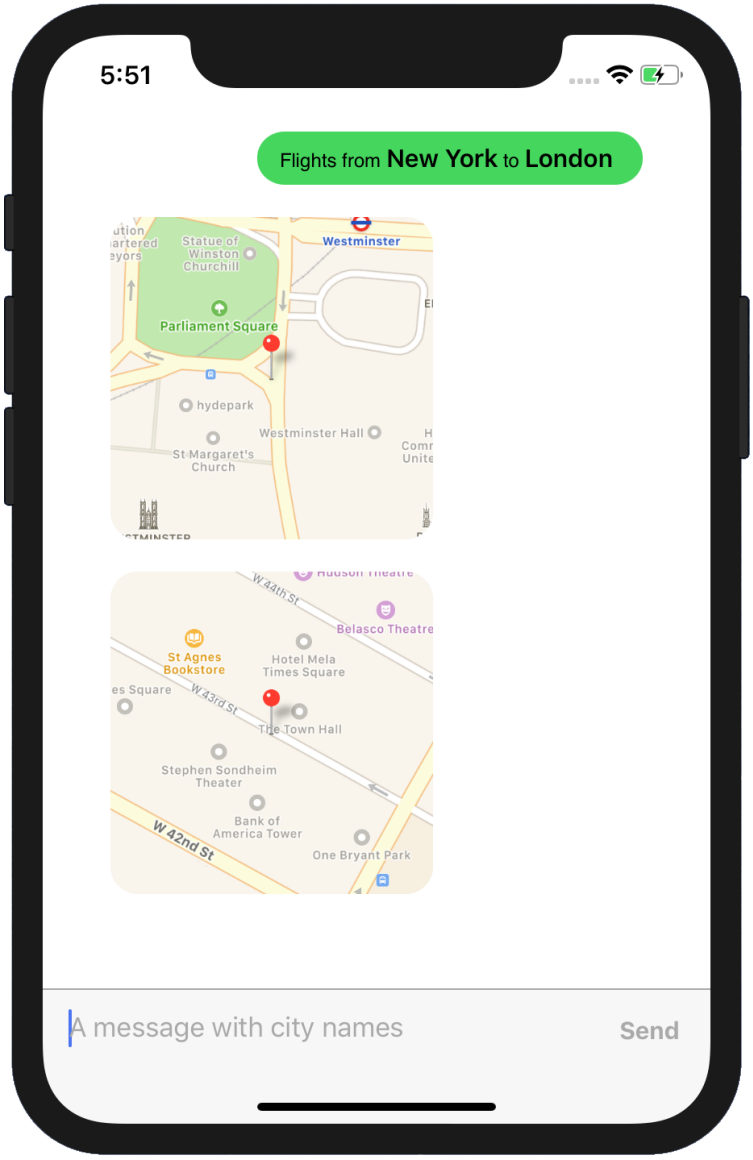
### 1.5.3 3. Implement the iOS app

Install Xcode and CocoaPods if you haven't already.

We use the open-source MessageKit to bootstrap our iOS app. Clone the app from our sister repository, and run -

```
$ pod install
$ open PyTextATIS.workspace
```

The comments in `ViewController.swift` explain the modifications over the base code. Change the IP address in that file to your instance's and run the app!




## 1.6 Serve Models in Production

We have seen how to use PyText models in an app using Flask in the previous tutorial, but the server implementation still requires a Python runtime. TorchScript models are designed to perform well even in production scenarios with high requirements for performance and scalability.

In this tutorial, we will implement a Thrift server in C++, in order to extract the maximum performance from our exported TorchScript text classification model trained on the demo dataset. We will also prepare a Docker image which can be deployed to your cloud provider of choice.

The full source code for the implemented server in this tutorial can be found in the demos directory.

To complete this tutorial, you will need to have Docker installed.

### 1.6.1 1. Create a Dockerfile and install dependencies

The first step is to prepare our Docker image with the necessary dependencies (including libtorch for running TorchScript models). In an empty, folder, create a *Dockerfile* with the following contents:

**Dockerfile**

```
FROM ubuntu:18.04

# Install dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
  build-essential \
  ca-certificates \
  cmake \
  curl \
  git \
  libcurl4-openssl-dev \
  libgflags-dev \
  unzip

# Install libtorch
WORKDIR /
RUN curl https://download.pytorch.org/libtorch/cpu/libtorch-cxx11-abi-shared-with-
↪deps-1.4.0%2Bcpu.zip --output libtorch.zip \
    && unzip libtorch.zip \
    && rm libtorch.zip
```

### 1.6.2 2. Add Thrift API

Thrift is a software library for developing scalable cross-language services. It comes with a client code generation engine enabling services to be interfaced across the network on multiple languages or devices. We will use Thrift to create a service which serves our model.

Add the dependency on Thrift in the Dockerfile:

```
# Install Thrift + dependencies
WORKDIR /
RUN apt-get update && apt-get install -y \
    libboost-dev \
    libboost-test-dev \
    libboost-program-options-dev \
    libboost-filesystem-dev \
    libboost-thread-dev \
    libevent-dev \
    automake \
    libtool \
    flex \
    bison \
    pkg-config \
    libssl-dev \
    && rm -rf /var/lib/apt/lists/*
RUN curl https://downloads.apache.org/thrift/0.13.0/thrift-0.13.0.tar.gz --output␣
↪thrift-0.13.0.tar.gz \
    && tar -xvf thrift-0.13.0.tar.gz \
    && rm thrift-0.13.0.tar.gz
WORKDIR /thrift-0.13.0
RUN ./bootstrap.sh \
```

(continues on next page)

(continued from previous page)

```
    && ./configure \
    && make \
    && make install
```

Our C++ server will expose a very simple API that receives a sentence/utterance as a string, and return a map of label names(*string*) -> scores(*double*). The corresponding thrift spec fo the API is below:

**predictor.thrift**

```
namespace cpp predictor_service

service Predictor {
  // Returns scores for each class
  map<string,double> predict(1:string doc),
}
```

### 1.6.3 3. Implement server code

Now, we will write our server's code. The first thing our server needs to be able to do is to load the model from a file path into the Caffe2 workspace and initialize it. We do that in the constructor of our `PredictorHandler` thrift server class:

**server.cpp**

```
class PredictorHandler : virtual public PredictorIf {
  private:
    torch::jit::script::Module mModule;
...
  public:
    PredictorHandler(string& modelFile) {
      mModule = torch::jit::load(modelFile);
    }
...
}
```

Now that our model is loaded, we need to implement the *predict* API method which is our main interface to clients. The implementation needs to do the following:

1. Pre-process the input sentence into tokens
2. Prepare input for the model as a batch
3. Run the model
4. Extract and populate the results into the response

**server.cpp**

```
class PredictorHandler : virtual public PredictorIf {
...
  public:
    void predict(map<string, double>& _return, const string& doc) {
      // Pre-process: tokenize input doc
      vector<string> tokens;
      string docCopy = doc;
      tokenize(tokens, docCopy);
```

(continues on next page)

(continued from previous page)

```
      // Prepare input for the model as a batch
      vector<vector<string>> batch{tokens};
      vector<torch::jit::IValue> inputs{
          mDummyVec, // texts in model.forward
          mDummyVecVec, // multi_texts in model.forward
          batch // tokens in model.forward
      };

      // Run the model
      auto output =
          mModule.forward(inputs).toGenericListRef().at(0).toGenericDict();

      // Extract and populate results into the response
      for (const auto& elem : output) {
        _return.insert({elem.key().toStringRef(), elem.value().toDouble()});
      }
    }
...
}
```

The full source code for *server.cpp* can be found here.

Note: The source code in the demo also implements a REST proxy for the Thrift server to make it easy to test and make calls over simple HTTP. Feel free to use that if you don't need to pass raw tensors into your model.

### 1.6.4 4. Build and compile scripts

To build our server, we need to provide necessary headers during compile time and the required dependent libraries during link time. The *Makefile* below does this:

**Makefile**

```
CPPFLAGS += -g -std=c++11 -std=c++14 \
  -I./gen-cpp \
  -I/libtorch/include \
  -Wno-deprecated-declarations
CLIENT_LDFLAGS += -lthrift
SERVER_LDFLAGS += -L/libtorch/lib \
  -lthrift -lpistache -lpthread -ltorch -lc10 -lcurl -lgflags

server: server.o gen-cpp/Predictor.o
  g++ $^ $(SERVER_LDFLAGS) -o $@

clean:
  rm -f *.o server
```

In our *Dockerfile*, we also add some steps to copy our local files into the docker image, compile the app, and add the necessary library search paths.

**Dockerfile**

```
# Copy local files to /app
COPY . /app
WORKDIR /app

# Compile app
```

(continues on next page)

(continued from previous page)

```
RUN thrift -r --gen cpp predictor.thrift
RUN make

# Add library search paths
ENV LD_LIBRARY_PATH /libtorch/lib:/usr/local/lib
```

### 1.6.5 5. Test/Run the server

To obtain a sample TorchScript model, run the following commads in your PyText directory:

```
(pytext) $ pytext train < demo/configs/docnn.json
(pytext) $ pytext torchscript-export < demo/configs/docnn.json
```

This creates a */tmp/model.pt.torchscript* file which you should copy into the server directory where you wrote the files in the previous section. This section assumes that this directory matches the one found here.

1. Build the Docker image:

```
$ docker build -t predictor_service .
```

If successful, you should see the message "Successfully tagged predictor_service:latest".

2. Run the server:

```
$ docker run -it -p 8080:8080 predictor_service:latest ./server model.pt.torchscript
```

If successful, you should see the message "Server running. Thrift port: 9090, REST port: 8080"

3. Test our server by sending a test utterance "set an alarm":

```
$ curl -G "http://localhost:8080" --data-urlencode "doc=set an alarm"
```

If successful, you should see the scores printed out on the console. On further inspection, the score for "alarm/set_alarm" is the highest among the classes.

```
alarm/modify_alarm:-1.99205
alarm/set_alarm:-1.8802
alarm/snooze_alarm:-1.89931
alarm/time_left_on_alarm:-2.00953
reminder/set_reminder:-2.00718
reminder/show_reminders:-1.91181
weather/find:-1.93019
```

Congratulations! You have now built your own server that can serve your PyText models in production!

We also provide a Docker image on Docker Hub with this example, which you can freely use and adapt to your needs.

## 1.7 Config Files Explained

PyText Models and training Tasks contain many components, and each components expects many parameters to define their behavior. PyText uses a `config` to specify those parameters. The `config` is can be loaded from a JSON file, which is what we describe here.

### 1.7.1 Structure of a Config File

A typical `config` file only contains the parameters specific to your project. Here's a fully working JSON file, and it does not need to be more complicated than this:

```
{
    "task": {
        "DocumentClassificationTask": {
            "data": {
              "source": {
                "TSVDataSource": {
                  "field_names": ["label", "text"],
                  "train_filename": "my/data/train.tsv",
                  "eval_filename": "my/data/eval.tsv",
                  "test_filename": "my/data/test.tsv"
                }
            },
            "model": {
                "embedding": {
                    "embed_dim": 200
                }
            }
        }
    },
    "version": 15
}
```

At the top level, the most important settings are the "task" and the "version". "task" defines the `Task` component to be used, which specifies where to get the "data", which "model" to train, which "trainer" to use, and which "metric_reporter" will present the results.

Each of those parameters can be a `Component` that is specified by its class name, or omitted to use the default class with its default parameters. In the example above, we specify `TSVDataSource` to use this class, but we skip the model class name because we want to use the default `DocModel`.

The "version" number helps PyText maintain backwards compatibility. PyText will use *config adapters* to internally try and update the configs to match the latest component parameters so you don't have to keep changing your configs at each PyText update. To manually update your config to the latest version, you can use the update-config command.

### 1.7.2 Parameters in Config File

Parameters are either a component or a value. In the config above, we see that "field_names" expects a list of strings, "train_filename" expects a string, and "embed_dim" expects an integer.

"source" and "model" however expect a component, and as we've seen in the previous section, we can optionally specify the class name of a component if we decide to use a component that is not the default. We can tell whether it's a class name or a parameter name by looking at the first letter: class names start with an upper case letter. For "source" we decided to specify `TSVDataSource`, but for "model" we did not and decided to let `DocumentClassificationTask` use its default `DocModel`. We could have specified the class name like this, and that would be equivalent:

```
"model": {
    "DocModel": {
        "embedding": {
            "embed_dim": 100
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

In the next example, the default representation for `DocModel` is `BiLSTMDocAttention`. We did not specify "representation" before because we were happy with this default. But if we decide to use `DocNNRepresentation` instead, we would modify the config like this:

```
"model": {
    "embedding": {
        "embed_dim": 100
    },
    "representation": {
        "DocNNRepresentation": {
        }
    }
}
```

In this example we just want to change the class of "representation" and use its default parameters, so we don't need to specify any of them and we can leave its parameters set empty *{}*.

To explore more components parameters and their possible values, you can use the help-config command or browse the class documentation.

### 1.7.3 Changing a Config File

Users typically start with an existing config file, or create one using the gen-default-config command, and then edit it to tweak the parameters.

The file generated by `gen-default-config` is very large, because it contains the default value of every parameter for every component. Any of those parameters can be omitted from the config file, because PyText can recover their default values.

In general, you should remove from your config file all the parameters you don't want to override and keep those you do want to override now, or you might want to tweak later.

For example, `TSVDataSource` can use a different "delimiter", but in most cases we want to use the default "\t" for tab-separated-values files (TSV), so the config above does not specify *"delimiter": "\t"*. If we wanted to load a CVS file, we could override this default by adding our own "delimiter" to our config (and since CVS fields can be "quoted", unlike TSV where this option's default is *false*, we'd also override it with *true*.)

```
"TSVDataSource": {
    "delimiter": ",",
    "quoted": true,
    "field_names": ["label", "text"],
    "train_filename": "my/data/train.csv",
    "eval_filename": "my/data/eval.csv",
    "test_filename": "my/data/test.csv"
}
```

The config at the top of this page is a fully working example. It could be simplified even further by removing the "model" section if you don't want to change any of the model parameters, but in this case I guess the author decided to tweak "embed_dim".

### 1.7.4 JSON Format Primer

A few notes about the JSON syntax and the differences with python:

- field names and string values should all be quoted with *"double-quotes"*
- booleans are lower case: *true*, *false*
- no trailing comma (after the last value of a block)
- empty value is: *null*
- indentation is optional but recommended for readability
- the first character must be *{* and the last one must be *}*
- obviously all brackets must be balanced: *{}*, *[]*

## 1.8 Config Commands

This page explains the usage of the commands `help-config` to explore PyText components, and `gen-default-config` to create a config file with custom components and parameters.

### 1.8.1 Exploring Config Options

You can explore PyText Components with the command `help-config`. This will print the documentation of the component, its full module name, its base class, as well as the list of its config parameters, their type and their default value.

```
$ pytext help-config LMTask
=== pytext.task.tasks.LMTask (NewTask) ===
    data = Data
    exporter = null
    features = FeatureConfig
    featurizer = SimpleFeaturizer
    metric_reporter: LanguageModelMetricReporter = LanguageModelMetricReporter
    model: LMLSTM = LMLSTM
    trainer = TaskTrainer
```

You can drill down to the component you're interested in. For example, if you want to know more about the model `LMLSTM`, you can use the same command. Notice how PyText lists the possible values for *Union* types (for example with *representation* below.)

```
$ pytext help-config LMLSTM
=== pytext.models.language_models.lmlstm.LMLSTM (BaseModel) ===
"""
`LMLSTM` implements a word-level language model that uses LSTMs to
    represent the document.
"""
    ModelInput = LMLSTM.Config.ModelInput
    caffe2_format: (ExporterType)
         PREDICTOR (default)
         INIT_PREDICT
    decoder: (one of)
         None
         MLPDecoder (default)
```

(continues on next page)

(continued from previous page)

```
    embedding: WordFeatConfig = WordEmbedding
    inputs: LMLSTM.Config.ModelInput = ModelInput
    output_layer: LMOutputLayer = LMOutputLayer
    representation: (one of)
         DeepCNNRepresentation
         BiLSTM (default)
    stateful: bool
    tied_weights: bool
```

PyText internally registers all the component classes, so we can look up and find any component using the class name or their aliases. For example somewhere in PyText we have `import DeepCNNRepresentation as CNN`, so we would normally look up `DeepCNNRepresentation`, but if we know that this class has an alias we can look up `CNN` instead, and print the information about this class:

```
$ pytext help-config CNN
=== pytext.models.representations.deepcnn.DeepCNNRepresentation (RepresentationBase)␣
↪===
"""
`DeepCNNRepresentation` implements CNN representation layer
    preceded by a dropout layer. CNN representation layer is based on the encoder
    in the architecture proposed by Gehring et. al. in Convolutional Sequence to
    Sequence Learning.

    Args:
        config (Config): Configuration object of type DeepCNNRepresentation.Config.
        embed_dim (int): The number of expected features in the input.
"""
    cnn: CNNParams = CNNParams
    dropout: float = 0.3
```

### 1.8.2 Creating a Config File

The command `gen-default-config` creates a json config files for a given `Task` using the default value for all the parameters. You must specify the class name of the `Task`. The json config will be printed in the terminal, so you need to send it to a file using of your choice (for example `my_config.json`) to be able to edit it and use it.

```
$ pytext gen-default-config LMTask > my_config.json
INFO - Applying task option: LMTask
...
```

In the `help-config LMLSTM` above, we see that *representation* is by default `BiLSTM`, but could also be `DeepCNNRepresentation`. (This can be because the type is declared as a *Union* of valid alternatives, or because the type is a base class.) Those two classes will have different parameters, so we can't just edit the *my_config.json* and replace the class name.

We can specify which components to use by adding any number of class names to the command. Let's create this config, and we'll use add `DeepCNNRepresentation` to our command. `gen-default-config` will look up this class name and find that it is a suitable *representation* component for the `LMLSTM` model in our `LMTask`.

```
$ pytext gen-default-config LMTask DeepCNNRepresentation > my_config.json
INFO - Applying task option: LMTask
INFO - Applying class option: task->model->representation = CNN
...
```

This also works with parameters which are not component class names. You can specify the parameter name and its value, and `gen-default-config` will automatically apply this parameter to the right component.

```
$ pytext gen-default-config LMTask epochs=200
INFO - Applying task option: LMTask
INFO - Applying parameter option to task.trainer.epochs : epochs=200
...
```

Sometimes the same parameter name is used by multiple components. In this case PyText prints the list of those parameters with their full config path. You can then simply use the last part of the path that is enough to differentiate them and pick the one you want. In the next example, we omit the prefix *task.model.* because we don't need it to find where to apply our parameter *representation.dropout*.

```
$ pytext gen-default-config LMTask dropout=0.7 > my_config.json
INFO - Applying task option: LMTask
...
Exception: Multiple possibilities for dropout=0.7: task.model.representation.dropout,␣
↪task.model.decoder.dropout

$ pytext gen-default-config LMTask representation.dropout=0.7 > my_config.json
INFO - Applying task option: LMTask
INFO - Applying parameter option to task.model.representation.dropout :␣
↪representation.dropout=0.7
...
```

You can add any number and combination of those parameters. Please note that they will be applied in order, so if you want to change a component class and some of its parameters, you must specify the parameters in this order (component first, then parameters). If you don't do that, your parameters changes will be ignored. For example, changing *representation.dropout* first, then overriding the representation component will replace the default representation with a new `CNN` component with all the parameter using the default value.

Look at this bad example: you can verify that the representation dropout is 0.3 (the default value for `CNN`) and not 0.7 as we specified, because CNN was applied after and replaced the component that had its dropout modified first.

```
$ pytext gen-default-config LMTask representation.dropout=0.7 CNN > my_config.json
INFO - Applying task option: LMTask
INFO - Applying parameter option to task.model.representation.dropout :␣
↪representation.dropout=0.7
INFO - Applying class option: task->model->representation = CNN
...
```

Now let's combine everything:

```
$ pytext gen-default-config LMTask BlockShardedTSVDataSource CNN dilated=True␣
↪epochs=200 representation.dropout=0.7 > my_config.json
INFO - Applying task option: LMTask
INFO - Applying class option: task->data->source = BlockShardedTSVDataSource
INFO - Applying class option: task->model->representation = CNN
INFO - Applying parameter option to task.model.representation.cnn.dilated :␣
↪dilated=True
INFO - Applying parameter option to task.trainer.epochs : epochs=200
INFO - Applying parameter option to task.model.representation.dropout :␣
↪representation.dropout=0.2
...
```

### 1.8.3 Updating a Config File

When there's a new release of PyText, some component parameters might change because of bug fixes or new features. While PyText has *config_adapters* that can internally transform old configs to map them to the latest components, it is sometimes useful to update your config file to the current version. This can be done with the command `update-config`:

```
$ pytext update-config < my_config_old.json > my_config_new.json
```

## 1.9 Train Intent-Slot model on ATIS Dataset

**OBSOLETE** This documentation is using the old API and needs to be updated with the new classes configs.

Intent detection and Slot filling are two common tasks in Natural Language Understanding for personal assistants. Given a user's "utterance" (e.g. Set an alarm for 10 pm), we detect its intent (set_alarm) and tag the slots required to fulfill the intent (10 pm).

The two tasks can be modeled as text classification and sequence labeling, respectively. We can train two separate models, but training a joint model has been shown to perform better.

In this tutorial, we will train a joint intent-slot model in PyText on the ATIS (Airline Travel Information System) dataset. Note that to download the dataset, you will need a Kaggle account for which you can sign up for free.

### 1.9.1 1. Prepare the data

The in-built PyText data-handler expects the data to be stored in a tab-separated file that contains the intent label, slot label and the raw utterance.

Download the data locally and use the script below to preprocess it into format PyText expects

```
$ unzip <download_dir>/atis.zip -d <download_dir>/atis
$ python3 demo/atis_joint_model/data_processor.py
  --download-folder <download_dir>/atis --output-directory demo/atis_joint_model/
```

The script will also randomly split the training data into training and validation sets. All the pre-processed data will be written to the output-directory argument specified in the command.

An alternative approach here would be to write a custom data-handler for your custom data format, but that is beyond the scope of this tutorial.

### 1.9.2 2. Download Pre-trained word embeddings

Word embeddings are the vector representations of the different words understood by your model. Pre-trained word embeddings can significantly improve the accuracy of your model, since they have been trained on vast amounts of data. In this tutorial, we'll use GloVe embeddings, which can be downloaded by:

```
$ curl https://nlp.stanford.edu/data/wordvecs/glove.6B.zip > demo/atis_joint_model/
↪glove.6B.zip
$ unzip demo/atis_joint_model/glove.6B.zip -d demo/atis_joint_model
```

The downloaded file size is ~800 MB.

### 1.9.3 3. Train the model

To train a PyText model, you need to pick the right task and model architecture, among other parameters. Default values are available for many parameters and can give reasonable results in most cases. The following is a sample config which can train a joint intent-slot model

```
{
  "config": {
    "task": {
      "IntentSlotTask": {
        "data": {
          "Data": {
            "source": {
              "TSVDataSource": {
                "field_names": [
                  "label",
                  "slots",
                  "text",
                  "doc_weight",
                  "word_weight"
                ],
                "train_filename": "demo/atis_joint_model/atis.processed.train.csv",
                "eval_filename": "demo/atis_joint_model/atis.processed.val.csv",
                "test_filename": "demo/atis_joint_model/atis.processed.test.csv"
              }
            },
            "batcher": {
              "PoolingBatcher": {
                "train_batch_size": 128,
                "eval_batch_size": 128,
                "test_batch_size": 128,
                "pool_num_batches": 10000
              }
            },
            "sort_key": "tokens",
            "in_memory": true
          }
        },
        "model": {
          "representation": {
            "BiLSTMDocSlotAttention": {
              "pooling": {
                "SelfAttention": {}
              }
            }
          },
          "output_layer": {
            "doc_output": {
              "loss": {
                "CrossEntropyLoss": {}
              }
            },
            "word_output": {
              "CRFOutputLayer": {}
            }
          },
          "word_embedding": {
            "embed_dim": 100,
```

(continues on next page)

(continued from previous page)

```
            "pretrained_embeddings_path": "demo/atis_joint_model/glove.6B.100d.txt"
          }
        },
        "trainer": {
          "epochs": 20,
          "optimizer": {
            "Adam": {
              "lr": 0.001
            }
          }
        }
      }
    }
  }
}
```

We explain some of the parameters involved:

- `IntentSlotTask` trains a joint model for document classification and word tagging.
- The `Model` has multiple layers - - We use BiLSTM model with attention as the representation layer. The pooling attribute decides the attention technique used. - We use different loss functions for document classification (Cross Entropy Loss) and slot filling (CRF layer)
- Pre-trained word embeddings are provided within the *word_embedding* attribute.

To train the PyText model,

```
(pytext) $ pytext train < sample_config.json
```

### 1.9.4 3. Tune the model and get final results

Tuning the model's hyper-parameters is key to obtaining the best model accuracy. Using hyper-parameter sweeps on learning rate, number of layers, dimension and dropout of BiLSTM etc., we can achieve a F1 score of ~95% on slot labels which is close to the state-of-the-art. The fine-tuned model config is available at `demos/atis_intent_slot/atis_joint_config.json`

To train the model using fine tuned model config,

```
(pytext) $ pytext train < demo/atis_joint_model/atis_joint_config.json
```

### 1.9.5 4. Generate predictions

Lets make the model run on some sample utterances! You can input one by running

```
(pytext) $ pytext --config-file demo/atis_joint_model/atis_joint_config.json \
  predict --exported-model /tmp/atis_joint_model.c2 <<< '{"text": "flights from␣
↪colorado"}'
```

The response from the model is log of probabilities for different intents and slots, with the correct intent and slot hopefully having the highest.

In the following snippet of the model's response, we see that the intent *doc_scores:flight* and slot *word_scores:fromloc.city_name* for third word "colorado" have the highest predictions.

```
{
 ....
 'doc_scores:flight': array([-0.00016726], dtype=float32),
 'doc_scores:ground_service+ground_fare': array([-25.865768], dtype=float32),
 'doc_scores:meal': array([-17.864975], dtype=float32),
 ..,
 'word_scores:airline_name': array([[-12.158762],
       [-15.142928],
       [ -8.991585]], dtype=float32),
 'word_scores:fromloc.city_name': array([[-1.5084317e+01],
       [-1.3880151e+01],
       [-1.4416825e-02]], dtype=float32),
 'word_scores:fromloc.state_code': array([[-17.824356],
       [-17.89767 ],
       [ -9.848984]], dtype=float32),
 'word_scores:meal': array([[-15.079164],
       [-17.229427],
       [-17.529446]], dtype=float32),
 'word_scores:transport_type': array([[-14.722928],
       [-16.700478],
       [-13.4414  ]], dtype=float32),
 ...
}
```

## 1.10 Hierarchical intent and slot filling

In this tutorial, we will train a semantic parser for task oriented dialog by modeling hierarchical intents and slots (Gupta et al. , Semantic Parsing for Task Oriented Dialog using Hierarchical Representations, EMNLP 2018). The underlying model used in the paper is the Recurrent Neural Network Grammar (Dyer et al., Recurrent Neural Network Grammar, NAACL 2016). RNNG is neural constituency parser that explicitly models the compositional tree structure of the words and phrases in an utterance.

### 1.10.1 1. Fetch the dataset

Download the dataset to a local directory. We will refer to this as *base_dir* in the next section.

```
$ curl -o top-dataset-semantic-parsing.zip -L https://fb.me/semanticparsingdialog
$ unzip top-dataset-semantic-parsing.zip
```

### 1.10.2 2. Prepare configuration file

Prepare the configuration file for training. A sample config file can be found in your PyText repository at `demo/configs/rnng.json`. If you haven't set up PyText, please follow *Installation*, then make the following changes in the config:

- Set *train_path* to *base_dir/top-dataset-semantic-parsing/train.tsv*.
- Set *eval_path* to *base_dir/top-dataset-semantic-parsing/eval.tsv*.
- Set *test_path* to *base_dir/top-dataset-semantic-parsing/test.tsv*.

### 1.10.3 3. Train a model with the downloaded dataset

Train the model using the command below

```
(pytext) $ pytext train < demo/configs/rnng.json
```

The output will look like:

```
Merged Intent and Slot Metrics
P = 24.03 R = 31.90, F1 = 27.41.
```

This will take about hour. If you want to train with a smaller dataset to make it quick then generate a subset of the dataset using the commands below and update the paths in `demo/configs/rnng.json`:

```
$ head -n 1000 base_dir/top-dataset-semantic-parsing/train.tsv > base_dir/top-dataset-
↪semantic-parsing/train_small.tsv
$ head -n 100 base_dir/top-dataset-semantic-parsing/eval.tsv > base_dir/top-dataset-
↪semantic-parsing/eval_small.tsv
$ head -n 100 base_dir/top-dataset-semantic-parsing/test.tsv > base_dir/top-dataset-
↪semantic-parsing/test_small.tsv
```

If you now train the model with smaller datasets, the output will look like:

```
Merged Intent and Slot Metrics
P = 24.03 R = 31.90, F1 = 27.41.
```

### 1.10.4 4. Test the model interactively against input utterances.

Load the model using the command below

```
(pytext) $ pytext predict-py --model-file=/tmp/model.pt
please input a json example, the names should be the same with column_to_read in␣
↪model training config:
```

This will give you a REPL prompt. You can enter an utterance to get back the model's prediction repeatedly. You should enter in a json format shown below. Once done press Ctrl+D.

```
{"text": "order coffee from starbucks"}
```

You should see an output like:

```
[{'prediction': [7, 0, 5, 0, 1, 0, 3, 0, 1, 1],
'score': [
        0.44425372408062447,
        0.8018286800064633,
        0.6880680051949267,
        0.9891564979506277,
        0.9999506231665385,
        0.9992705616574005,
        0.34512090135492923,
        0.9999979545618913,
        0.9999998668826438,
        0.9999998686418744]}]
```

We have also provided a pre-trained model which you may download here

## 1.11 Multitask training with disjoint datasets

In this tutorial, we will jointly train a classification task with a language modeling task in a multitask setting. The models will share the embedding and representation layers.

We will use the following datasets:

1. Binarized Stanford Sentiment Treebank (SST-2), which is part of the GLUE benchmark. This dataset contains segments from movie reviews labeled with their binary sentiment.
2. WikiText-2, a medium-size language modeling dataset with text extracted from Wikipedia.

### 1.11.1 1. Fetch and prepare the dataset

Download the dataset in a local directory. We will refer to this as *base_dir* in the next section.

```
$ curl "https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-2-v1.zip" -o␣
↪wikitext-2-v1.zip
$ unzip wikitext-2-v1.zip
$ curl "https://firebasestorage.googleapis.com/v0/b/mtl-sentence-representations.
↪appspot.com/o/data%2FSST-2.zip?alt=media&token=aabc5f6b-e466-44a2-b9b4-cf6337f84ac8
↪" -o SST-2.zip
$ unzip SST-2.zip
```

Remove headers from SST-2 data:

```
$ cd base_dir/SST-2
$ sed -i '1d' train.tsv
$ sed -i '1d' dev.tsv
```

Remove empty lines from WikiText:

```
$ cd base_dir/wikitext-2
$ sed -i '/^\s*$/d' train.tsv
$ sed -i '/^\s*$/d' valid.tsv
$ sed -i '/^\s*$/d' test.tsv
```

### 1.11.2 2. Train a base model

Prepare the configuration file for training. A sample config file for the base document classification model can be found in your PyText repository at `demo/configs/sst2.json`. If you haven't set up PyText, please follow *Installation*, then make the following changes in the config:

- Set *train_path* to *base_dir/SST-2/train.tsv*.
- Set *eval_path* to *base_dir/SST-2/eval.tsv*.
- Set *test_path* to *base_dir/SST-2/test.tsv*.

The test set labels for this tasks are not openly available, therefore we will use the dev set. Train the model using the command below.

```
(pytext) $ pytext train < demo/configs/sst2.json
```

The output will look like:

```
Stage.EVAL
loss: 0.472868
Accuracy: 85.67
```

### 1.11.3 3. Configure for multitasking

The example configuration for this tutorial is at `demo/configs/multitask_sst_lm.json`. The main configuration is under *tasks*, which is a dictionary of task name to task config:

```
      "task_weights": {
              "SST2": 1,
              "LM": 1
      },
"tasks": {
  "SST2": {
    "DocClassificationTask": { ... }
  },
  "LM": {
    "LMTask": { ... }
  }
}
```

You can also modify *task_weights* to weight the loss for each task. The sub-tasks can be configured as you would in a single task setting, with the exception of changes described in the next sections.

### 1.11.4 3. Specify which parameters to share

Parameter sharing is specified at module level with the *shared_module_key* parameter, which is an arbitrary string. Modules with identical *shared_module_key* share parameters.

Here we will share the BiLSTM module. Under the *SST* task, we set

```
"representation": {
  "BiLSTMDocAttention": {
    "lstm": {
      "shared_module_key": "SHARED_LSTM"
    }
  }
}
```

Under the *LM* task, we set

```
"representation": {
  "shared_module_key": "SHARED_LSTM"
},
```

In this case, *BiLSTMDocAttention.lstm* of `DocClassificationTask` and *representation* of `LMTask` are both of type *BiLSTM*, therefore parameter sharing is possible.

### 1.11.5 3. Share the embedding layer

The embedding is also a module, and can be similarly shared. This is configured under the *features* section. However, we need to ensure that we use the same vocabulary for both tasks, by specifying a pre-built vocabulary file. First create the vocabulary from the classification task data:

```
$ cd base_dir/SST-2
$ cat train.tsv dev.tsv | tr ' ' '\n' | sort | uniq > sst_vocab.txt
```

Then point to this file in configuration:

```
"features": {
    "shared_module_key": "SHARED_EMBEDDING",
    "word_feat": {
      "vocab_file": "base_dir/SST-2/sst_vocab.txt",
      "vocab_size": 15000,
      "vocab_from_train_data": false
    }
  }
```

### 1.11.6 3. Train the model

You can train the model with

```
(pytext) $ pytext train < demo/configs/multitask_sst_lm.json
```

The output will look like

```
Stage.EVAL
loss: 0.455871
Accuracy: 86.12
```

Not a great improvement, but we used a very primitive language modeling task (bi-directional with no masking) for the purposes of this tutorial. Happy multitasking!

## 1.12 Data Parallel Distributed Training

Distributed training enables one to easily parallelize computations across processes and clusters of machines. To do so, it leverages messaging passing semantics allowing each process to communicate data to any of the other processes.

PyText exploits `DistributedDataParallel` for synchronizing gradients and `torch.multiprocessing` to spawn multiple processes which each setup the distributed environment with NCCL as default backend, initialize the process group, and finally execute the given run function. The module is replicated on each machine and each device (e.g every single process), and each such replica handles a portion of the input partitioned by PyText's `DataHandler`. For more on distributed training in PyTorch, refer to Writing distributed applications with PyTorch.

In this tutorial, we will train a DocNN model on a single node with 8 GPUs using the SST dataset.

### 1.12.1 1. Requirement

Distributed training is only available for GPUs, so you'll need GPU-equipped server or virtual machine to run this tutorial.

**Notes:**

- This demo use a local temporary file for initializing the distributed processes group, which means it only works on a single node. Please make sure to set *distributed_world_size* less than or equal to the maximum available GPUs on the server.

- For distributed training on clusters of machines, you can use a shared file accessible to all the hosts (ex: file:///mnt/nfs/sharedfile) or the TCP init method. More info on distributed initialization.
- In `demo/configs/distributed_docnn.json`, set *distributed_world_size* to 1 to disable distributed training, and set *use_cuda_if_available* to *false* to disable training on GPU.

### 1.12.2 2. Fetch the dataset

Download the SST dataset (The Stanford Sentiment Treebank) to a local directory. We will refer to this as *base_dir* in the next section.

```
$ unzip SST-2.zip && cd SST-2
$ sed 1d train.tsv | head -1000 > train_tiny.tsv
$ sed 1d dev.tsv | head -100 > eval_tiny.tsv
```

### 1.12.3 3. Prepare configuration file

Prepare the configuration file for training. A sample config file can be found in your PyText repository at `demo/configs/distributed_docnn.json`. If you haven't set up PyText, please follow *Installation*.

The two parameters that are used for distributed training are:

- *distributed_world_size*: total number of GPUs used for distributed training, e.g. if set to 40 with every server having 8 GPU, 5 servers will be fully used.
- *use_cuda_if_available*: set to *true* for training on GPUs.

For this tutorial, please change the following in the config file.

- Set *train_path* to *base_dir/train_tiny.tsv*.
- Set *eval_path* to *base_dir/eval_tiny.tsv*.
- Set *test_path* to *base_dir/eval_tiny.tsv*.

### 1.12.4 4. Train model with the downloaded dataset

Train the model using the command below

```
(pytext) $ pytext train < demo/configs/distributed_docnn.json
```

## 1.13 XLM-RoBERTa

### 1.13.1 Introduction

XLM-R (XLM-RoBERTa, Unsupervised Cross-lingual Representation Learning at Scale) is a scaled cross lingual sentence encoder. It is trained on `2.5T` of data across `100` languages data filtered from Common Crawl. XLM-R achieves state-of-the-arts results on multiple cross lingual benchmarks.

### 1.13.2 Tutorial

Tutorial in Notebook

Run the tutorial in Google Colab

### 1.13.3 Pre-trained models

| Model | Description | #params | vocab size | Download |
|---|---|---|---|---|
| `xlmr.base.v0` | XLM-R using the BERT-base architecture | 250M | 250k | xlm.base.v0.tar.gz |
| `xlmr.large.v0` | XLM-R using the BERT-large architecture | 560M | 250k | xlm.large.v0.tar.gz |

(Note: The above models are still under training, we will update the weights, once fully trained, the results are based on the above checkpoints.)

### 1.13.4 Results

XNLI (Conneau et al., 2018):

| Model | average | en | fr | es | de | el | bg | ru | tr | ar | vi | th | zh | hi | sw | ur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `roberta.large.mnli` *(TRANSLATE-TEST)* | 77.8 | 91.3 | 82.9 | 84.3 | 81.2 | 81.7 | 83.1 | 78.3 | 76.8 | 76.6 | 74.2 | 74.1 | 77.5 | 70.9 | 66.7 | 66.8 |
| `xlmr.large.v0` *(TRANSLATE-TRAIN-ALL)* | **82.4** | 88.7 | 85.2 | 85.6 | 84.6 | 83.6 | 85.5 | 82.4 | 81.6 | 80.9 | 83.4 | 80.9 | 83.3 | 79.8 | 75.9 | 74.3 |

MLQA (Lewis et al., 2018)

| Model | average | en | es | de | ar | hi | vi | zh |
|---|---|---|---|---|---|---|---|---|
| `BERT-large` | • | 80.2/67.4 | • | • | • | • | • | • |
| `mBERT` | 57.7 / 41.6 | 77.7 / 65.2 | 64.3 / 46.6 | 57.9 / 44.3 | 45.7 / 29.8 | 43.8 / 29.7 | 57.1 / 38.6 | 57.5 / 37.3 |
| `xlmr.large.v0` | **70.0 / 52.2** | 80.1 / 67.7 | 73.2 / 55.1 | 68.3 / 53.7 | 62.8 / 43.7 | 68.3 / 51.0 | 70.5 / 50.1 | 67.1 / 44.4 |

### 1.13.5 Citation

```bibtex
@article{
    title = {Unsupervised Cross-lingual Representation Learning at Scale},
    author = {Alexis Conneau and Kartikay Khandelwal
        and Naman Goyal and Vishrav Chaudhary and Guillaume Wenzek
        and Francisco Guzm\'an and Edouard Grave and Myle Ott
        and Luke Zettlemoyer and Veselin Stoyanov
    },
    journal={},
    year = {2019},
}
```

## 1.14 Semantic parsing with sequence-to-sequence models

### 1.14.1 Introduction

PyText provides an encoder-decoder framework that is suitable for any task that requires mapping a sequence of input tokens to a sequence of output tokens. The default implementation is based on recurrent neural networks (RNNs), which have been shown to be unreasonably effective at sequence processing tasks. The default implementation includes three major components

1. A bidirectional LSTM sequence encoder
2. An LSTM sequence decoder
3. A sequence generator that supports incremental decoding and beam search

All of these components are Torchscript-friendly, so that the trained model can be exported directly as-is. Following the general design of PyText, each of these components may be customized via their respective config objects or replaced entirely by custom components.

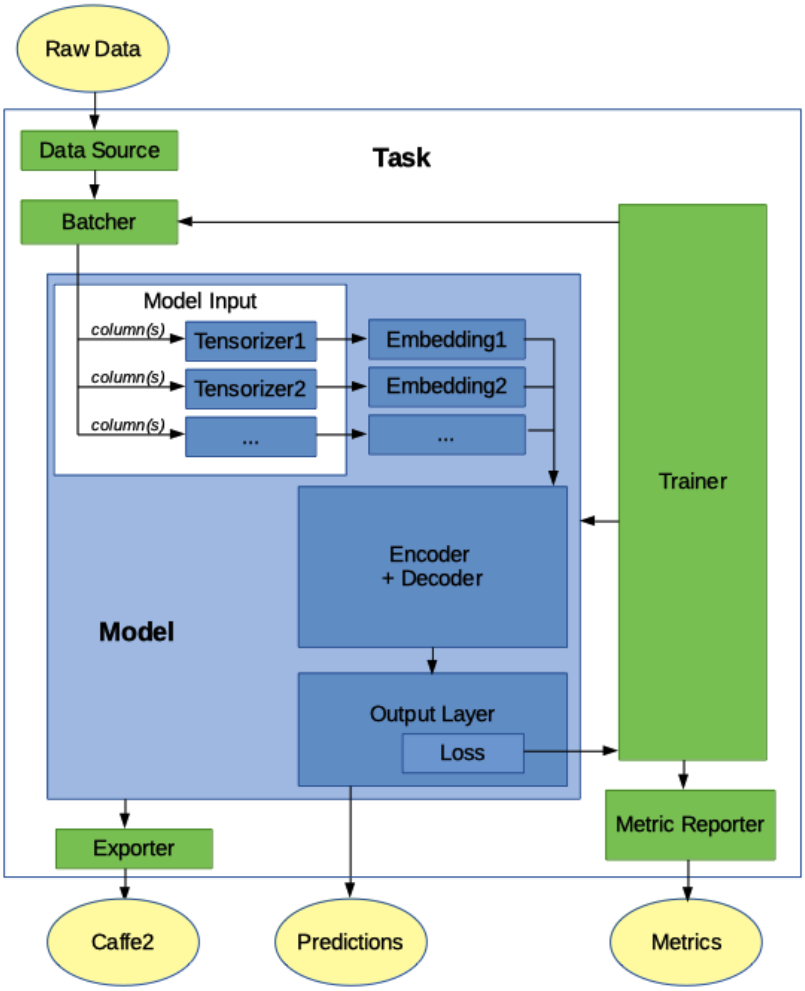
### 1.14.2 Tutorial

Tutorial in notebook Run the tutorial in Google Colab

## 1.15 Architecture Overview

PyText is designed to help users build end to end pipelines for training and inference. A number of default pipelines are implemented for popular tasks which can be used as-is. Users are free to extend or replace one or more of the pipelines's components.

The following figure describes the relationship between the major components of PyText:




Note: some models might implement a single "encoder_decoder" component while others implement two components: a representation and a decoder.

### 1.15.1 Model

The Model class is the central concept in PyText. It defines the neural network architecture. PyText provides models for common NLP jobs. Users can implement their custom model in two ways:

- subclassing `Model` will give you most of the functions for the common architecture *embedding -> representation -> decoder -> output_layer*.
- if you need more flexibility, you can subclass the more basic `BaseModel` which makes no assumptions about architectures, allowing you to implement any model.

Most PyText models implement `Model` and use the following architecture:

```
- model
  - model_input
    - tensorizers
  - embeddings
  - encoder+decoder
  - output_layer
```

(continues on next page)

(continued from previous page)

```
  - loss
  - prediction
```

- **model_input**: defines how the input strings will be transformed into tensors. This is done by input-specific "Tensorizers". For example, the `TokenTensorizer` takes a sentence, tokenize it and looks up in its vocabulary to create the corresponding tensor. (The vocabulary is created during initialization by doing a first pass on the inputs.) In addition to the inputs, we also define here how to handle other data that can be found in the input files, such as the "labels" (arguably an output, but true labels are used an input during training).
- **embeddings**: this step transforms the tensors created by model_input into embeddings. Each model_input (tensorizer) will be associated to a compatible embedding class (for example: `WordEmbedding`, or `CharacterEmbedding`). (see *pytext/models/embeddings/*)
- **representation**: also called "encoder", this can be one of the provided classes, such as those using a CNN (for example `DocNNRepresentation`), those using an LSTM (for example `BiLSTMDocAttention`), or any other type of representation. The parameters will depend on the representation selected. (see *pytext/models/representations/*)
- **decoder**: this is typically an MLP (Multi-Layer Perceptron). If you use the default `MLPDecoder`, *hidden_dims* is the most useful parameter, which is an array containing the number of nodes in each hidden layer. (see *pytext/models/decoders/*)
- **output_layer**: this is where the human-understandable output of the model is defined. For example, a document classification can automatically use the "labels" vocabulary defined in model_input as outputs. output_layer also defines the loss function to use during training. (see *pytext/models/output_layers/*)

### 1.15.2 Task: training definition

To train the model, we define a `Task`, which will tell PyText how to load the data, which model to use, how to train it, as well as the how to measure metrics.

The Task is defined with the following information:

- **data**: defines where to find and how to handle the data: see **data_source** and **batcher**.
- **data -> data_source**: The format of the input data (training, eval and testing) can differ a lot depending on the source. PyText provides `TSVDataSource` to read from the common tab-separated files. Users can easily write their own custom implementation if their files have a different format.
- **data -> batcher**: The batcher is responsible for grouping the input data into batches that will be processed one at a time. *train_batch_size*, *eval_batch_size* and *test_batch_size* can be changed to reduce the running time (while increasing the memory requirements). The default `Batcher` takes the input sequentially, which is adequat in most cases. Alternatively, *PoolingBatcher* shuffles the inputs to make sure the data is not in order, which could introduce a biais in the results.
- **trainer**: This defines a number of useful options for the training runs, like number of *epochs*, whether to *report_train_metrics* only during eval, and the *random_seed* to use.
- **metric_reporter**: different models will need to report different metrics. (For example, common metrics for document classification are precision, recall, f1 score.) Each PyText task can use a corresponding default metric reporter class, but users might want to use alternatives or implement their own.
- **exporter**: defines how to export the model so it can be used in production. PyText currently exports to caffe2 via onnx or torchscript.
- **model**: (see above)

### 1.15.3 How Data is Consumed

1. **data_source**: Defines where the data can be found (for example: one training file, one eval file, and one test file) and the schema (field names). The data_source class will read each entry one by one (for example: each line in a TSV file) and convert each one into a **row**, which is a python dict of field name to entry value. Values are converted automatically if their type is specified.
2. **tensorizer**: Defines how **rows** are transformed into tensors. Tensorizers listed in the model will use one or more fields in the **row** to create a tensor or a tuple of tensors. To do that, some tensorizers will split the field values using a tokenizer that can be overridden in the config. Tensorizers typically have a vocabulary that allows them to map words or labels to numbers, and it's built during the initialization phase by scanning the data once. (Alternatively, it can be loaded from file.)
3. **model -> arrange_model_inputs()**: At this point, we have a python dict of tensorizer name to tensor or a tuple of tensors. Model has the method arrange_model_inputs() which flattens this python dict into a list tensors or tuple of tensors in the right order for the Model's forward method.
4. **model -> forward()**: This is where the magic happens. Input tensors are passed to the embbedings forward methods, then the results are passed to the encoder/decoder forward methods, and finally the ouput layer produces a prediction.

### 1.15.4 Config Example

We only specify the options we want to override. Everything else will use the default values. A typical config might look like this:

```
{
  "task": {
    "MyTask": {
      "data": {
        "source": {
          "TSVDataSource": {
            "field_names": ["label", "slots", "text"],
            "train_filename": "data/my_train_data.tsv",
            "test_filename": "data/my_test_data.tsv",
            "eval_filename": "data/my_eval_data.tsv"
          }
        }
      }
    }
  }
}
```

### 1.15.5 Code Example

```
class MyTask(NewTask):
    class Config(NewTask.Config):
        model: MyModel.Config = MyModel.Config()

class MyModel(Model):
    class Config(Model.Config):
        class ModelInput(Model.Config.ModelInput):
            tokens: TokenTensorizer.Config = TokenTensorizer.Config()
            labels: SlotLabelTensorizer.Config = SlotLabelTensorizer.Config()
```

(continues on next page)

(continued from previous page)

```
        inputs: ModelInput = ModelInput()
        embedding: WordEmbedding.Config = WordEmbedding.Config()

        representation: Union[
            BiLSTMSlotAttention.Config,
            BSeqCNNRepresentation.Config,
            PassThroughRepresentation.Config,
        ] = BiLSTMSlotAttention.Config()
        output_layer: Union[
            WordTaggingOutputLayer.Config, CRFOutputLayer.Config
        ] = WordTaggingOutputLayer.Config()
        decoder: MLPDecoder.Config = MLPDecoder.Config()

    @classmethod
    def from_config(cls, config, tensorizers):
        vocab = tensorizers["tokens"].vocab
        embedding = create_module(config.embedding, vocab=vocab)

        labels = tensorizers["labels"].vocab
        representation = create_module(
            config.representation, embed_dim=embedding.embedding_dim
        )
        decoder = create_module(
            config.decoder,
            in_dim=representation.representation_dim,
            out_dim=len(labels),
        )
        output_layer = create_module(config.output_layer, labels=labels)
        return cls(embedding, representation, decoder, output_layer)

    def arrange_model_inputs(self, tensor_dict):
        tokens, seq_lens, _ = tensor_dict["tokens"]
        return (tokens, seq_lens)

    def arrange_targets(self, tensor_dict):
        return tensor_dict["labels"]

    def forward(
        self,
        tokens: torch.Tensor,
    ) -> List[torch.Tensor]:
        embeddings = [self.token_embedding(tokens)]

        final_embedding = torch.cat(embeddings, -1)
        representation = self.representation(final_embedding)

        return self.decoder(representation)
```

## 1.16 Custom Data Format

PyText's default reader is `TSVDataSource` to read your dataset if it's in tsv format (tab-separated values). In many cases, your data is going to be in a different format. You could write a pre-processing script to format your data into tsv format, but it's easier and more convenient to implement your own `DataSource` component so that PyText can read your data directly, without any preprocessing.

This tutorial explains how to implement a simple `DataSource` that can read the ATIS data and to perform a classification task using the "intent" labels.

### 1.16.1 1. Download the data

Download the ATIS (Airline Travel Information System) dataset and unzip it in a directory. Note that to download the dataset, you will need a Kaggle account for which you can sign up for free. The zip file is about 240KB.

```
$ unzip <download_dir>/atis.zip -d <download_dir>/atis
```

### 1.16.2 2. The data format

The ATIS dataset has a few defining characterics:

1. it has a train set and a test set, but not eval set
2. the data is split into a "dict" file, which is a vocab file containing the words or labels, and the train and test sets, which only contain integers representing the word indexes.
3. sentences always start with the token 178 = BOS (Beginning Of Sentence) and end with the token 179 = EOS (End Of Sentence).

```
$ tail atis/atis.dict.vocab.csv
y
year
yes
yn
york
you
your
yx
yyz
zone
$ tail atis/atis.test.query.csv
178 479 0 545 851 264 882 429 851 915 330 179
178 479 902 851 264 180 428 444 736 521 301 851 915 330 179
178 818 581 207 827 204 616 915 330 179
178 479 0 545 851 264 180 428 444 299 851 619 937 301 654 887 200 435 621 740 179
178 818 581 207 827 204 482 827 619 937 301 229 179
178 688 423 207 827 429 444 299 851 218 203 482 827 619 937 301 229 826 236 621 740␣
↪253 130 689 179
178 423 581 180 428 444 299 851 218 203 482 827 619 937 301 229 179
178 479 0 545 851 431 444 589 851 297 654 212 200 179
178 479 932 545 851 264 180 730 870 428 444 511 301 851 297 179
178 423 581 180 428 826 427 444 587 851 810 179
```

Our `DataSource` must then resolve the words from the vocab files to rebuild the sentences and labels as strings. It must also take a subset the one of train or test dataset to create the eval dataset. Since the test set is pretty small, we'll use the train set for that purpose and randomly take a small fraction (say 25%) to create the eval set. Finally, we can safely remove the first and last tokens of every query (BOS and EOS), as they don't add any value for classification.

The ATIS dataset also has information for slots tagging that we'll ignore because we only care about classification in this tutorial.

### 1.16.3 3. DataSource

PyText defines a `DataSource` to read the data. It expect each row of data to be represented as a python dict where the keys are the column names and the values are the columns properly typed.

Most of the time, the dataset will come as strings and the casting to the proper types can be inferred automatically from the other components in the config. To make the implementation of a new `DataSource` easier, PyText provides the class `RootDataSource` that does this type lookup for you. Most users should use `RootDataSource` as a base class.

### 1.16.4 4. Implementing *AtisIntentDataSource*

We will write the all the code for our `AtisIntentDataSource` in the file *my_classifier/source.py*.

First, let's write the utilities that will help us read the data: a function to load the vocab files, and the generator that uses them to rebuild the sentences and labels. We return pytext.data.utils.UNK for unknown words. We store the indexes as strings to avoid casting from and to ints when reading the inputs.:

```
def load_vocab(file_path):
    """
    Given a file, prepare the vocab dictionary where each line is the value and
    (line_no - 1) is the key
    """
    vocab = {}
    with open(file_path, "r") as file_contents:
        for idx, word in enumerate(file_contents):
            vocab[str(idx)] = word.strip()
    return vocab

def reader(file_path, vocab):
    with open(file_path, "r") as reader:
        for line in reader:
            yield " ".join(
                vocab.get(s.strip(), UNK)
                # ATIS every row starts/ends with BOS/EOS: remove them
                for s in line.split()[1:-1]
            )
```

Then we declate the `DataSource` class itself: `AtisIntentDataSource`. It inherits from `RootDataSource`, which gives us the automatic lookup of data types. We declare all the config parameters that will be useful, and give sensible default values so that the general case where users provide only *path* and *field_names* will likely work. We load the vocab files for queries and intent only once in the constructor and keep them in memory for the entire run:

```
class AtisIntentDataSource(RootDataSource):

    def __init__(
        self,
        path="my_directory",
        field_names=None,
        validation_split=0.25,
        random_seed=12345,
        # Filenames can be overridden if necessary
        intent_filename="atis.dict.intent.csv",
        vocab_filename="atis.dict.vocab.csv",
        test_queries_filename="atis.test.query.csv",
        test_intent_filename="atis.test.intent.csv",
```

(continues on next page)

(continued from previous page)

```
        train_queries_filename="atis.train.query.csv",
        train_intent_filename="atis.train.intent.csv",
        **kwargs,
    ):
        super().__init__(**kwargs)

        field_names = field_names or ["text", "label"]
        assert len(field_names or []) == 2, \
           "AtisIntentDataSource only handles 2 field_names: {}".format(field_names)

        self.random_seed = random_seed
        self.eval_split = eval_split

        # Load the vocab dict in memory for the readers
        self.words = load_vocab(os.path.join(path, vocab_filename))
        self.intents = load_vocab(os.path.join(path, intent_filename))

        self.query_field = field_names[0]
        self.intent_field = field_names[1]

        self.test_queries_filepath = os.path.join(path, test_queries_filename)
        self.test_intent_filepath = os.path.join(path, test_intent_filename)
        self.train_queries_filepath = os.path.join(path, train_queries_filename)
        self.train_intent_filepath = os.path.join(path, train_intent_filename)
```

To generate the eval data set, we need to randomly select some of the rows in training, but in a consistent and repeatable way. This is not strictly needed, and the training will work if the selection were completely random, but having a consistent sequence will help with debugging and give comparable results from training to training. In order to do that, we need to use the same seed for a new random number generator each time we start reading the train data set. The function below can be used for either training or eval and ensures that those two sets are complement of each other, with the ratio determined by eval_split. This function returns True or False depending on whether the row should be included or not:

```
def _selector(self, select_eval):
    """
    This selector ensures that the same pseudo-random sequence is
    always the used from the Beginning. The `select_eval` parameter
    guarantees that the training set and eval set are exact complements.
    """
    rng = Random(self.random_seed)
    def fn():
        return select_eval ^ (rng.random() >= self.eval_split)
    return fn
```

Next, we write the function that iterates through both the *reader* for the queries (sentences) and the *reader* for the intents (labels) simultaneously. It yields each row in the form a python dictionnary, where the keys are the *field_names*. We can pass an optional function to select a subset of the row (ie: _selector defined above); the default is to select all the rows:

```
def _iter_rows(self, query_reader, intent_reader, select_fn=lambda: True):
    for query_str, intent_str in zip(query_reader, intent_reader):
        if select_fn():
            yield {
                # in ATIS every row starts/ends with BOS/EOS: remove them
                self.query_field: query_str[4:-4],
                self.intent_field: intent_str,
```

(continues on next page)

(continued from previous page)

```
            }
```

Finally, we tie everything toghether by implementing the 3 API methods of `RootDataSource`. Each of those methods should return a generator that can iterate through the specific dataset entirely. For the test dataset, we simply return all the row presented by the data in test_queries_filepath and test_intent_filepath, using the corresponding vocab:

```
def raw_test_data_generator(self):
    return iter(self._iter_rows(
        query_reader=reader(
            self.test_queries_filepath,
            self.words,
        ),
        intent_reader=reader(
            self.test_intent_filepath,
            self.intents,
        ),
    ))
```

For the eval and train datasets, we read the same files train_queries_filepath and train_intent_filepath, but we select some of the rows for eval and the rest for train:

```
def raw_train_data_generator(self):
    return iter(self._iter_rows(
        query_reader=reader(
            self.train_queries_filepath,
            self.words,
        ),
        intent_reader=reader(
            self.train_intent_filepath,
            self.intents,
        ),
        select_fn=self._selector(select_eval=False),
    ))

def raw_eval_data_generator(self):
    return iter(self._iter_rows(
        query_reader=reader(
            self.train_queries_filepath,
            self.words,
        ),
        intent_reader=reader(
            self.train_intent_filepath,
            self.intents,
        ),
        select_fn=self._selector(select_eval=True),
    ))
```

`RootDataSource` needs to know how it should transform the values in the dictionnaries created by the raw generators into the types matching the tensorizers used in the model. Fortunately, `RootDataSource` already provides a number of type conversion functions like the one below, so we don't need to do it for strings. If we did need to do it, we would declare one like this for `AtisIntentDataSource`.:

```
@AtisIntentDataSource.register_type(str)
def load_string(s):
    return s
```

The full source code for this tutorial can be found in *demo/datasource/source.py*, which include the *imports* needed.

### 1.16.5 5. Testing *AtisIntentDataSource*

For rapid dev-test cycles, we add a simple main code printing the generated data in the terminal:

```
if __name__ == "__main__":
    import sys
    src = AtisIntentDataSource(
        path=sys.argv[1],
        field_names=["query", "intent"],
        schema={},
    )
    for row in src.raw_train_data_generator():
        print("TRAIN", row)
    for row in src.raw_eval_data_generator():
        print("EVAL", row)
    for row in src.raw_test_data_generator():
        print("TEST", row)
```

We test our class to make sure we're getting the right data.

```
$ python3 my_classifier/source.py atis | head -n 3
TRAIN {'query': 'what flights are available from pittsburgh to baltimore on thursday␣
↪morning', 'intent': 'flight'}
TRAIN {'query': 'cheapest airfare from tacoma to orlando', 'intent': 'airfare'}
TRAIN {'query': 'round trip fares from pittsburgh to philadelphia under 1000 dollars',
↪ 'intent': 'airfare'}

$ python3 my_classifier/source.py atis | cut -d " " -f 1 | uniq -c
3732 TRAIN
1261 EVAL
 893 TEST
```

### 1.16.6 6. Training the Model

First let's get a config using our new `AtisIntentDataSource`

```
$ pytext --include my_classifier gen-default-config DocumentClassificationTask␣
↪AtisIntentDataSource > my_classifier/config.json
Including: my_classifier
... importing module: my_classifier.source
... importing: <class 'my_classifier.source.AtisIntentDataSource'>
INFO - Applying option: task->data->source = AtisIntentDataSource
```

This default config contains all the parameters with their default value. So we edit the config to remove the parameters that we don't care about, and we edit the ones we care about. We only want to run 3 epochs for now. It looks like this.

```
$ cat my_classifier/config.json
{
  "debug_path": "my_classifier.debug",
  "export_caffe2_path": "my_classifier.caffe2.predictor",
  "export_onnx_path": "my_classifier.onnx",
  "save_snapshot_path": "my_classifier.pt",
  "task": {
    "DocumentClassificationTask": {
      "data": {
        "Data": {
```

(continues on next page)

(continued from previous page)

```
          "source": {
            "AtisIntentDataSource": {
              "field_names": ["text", "label"],
              "path": "atis",
              "random_seed": 12345,
              "validation_split": 0.25
            }
          }
        }
      },
      "metric_reporter": {
        "output_path": "my_classifier.out"
      },
      "trainer": {
        "epochs": 3
      }
    }
  },
  "test_out_path": "my_classifier_test.out",
  "version": 12
}
```

And, at last, we can train the model

```
$ pytext --include my_classifier train < my_classifier/config.json
```

### 1.16.7 Notes

In the current version of PyText, we need to explicitly declare a few more things, like the *Config* class (that looks like the __init__ parameters) and the from_config method:

```
class Config(RootDataSource.Config):
    path: str = "."
    field_names: List[str] = ["text", "label"]
    validation_split: float = 0.25
    random_seed: int = 12345
    # Filenames can be overridden if necessary
    intent_filename: str = "atis.dict.intent.csv"
    vocab_filename: str = "atis.dict.vocab.csv"
    test_queries_filename: str = "atis.test.query.csv"
    test_intent_filename: str = "atis.test.intent.csv"
    train_queries_filename: str = "atis.train.query.csv"
    train_intent_filename: str = "atis.train.intent.csv"

# Config mimics the constructor
# This will be the default in future pytext.
@classmethod
def from_config(cls, config: Config, schema: Dict[str, Type]):
    return cls(schema=schema, **config._asdict())
```

## 1.17 Custom Tensorizer

`Tensorizer` is the class that prepares your data coming out of the data source and transforms it into tensors suitable for processing. Each tensorizer knows how to prepare the input data from specific columns. In order to do that, the tensorizer (after initialization, such as creating or loading the vocabulary for look-ups) executes the following steps:

1. Its `Config` defines which column name(s) the tensorizer will look at
2. `numberize()` takes one row and transform the strings into numbers
3. `tensorize()` takes a batch of rows and creates the tensors

PyText provides a number of tensorizers for the most common cases. However, if you have your own custom features that don't have a suitable `Tensorizer`, you will need to write your own class. Fortunately it's quite easy: you simply need to create a class that inherits from `Tensorizer` (or one of its subclasses), and implement a few functions.

First a `Config` inner class, from_config class method, and the constructor *__init__*. This is just to declare member variables.

The tensorizer should declare the schema of your `Tensorizer` by defining a *column_schema* property which returns a list of tuples, one for each field/column read from the data source. Each tuple specifies the name of the column, and the type of the data. By specifying the type of your data, the data source will automatically parse the inputs and pass objects of those types to the tensorizers. You don't need to parse your own inputs.

For example, `SeqTokenTensorizer` reads one column from the input data. The data is formatted like a json list of strings: *["where do you wanna meet?", "MPK"]*. The schema declaration is like this:

```
@property
def column_schema(self):
    return [(self.column, List[str])]
```

Another example with `GazetteerTensorizer`: it needs 2 columns, one string for the text itself, and one for the gazetteer features formatted like a complex json object. (The Gazetteer type is registered in the data source to automatically convert the raw strings from the input to this type.) The schema declaration is like this:

```
Gazetteer = List[Dict[str, Dict[str, float]]]

@property
def column_schema(self):
    return [(self.text_column, str), (self.dict_column, Gazetteer)]
```

### 1.17.1 Example Implementation

Let's implement a simple word tensorizer that creates a tensor with the word indexes from a vocabulary.

```
class MyWordTensorizer(Tensorizer):

    class Config(Tensorizer.Config):
        #: The name of the text column to read from the data source.
        column: str = "text"

    @classmethod
    def from_config(cls, config: Config):
        return cls(column=config.column)

    def __init__(self, column):
        self.column = column
```

(continues on next page)

(continued from previous page)

```
        self.vocab = vocab

    @property
    def column_schema(self):
        return [(self.column, str)]
```

Next we need to build the vocabulary by reading the training data and count the words. Since multiple tensorizers might need to read the data, we parallelize the reading part and the tensorizers use the pattern *row = yield* to read their inputs. In this simple example, our "tokenize" function is just going to split on spaces.

```
def _tokenize(self, row):
    raw_text = row[self.column]
    return raw_text.split()

def initialize(self):
    """Build vocabulary based on training corpus."""
    vocab_builder = VocabBuilder()

    try:
        while True:
            row = yield
            words = _tokenize(row)
            vocab_builder.add_all(words)
    except GeneratorExit:
        self.vocab = vocab_builder.make_vocab()
```

The most important method is numberize, which takes a row and transforms it into list of numbers. The exact meaning of those numbers is arbitrary and depends on the design of the model. In our case, we look up the word indexes in the vocabulary.

```
def numberize(self, row):
    """Look up tokens in vocabulary to get their corresponding index"""
    words = _tokenize(row)
    idx = self.vocab.lookup_all(words)
    # LSTM representations need the length of the sequence
    return idx, len(idx)
```

Because LSTM-based representations need the length of the sequence to only consider the useful values and ignore the padding, we also return the length of each sequence.

Finally, the last function will create properly padded torch.Tensors from the batches produced by *numberize*. Numberized results can be cached for performance. We have a separate function to tensorize them because they are shuffled and batched differently (at each epoch), and then they will need different padding (because padding dimensions depend on the batch).

```
def tensorize(self, batch):
    tokens, seq_lens = zip(*batch)
    return (
        pad_and_tensorize(tokens, self.vocab.get_pad_index()),
        pad_and_tensorize(seq_lens),
    )
```

LSTM-based representations implemented in Torch also need the batches to be sorted by sequence length descending, so we're add in a sort function.

```
def sort_key(self, row):
    # LSTM representations need the batches to be sorted by descending seq_len
    return row[1]
```

The full code is in *demo/examples/tensorizer.py*

### 1.17.2 Testing

We can test our tensorizer with the following code that initializes the vocab, then tries the *numberize* function:

```
rows = [
    {"text": "I want some coffee"},
    {"text": "Turn it up"},
]
tensorizer = MyWordTensorizer(column="text")

# Vocabulary starts with 0 and 1 for Unknown and Padding.
# The rest of the vocabulary is built by the rows in order.
init = tensorizer.initialize()
init.send(None)  # start the loop
for row in rows:
    init.send(row)
init.close()

# Verify numberize.
numberized_rows = (tensorizer.numberize(r) for r in rows)
words, seq_len = next(numberized_rows)
assert words == [2, 3, 4, 5]
assert seq_len == 4  # "I want some coffee" has 4 words
words, seq_len = next(numberized_rows)
assert words == [6, 7, 8]
assert seq_len == 3  # "Turn it up" has 3 words

# test again, this time also make the tensors
numberized_rows = (tensorizer.numberize(r) for r in rows)
words_tensors, seq_len_tensors = tensorizer.tensorize(numberized_rows)
# Notice the padding (1) of the 2nd tensor to match the dimension
assert words_tensors.equal(torch.tensor([[2, 3, 4, 5], [6, 7, 8, 1]]))
assert seq_len_tensors.equal(torch.tensor([4, 3]))
```

## 1.18 Using External Dense Features

Sometime you want to add external features to augment the inputs to your model. For example, if you want to classify a text that has an image associated to it, you might want to process the image separately and use features of this image along with the text to help the classifier. Those features are added in the input data as one extra field (column) and should look like a list of floats (json).

Let's look at a simple example, first without the dense feature, then we add dense features.

### 1.18.1 Example: Simple Model

First, here's an example of a simple classifier that uses just the text and no dense features. (This is only showing the relevant parts of the model code for simplicity.)

```
class MyModel(Model):
  class Config(Model.Config):
    class ModelInput(Model.Config.InputConfig):
      tokens: TokenTensorizer.Config = TokenTensorizer.Config()
      labels: LabelTensorizer.Config = LabelTensorizer.Config()

    inputs: ModelInput = ModelInput()
    token_embedding: WordEmbedding.Config = WordEmbedding.Config()

    representation: RepresentationBase.Config = DocNNRepresentation.Config()
    decoder: DecoderBase.Config = MLPDecoder.Config()
    output_layer: OutputLayerBase.Config = ClassificationOutputLayer.Config()

  def from_config(cls, config, tensorizers):
    token_embedding = create_module(config.token_embedding, tensorizer=tensorizers[
↪"tokens"])
    representation = create_module(config.representation, embed_dim=token_embedding.
↪embedding_dim)
    labels = tensorizers["labels"].vocab
    decoder = create_module(
        config.decoder,
        in_dim=representation.representation_dim
        out_dim=len(labels),
    )
    output_layer = create_module(config.output_layer, labels=labels)
    return cls(token_embedding, representation, decoder, output_layer)

  def arrange_model_inputs(self, tensor_dict):
    return (tensor_dict["tokens"],)

  def forward(
      self,
      tokens_in: Tuple[torch.Tensor, torch.Tensor],
  ) -> List[torch.Tensor]:
        word_tokens, seq_lens = tokens
        embedding_out = self.embedding(word_tokens)
        representation_out = self.representation(embedding_out, seq_lens)
        return self.decoder(representation_out)
```

### 1.18.2 Example: Simple Model With Dense Features

To use the dense features, you will typically write your model to use them directly in the decoder, bypassing the embeddings and representation stages that process the text part of your inputs. Here's the same example again, this time with the dense features added (see lines marked with <–).

```
class MyModel(Model):
  class Config(Model.Config):
    class ModelInput(Model.Config.InputConfig):
      tokens: TokenTensorizer.Config = TokenTensorizer.Config()
      dense: FloatListTensorizer.Config = FloatListTensorizer.Config()   # <--
      labels: LabelTensorizer.Config = LabelTensorizer.Config()

    inputs: ModelInput = ModelInput()
    token_embedding: WordEmbedding.Config = WordEmbedding.Config()
```

(continues on next page)

(continued from previous page)

```
    representation: RepresentationBase.Config = DocNNRepresentation.Config()
    decoder: DecoderBase.Config = MLPDecoder.Config()
    output_layer: OutputLayerBase.Config = ClassificationOutputLayer.Config()

  def from_config(cls, config, tensorizers):
    token_embedding = create_module(config.token_embedding, tensorizer=tensorizers[
↪"tokens"])
    representation = create_module(config.representation, embed_dim=token_embedding.
↪embedding_dim)
    dense_dim = tensorizers["dense"].out_dim    # <--
    labels = tensorizers["labels"].vocab
    decoder = create_module(
        config.decoder,
        in_dim=representation.representation_dim + dense_dim    # <--
        out_dim=len(labels),
    )
    output_layer = create_module(config.output_layer, labels=labels)
    return cls(token_embedding, representation, decoder, output_layer)

  def arrange_model_inputs(self, tensor_dict):
    return (tensor_dict["tokens"], tensor_dict["dense"])  # <--

  def forward(
      self,
      tokens_in: Tuple[torch.Tensor, torch.Tensor],
      dense_in: torch.Tensor,    # <--
  ) -> List[torch.Tensor]:
        word_tokens, seq_lens = tokens
        embedding_out = self.embedding(word_tokens)
        representation_out = self.representation(embedding_out, seq_lens)
        representation_out = torch.cat((representation_out, dense_in), 1)    # <--
        return self.decoder(representation_out)
```

## 1.19 Creating A New Model

PyText uses a `Model` class as a central place to define components for data processing, model training, etc. and wire up those components.

In this tutorial, we will create a word tagging model for the ATIS dataset. The format of the ATIS dataset is explained in the *Custom Data Format*, so we will not repeat it here. We are going to create a similar data source that uses the slot tagging information rather than the intent information. We won't describe in detail how this data source is created but you can look at the *Custom Data Format*, and the full source code for this tutorial in `demo/my_tagging` for more information.

This model will predict a "slot", also called "tag" or "label", for each word in the utterance, using the IOB2 format), where the O tag is used for Outside (no match), B- for Beginning and I- for Inside (continuation). Here's an example:

```
{
  "text": "please list the flights from newark to los angeles",
  "slots": "O O O O O B-fromloc.city_name O B-toloc.city_name I-toloc.city_name"
}
```

### 1.19.1 1. The Components

The first step is to specify the components used in this model by listing them in the Config class, the corresponding `from_config` function, and the constructor `__init__`.

Thanks to the modular nature of PyText, we can simply use many included common components, such as `TokenTensorizer`, `WordEmbedding`, `BiLSTMSlotAttention` and `MLPDecoder`. Since we're also using the common pattern of *embedding -> representation -> decoder -> output_layer*, we use `Model` as a base class, so we don't need to write `__init__`.

ModelInput defines how the data that is read will be transformed into tensors. This is done using a `Tensorizer`. These components take one or several columns (often strings) from each input row and create the corresponding numeric features in a properly padded tensor. The tensorizers will to be initialized first, and in this step they will often parse the training data to create their `Vocabulary`.

In our case, the utterance is in the column "text" (which is the default column name for this tensorizer), and is composed of tokens (words), so we can use the `TokenTensorizer`. The `Vocabulary` will be created from all the utterances.

The slots are also composed of tokens: the IOB2 tags. We can also use `TokenTensorizer` for the column "slots". This `Vocabulary` will be the list of IOB2 tags found in the "slots" column of the training data. This is a different column name, so we specify it.

```
class MyTaggingModel(Model):
    class Config(ConfigBase):
        class ModelInput(Model.Config.ModelInput):
            tokens: TokenTensorizer.Config = TokenTensorizer.Config()
            slots: TokenTensorizer.Config = TokenTensorizer.Config(column="slots")

        inputs: ModelInput = ModelInput()
        embedding: WordEmbedding.Config = WordEmbedding.Config()
        representation: BiLSTMSlotAttention.Config = BiLSTMSlotAttention.Config,
        decoder: MLPDecoder.Config = MLPDecoder.Config()
        output_layer: MyTaggingOutputLayer.Config = MyTaggingOutputLayer.Config()
```

### 1.19.2 2. from_config method

`from_config` is where the components are created with the proper parameters. Some come from the Config (passed by the user in json format), some use the default values, others are dicated by the model's architecture so that the different components fit with each other. For example, the representation layer needs to know the dimension of the embeddings it will receive, the decoder needs to know the dimension of the representation layer before it and the size of the slots vocab to output.

In this model, we only need one embedding: the one of the tokens. The slots don't have embeddings because while they are listed as input (in ModelInput), they are actually outputs and the will be used in the output layer. (During training, they are inputs as true values.)

```
@classmethod
def from_config(cls, config, tensorizers):
    embedding = create_module(config.embedding, tensorizer=tensorizers["tokens"])
    representation = create_module(
        config.representation, embed_dim=embedding.embedding_dim
    )
    slots = tensorizers["slots"].vocab
    decoder = create_module(
        config.decoder,
```

(continues on next page)

(continued from previous page)

```
        in_dim=representation.representation_dim,
        out_dim=len(slots),
    )
    output_layer = MyTaggingOutputLayer(slots, CrossEntropyLoss(None))
    # call __init__ constructor from super class Model
    return cls(embedding, representation, decoder, output_layer)
```

### 1.19.3 3. Forward method

The `forward` method contains the execution logic calling each of those components and passing the results of one to the next. It will be called for every row transformed into tensors.

`TokenTensorizer` returns the tensor for the tokens themselves and also the sequence length, which is the number of tokens in the utterances. This is because we need to pad the tensors in a batch to give them all the same dimensions, and LSTM-based reprentations need to differentiate the padding from the actual tokens.

```
def forward(
    self,
    word_tokens: torch.Tensor,
    seq_lens: torch.Tensor,
) -> List[torch.Tensor]:
    # fetch embeddings for the tokens in the utterance
    embedding = self.embedding(word_tokens)

    # pass the embeddings to the BiLSTMSlotAttention layer.
    # LSTM-based representations also need seq_lens.
    representation = self.representation(embedding, seq_lens)

    # some LSTM representations return extra tensors, we don't use those.
    if isinstance(representation, tuple):
        representation = representation[0]

    # finally run the results through the decoder
    return self.decoder(representation)
```

### 1.19.4 4. Complete MyTaggingModel

To finish this class, we need to define a few more functions.

All the inputs are placed in a python dict where the key is the name of the tensorizer as defined in ModelInput, and the value is the tensor for this input row.

First, we define how the inputs will be passed to the `forward` function in `arrange_model_inputs`. In our case, the only input passed to the `forward` function is the tensors from the "tokens" input. As explained above, `TokenTensorizer` returns 2 tensors: the tokens and the sequence length. (Actually it returns 3 tensors, we'll ignore the 3rd one, the token ranges, in this tutorial)

Then we define `arrange_targets`, which is doing something similar for the targets, which are passed to the loss function during training. In our case, it's the "slots" tensorizer doing that. The padding value can be passed to the loss function (unlike LSTM representations), so we only need the first tensor.

```
def arrange_model_inputs(self, tensor_dict):
    tokens, seq_lens, _ = tensor_dict["tokens"]
    return (tokens, seq_lens)
```

(continues on next page)

(continued from previous page)

```
def arrange_targets(self, tensor_dict):
    slots, _, _ = tensor_dict["slots"]
    return slots
```

### 1.19.5 5. Output Layer

So far, our model is using the same components as any other model, including a common classification model, except for two things: the BiLSTMSlotAttention and the output layer.

BiLSTMSlotAttention is a multi-layer bidirectional LSTM based representation with attention over slots. The implementation of this representation is outside the scope of this tutorial, and this component is already included in PyText, so we'll just use it.

The output layer can be simple enough and demonstrates a few important notions in PyText, like how the loss function is tied to the output layer. We implement it like this:

```
class MyTaggingOutputLayer(OutputLayerBase):

    class Config(OutputLayerBase.Config):
        loss: CrossEntropyLoss.Config = CrossEntropyLoss.Config()

    @classmethod
    def from_config(cls, config, vocab, pad_token):
        return cls(
            vocab,
            create_loss(config.loss, ignore_index=pad_token),
        )

    def get_loss(self, logit, target, context, reduce=True):
        # flatten the logit from [batch_size, seq_lens, dim] to
        # [batch_size * seq_lens, dim]
        return self.loss_fn(logit.view(-1, logit.size()[-1]), target.view(-1), reduce)

    def get_pred(self, logit, *args, **kwargs):
        preds = torch.max(logit, 2)[1]
        scores = F.log_softmax(logit, 2)
        return preds, scores
```

### 1.19.6 6. Metric Reporter

Next we need to write a `MetricReporter` to calculate metrics and report model training/test results:

The `MetricReporter` base class aggregates all the output from Trainer, including predictions, scores and targets. The default aggregation behavior is concatenating the tensors from each batch and converting it to list. If you want different aggregation behavior, you can override it with your own implementation. Here we use the compute_classification_metrics method provided in pytext.metrics to get the precision/recall/F1 scores. PyText ships with a few common metric calculation methods, but you can easily incorporate other libraries, such as sklearn.

In the `__init__` method, we can pass a list of `Channel` to report the results to any output stream. We use a simple `ConsoleChannel` that prints everything to stdout and a `TensorBoardChannel` that outputs metrics to TensorBoard:

```
class MyTaggingMetricReporter(MetricReporter):

    @classmethod
    def from_config(cls, config, vocab):
        return MyTaggingMetricReporter(
            channels=[ConsoleChannel(), TensorBoardChannel()],
            label_names=vocab
        )

    def __init__(self, label_names, channels):
        super().__init__(channels)
        self.label_names = label_names

    def calculate_metric(self):
        return compute_classification_metrics(
            list(
                itertools.chain.from_iterable(
                    (
                        LabelPrediction(s, p, e)
                        for s, p, e in zip(scores, pred, expect)
                    )
                    for scores, pred, expect in zip(
                        self.all_scores, self.all_preds, self.all_targets
                    )
                )
            ),
            self.label_names,
            self.calculate_loss(),
        )
```

### 1.19.7 7. Task

Finally, we declare a task by inheriting from `NewTask`. This base class specifies the training parameters of the model: the data source and batcher, the trainer class (most models will use the default one), and the metric reporter.

Since our metric reporter needs to be initialized with a specific vocab, we need to define the classmethod *create_metric_reporter* so that PyText can construct it properly.

```
class MyTaggingTask(NewTask):
    class Config(NewTask.Config):
        model: MyTaggingModel.Config = MyTaggingModel.Config()
        metric_reporter: MyTaggingMetricReporter.Config = MyTaggingMetricReporter.
↪Config()

    @classmethod
    def create_metric_reporter(cls, config, tensorizers):
        return MyTaggingMetricReporter(
            channels=[ConsoleChannel(), TensorBoardChannel()],
            label_names=list(tensorizers["slots"].vocab),
        )
```

### 1.19.8 8. Generate sample config and train the model

Save all your files in the same directory. For example, I saved all my files in `my_tagging/`.Now you can tell PyText to include your classes with the parameter `--include my_tagging`

Now that we have a fully functional class:~*Task*, we can generate a default JSON config for it by using the pytext cli tool.

```
(pytext) $ pytext --include my_tagging gen-default-config MyTaggingTask > my_config.
↪json
```

Tweak the config as you like, for instance change the number of epochs. Most importantly, specify the path to your ATIS dataset. Then train the model with:

```
(pytext) $ pytext --include my_tagging train < my_config.json
```

## 1.20 Hacking PyText

### 1.20.1 Using your own classes in PyText

Most people just want to create their own components and use them to load their data, train models, etc. In this case, you just need to put all your *.py* files in a directory and include it with the option *–include <my directory>*. PyText will be able to find your code and import your classes. This works with PyText from *pip install* or from github sources.

Example with Custom Data Source

### 1.20.2 Changing PyText

Why would you want to change PyText? Maybe you want to fix one of the github issues, or you want to experiment with your own changes that you can't simply include and you would like to see included in PyText's future releases. In this case you need to download the sources and submit them back to github. Since getting your changes ready and integrated can take some time, you might need to keep your sources up to date. Here's how to do it.

#### Installation

First, make a copy of the PyText repo into your github account. For that (you need a github account), go to the PyText repo and click the Fork button at top-right of the page.

Once the fork is complete, clone your fork onto your computer by clicking the "Clone or download" button and copying the URL. Then, in a terminal, use the *git clone* command to clone the repo in the directory of your choice.

```
$ git clone https://github.com/<your_account>/pytext.git
```

To be able to update your github fork with the latest changes from Facebook's PyText sources, you need to add it as a "remote" with this command. (This can be done later.) The name "upstream" is what's typically used, but you can use any name you want for your remotes.

```
$ git remote add upstream https://github.com/facebookresearch/pytext.git
```

Now you should have 2 remotes: *origin* is your own github fork, and *upstream* is facebook's github repo.

Now you can install the PyText dependencies in a virtual environment. (This means the dependencies will be installed inside the directory *pytext_venv* under *pytext/*, not in your machine's system directory.) Notice the *(pytext_venv)* in the terminal prompt when it's active.

```
$ cd pytext
$ source activation_venv
(pytext_venv) $ ./install_deps
```

To exit the virtual environment:

```
(pytext_venv) $ deactivate
```

### Writing Code

After you've made some code changes, you need to create a branch to commit your code. Do not commit your code in your *master* branch! Give your branch a name that represents what your experiment is about. Then add your changes and commit them.

```
$ git checkout -b <my_experiment>
$ git status -sb
... # list of files you changed
$ git add <file1> <file2>
$ git diff --cached  # see the code changes you added
# ... maybe keep changing and run git add again
$ git commit  # save your changes
$ git show # optional, look at the code changes
$ git push --set-upstream origin <my_experiment>  # send your branch to your github␣
↪fork
```

At this point you should be able to see your branch in your github repo and create a Pull Request to Facebook's github if you want to submit it for review and later be integrated.

### Keeping Up-to-Date

To resume development in an already cloned repo, you might need re-activate the virtual environment:

```
$ cd pytext
$ source activation_venv
```

If you need to update your github repo with the latest changes in the Facebook upstream repo, fetch the changes with this command, merge your master with those changes, and push the changes to your github forks. In order to do that, you can't have any pending changes, so make sure you commit your current work to a branch.

```
$ git fetch upstream
$ git checkout master
$ git merge upstream/master
$ git push
```

Important: never commit changes in your master. Doing this would prevent further updates. Instead, always commit changes to a branch. (See below for more on this.)

Finally, you might need to rebase your branches to the latest master. Check out the branch, rebase it, and (optionally) push it again to your github fork.

```
$ git checkout <my_experiment>
$ git rebase master
$ git push  # optional
```

### Modifying your Pull Request

Many times you will need to modify your code and submit your pull request again. Maybe you found a bug that you need to fix, or you want to integrate some feedback you got in the pull request, or after you rebased your branch you

had to solve a conflict.

If you're going to change your pull request, it's always a good idea to start by rebasing your branch on the lastest upstream/master (see above.)

After making your changes, amend to your existing commit rather than creating a new commit on top of it. This is to ensure your changes are in a single clean commit that does not contain your failed experiments. At this point, you will have a branch *<my_experiment>*, and the branch you pushed to your github forked *origin/<my_experiment>*. Then you will need to force the push to replace the github branch with your changes. The pull request will be automatically updated upstream.

```
$ git commit --amend
$ git push --force
```

### 1.20.3 Addendum

#### One commit or multiple commits?

For most contributions, you will want to keep your pull request as a single, clean commit. It's better to amend the same commit rather than keeping the entire history of intermediate edits.

If your change is more involved, it might be better to create multiple commits, as long as each commit does one thing and is self contained.

#### Code Quality

In order to get your pull request integrated with PyText, it needs to pass the tests and be reviewed. The pull requests will automatically run the circleci tests, and they must be all green for your pull request to be accepted. These tests include building the documentation, run the unit tests under python 3.6 and 3.7, and run the linter *black* to verify code formatting. You can run the linter yourself after installing it with *pip install black*.

If all the tests are green, people will start reviewing your changes. (You too can review other pull requests and make comments and suggestions.) If reviewers ask questions or make suggestions, try your best to answer them with comments or code changes.

A very common reason to reject a pull request is lack of unit testing. Make sure your code is covered by unit tests (add your own tests) to make sure they work now and also in the future when other people make changes to your code!

#### Creating Documentation

Whether you want to add documentation for your feature in code, or just change the existing the documentation, you will need to test it locally. First install extra dependencies needed to build the documentation:

```
$ pip install --upgrade -r docs_requirements.txt
$ pip install --upgrade -r pytext/docs/requirements.txt
```

Then you can build the documentation

```
$ cd pytext/docs
$ make html
```

Finally you can look at the documentation produced with a URL like this *file:///<path_to_pytext_sources>/pytext/docs/build/html/hacking_pytext.html*

### Useful git alias

One of the most useful command for git is one where you print the commits and branches like a tree. This is a complex command most useful when stored as an alias, so we're giving it here.

```
$ git config --global alias.lg "log --pretty=tformat:'%C(yellow)%h %Cgreen(%ad)%Cred
↪%d %Creset%s %C(bold blue)<%cn>%Creset' --decorate --date=short --date=local --
↪graph --all"

$ # try it
$ git lg
```

## 1.21 pytext

### 1.21.1 config

#### field_config

#### FeatureConfig

**Component:** *Module*

**class** pytext.config.field_config.**FeatureConfig**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**word_feat:** *WordEmbedding.Config* **=** *WordEmbedding.Config*()

**seq_word_feat: Optional[***WordEmbedding.Config***] = None**

**dict_feat: Optional[***DictEmbedding.Config***] = None**

**char_feat: Optional[***CharacterEmbedding.Config***] = None**

**dense_feat: Optional[***FloatVectorConfig***] = None**

**contextual_token_embedding: Optional[***ContextualTokenEmbedding.Config***] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "word_feat": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
```

(continues on next page)

(continued from previous page)

```
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "seq_word_feat": null,
    "dict_feat": null,
    "char_feat": null,
    "dense_feat": null,
    "contextual_token_embedding": null
}
```

### FloatVectorConfig

**class** pytext.config.field_config.**FloatVectorConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**dim: int = 0**

**export_input_names: list[str] = `['float_vec_vals']`**

**dim_error_check: bool = `False`**

**Default JSON**

```
{
    "dim": 0,
    "export_input_names": [
        "float_vec_vals"
    ],
    "dim_error_check": false
}
```

### module_config

### CNNParams

**class** pytext.config.module_config.**CNNParams**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**kernel_num: int = `100`**

**kernel_sizes: list[int] = `[3, 4]`**

**weight_norm: bool = `False`**

**dilated: bool = `False`**

**causal: bool = `False`**

**Default JSON**

```
{
    "kernel_num": 100,
    "kernel_sizes": [
        3,
        4
    ],
    "weight_norm": false,
    "dilated": false,
    "causal": false
}
```

## pytext_config

### ExportConfig

**class** pytext.config.pytext_config.**ExportConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**export_caffe2_path: Optional[str] = `None`**

**export_onnx_path: str = `'/tmp/model.onnx'`**

**export_torchscript_path: Optional[str] = `None`**

**export_lite_path: Optional[str] = `None`**

**torchscript_quantize: Optional[bool] = `False`**

**accelerate: list[str] = `[]`**

**inference_interface: Optional[str] = `None`**

**seq_padding_control: Optional[list[int]] = `None`**

**batch_padding_control: Optional[list[int]] = `None`**

**target: str = `''`**

**Default JSON**

```
{
    "export_caffe2_path": null,
    "export_onnx_path": "/tmp/model.onnx",
    "export_torchscript_path": null,
    "export_lite_path": null,
    "torchscript_quantize": false,
    "accelerate": [],
    "inference_interface": null,
    "seq_padding_control": null,
    "batch_padding_control": null,
    "target": ""
}
```

### PyTextConfig

**class** pytext.config.pytext_config.**PyTextConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**task: Union[*TaskBase.Config*, *Task_Deprecated.Config*, *_NewTask.Config*, *NewTask.Config*, *DisjointMultitask.Config*, *NewD***

**use_cuda_if_available: bool = `True`**

**use_fp16: bool = `False`**

**distributed_world_size: int = `1`**

**gpu_streams_for_distributed_training: int = `1`**

**load_snapshot_path: str = `''`**

**save_snapshot_path: str = `'/tmp/model.pt'`**

**use_config_from_snapshot: bool = `True`**

**auto_resume_from_snapshot: bool = `False`**

**export: *ExportConfig* = *ExportConfig*()**

**export_list: list[*ExportConfig*] = `[]`**

**modules_save_dir: str = `''`**

**save_module_checkpoints: bool = `False`**

**save_all_checkpoints: bool = `False`**

**use_tensorboard: bool = `True`**

**random_seed: Optional[int] = `0`** Seed value to seed torch, python, and numpy random generators.

**use_deterministic_cudnn: bool = `False`** Whether to allow CuDNN to behave deterministically.

**report_eval_results: bool = `False`**

**report_test_results: bool = `True`**

**include_dirs: Optional[list[str]] = `None`**

**version: int**

**use_cuda_for_testing: bool = `True`**

**read_chunk_size: Optional[int] = `1000000000`**

**test_out_path: str = `'/tmp/test_out.txt'`**

**debug_path: str = `'/tmp/model.debug'`**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

## 1.21.2 data

### batch_sampler

#### AlternatingRandomizedBatchSampler.Config

**Component:** *`AlternatingRandomizedBatchSampler`*

**class** `AlternatingRandomizedBatchSampler.`**`Config`**

**Bases:** `Component.Config`

**All Attributes (including base classes)**

**unnormalized_iterator_probs: dict[str, float]**

**second_unnormalized_iterator_probs: dict[str, float]**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

#### BaseBatchSampler.Config

**Component:** *`BaseBatchSampler`*

**class** `BaseBatchSampler.`**`Config`**

**Bases:** `Component.Config`

**All Attributes (including base classes)**

**Subclasses**

- `EvalBatchSampler.Config`

**Default JSON**

```
{}
```

#### EvalBatchSampler.Config

**Component:** *`EvalBatchSampler`*

**class** `EvalBatchSampler.`**`Config`**

**Bases:** `BaseBatchSampler.Config`

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### NaturalBatchSampler.Config

**Component:** *NaturalBatchSampler*

**class** NaturalBatchSampler.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**dataset_counts: dict[str, int] = { }**

**Default JSON**

```
{
    "dataset_counts": {}
}
```

### RandomizedBatchSampler.Config

**Component:** *RandomizedBatchSampler*

**class** RandomizedBatchSampler.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**unnormalized_iterator_probs: dict[str, float]**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### RoundRobinBatchSampler.Config

**Component:** *RoundRobinBatchSampler*

**class** RoundRobinBatchSampler.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**iter_to_set_epoch: str = ''**

**Default JSON**

```
{
    "iter_to_set_epoch": ""
}
```

### bert_tensorizer

#### BERTTensorizer.Config

**Component:** *BERTTensorizer*

**class** BERTTensorizer.**Config**

**Bases:** BERTTensorizerBase.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**columns: list[str] = ['text']**

**tokenizer:** *Tokenizer.Config* = *WordPieceTokenizer.Config*()

**base_tokenizer: Optional[***Tokenizer.Config***] = None**

**vocab_file: str = 'manifold://nlp_technologies/tree/huggingface-models/bert-base-uncased/voca**

**max_seq_len: int = 256**

**Subclasses**

- BERTContextTensorizerForDenseRetrieval.Config
- SquadForBERTTensorizer.Config
- SquadForBERTTensorizerForKD.Config

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "text"
    ],
    "tokenizer": {
        "WordPieceTokenizer": {
            "basic_tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            },
            "wordpiece_vocab_path": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt"
        }
    },
    "base_tokenizer": null,
    "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-base-
↪uncased/vocab.txt",
    "max_seq_len": 256
}
```

#### BERTTensorizerBase.Config

**Component:** *BERTTensorizerBase*

**class** BERTTensorizerBase.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**columns: list[str] = ['text']**

**tokenizer:** *Tokenizer.Config* = *Tokenizer.Config*()

**base_tokenizer: Optional[***Tokenizer.Config***] = None**

**vocab_file: str = ''**

**max_seq_len: int = 256**

**Subclasses**

- BERTTensorizer.Config
- BERTContextTensorizerForDenseRetrieval.Config
- RoBERTaContextTensorizerForDenseRetrieval.Config
- RoBERTaTensorizer.Config
- RoBERTaTokenLevelTensorizer.Config
- SquadForBERTTensorizer.Config
- SquadForBERTTensorizerForKD.Config
- SquadForRoBERTaTensorizer.Config
- SquadForRoBERTaTensorizerForKD.Config

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "text"
    ],
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "base_tokenizer": null,
    "vocab_file": "",
    "max_seq_len": 256
}
```

### data

### Batcher.Config

**Component:** *Batcher*

**class** Batcher.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**train_batch_size: int = 16** Make batches of this size when possible. If there's not enough data, might generate some smaller batches.

**eval_batch_size: int = 16**

**test_batch_size: int = 16**

**Subclasses**

- PoolingBatcher.Config
- DynamicPoolingBatcher.Config
- ExponentialDynamicPoolingBatcher.Config
- LinearDynamicPoolingBatcher.Config

**Default JSON**

```
{
    "train_batch_size": 16,
    "eval_batch_size": 16,
    "test_batch_size": 16
}
```

### Data.Config

**Component:** *Data*

**class** Data.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**source:** *DataSource.Config* = *TSVDataSource.Config*() Specify where training/test/eval data come from. The default value will not provide any data.

**batcher:** *Batcher.Config* = *PoolingBatcher.Config*() How training examples are split into batches for the optimizer.

**sort_key: Optional[str] = None**

**in_memory: Optional[bool] = True** cache numberized result in memory, turn off when CPU memory bound.

**Subclasses**

- PackedLMData.Config

**Default JSON**

```
{
    "source": {
        "TSVDataSource": {
            "column_mapping": {},
            "train_filename": null,
            "test_filename": null,
```

(continues on next page)

(continued from previous page)

```
            "eval_filename": null,
            "field_names": null,
            "delimiter": "\t",
            "quoted": false,
            "drop_incomplete_rows": false
        }
    },
    "batcher": {
        "PoolingBatcher": {
            "train_batch_size": 16,
            "eval_batch_size": 16,
            "test_batch_size": 16,
            "pool_num_batches": 1000,
            "num_shuffled_pools": 1
        }
    },
    "sort_key": null,
    "in_memory": true
}
```

### PoolingBatcher.Config

**Component:** *PoolingBatcher*

**class** PoolingBatcher.**Config**

**Bases:** Batcher.Config

**All Attributes (including base classes)**

**train_batch_size: int = 16**

**eval_batch_size: int = 16**

**test_batch_size: int = 16**

**pool_num_batches: int = 1000** Size of a pool expressed in number of batches

**num_shuffled_pools: int = 1** How many pool-sized chunks to load at a time for shuffling

**Subclasses**

- DynamicPoolingBatcher.Config
- ExponentialDynamicPoolingBatcher.Config
- LinearDynamicPoolingBatcher.Config

**Default JSON**

```
{
    "train_batch_size": 16,
    "eval_batch_size": 16,
    "test_batch_size": 16,
    "pool_num_batches": 1000,
    "num_shuffled_pools": 1
}
```

### data_handler

#### DataHandler.Config

**Component:** *DataHandler*

**class** DataHandler.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**columns_to_read: list[str] = `[]`**

**shuffle: bool = `True`**

**sort_within_batch: bool = `True`**

**train_path: str = `'train.tsv'`**

**eval_path: str = `'eval.tsv'`**

**test_path: str = `'test.tsv'`**

**train_batch_size: int = `128`**

**eval_batch_size: int = `128`**

**test_batch_size: int = `128`**

**column_mapping: dict[str, str] = `{}`**

**Subclasses**

- DisjointMultitaskDataHandler.Config

**Default JSON**

```
{
    "columns_to_read": [],
    "shuffle": true,
    "sort_within_batch": true,
    "train_path": "train.tsv",
    "eval_path": "eval.tsv",
    "test_path": "test.tsv",
    "train_batch_size": 128,
    "eval_batch_size": 128,
    "test_batch_size": 128,
    "column_mapping": {}
}
```

### dense_retrieval_tensorizer

#### BERTContextTensorizerForDenseRetrieval.Config

**Component:** *BERTContextTensorizerForDenseRetrieval*

**class** BERTContextTensorizerForDenseRetrieval.**Config**

**Bases:** BERTTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**columns: list[str] = `['text']`**

**tokenizer: *Tokenizer.Config* = *WordPieceTokenizer.Config*()**

**base_tokenizer: Optional[*Tokenizer.Config*] = `None`**

**vocab_file: str = `'manifold://nlp_technologies/tree/huggingface-models/bert-base-uncased/voca`**

**max_seq_len: int = `256`**

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "text"
    ],
    "tokenizer": {
        "WordPieceTokenizer": {
            "basic_tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            },
            "wordpiece_vocab_path": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt"
        }
    },
    "base_tokenizer": null,
    "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-base-
↪uncased/vocab.txt",
    "max_seq_len": 256
}
```

### PositiveLabelTensorizerForDenseRetrieval.Config

**Component:** *`PositiveLabelTensorizerForDenseRetrieval`*

**class** `PositiveLabelTensorizerForDenseRetrieval.`**`Config`**

**Bases:** `LabelTensorizer.Config`

**All Attributes (including base classes)**

**is_input: bool = `False`**

**column: str = `'label'`**

**allow_unknown: bool = `False`**

**pad_in_vocab: bool = `False`**

**label_vocab: Optional[list[str]] = `None`**

**label_vocab_file: Optional[str] = `None`**

**add_labels: Optional[list[str]] = `None`**

**Default JSON**

```
{
    "is_input": false,
    "column": "label",
    "allow_unknown": false,
    "pad_in_vocab": false,
    "label_vocab": null,
    "label_vocab_file": null,
    "add_labels": null
}
```

### RoBERTaContextTensorizerForDenseRetrieval.Config

**Component:** *RoBERTaContextTensorizerForDenseRetrieval*

**class** RoBERTaContextTensorizerForDenseRetrieval.**Config**

**Bases:** RoBERTaTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**columns: list[str] = `['text']`**

**tokenizer: *Tokenizer.Config* = *GPT2BPETokenizer.Config*()**

**base_tokenizer: Optional[*Tokenizer.Config*] = `None`**

**vocab_file: str = `'gpt2_bpe_dict'`**

**max_seq_len: int = `256`**

**add_selfie_token: bool = `False`**

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "text"
    ],
    "tokenizer": {
        "GPT2BPETokenizer": {
            "bpe_encoder_path": "manifold://pytext_training/tree/static/vocabs/bpe/
↪gpt2/encoder.json",
            "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/bpe/gpt2/
↪vocab.bpe",
            "lowercase": false
        }
    },
    "base_tokenizer": null,
    "vocab_file": "gpt2_bpe_dict",
    "max_seq_len": 256,
    "add_selfie_token": false
}
```

### disjoint_multitask_data

### DisjointMultitaskData.Config

**Component:** *DisjointMultitaskData*

**class** DisjointMultitaskData.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**sampler: *BaseBatchSampler.Config* = *RoundRobinBatchSampler.Config*()**

**test_key: Optional[str] = None**

**Default JSON**

```
{
    "sampler": {
        "RoundRobinBatchSampler": {
            "iter_to_set_epoch": ""
        }
    },
    "test_key": null
}
```

### disjoint_multitask_data_handler

### DisjointMultitaskDataHandler.Config

**Component:** *DisjointMultitaskDataHandler*

**class** DisjointMultitaskDataHandler.**Config**

**Bases:** DataHandler.Config

Configuration class for *DisjointMultitaskDataHandler*.

**upsample**

If upsample, keep cycling over each iterator in round-robin. Iterators with less batches will get more passes. If False, we do single pass over each iterator, the ones which run out will sit idle. This is used for evaluation. Default True.

**Type** bool

**All Attributes (including base classes)**

**columns_to_read: list[str] = []**

**shuffle: bool = True**

**sort_within_batch: bool = True**

**train_path: str = 'train.tsv'**

**eval_path: str = 'eval.tsv'**

**test_path: str = 'test.tsv'**

**train_batch_size: int = 128**

**eval_batch_size: int = 128**

**test_batch_size: int = 128**

**column_mapping: dict[str, str] = {}**

**upsample: bool = `True`**

**Default JSON**

```
{
    "columns_to_read": [],
    "shuffle": true,
    "sort_within_batch": true,
    "train_path": "train.tsv",
    "eval_path": "eval.tsv",
    "test_path": "test.tsv",
    "train_batch_size": 128,
    "eval_batch_size": 128,
    "test_batch_size": 128,
    "column_mapping": {},
    "upsample": true
}
```

### dynamic_pooling_batcher

### BatcherSchedulerConfig

**Component:** *Module*

**class** pytext.data.dynamic_pooling_batcher.**BatcherSchedulerConfig**

**Bases:** `Module.Config`

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**start_batch_size: int = `32`**

**end_batch_size: int = `256`**

**epoch_period: int = `10`**

**step_size: int = `1`**

**Subclasses**

- *ExponentialBatcherSchedulerConfig*

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "start_batch_size": 32,
    "end_batch_size": 256,
    "epoch_period": 10,
    "step_size": 1
}
```

### DynamicPoolingBatcher.Config

**Component:** *DynamicPoolingBatcher*

**class** DynamicPoolingBatcher.**Config**

**Bases:** PoolingBatcher.Config

**All Attributes (including base classes)**

**train_batch_size: int = 16**

**eval_batch_size: int = 16**

**test_batch_size: int = 16**

**pool_num_batches: int = 1000**

**num_shuffled_pools: int = 1**

**scheduler_config:** *BatcherSchedulerConfig* **=** *BatcherSchedulerConfig*()

**Subclasses**

- ExponentialDynamicPoolingBatcher.Config
- LinearDynamicPoolingBatcher.Config

**Default JSON**

```
{
    "train_batch_size": 16,
    "eval_batch_size": 16,
    "test_batch_size": 16,
    "pool_num_batches": 1000,
    "num_shuffled_pools": 1,
    "scheduler_config": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "start_batch_size": 32,
        "end_batch_size": 256,
        "epoch_period": 10,
        "step_size": 1
    }
}
```

### ExponentialBatcherSchedulerConfig

**Component:** *Module*

**class** pytext.data.dynamic_pooling_batcher.**ExponentialBatcherSchedulerConfig**

**Bases:** *BatcherSchedulerConfig*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**start_batch_size: int = 32**

**end_batch_size: int = 256**

**epoch_period: int = 10**

**step_size: int = 1**

**gamma: float = 5**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "start_batch_size": 32,
    "end_batch_size": 256,
    "epoch_period": 10,
    "step_size": 1,
    "gamma": 5
}
```

### ExponentialDynamicPoolingBatcher.Config

**Component:** *ExponentialDynamicPoolingBatcher*

**class** ExponentialDynamicPoolingBatcher.**Config**

**Bases:** DynamicPoolingBatcher.Config

**All Attributes (including base classes)**

**train_batch_size: int = 16**

**eval_batch_size: int = 16**

**test_batch_size: int = 16**

**pool_num_batches: int = 1000**

**num_shuffled_pools: int = 1**

**scheduler_config:** *ExponentialBatcherSchedulerConfig* **=** *BatcherSchedulerConfig*()

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### LinearDynamicPoolingBatcher.Config

**Component:** *LinearDynamicPoolingBatcher*

**class** LinearDynamicPoolingBatcher.**Config**

**Bases:** DynamicPoolingBatcher.Config

**All Attributes (including base classes)**

**train_batch_size: int = 16**

**eval_batch_size: int = 16**

**test_batch_size: int = 16**

**pool_num_batches: int = 1000**

**num_shuffled_pools: int = 1**

**scheduler_config:** *BatcherSchedulerConfig* = *BatcherSchedulerConfig*()

**Default JSON**

```
{
    "train_batch_size": 16,
    "eval_batch_size": 16,
    "test_batch_size": 16,
    "pool_num_batches": 1000,
    "num_shuffled_pools": 1,
    "scheduler_config": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "start_batch_size": 32,
        "end_batch_size": 256,
        "epoch_period": 10,
        "step_size": 1
    }
}
```

### featurizer

### featurizer

### Featurizer.Config

**Component:** *Featurizer*

**class** Featurizer.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### simple_featurizer

### SimpleFeaturizer.Config

**Component:** *SimpleFeaturizer*

**class** SimpleFeaturizer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**sentence_markers: Optional[tuple[str, str]] = None**

**lowercase_tokens: bool = `True`**

**split_regex: str = `'\\s+'`**

**convert_to_bytes: bool = `False`**

**Default JSON**

```
{
    "sentence_markers": null,
    "lowercase_tokens": true,
    "split_regex": "\\s+",
    "convert_to_bytes": false
}
```

### packed_lm_data

### PackedLMData.Config

**Component:** *PackedLMData*

**class** PackedLMData.**Config**

**Bases:** Data.Config

**All Attributes (including base classes)**

**source:** *DataSource.Config* = *TSVDataSource.Config*()

**batcher:** *Batcher.Config* = *PoolingBatcher.Config*()

**sort_key: Optional[str] = `None`**

**in_memory: Optional[bool] = `True`**

**max_seq_len: int = `128`**

**Default JSON**

```
{
    "source": {
        "TSVDataSource": {
            "column_mapping": {},
            "train_filename": null,
            "test_filename": null,
            "eval_filename": null,
            "field_names": null,
            "delimiter": "\t",
            "quoted": false,
            "drop_incomplete_rows": false
        }
    },
    "batcher": {
        "PoolingBatcher": {
            "train_batch_size": 16,
            "eval_batch_size": 16,
            "test_batch_size": 16,
            "pool_num_batches": 1000,
            "num_shuffled_pools": 1
        }
    },
```

(continues on next page)

(continued from previous page)

```
    "sort_key": null,
    "in_memory": true,
    "max_seq_len": 128
}
```

### roberta_tensorizer

### RoBERTaTensorizer.Config

**Component:** *RoBERTaTensorizer*

**class** RoBERTaTensorizer.**Config**

**Bases:** BERTTensorizerBase.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**columns: list[str] = `['text']`**

**tokenizer:** *Tokenizer.Config* = *GPT2BPETokenizer.Config*()

**base_tokenizer: Optional[***Tokenizer.Config***] = `None`**

**vocab_file: str = `'gpt2_bpe_dict'`**

**max_seq_len: int = `256`**

**add_selfie_token: bool = `False`**

**Subclasses**

- RoBERTaContextTensorizerForDenseRetrieval.Config
- RoBERTaTokenLevelTensorizer.Config
- SquadForRoBERTaTensorizer.Config
- SquadForRoBERTaTensorizerForKD.Config

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "text"
    ],
    "tokenizer": {
        "GPT2BPETokenizer": {
            "bpe_encoder_path": "manifold://pytext_training/tree/static/vocabs/bpe/
↪gpt2/encoder.json",
            "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/bpe/gpt2/
↪vocab.bpe",
            "lowercase": false
        }
    },
    "base_tokenizer": null,
    "vocab_file": "gpt2_bpe_dict",
    "max_seq_len": 256,
```

(continues on next page)

(continued from previous page)

```
    "add_selfie_token": false
}
```

### RoBERTaTokenLevelTensorizer.Config

**Component:** *RoBERTaTokenLevelTensorizer*

**class** RoBERTaTokenLevelTensorizer.**Config**

**Bases:** RoBERTaTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**columns: list[str] = `['text']`**

**tokenizer: *Tokenizer.Config* = *GPT2BPETokenizer.Config*()**

**base_tokenizer: Optional[*Tokenizer.Config*] = `None`**

**vocab_file: str = `'gpt2_bpe_dict'`**

**max_seq_len: int = `256`**

**add_selfie_token: bool = `False`**

**labels_columns: list[str] = `['label']`**

**labels: list[str] = `[]`**

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "text"
    ],
    "tokenizer": {
        "GPT2BPETokenizer": {
            "bpe_encoder_path": "manifold://pytext_training/tree/static/vocabs/bpe/
↪gpt2/encoder.json",
            "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/bpe/gpt2/
↪vocab.bpe",
            "lowercase": false
        }
    },
    "base_tokenizer": null,
    "vocab_file": "gpt2_bpe_dict",
    "max_seq_len": 256,
    "add_selfie_token": false,
    "labels_columns": [
        "label"
    ],
    "labels": []
}
```

### sources

### conllu

#### CoNLLUNERDataSource.Config

**Component:** *CoNLLUNERDataSource*

**class** CoNLLUNERDataSource.**Config**
**Bases:** CoNLLUPOSDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = { }**

**language: Optional[str] = None**

**train_filename: Optional[str] = None**

**test_filename: Optional[str] = None**

**eval_filename: Optional[str] = None**

**field_names: Optional[list[str]] = None**

**delimiter: str = '\t'**

**Default JSON**

```
{
    "column_mapping": {},
    "language": null,
    "train_filename": null,
    "test_filename": null,
    "eval_filename": null,
    "field_names": null,
    "delimiter": "\t"
}
```

#### CoNLLUPOSDataSource.Config

**Component:** *CoNLLUPOSDataSource*

**class** CoNLLUPOSDataSource.**Config**
**Bases:** RootDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = { }**

**language: Optional[str] = None** Name of the language. If not set, languages will be empty.

**train_filename: Optional[str] = None** Filename of training set. If not set, iteration will be empty.

**test_filename: Optional[str] = None** Filename of testing set. If not set, iteration will be empty.

**eval_filename: Optional[str] = None** Filename of eval set. If not set, iteration will be empty.

**field_names: Optional[list[str]] = None** Field names for the TSV. If this is not set, the first line of each file will be assumed to be a header containing the field names.

**delimiter: str = '\t'** The column delimiter. CoNLL-U file default is t.

**Subclasses**

- CoNLLUNERDataSource.Config

**Default JSON**

```
{
    "column_mapping": {},
    "language": null,
    "train_filename": null,
    "test_filename": null,
    "eval_filename": null,
    "field_names": null,
    "delimiter": "\t"
}
```

### data_source

### DataSource.Config

**Component:** *DataSource*

**class** DataSource.**Config**
**Bases:** Component.Config

**All Attributes (including base classes)**

**Subclasses**

- RowShardedDataSource.Config
- ShardedDataSource.Config
- DenseRetrievalDataSource.Config
- SquadDataSource.Config
- SquadDataSourceForKD.Config

**Default JSON**

```
{}
```

### RootDataSource.Config

**Component:** *RootDataSource*

**class** RootDataSource.**Config**
**Bases:** Component.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = { }** An optional column mapping, allowing the columns in the raw data source to not map directly to the column names in the schema. This mapping will remap names from the raw data source to names in the schema.

**Subclasses**

- CoNLLUNERDataSource.Config
- CoNLLUPOSDataSource.Config

- PandasDataSource.Config
- SessionPandasDataSource.Config
- SessionDataSource.Config
- BlockShardedTSVDataSource.Config
- MultilingualTSVDataSource.Config
- SessionTSVDataSource.Config
- TSVDataSource.Config

**Default JSON**

```
{
    "column_mapping": {}
}
```

### RowShardedDataSource.Config

**Component:** *RowShardedDataSource*

**class** RowShardedDataSource.**Config**
**Bases:** ShardedDataSource.Config

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### ShardedDataSource.Config

**Component:** *ShardedDataSource*

**class** ShardedDataSource.**Config**
**Bases:** DataSource.Config

**All Attributes (including base classes)**

**Subclasses**

- RowShardedDataSource.Config

**Default JSON**

```
{}
```

### dense_retrieval

### DenseRetrievalDataSource.Config

**Component:** *DenseRetrievalDataSource*

**class** DenseRetrievalDataSource.**Config**
**Bases:** DataSource.Config

**All Attributes (including base classes)**

**train_filename: Optional[str] = `'train-v2.0.json'`**

**test_filename: Optional[str] = `'dev-v2.0.json'`**

**eval_filename: Optional[str] = `'dev-v2.0.json'`**

**num_negative_ctxs: int = `1`**

**use_title: bool = `True`**

**use_cache: bool = `False`**

**Default JSON**

```
{
    "train_filename": "train-v2.0.json",
    "test_filename": "dev-v2.0.json",
    "eval_filename": "dev-v2.0.json",
    "num_negative_ctxs": 1,
    "use_title": true,
    "use_cache": false
}
```

## pandas

### PandasDataSource.Config

**Component:** *PandasDataSource*

**class** PandasDataSource.**Config**

**Bases:** RootDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = { }**

**train_df: Optional[DataFrame] = `None`**

**test_df: Optional[DataFrame] = `None`**

**eval_df: Optional[DataFrame] = `None`**

**Subclasses**

- SessionPandasDataSource.Config

**Default JSON**

```
{
    "column_mapping": {},
    "train_df": null,
    "test_df": null,
    "eval_df": null
}
```

### SessionPandasDataSource.Config

**Component:** *SessionPandasDataSource*

**class** SessionPandasDataSource.**Config**

**Bases:** PandasDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = {}**

**train_df: Optional[DataFrame] = None**

**test_df: Optional[DataFrame] = None**

**eval_df: Optional[DataFrame] = None**

**Default JSON**

```
{
    "column_mapping": {},
    "train_df": null,
    "test_df": null,
    "eval_df": null
}
```

### session

### SessionDataSource.Config

**Component:** *SessionDataSource*

**class** SessionDataSource.**Config**

**Bases:** RootDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = {}**

**Default JSON**

```
{
    "column_mapping": {}
}
```

### squad

### SquadDataSource.Config

**Component:** *SquadDataSource*

**class** SquadDataSource.**Config**

**Bases:** DataSource.Config

**All Attributes (including base classes)**

**train_filename: Optional[str] = 'train-v2.0.json'**

**test_filename: Optional[str] = 'dev-v2.0.json'**

**eval_filename: Optional[str] = 'dev-v2.0.json'**

**ignore_impossible: bool = True**

**max_character_length: int = 1048576**

**min_overlap: float = 0.1**

**delimiter: str = '\t'**

**quoted: bool = False**

**Subclasses**

- SquadDataSourceForKD.Config

**Default JSON**

```
{
    "train_filename": "train-v2.0.json",
    "test_filename": "dev-v2.0.json",
    "eval_filename": "dev-v2.0.json",
    "ignore_impossible": true,
    "max_character_length": 1048576,
    "min_overlap": 0.1,
    "delimiter": "\t",
    "quoted": false
}
```

### SquadDataSourceForKD.Config

**Component:** *SquadDataSourceForKD*

**class** SquadDataSourceForKD.**Config**

**Bases:** SquadDataSource.Config

**All Attributes (including base classes)**

**train_filename: Optional[str] = 'train-v2.0.json'**

**test_filename: Optional[str] = 'dev-v2.0.json'**

**eval_filename: Optional[str] = 'dev-v2.0.json'**

**ignore_impossible: bool = True**

**max_character_length: int = 1048576**

**min_overlap: float = 0.1**

**delimiter: str = '\t'**

**quoted: bool = False**

**Default JSON**

```
{
    "train_filename": "train-v2.0.json",
    "test_filename": "dev-v2.0.json",
    "eval_filename": "dev-v2.0.json",
    "ignore_impossible": true,
    "max_character_length": 1048576,
    "min_overlap": 0.1,
    "delimiter": "\t",
    "quoted": false
}
```

### tsv

#### BlockShardedTSVDataSource.Config

**Component:** *BlockShardedTSVDataSource*

**class** BlockShardedTSVDataSource.**Config**

**Bases:** TSVDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = { }**

**train_filename: Optional[str] = None**

**test_filename: Optional[str] = None**

**eval_filename: Optional[str] = None**

**field_names: Optional[list[str]] = None**

**delimiter: str = '\t'**

**quoted: bool = False**

**drop_incomplete_rows: bool = False**

**Default JSON**

```
{
    "column_mapping": {},
    "train_filename": null,
    "test_filename": null,
    "eval_filename": null,
    "field_names": null,
    "delimiter": "\t",
    "quoted": false,
    "drop_incomplete_rows": false
}
```

#### MultilingualTSVDataSource.Config

**Component:** *MultilingualTSVDataSource*

**class** MultilingualTSVDataSource.**Config**

**Bases:** TSVDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = { }**

**train_filename: Optional[str] = None**

**test_filename: Optional[str] = None**

**eval_filename: Optional[str] = None**

**field_names: Optional[list[str]] = None**

**delimiter: str = '\t'**

**quoted: bool = False**

**drop_incomplete_rows: bool = `False`**

**data_source_languages: dict[str, list[str]] = `{'train': ['en'], 'eval': ['en'], 'test': ['en']}`**

**language_columns: list[str] = `['language']`**

**Default JSON**

```
{
    "column_mapping": {},
    "train_filename": null,
    "test_filename": null,
    "eval_filename": null,
    "field_names": null,
    "delimiter": "\t",
    "quoted": false,
    "drop_incomplete_rows": false,
    "data_source_languages": {
        "train": [
            "en"
        ],
        "eval": [
            "en"
        ],
        "test": [
            "en"
        ]
    },
    "language_columns": [
        "language"
    ]
}
```

### SessionTSVDataSource.Config

**Component:** *SessionTSVDataSource*

**class** SessionTSVDataSource.**Config**

**Bases:** TSVDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = `{}`**

**train_filename: Optional[str] = `None`**

**test_filename: Optional[str] = `None`**

**eval_filename: Optional[str] = `None`**

**field_names: Optional[list[str]] = `None`**

**delimiter: str = `'\t'`**

**quoted: bool = `False`**

**drop_incomplete_rows: bool = `False`**

**Default JSON**

```
{
    "column_mapping": {},
    "train_filename": null,
    "test_filename": null,
    "eval_filename": null,
    "field_names": null,
    "delimiter": "\t",
    "quoted": false,
    "drop_incomplete_rows": false
}
```

### TSVDataSource.Config

**Component:** *TSVDataSource*

**class** TSVDataSource.**Config**

**Bases:** RootDataSource.Config

**All Attributes (including base classes)**

**column_mapping: dict[str, str] = { }**

**train_filename: Optional[str] = None** Filename of training set. If not set, iteration will be empty.

**test_filename: Optional[str] = None** Filename of testing set. If not set, iteration will be empty.

**eval_filename: Optional[str] = None** Filename of eval set. If not set, iteration will be empty.

**field_names: Optional[list[str]] = None** Field names for the TSV. If this is not set, the first line of each file will be assumed to be a header containing the field names.

**delimiter: str = '\t'** The column delimiter passed to Python's csv library. Change to "," for csv.

**quoted: bool = False** Whether the columns can use quotes to include delimiters or not. Rows with unclosed quotes will be merged with n inside. Change to True for quoted csv.

**drop_incomplete_rows: bool = False**

**Subclasses**

- BlockShardedTSVDataSource.Config
- MultilingualTSVDataSource.Config
- SessionTSVDataSource.Config

**Default JSON**

```
{
    "column_mapping": {},
    "train_filename": null,
    "test_filename": null,
    "eval_filename": null,
    "field_names": null,
    "delimiter": "\t",
    "quoted": false,
    "drop_incomplete_rows": false
}
```

### squad_for_bert_tensorizer

### SquadForBERTTensorizer.Config

**Component:** *SquadForBERTTensorizer*

**class** SquadForBERTTensorizer.**Config**

**Bases:** BERTTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**columns: list[str] = ['question', 'doc']**

**tokenizer:** *Tokenizer.Config* = *WordPieceTokenizer.Config*()

**base_tokenizer: Optional[***Tokenizer.Config***] = None**

**vocab_file: str = 'manifold://nlp_technologies/tree/huggingface-models/bert-base-uncased/voca**

**max_seq_len: int = 256**

**answers_column: str = 'answers'**

**answer_starts_column: str = 'answer_starts'**

**Subclasses**

- SquadForBERTTensorizerForKD.Config

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "question",
        "doc"
    ],
    "tokenizer": {
        "WordPieceTokenizer": {
            "basic_tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            },
            "wordpiece_vocab_path": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt"
        }
    },
    "base_tokenizer": null,
    "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-base-
↪uncased/vocab.txt",
    "max_seq_len": 256,
    "answers_column": "answers",
    "answer_starts_column": "answer_starts"
}
```

### SquadForBERTTensorizerForKD.Config

**Component:** *SquadForBERTTensorizerForKD*

**class** SquadForBERTTensorizerForKD.**Config**

**Bases:** SquadForBERTTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**columns: list[str] = `['question', 'doc']`**

**tokenizer: *Tokenizer.Config* = *WordPieceTokenizer.Config*()**

**base_tokenizer: Optional[*Tokenizer.Config*] = `None`**

**vocab_file: str = `'manifold://nlp_technologies/tree/huggingface-models/bert-base-uncased/voca`**

**max_seq_len: int = `256`**

**answers_column: str = `'answers'`**

**answer_starts_column: str = `'answer_starts'`**

**start_logits_column: str = `'start_logits'`**

**end_logits_column: str = `'end_logits'`**

**has_answer_logits_column: str = `'has_answer_logits'`**

**pad_mask_column: str = `'pad_mask'`**

**segment_labels_column: str = `'segment_labels'`**

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "question",
        "doc"
    ],
    "tokenizer": {
        "WordPieceTokenizer": {
            "basic_tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            },
            "wordpiece_vocab_path": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt"
        }
    },
    "base_tokenizer": null,
    "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-base-
↪uncased/vocab.txt",
    "max_seq_len": 256,
    "answers_column": "answers",
    "answer_starts_column": "answer_starts",
    "start_logits_column": "start_logits",
    "end_logits_column": "end_logits",
```

(continues on next page)

(continued from previous page)

```
    "has_answer_logits_column": "has_answer_logits",
    "pad_mask_column": "pad_mask",
    "segment_labels_column": "segment_labels"
}
```

### SquadForRoBERTaTensorizer.Config

**Component:** *SquadForRoBERTaTensorizer*

**class** SquadForRoBERTaTensorizer.**Config**

**Bases:** RoBERTaTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**columns: list[str] = ['question', 'doc']**

**tokenizer: *Tokenizer.Config* = *GPT2BPETokenizer.Config*()**

**base_tokenizer: Optional[*Tokenizer.Config*] = None**

**vocab_file: str = 'gpt2_bpe_dict'**

**max_seq_len: int = 256**

**add_selfie_token: bool = False**

**answers_column: str = 'answers'**

**answer_starts_column: str = 'answer_starts'**

**Subclasses**

- SquadForRoBERTaTensorizerForKD.Config

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "question",
        "doc"
    ],
    "tokenizer": {
        "GPT2BPETokenizer": {
            "bpe_encoder_path": "manifold://pytext_training/tree/static/vocabs/bpe/
↪gpt2/encoder.json",
            "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/bpe/gpt2/
↪vocab.bpe",
            "lowercase": false
        }
    },
    "base_tokenizer": null,
    "vocab_file": "gpt2_bpe_dict",
    "max_seq_len": 256,
    "add_selfie_token": false,
    "answers_column": "answers",
    "answer_starts_column": "answer_starts"
}
```

### SquadForRoBERTaTensorizerForKD.Config

**Component:** *SquadForRoBERTaTensorizerForKD*

**class** SquadForRoBERTaTensorizerForKD.**Config**

**Bases:** SquadForRoBERTaTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**columns: list[str] = `['question', 'doc']`**

**tokenizer: *Tokenizer.Config* = *GPT2BPETokenizer.Config*()**

**base_tokenizer: Optional[*Tokenizer.Config*] = `None`**

**vocab_file: str = `'gpt2_bpe_dict'`**

**max_seq_len: int = `256`**

**add_selfie_token: bool = `False`**

**answers_column: str = `'answers'`**

**answer_starts_column: str = `'answer_starts'`**

**start_logits_column: str = `'start_logits'`**

**end_logits_column: str = `'end_logits'`**

**has_answer_logits_column: str = `'has_answer_logits'`**

**pad_mask_column: str = `'pad_mask'`**

**segment_labels_column: str = `'segment_labels'`**

**Default JSON**

```
{
    "is_input": true,
    "columns": [
        "question",
        "doc"
    ],
    "tokenizer": {
        "GPT2BPETokenizer": {
            "bpe_encoder_path": "manifold://pytext_training/tree/static/vocabs/bpe/
↪gpt2/encoder.json",
            "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/bpe/gpt2/
↪vocab.bpe",
            "lowercase": false
        }
    },
    "base_tokenizer": null,
    "vocab_file": "gpt2_bpe_dict",
    "max_seq_len": 256,
    "add_selfie_token": false,
    "answers_column": "answers",
    "answer_starts_column": "answer_starts",
    "start_logits_column": "start_logits",
    "end_logits_column": "end_logits",
    "has_answer_logits_column": "has_answer_logits",
    "pad_mask_column": "pad_mask",
```

(continues on next page)

(continued from previous page)

```
    "segment_labels_column": "segment_labels"
}
```

### squad_tensorizer

### SquadTensorizer.Config

**Component:** *SquadTensorizer*

**class** SquadTensorizer.**Config**

**Bases:** TokenTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str = 'text'**

**tokenizer:** *Tokenizer.Config* = *Tokenizer.Config*(**split_regex='\\W+'**)

**add_bos_token: bool = False**

**add_eos_token: bool = False**

**use_eos_token_for_bos: bool = False**

**max_seq_len: Optional[int] = None**

**vocab:** *VocabConfig* = *VocabConfig*()

**vocab_file_delimiter: str = ' '**

**doc_column: str = 'doc'**

**ques_column: str = 'question'**

**answers_column: str = 'answers'**

**answer_starts_column: str = 'answer_starts'**

**max_ques_seq_len: int = 64**

**max_doc_seq_len: int = 256**

**Subclasses**

- SquadTensorizerForKD.Config

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\W+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "add_bos_token": false,
    "add_eos_token": false,
```

(continues on next page)

(continued from previous page)

```
    "use_eos_token_for_bos": false,
    "max_seq_len": null,
    "vocab": {
        "build_from_data": true,
        "size_from_data": 0,
        "min_counts": 0,
        "vocab_files": []
    },
    "vocab_file_delimiter": " ",
    "doc_column": "doc",
    "ques_column": "question",
    "answers_column": "answers",
    "answer_starts_column": "answer_starts",
    "max_ques_seq_len": 64,
    "max_doc_seq_len": 256
}
```

### SquadTensorizerForKD.Config

**Component:** *SquadTensorizerForKD*

**class** SquadTensorizerForKD.**Config**

**Bases:** SquadTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'text'`**

**tokenizer:** *Tokenizer.Config* = *Tokenizer.Config*(**split_regex=`'\\W+'`**)

**add_bos_token: bool = `False`**

**add_eos_token: bool = `False`**

**use_eos_token_for_bos: bool = `False`**

**max_seq_len: Optional[int] = `None`**

**vocab:** *VocabConfig* = *VocabConfig*()

**vocab_file_delimiter: str = `' '`**

**doc_column: str = `'doc'`**

**ques_column: str = `'question'`**

**answers_column: str = `'answers'`**

**answer_starts_column: str = `'answer_starts'`**

**max_ques_seq_len: int = `64`**

**max_doc_seq_len: int = `256`**

**start_logits_column: str = `'start_logits'`**

**end_logits_column: str = `'end_logits'`**

**has_answer_logits_column: str = `'has_answer_logits'`**

**pad_mask_column: str = `'pad_mask'`**

**segment_labels_column: str = `'segment_labels'`**

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\W+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false,
    "max_seq_len": null,
    "vocab": {
        "build_from_data": true,
        "size_from_data": 0,
        "min_counts": 0,
        "vocab_files": []
    },
    "vocab_file_delimiter": " ",
    "doc_column": "doc",
    "ques_column": "question",
    "answers_column": "answers",
    "answer_starts_column": "answer_starts",
    "max_ques_seq_len": 64,
    "max_doc_seq_len": 256,
    "start_logits_column": "start_logits",
    "end_logits_column": "end_logits",
    "has_answer_logits_column": "has_answer_logits",
    "pad_mask_column": "pad_mask",
    "segment_labels_column": "segment_labels"
}
```

### tensorizers

### AnnotationNumberizer.Config

**Component:** *AnnotationNumberizer*

**class** AnnotationNumberizer.**Config**

**Bases:** `Tensorizer.Config`

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'seqlogical'`**

**Default JSON**

```
{
    "is_input": true,
```

(continues on next page)

(continued from previous page)

```
    "column": "seqlogical"
}
```

### ByteTensorizer.Config

**Component:** *ByteTensorizer*

**class** ByteTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'text'`** The name of the text column to parse from the data source.

**lower: bool = `True`**

**max_seq_len: Optional[int] = `None`**

**add_bos_token: Optional[bool] = `False`**

**add_eos_token: Optional[bool] = `False`**

**use_eos_token_for_bos: Optional[bool] = `False`**

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "lower": true,
    "max_seq_len": null,
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false
}
```

### ByteTokenTensorizer.Config

**Component:** *ByteTokenTensorizer*

**class** ByteTokenTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'text'`** The name of the text column to parse from the data source.

**tokenizer:** ***Tokenizer.Config*** **=** ***Tokenizer.Config*****()** The tokenizer to use to split input text into tokens.

**max_seq_len: Optional[int] = `None`** The max token length for input text.

**max_byte_len: int = `15`** The max byte length for a token.

**offset_for_non_padding: int = `0`** Offset to add to all non-padding bytes

**add_bos_token: bool = `False`**

**add_eos_token: bool = `False`**

**use_eos_token_for_bos: bool = `False`**

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "max_seq_len": null,
    "max_byte_len": 15,
    "offset_for_non_padding": 0,
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false
}
```

### CharacterTokenTensorizer.Config

**Component:** *`CharacterTokenTensorizer`*

**class** `CharacterTokenTensorizer.`**`Config`**

**Bases:** `TokenTensorizer.Config`

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'text'`**

**tokenizer:** ***Tokenizer.Config*** **=** ***Tokenizer.Config*****()**

**add_bos_token: bool = `False`**

**add_eos_token: bool = `False`**

**use_eos_token_for_bos: bool = `False`**

**max_seq_len: Optional[int] = `None`**

**vocab:** ***VocabConfig*** **=** ***VocabConfig*****()**

**vocab_file_delimiter: str = `' '`**

**max_char_length: int = `20`** The max character length for a token.

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
```

(continues on next page)

(continued from previous page)

```
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false,
    "max_seq_len": null,
    "vocab": {
        "build_from_data": true,
        "size_from_data": 0,
        "min_counts": 0,
        "vocab_files": []
    },
    "vocab_file_delimiter": " ",
    "max_char_length": 20
}
```

### CharacterVocabTokenTensorizer.Config

**Component:** *CharacterVocabTokenTensorizer*

**class** CharacterVocabTokenTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'text'`** The name of the text column to parse from the data source.

**tokenizer: *Tokenizer.Config* = *Tokenizer.Config*()** The tokenizer to use to split input text into tokens.

**add_bos_token: bool = `False`**

**add_eos_token: bool = `False`**

**use_eos_token_for_bos: bool = `False`**

**max_seq_len: Optional[int] = `None`**

**vocab: *VocabConfig* = *VocabConfig*()**

**vocab_file_delimiter: str = `' '`**

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "add_bos_token": false,
    "add_eos_token": false,
```

(continues on next page)

(continued from previous page)

```
    "use_eos_token_for_bos": false,
    "max_seq_len": null,
    "vocab": {
        "build_from_data": true,
        "size_from_data": 0,
        "min_counts": 0,
        "vocab_files": []
    },
    "vocab_file_delimiter": " "
}
```

### Float1DListTensorizer.Config

**Component:** *Float1DListTensorizer*

**class** Float1DListTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str = 'float_list_column'**

**pad_token: float = 1.0**

**Default JSON**

```
{
    "is_input": true,
    "column": "float_list_column",
    "pad_token": 1.0
}
```

### FloatListSeqTensorizer.Config

**Component:** *FloatListSeqTensorizer*

**class** FloatListSeqTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str** The name of the label column to parse from the data source.

**error_check: bool = False**

**dim: Optional[int] = None**

**pad_token: float = -1.0**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### FloatListTensorizer.Config

**Component:** *FloatListTensorizer*

**class** FloatListTensorizer.**Config**
**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str** The name of the label column to parse from the data source.

**error_check: bool = False**

**dim: Optional[int] = None**

**normalize: bool = False**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### FloatTensorizer.Config

**Component:** *FloatTensorizer*

**class** FloatTensorizer.**Config**
**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str** The name of the column to parse from the data source.

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### GazetteerTensorizer.Config

**Component:** *GazetteerTensorizer*

**class** GazetteerTensorizer.**Config**
**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**text_column: str = 'text'**

**dict_column: str = 'dict'**

**tokenizer:** ***Tokenizer.Config*** **=** ***Tokenizer.Config*****()** tokenizer to split text and create dict tensors of the same size.

**Default JSON**

```
{
    "is_input": true,
    "text_column": "text",
    "dict_column": "dict",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    }
}
```

### Integer1DListTensorizer.Config

**Component:** *Integer1DListTensorizer*

**class** Integer1DListTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str = 'int_list_column'**

**Default JSON**

```
{
    "is_input": true,
    "column": "int_list_column"
}
```

### LabelListRankTensorizer.Config

**Component:** *LabelListRankTensorizer*

**class** LabelListRankTensorizer.**Config**

**Bases:** LabelTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = False**

**column: str = 'label'**

**allow_unknown: bool = False**

**pad_in_vocab: bool = False**

**label_vocab: Optional[list[str]] = None**

**label_vocab_file: Optional[str] = None**

**add_labels: Optional[list[str]] = None**

**pad_missing: bool = False**

**Default JSON**

```
{
    "is_input": false,
    "column": "label",
    "allow_unknown": false,
    "pad_in_vocab": false,
    "label_vocab": null,
    "label_vocab_file": null,
    "add_labels": null,
    "pad_missing": false
}
```

### LabelListTensorizer.Config

**Component:** *LabelListTensorizer*

**class** LabelListTensorizer.**Config**

**Bases:** LabelTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = False**

**column: str = 'label'**

**allow_unknown: bool = False**

**pad_in_vocab: bool = False**

**label_vocab: Optional[list[str]] = None**

**label_vocab_file: Optional[str] = None**

**add_labels: Optional[list[str]] = None**

**pad_missing: bool = False**

**Default JSON**

```
{
    "is_input": false,
    "column": "label",
    "allow_unknown": false,
    "pad_in_vocab": false,
    "label_vocab": null,
    "label_vocab_file": null,
    "add_labels": null,
    "pad_missing": false
}
```

### LabelTensorizer.Config

**Component:** *LabelTensorizer*

**class** LabelTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = False**

**column: str = `'label'`** The name of the label column to parse from the data source.

**allow_unknown: bool = `False`** Whether to allow for unknown labels at test/prediction time.

**pad_in_vocab: bool = `False`** Whether vocab should have pad, usually false when label is used as target.

**label_vocab: Optional[list[str]] = `None`** The label values, if known. Will skip initialization step if provided.

**label_vocab_file: Optional[str] = `None`** File with the label values. This can be used when the label space is too large to specify these as a list. The file should not contain a header.

**add_labels: Optional[list[str]] = `None`** Add these labels to the vocabulary during the initialization step (only if the initialization step is not skipped). Useful when the dataset may not include all labels, as for incremental trainings.

**Subclasses**

- `PositiveLabelTensorizerForDenseRetrieval.Config`
- `LabelListRankTensorizer.Config`
- `LabelListTensorizer.Config`
- `SoftLabelTensorizer.Config`

**Default JSON**

```
{
    "is_input": false,
    "column": "label",
    "allow_unknown": false,
    "pad_in_vocab": false,
    "label_vocab": null,
    "label_vocab_file": null,
    "add_labels": null
}
```

### MetricTensorizer.Config

**Component:** *`MetricTensorizer`*

**class** `MetricTensorizer.`**`Config`**

**Bases:** `Tensorizer.Config`

**All Attributes (including base classes)**

**is_input: bool = `False`**

**names: list[str]**

**indexes: list[int]**

**Subclasses**

- `NtokensTensorizer.Config`

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### NtokensTensorizer.Config

**Component:** *NtokensTensorizer*

**class** NtokensTensorizer.**Config**
**Bases:** MetricTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `False`**

**names: list[str]**

**indexes: list[int]**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### NumericLabelTensorizer.Config

**Component:** *NumericLabelTensorizer*

**class** NumericLabelTensorizer.**Config**
**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `False`**

**column: str = `'label'`** The name of the label column to parse from the data source.

**rescale_range: Optional[list[float]] = `None`** If provided, the range of values the raw label can be. Will rescale the label values to be within [0, 1].

**Default JSON**

```
{
    "is_input": false,
    "column": "label",
    "rescale_range": null
}
```

### SeqTokenTensorizer.Config

**Component:** *SeqTokenTensorizer*

**class** SeqTokenTensorizer.**Config**
**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'text_seq'`**

**max_seq_len: Optional[int] = `None`**

**add_bos_token: bool = `False`** sentence markers

**add_eos_token: bool = `False`**

**use_eos_token_for_bos: bool = `False`**

**add_bol_token: bool = `False`** list markers

**add_eol_token: bool = `False`**

**use_eol_token_for_bol: bool = `False`**

**tokenizer:** *Tokenizer.Config* = *Tokenizer.Config*() The tokenizer to use to split input text into tokens.

**max_turn: int = `50`**

**Default JSON**

```
{
    "is_input": true,
    "column": "text_seq",
    "max_seq_len": null,
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false,
    "add_bol_token": false,
    "add_eol_token": false,
    "use_eol_token_for_bol": false,
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "max_turn": 50
}
```

### SlotLabelTensorizer.Config

**Component:** *SlotLabelTensorizer*

**class** `SlotLabelTensorizer.`**`Config`**

**Bases:** `Tensorizer.Config`

**All Attributes (including base classes)**

**is_input: bool = `False`**

**slot_column: str = `'slots'`** The name of the slot label column to parse from the data source.

**text_column: str = `'text'`** The name of the text column to parse from the data source. We need this to be able to generate tensors which correspond to input text.

**tokenizer:** *Tokenizer.Config* = *Tokenizer.Config*() The tokenizer to use to split input text into tokens. This should be configured in a way which yields tokens consistent with the tokens input to or output by a model, so that the labels generated by this tensorizer will match the indices of the model's tokens.

**allow_unknown: bool = `False`** Whether to allow for unknown labels at test/prediction time

**Subclasses**

- `SlotLabelTensorizerExpansible.Config`

**Default JSON**

```
{
    "is_input": false,
    "slot_column": "slots",
    "text_column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "allow_unknown": false
}
```

### SlotLabelTensorizerExpansible.Config

**Component:** *SlotLabelTensorizerExpansible*

**class** SlotLabelTensorizerExpansible.**Config**

**Bases:** SlotLabelTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `False`**

**slot_column: str = `'slots'`**

**text_column: str = `'text'`**

**tokenizer:** *Tokenizer.Config* = *Tokenizer.Config*()

**allow_unknown: bool = `False`**

**Default JSON**

```
{
    "is_input": false,
    "slot_column": "slots",
    "text_column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "allow_unknown": false
}
```

### SoftLabelTensorizer.Config

**Component:** *SoftLabelTensorizer*

**class** SoftLabelTensorizer.**Config**

**Bases:** LabelTensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = `False`**

**column: str = `'label'`**

**allow_unknown: bool = `False`**

**pad_in_vocab: bool = `False`**

**label_vocab: Optional[list[str]] = `None`**

**label_vocab_file: Optional[str] = `None`**

**add_labels: Optional[list[str]] = `None`**

**probs_column: str = `'target_probs'`**

**logits_column: str = `'target_logits'`**

**labels_column: str = `'target_labels'`**

**Default JSON**

```
{
    "is_input": false,
    "column": "label",
    "allow_unknown": false,
    "pad_in_vocab": false,
    "label_vocab": null,
    "label_vocab_file": null,
    "add_labels": null,
    "probs_column": "target_probs",
    "logits_column": "target_logits",
    "labels_column": "target_labels"
}
```

### String2DListTensorizer.Config

**Component:** *`String2DListTensorizer`*

**class** `String2DListTensorizer.`**`Config`**

**Bases:** `Tensorizer.Config`

**All Attributes (including base classes)**

**is_input: bool = `True`**

**column: str = `'text'`** The name of the text column to parse from the data source.

**vocab: *VocabConfig* = *VocabConfig*()**

**vocab_file_delimiter: str = `' '`**

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "vocab": {
        "build_from_data": true,
        "size_from_data": 0,
        "min_counts": 0,
        "vocab_files": []
```

(continues on next page)

(continued from previous page)

```
    },
    "vocab_file_delimiter": " "
}
```

### Tensorizer.Config

**Component:** *Tensorizer*

**class** Tensorizer.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**Subclasses**

- BERTTensorizer.Config
- BERTTensorizerBase.Config
- BERTContextTensorizerForDenseRetrieval.Config
- PositiveLabelTensorizerForDenseRetrieval.Config
- RoBERTaContextTensorizerForDenseRetrieval.Config
- RoBERTaTensorizer.Config
- RoBERTaTokenLevelTensorizer.Config
- SquadForBERTTensorizer.Config
- SquadForBERTTensorizerForKD.Config
- SquadForRoBERTaTensorizer.Config
- SquadForRoBERTaTensorizerForKD.Config
- SquadTensorizer.Config
- SquadTensorizerForKD.Config
- AnnotationNumberizer.Config
- ByteTensorizer.Config
- ByteTokenTensorizer.Config
- CharacterTokenTensorizer.Config
- CharacterVocabTokenTensorizer.Config
- Float1DListTensorizer.Config
- FloatListSeqTensorizer.Config
- FloatListTensorizer.Config
- FloatTensorizer.Config
- GazetteerTensorizer.Config
- Integer1DListTensorizer.Config
- LabelListRankTensorizer.Config

- LabelListTensorizer.Config
- LabelTensorizer.Config
- MetricTensorizer.Config
- NtokensTensorizer.Config
- NumericLabelTensorizer.Config
- SeqTokenTensorizer.Config
- SlotLabelTensorizer.Config
- SlotLabelTensorizerExpansible.Config
- SoftLabelTensorizer.Config
- String2DListTensorizer.Config
- TokenTensorizer.Config
- UidTensorizer.Config

**Default JSON**

```
{
    "is_input": true
}
```

### TokenTensorizer.Config

**Component:** *TokenTensorizer*

**class** TokenTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str = 'text'** The name of the text column to parse from the data source.

**tokenizer:** *Tokenizer.Config* = *Tokenizer.Config*() The tokenizer to use to split input text into tokens.

**add_bos_token: bool = False**

**add_eos_token: bool = False**

**use_eos_token_for_bos: bool = False**

**max_seq_len: Optional[int] = None**

**vocab:** *VocabConfig* = *VocabConfig*()

**vocab_file_delimiter: str = ' '**

**Subclasses**

- SquadTensorizer.Config
- SquadTensorizerForKD.Config
- CharacterTokenTensorizer.Config

**Default JSON**

```
{
    "is_input": true,
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true,
            "use_byte_offsets": false
        }
    },
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false,
    "max_seq_len": null,
    "vocab": {
        "build_from_data": true,
        "size_from_data": 0,
        "min_counts": 0,
        "vocab_files": []
    },
    "vocab_file_delimiter": " "
}
```

### UidTensorizer.Config

**Component:** *UidTensorizer*

**class** UidTensorizer.**Config**

**Bases:** Tensorizer.Config

**All Attributes (including base classes)**

**is_input: bool = True**

**column: str = 'uid'**

**allow_unknown: bool = True**

**Default JSON**

```
{
    "is_input": true,
    "column": "uid",
    "allow_unknown": true
}
```

### VocabConfig

**Component:** *Component*

**class** pytext.data.tensorizers.**VocabConfig**

**Bases:** Component.Config

**All Attributes (including base classes)**

**build_from_data: bool = True** Whether to add tokens from training data to vocab.

**size_from_data: int = 0** Add *size_from_data* most frequent tokens in training data to vocab (if this is 0, add all tokens from training data).

**min_counts: int = 0** Add *min_counts* filter out tokens in training data that with count smaller than min_counts.

**vocab_files: list[*VocabFileConfig*] = []**

**Default JSON**

```
{
    "build_from_data": true,
    "size_from_data": 0,
    "min_counts": 0,
    "vocab_files": []
}
```

### VocabFileConfig

**Component:** *Component*

**class** pytext.data.tensorizers.**VocabFileConfig**

**Bases:** Component.Config

**All Attributes (including base classes)**

**filepath: str = ''** File containing tokens to add to vocab (first whitespace-separated entry per line)

**skip_header_line: bool = False** Whether to skip the first line of the file (e.g. if it is a header line)

**lowercase_tokens: bool = False** Whether to lowercase each of the tokens in the file

**size_limit: int = 0** The max number of tokens to add to vocab

**Default JSON**

```
{
    "filepath": "",
    "skip_header_line": false,
    "lowercase_tokens": false,
    "size_limit": 0
}
```

### tokenizers

### tokenizer

### BERTInitialTokenizer.Config

**Component:** *BERTInitialTokenizer*

**class** BERTInitialTokenizer.**Config**

**Bases:** Tokenizer.Config

Config for this class.

**All Attributes (including base classes)**

**split_regex: str = '\\s+'**

**lowercase: bool = `True`**

**use_byte_offsets: bool = `False`**

**Default JSON**

```
{
    "split_regex": "\\s+",
    "lowercase": true,
    "use_byte_offsets": false
}
```

### DoNothingTokenizer.Config

**Component:** *DoNothingTokenizer*

**class** DoNothingTokenizer.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**do_nothing: str = `''`**

**Default JSON**

```
{
    "do_nothing": ""
}
```

### GPT2BPETokenizer.Config

**Component:** *GPT2BPETokenizer*

**class** GPT2BPETokenizer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**bpe_encoder_path: str = `'manifold://pytext_training/tree/static/vocabs/bpe/gpt2/encoder.json`**

**bpe_vocab_path: str = `'manifold://pytext_training/tree/static/vocabs/bpe/gpt2/vocab.bpe'`**

**lowercase: bool = `False`**

**Default JSON**

```
{
    "bpe_encoder_path": "manifold://pytext_training/tree/static/vocabs/bpe/gpt2/
↪encoder.json",
    "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/bpe/gpt2/vocab.
↪bpe",
    "lowercase": false
}
```

### SentencePieceTokenizer.Config

**Component:** *SentencePieceTokenizer*

**class** SentencePieceTokenizer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**sp_model_path: str = ''**

**max_input_text_length: Optional[int] = None**

**Default JSON**

```
{
    "sp_model_path": "",
    "max_input_text_length": null
}
```

### Tokenizer.Config

**Component:** *Tokenizer*

**class** Tokenizer.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**split_regex: str = '\\s+'** A regular expression for the tokenizer to split on. Tokens are the segments between the regular expression matches. The start index is inclusive of the unmatched region, and the end index is exclusive (matching the first character of the matched split region).

**lowercase: bool = True** Whether token values should be lowercased or not.

**use_byte_offsets: bool = False** Whether to use utf8 byte offsets

**Subclasses**

- BERTInitialTokenizer.Config

**Default JSON**

```
{
    "split_regex": "\\s+",
    "lowercase": true,
    "use_byte_offsets": false
}
```

### WordPieceTokenizer.Config

**Component:** *WordPieceTokenizer*

**class** WordPieceTokenizer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**basic_tokenizer:** *BERTInitialTokenizer.Config* = *BERTInitialTokenizer.Config*()

**wordpiece_vocab_path: str = 'manifold://nlp_technologies/tree/huggingface-models/bert-base-un**

**Default JSON**

```
{
    "basic_tokenizer": {
        "split_regex": "\\s+",
        "lowercase": true,
        "use_byte_offsets": false
    },
    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt"
}
```

### 1.21.3 exporters

#### custom_exporters

#### DenseFeatureExporter.Config

**Component:** *DenseFeatureExporter*

**class** DenseFeatureExporter.**Config**

**Bases:** ModelExporter.Config

**All Attributes (including base classes)**

**export_logits: bool = False**

**export_raw_to_metrics: bool = False**

**Default JSON**

```
{
    "export_logits": false,
    "export_raw_to_metrics": false
}
```

#### InitPredictNetExporter.Config

**Component:** *InitPredictNetExporter*

**class** InitPredictNetExporter.**Config**

**Bases:** ModelExporter.Config

**All Attributes (including base classes)**

**export_logits: bool = False**

**export_raw_to_metrics: bool = False**

**Default JSON**

```
{
    "export_logits": false,
    "export_raw_to_metrics": false
}
```

### exporter

#### ModelExporter.Config

**Component:** *ModelExporter*

**class** ModelExporter.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**export_logits: bool = `False`**

**export_raw_to_metrics: bool = `False`**

**Subclasses**

- `DenseFeatureExporter.Config`
- `InitPredictNetExporter.Config`

**Default JSON**

```
{
    "export_logits": false,
    "export_raw_to_metrics": false
}
```

## 1.21.4 loss

### loss

#### AUCPRHingeLoss.Config

**Component:** *AUCPRHingeLoss*

**class** AUCPRHingeLoss.**Config**

**Bases:** *ConfigBase*

**precision_range_lower**

the lower range of precision values over which to compute AUC. Must be nonnegative, *leq precision_range_upper*, and *leq 1.0*.

**Type** float

**precision_range_upper**

the upper range of precision values over which to compute AUC. Must be nonnegative, *geq precision_range_lower*, and *leq 1.0*.

**Type** float

**num_classes**

number of classes(aka labels)

**Type** int

**num_anchors**

The number of grid points used to approximate the Riemann sum.

**Type** int

**All Attributes (including base classes)**

**precision_range_lower: float = `0.0`**

**precision_range_upper: float = `1.0`**

**num_classes: int = `1`**

**num_anchors: int = `20`**

**Default JSON**

```
{
    "precision_range_lower": 0.0,
    "precision_range_upper": 1.0,
    "num_classes": 1,
    "num_anchors": 20
}
```

### BinaryCrossEntropyLoss.Config

**Component:** *BinaryCrossEntropyLoss*

**class** BinaryCrossEntropyLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**reweight_negative: bool = `True`**

**reduce: bool = `True`**

**Default JSON**

```
{
    "reweight_negative": true,
    "reduce": true
}
```

### BinaryCrossEntropyWithLogitsLoss.Config

**Component:** *BinaryCrossEntropyWithLogitsLoss*

**class** BinaryCrossEntropyWithLogitsLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**reduce: bool = `True`**

**Default JSON**

```
{
    "reduce": true
}
```

### CosineEmbeddingLoss.Config

**Component:** *CosineEmbeddingLoss*

**class** CosineEmbeddingLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**margin: float = 0.0**

**Default JSON**

```
{
    "margin": 0.0
}
```

### CrossEntropyLoss.Config

**Component:** *CrossEntropyLoss*

**class** CrossEntropyLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### HingeLoss.Config

**Component:** *HingeLoss*

**class** HingeLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**margin: float = 1.0**

**Default JSON**

```
{
    "margin": 1.0
}
```

### KLDivergenceBCELoss.Config

**Component:** *KLDivergenceBCELoss*

**class** KLDivergenceBCELoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**temperature: float = 1.0**

**hard_weight: float = 0.0**

**Default JSON**

```
{
    "temperature": 1.0,
    "hard_weight": 0.0
}
```

### KLDivergenceCELoss.Config

**Component:** *KLDivergenceCELoss*

**class** KLDivergenceCELoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**temperature: float = 1.0**

**hard_weight: float = 0.0**

**Default JSON**

```
{
    "temperature": 1.0,
    "hard_weight": 0.0
}
```

### LabelSmoothedCrossEntropyLoss.Config

**Component:** *LabelSmoothedCrossEntropyLoss*

**class** LabelSmoothedCrossEntropyLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**beta: float = 0.1**

**source: SourceType = <SourceType.LOGITS: 'logits'>**

**use_entropy: bool = False**

**Default JSON**

```
{
    "beta": 0.1,
    "source": "logits",
    "use_entropy": false
}
```

### Loss.Config

**Component:** *Loss*

**class** Loss.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**Subclasses**

- `EntropyRegularizer.Config`
- `Regularizer.Config`
- `UniformRegularizer.Config`
- `StructuredLoss.Config`

**Default JSON**

```
{}
```

### MAELoss.Config

**Component:** *MAELoss*

**class** MAELoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### MSELoss.Config

**Component:** *MSELoss*

**class** MSELoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### MultiLabelSoftMarginLoss.Config

**Component:** *MultiLabelSoftMarginLoss*

**class** MultiLabelSoftMarginLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### NLLLoss.Config

**Component:** *NLLLoss*

**class** NLLLoss.**Config**
**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### PairwiseRankingLoss.Config

**Component:** *PairwiseRankingLoss*

**class** PairwiseRankingLoss.**Config**
**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**margin: float = 1.0**

**Default JSON**

```
{
    "margin": 1.0
}
```

### regularized_loss

### LabelSmoothingLoss.Config

**Component:** *LabelSmoothingLoss*

**class** LabelSmoothingLoss.**Config**
**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**beta: float = 0.1**

**label_loss: Union[*NLLLoss.Config*, *StructuredMarginLoss.Config*, *HingeLoss.Config*] = *NLLLoss.Config*()**

**smoothing_loss: Union[*UniformRegularizer.Config*, *EntropyRegularizer.Config*, *AdaptiveRegularizer.Config*] = *UniformRegu***

**Subclasses**

- SamplewiseLabelSmoothingLoss.Config

**Default JSON**

```
{
    "beta": 0.1,
    "label_loss": {
```

(continues on next page)

(continued from previous page)

```
        "NLLLoss": {}
    },
    "smoothing_loss": {
        "UniformRegularizer": {}
    }
}
```

### NARSamplewiseSequenceLoss.Config

**Component:** *NARSamplewiseSequenceLoss*

**class** NARSamplewiseSequenceLoss.**Config**

**Bases:** NARSequenceLoss.Config

**All Attributes (including base classes)**

**beta: float = 0.1**

**assert_valid_targets: bool = True**

**label_type: SourceType = <SourceType.LOG_PROBS: 'log_probs'>**

**length_type: SourceType = <SourceType.LOG_PROBS: 'log_probs'>**

**label_loss:** *SamplewiseLabelSmoothingLoss.Config* **=** *SamplewiseLabelSmoothingLoss.Config*()

**length_loss:** *SamplewiseLabelSmoothingLoss.Config* **=** *SamplewiseLabelSmoothingLoss.Config*()

**Default JSON**

```
{
    "beta": 0.1,
    "assert_valid_targets": true,
    "label_type": "log_probs",
    "length_type": "log_probs",
    "label_loss": {
        "beta": 0.1,
        "label_loss": {
            "NLLLoss": {}
        },
        "smoothing_loss": {
            "UniformRegularizer": {}
        }
    },
    "length_loss": {
        "beta": 0.1,
        "label_loss": {
            "NLLLoss": {}
        },
        "smoothing_loss": {
            "UniformRegularizer": {}
        }
    }
}
```

### NARSequenceLoss.Config

**Component:** *NARSequenceLoss*

**class** NARSequenceLoss.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**beta: float = 0.1**

**assert_valid_targets: bool = True**

**label_type: SourceType = <SourceType.LOG_PROBS: 'log_probs'>**

**length_type: SourceType = <SourceType.LOG_PROBS: 'log_probs'>**

**label_loss:** *LabelSmoothingLoss.Config* = *LabelSmoothingLoss.Config*()

**length_loss:** *LabelSmoothingLoss.Config* = *LabelSmoothingLoss.Config*()

**Subclasses**

- NARSamplewiseSequenceLoss.Config

**Default JSON**

```
{
    "beta": 0.1,
    "assert_valid_targets": true,
    "label_type": "log_probs",
    "length_type": "log_probs",
    "label_loss": {
        "beta": 0.1,
        "label_loss": {
            "NLLLoss": {}
        },
        "smoothing_loss": {
            "UniformRegularizer": {}
        }
    },
    "length_loss": {
        "beta": 0.1,
        "label_loss": {
            "NLLLoss": {}
        },
        "smoothing_loss": {
            "UniformRegularizer": {}
        }
    }
}
```

### SamplewiseLabelSmoothingLoss.Config

**Component:** *SamplewiseLabelSmoothingLoss*

**class** SamplewiseLabelSmoothingLoss.**Config**

**Bases:** LabelSmoothingLoss.Config

**All Attributes (including base classes)**

**beta: float = 0.1**

**label_loss: Union[*NLLLoss.Config*, *StructuredMarginLoss.Config*, *HingeLoss.Config*] = *NLLLoss.Config*()**

**smoothing_loss: Union[*UniformRegularizer.Config*, *EntropyRegularizer.Config*, *AdaptiveRegularizer.Config*] = *UniformRegu***

**Default JSON**

```
{
    "beta": 0.1,
    "label_loss": {
        "NLLLoss": {}
    },
    "smoothing_loss": {
        "UniformRegularizer": {}
    }
}
```

### regularizer

#### AdaptiveRegularizer.Config

**Component:** *AdaptiveRegularizer*

**class** AdaptiveRegularizer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**eta: float = 0.1**

**label_embedding_dim: int = 20**

**label_embedding_dropout: float = 0.4**

**Default JSON**

```
{
    "eta": 0.1,
    "label_embedding_dim": 20,
    "label_embedding_dropout": 0.4
}
```

#### EntropyRegularizer.Config

**Component:** *EntropyRegularizer*

**class** EntropyRegularizer.**Config**

**Bases:** Regularizer.Config

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### Regularizer.Config

**Component:** *Regularizer*

**class** Regularizer.**Config**
**Bases:** Loss.Config

**All Attributes (including base classes)**

**Subclasses**

- EntropyRegularizer.Config
- UniformRegularizer.Config

**Default JSON**

```
{}
```

### UniformRegularizer.Config

**Component:** *UniformRegularizer*

**class** UniformRegularizer.**Config**
**Bases:** Regularizer.Config

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

## structured_loss

### StructuredLoss.Config

**Component:** *StructuredLoss*

**class** StructuredLoss.**Config**
**Bases:** Loss.Config

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### StructuredMarginLoss.Config

**Component:** *StructuredMarginLoss*

**class** StructuredMarginLoss.**Config**
**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**cost_scale: float** = 1.0

**cost_fn: CostFunctionType = <CostFunctionType.HAMMING: 'hamming'>**

**label_loss: Union[*NLLLoss.Config*, *HingeLoss.Config*] = *NLLLoss.Config*()**

**Default JSON**

```
{
    "cost_scale": 1.0,
    "cost_fn": "hamming",
    "label_loss": {
        "NLLLoss": {}
    }
}
```

### 1.21.5 metric_reporters

#### calibration_metric_reporter

#### CalibrationMetricReporter.Config

**Component:** *CalibrationMetricReporter*

**class** CalibrationMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

#### classification_metric_reporter

#### ClassificationMetricReporter.Config

**Component:** *ClassificationMetricReporter*

**class** ClassificationMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = `[]`**

**log_gradient: bool = `False`**

**model_select_metric: ComparableClassificationMetric = `<ComparableClassificationMetric.ACCURACY: 'acc`**

**target_label: Optional[str] = `None`**

**text_column_names: list[str] = `['text']`** These column names correspond to raw input data columns. Text in these columns (usually just 1 column) will be concatenated and output in the IntentModelChannel as an evaluation tsv.

**additional_column_names: list[str] = `[]`** These column names correspond to raw input data columns, that will be read by data_source into context, and included in the run_model output file along with other saving results.

**recall_at_precision_thresholds: list[float] = `[0.2, 0.4, 0.6, 0.8, 0.9]`**

**is_memory_efficient: bool = `False`**

**Subclasses**

- `MultiLabelClassificationMetricReporter.Config`

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "model_select_metric": "accuracy",
    "target_label": null,
    "text_column_names": [
        "text"
    ],
    "additional_column_names": [],
    "recall_at_precision_thresholds": [
        0.2,
        0.4,
        0.6,
        0.8,
        0.9
    ],
    "is_memory_efficient": false
}
```

### MultiLabelClassificationMetricReporter.Config

**Component:** *`MultiLabelClassificationMetricReporter`*

**class** `MultiLabelClassificationMetricReporter.`**`Config`**

**Bases:** `ClassificationMetricReporter.Config`

**All Attributes (including base classes)**

**output_path: str = `'/tmp/test_out.txt'`**

**pep_format: bool = `False`**

**student_column_names: list[str] = `[]`**

**log_gradient: bool = `False`**

**model_select_metric: ComparableClassificationMetric = `<ComparableClassificationMetric.ACCURACY: 'acc`**

**target_label: Optional[str] = `None`**

**text_column_names: list[str] = `['text']`**

**additional_column_names: list[str] = `[]`**

**recall_at_precision_thresholds: list[float] = `[0.2, 0.4, 0.6, 0.8, 0.9]`**

**is_memory_efficient: bool = `False`**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "model_select_metric": "accuracy",
    "target_label": null,
    "text_column_names": [
        "text"
    ],
    "additional_column_names": [],
    "recall_at_precision_thresholds": [
        0.2,
        0.4,
        0.6,
        0.8,
        0.9
    ],
    "is_memory_efficient": false
}
```

### compositional_metric_reporter

### CompositionalMetricReporter.Config

**Component:** *`CompositionalMetricReporter`*

**class** `CompositionalMetricReporter.`**`Config`**

**Bases:** `MetricReporter.Config`

**All Attributes (including base classes)**

**output_path: str = `'/tmp/test_out.txt'`**

**pep_format: bool = `False`**

**student_column_names: list[str] = `[]`**

**log_gradient: bool = `False`**

**text_column_name: str = `'tokenized_text'`**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "text_column_name": "tokenized_text"
}
```

### dense_retrieval_metric_reporter

#### DenseRetrievalMetricReporter.Config

**Component:** *DenseRetrievalMetricReporter*

**class** DenseRetrievalMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**text_column_names: list[str] = ['question', 'positive_ctx', 'negative_ctxs']**

**model_select_metric: DenseRetrievalMetricNames = <DenseRetrievalMetricNames.ACCURACY: 'accuracy'>**

**task_batch_size: int = 0**

**num_negative_ctxs: int = 0**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "text_column_names": [
        "question",
        "positive_ctx",
        "negative_ctxs"
    ],
    "model_select_metric": "accuracy",
    "task_batch_size": 0,
    "num_negative_ctxs": 0
}
```

### disjoint_multitask_metric_reporter

#### DisjointMultitaskMetricReporter.Config

**Component:** *DisjointMultitaskMetricReporter*

**class** DisjointMultitaskMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**use_subtask_select_metric: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "use_subtask_select_metric": false
}
```

### intent_slot_detection_metric_reporter

### IntentSlotMetricReporter.Config

**Component:** *IntentSlotMetricReporter*

**class** IntentSlotMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

### language_model_metric_reporter

### LanguageModelMetricReporter.Config

**Component:** *LanguageModelMetricReporter*

**class** LanguageModelMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**aggregate_metrics: bool = True**

**perplexity_type: PerplexityType = <PerplexityType.MEDIAN: 'median'>**

**Subclasses**

- MaskedLMMetricReporter.Config

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "aggregate_metrics": true,
    "perplexity_type": "median"
}
```

### MaskedLMMetricReporter.Config

**Component:** *MaskedLMMetricReporter*

**class** MaskedLMMetricReporter.**Config**

**Bases:** LanguageModelMetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**aggregate_metrics: bool = True**

**perplexity_type: PerplexityType = <PerplexityType.MEDIAN: 'median'>**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "aggregate_metrics": true,
    "perplexity_type": "median"
}
```

### metric_reporter

### MetricReporter.Config

**Component:** *MetricReporter*

**class** MetricReporter.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []** Useful for KD training, column names that used by student but not teacher.

**log_gradient: bool = False**

**Subclasses**

- CalibrationMetricReporter.Config
- ClassificationMetricReporter.Config
- MultiLabelClassificationMetricReporter.Config
- CompositionalMetricReporter.Config
- DenseRetrievalMetricReporter.Config
- DisjointMultitaskMetricReporter.Config
- IntentSlotMetricReporter.Config
- LanguageModelMetricReporter.Config
- MaskedLMMetricReporter.Config
- PureLossMetricReporter.Config
- PairwiseRankingMetricReporter.Config
- RegressionMetricReporter.Config
- Seq2SeqCompositionalMetricReporter.Config
- Seq2SeqMetricReporter.Config
- SquadMetricReporter.Config
- MultiLabelSequenceTaggingMetricReporter.Config
- NERMetricReporter.Config
- SequenceTaggingMetricReporter.Config
- WordTaggingMetricReporter.Config

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
```

(continues on next page)

(continued from previous page)

```
    "log_gradient": false
}
```

### PureLossMetricReporter.Config

**Component:** *PureLossMetricReporter*

**class** PureLossMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

### pairwise_ranking_metric_reporter

### PairwiseRankingMetricReporter.Config

**Component:** *PairwiseRankingMetricReporter*

**class** PairwiseRankingMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

### regression_metric_reporter

#### RegressionMetricReporter.Config

**Component:** *RegressionMetricReporter*

**class** RegressionMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

### seq2seq_compositional

#### Seq2SeqCompositionalMetricReporter.Config

**Component:** *Seq2SeqCompositionalMetricReporter*

**class** Seq2SeqCompositionalMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**accept_flat_intents_slots: Optional[bool] = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false,
    "accept_flat_intents_slots": false
}
```

### seq2seq_metric_reporter

#### Seq2SeqMetricReporter.Config

**Component:** *Seq2SeqMetricReporter*

**class** Seq2SeqMetricReporter.**Config**
**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

### squad_metric_reporter

#### SquadMetricReporter.Config

**Component:** *SquadMetricReporter*

**class** SquadMetricReporter.**Config**
**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**n_best_size: int = 5**

**max_answer_length: int = 16**

**ignore_impossible: bool = True**

**false_label: str = 'False'**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
```

(continues on next page)

(continued from previous page)

```
    "log_gradient": false,
    "n_best_size": 5,
    "max_answer_length": 16,
    "ignore_impossible": true,
    "false_label": "False"
}
```

### word_tagging_metric_reporter

#### MultiLabelSequenceTaggingMetricReporter.Config

**Component:** *MultiLabelSequenceTaggingMetricReporter*

**class** MultiLabelSequenceTaggingMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

#### NERMetricReporter.Config

**Component:** *NERMetricReporter*

**class** NERMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
```

(continues on next page)

(continued from previous page)

```
    "log_gradient": false
}
```

### SequenceTaggingMetricReporter.Config

**Component:** *SequenceTaggingMetricReporter*

**class** SequenceTaggingMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

### WordTaggingMetricReporter.Config

**Component:** *WordTaggingMetricReporter*

**class** WordTaggingMetricReporter.**Config**

**Bases:** MetricReporter.Config

**All Attributes (including base classes)**

**output_path: str = '/tmp/test_out.txt'**

**pep_format: bool = False**

**student_column_names: list[str] = []**

**log_gradient: bool = False**

**Default JSON**

```
{
    "output_path": "/tmp/test_out.txt",
    "pep_format": false,
    "student_column_names": [],
    "log_gradient": false
}
```

### 1.21.6 models

#### bert_classification_models

#### BertModelInput

**class** pytext.models.bert_classification_models.**BertModelInput**

**Bases:** `EncoderModelInput`

**All Attributes (including base classes)**

**tokens:** *BERTTensorizer.Config* = *BERTTensorizer.Config*(**max_seq_len=`128`**)

**dense:** Optional[*FloatListTensorizer.Config*] = **`None`**

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**num_tokens:** *NtokensTensorizer.Config* = *NtokensTensorizer.Config*(**names=`['tokens']`, indexes=`[2]`**)

**Default JSON**

```
{
    "tokens": {
        "BERTTensorizer": {
            "is_input": true,
            "columns": [
                "text"
            ],
            "tokenizer": {
                "WordPieceTokenizer": {
                    "basic_tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    },
                    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                }
            },
            "base_tokenizer": null,
            "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt",
            "max_seq_len": 128
        }
    },
    "dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    },
    "num_tokens": {
```

(continues on next page)

(continued from previous page)

```
        "is_input": false,
        "names": [
            "tokens"
        ],
        "indexes": [
            2
        ]
    }
}
```

### BertPairwiseModel.Config

**Component:** *BertPairwiseModel*

**class** BertPairwiseModel.**Config**

**Bases:** _EncoderPairwiseModel.Config

**All Attributes (including base classes)**

**inputs:** *BertPairwiseModelInput* = *BertPairwiseModelInput*()

**decoder:** Optional[*MLPDecoder.Config*] = *MLPDecoder.Config*()

**output_layer:** Union[*ClassificationOutputLayer.Config*, *PairwiseCosineDistanceOutputLayer.Config*] = *ClassificationOutputL*

**encode_relations:** bool = **True**

**encoder:** *TransformerSentenceEncoderBase.Config* = *HuggingFaceBertSentenceEncoder.Config*()

**shared_encoder:** bool = **True**

**Subclasses**

- BertPairwiseRegressionModel.Config

**Default JSON**

```
{
    "inputs": {
        "tokens1": {
            "BERTTensorizer": {
                "is_input": true,
                "columns": [
                    "text1"
                ],
                "tokenizer": {
                    "WordPieceTokenizer": {
                        "basic_tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        },
                        "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/vocab.txt",
```

(continues on next page)

(continued from previous page)

```
                "max_seq_len": 128
            }
        },
        "tokens2": {
            "BERTTensorizer": {
                "is_input": true,
                "columns": [
                    "text2"
                ],
                "tokenizer": {
                    "WordPieceTokenizer": {
                        "basic_tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        },
                        "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/vocab.txt",
                "max_seq_len": 128
            }
        },
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        },
        "num_tokens": {
            "is_input": false,
            "names": [
                "tokens1",
                "tokens2"
            ],
            "indexes": [
                2,
                2
            ]
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
```

(continues on next page)

(continued from previous page)

```
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "ClassificationOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        }
    },
    "encode_relations": true,
    "encoder": {
        "HuggingFaceBertSentenceEncoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/",
            "load_weights": true
        }
    },
    "shared_encoder": true
}
```

### BertPairwiseModelInput

**class** pytext.models.bert_classification_models.**BertPairwiseModelInput**

**Bases:** EncoderPairwiseModelInput

**All Attributes (including base classes)**

**tokens1:** *BERTTensorizerBase.Config* = *BERTTensorizer.Config*(**columns=['text1'], max_seq_len=128**)

**tokens2:** *BERTTensorizerBase.Config* = *BERTTensorizer.Config*(**columns=['text2'], max_seq_len=128**)

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**num_tokens:** *NtokensTensorizer.Config* = *NtokensTensorizer.Config*(**names=['tokens1', 'tokens2'], indexes=[2,**

### Default JSON

```
{
    "tokens1": {
        "BERTTensorizer": {
            "is_input": true,
            "columns": [
                "text1"
            ],
            "tokenizer": {
                "WordPieceTokenizer": {
                    "basic_tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    },
                    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                }
            },
            "base_tokenizer": null,
            "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt",
            "max_seq_len": 128
        }
    },
    "tokens2": {
        "BERTTensorizer": {
            "is_input": true,
            "columns": [
                "text2"
            ],
            "tokenizer": {
                "WordPieceTokenizer": {
                    "basic_tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    },
                    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                }
            },
            "base_tokenizer": null,
            "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt",
            "max_seq_len": 128
        }
    },
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
```

(continues on next page)

(continued from previous page)

```
        }
    },
    "num_tokens": {
        "is_input": false,
        "names": [
            "tokens1",
            "tokens2"
        ],
        "indexes": [
            2,
            2
        ]
    }
}
```

### EncoderModelInput

**class** pytext.models.bert_classification_models.**EncoderModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *Tensorizer.Config* = *Tensorizer.Config*()

**dense:** Optional[*FloatListTensorizer.Config*] = `None`

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**num_tokens:** *NtokensTensorizer.Config* = *NtokensTensorizer.Config*(names=`['tokens']`, indexes=`[2]`)

**Default JSON**

```
{
    "tokens": {
        "Tensorizer": {
            "is_input": true
        }
    },
    "dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    },
    "num_tokens": {
        "is_input": false,
        "names": [
            "tokens"
        ],
        "indexes": [
```

(continues on next page)

(continued from previous page)

```
            2
        ]
    }
}
```

### EncoderPairwiseModelInput

**class** pytext.models.bert_classification_models.**EncoderPairwiseModelInput**

**Bases:** *ModelInputBase*

**All Attributes (including base classes)**

**tokens1:** *Tensorizer.Config* = *Tensorizer.Config*()

**tokens2:** *Tensorizer.Config* = *Tensorizer.Config*()

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**num_tokens:** *NtokensTensorizer.Config* = *NtokensTensorizer.Config*(**names=['tokens1', 'tokens2'], indexes=[2,**

**Default JSON**

```
{
    "tokens1": {
        "Tensorizer": {
            "is_input": true
        }
    },
    "tokens2": {
        "Tensorizer": {
            "is_input": true
        }
    },
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    },
    "num_tokens": {
        "is_input": false,
        "names": [
            "tokens1",
            "tokens2"
        ],
        "indexes": [
            2,
            2
        ]
    }
}
```

### NewBertModel.Config

**Component:** *NewBertModel*

**class** NewBertModel.**Config**

**Bases:** _EncoderBaseModel.Config

**All Attributes (including base classes)**

**inputs:** *BertModelInput* = *BertModelInput*()

**encoder:** *TransformerSentenceEncoderBase.Config* = *HuggingFaceBertSentenceEncoder.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**Subclasses**

- NewBertRegressionModel.Config
- BertSquadQAModel.Config
- RoBERTa.Config
- RoBERTaR3F.Config
- RoBERTaRegression.Config
- SELFIE.Config

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "BERTTensorizer": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "WordPieceTokenizer": {
                        "basic_tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        },
                        "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/vocab.txt",
                "max_seq_len": 128
            }
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
```

(continues on next page)

(continued from previous page)

```
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        },
        "num_tokens": {
            "is_input": false,
            "names": [
                "tokens"
            ],
            "indexes": [
                2
            ]
        }
    },
    "encoder": {
        "HuggingFaceBertSentenceEncoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/",
            "load_weights": true
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
```

(continues on next page)

(continued from previous page)

```
        "label_weights": null
    }
}
```

### _EncoderBaseModel.Config

**Component:** `_EncoderBaseModel`

**class** `_EncoderBaseModel.`**`Config`**

**Bases:** `BaseModel.Config`

**All Attributes (including base classes)**

**inputs:** *EncoderModelInput* = *EncoderModelInput*()

**encoder:** *RepresentationBase.Config*

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**Subclasses**

- `NewBertModel.Config`
- `NewBertRegressionModel.Config`
- `BertSquadQAModel.Config`
- `RoBERTa.Config`
- `RoBERTaR3F.Config`
- `RoBERTaRegression.Config`
- `SELFIE.Config`

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### _EncoderPairwiseModel.Config

**Component:** `_EncoderPairwiseModel`

**class** `_EncoderPairwiseModel.`**`Config`**

**Bases:** `BasePairwiseModel.Config`

**All Attributes (including base classes)**

**inputs:** *EncoderPairwiseModelInput* = *EncoderPairwiseModelInput*()

**decoder:** Optional[*MLPDecoder.Config*] = *MLPDecoder.Config*()

**output_layer:** Union[*ClassificationOutputLayer.Config*, *PairwiseCosineDistanceOutputLayer.Config*] = *ClassificationOutput*

**encode_relations:** bool = `True`

**encoder:** *RepresentationBase.Config*

**shared_encoder: bool = `True`**

**Subclasses**

- `BertPairwiseModel.Config`
- `BertPairwiseRegressionModel.Config`

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### bert_regression_model

### BertPairwiseRegressionModel.Config

**Component:** *`BertPairwiseRegressionModel`*

**class** `BertPairwiseRegressionModel.`**`Config`**

**Bases:** `BertPairwiseModel.Config`

**All Attributes (including base classes)**

**inputs: *ModelInput* = *ModelInput*()**

**decoder: Optional[*MLPDecoder.Config*] = `None`**

**output_layer: *PairwiseCosineRegressionOutputLayer.Config* = *PairwiseCosineRegressionOutputLayer.Config*()**

**encode_relations: bool = `False`**

**encoder: *TransformerSentenceEncoderBase.Config* = *HuggingFaceBertSentenceEncoder.Config*()**

**shared_encoder: bool = `True`**

**Default JSON**

```
{
    "inputs": {
        "tokens1": {
            "BERTTensorizer": {
                "is_input": true,
                "columns": [
                    "text1"
                ],
                "tokenizer": {
                    "WordPieceTokenizer": {
                        "basic_tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        },
                        "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/vocab.txt",
```

(continues on next page)

(continued from previous page)

```
                "max_seq_len": 128
            }
        },
        "tokens2": {
            "BERTTensorizer": {
                "is_input": true,
                "columns": [
                    "text2"
                ],
                "tokenizer": {
                    "WordPieceTokenizer": {
                        "basic_tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        },
                        "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/vocab.txt",
                "max_seq_len": 128
            }
        },
        "labels": {
            "is_input": false,
            "column": "label",
            "rescale_range": null
        },
        "num_tokens": {
            "is_input": false,
            "names": [
                "tokens1",
                "tokens2"
            ],
            "indexes": [
                2,
                2
            ]
        }
    },
    "decoder": null,
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "MSELoss": {}
        }
    },
    "encode_relations": false,
    "encoder": {
        "HuggingFaceBertSentenceEncoder": {
            "load_path": null,
```

(continues on next page)

(continued from previous page)

```
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/",
            "load_weights": true
        }
    },
    "shared_encoder": true
}
```

### InputConfig

**class** pytext.models.bert_regression_model.**InputConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**tokens:** *BERTTensorizer.Config* = *BERTTensorizer.Config*(**columns=['text1', 'text2'], max_seq_len=128**)

**labels:** *NumericLabelTensorizer.Config* = *NumericLabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "BERTTensorizer": {
            "is_input": true,
            "columns": [
                "text1",
                "text2"
            ],
            "tokenizer": {
                "WordPieceTokenizer": {
                    "basic_tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    },
                    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                }
            },
            "base_tokenizer": null,
            "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt",
            "max_seq_len": 128
        }
    },
```

(continues on next page)

(continued from previous page)

```
    "labels": {
        "is_input": false,
        "column": "label",
        "rescale_range": null
    }
}
```

### ModelInput

**class** pytext.models.bert_regression_model.**ModelInput**

**Bases:** BertPairwiseModelInput

**All Attributes (including base classes)**

**tokens1:** *BERTTensorizerBase.Config* = *BERTTensorizer.Config*(**columns=['text1'], max_seq_len=128**)

**tokens2:** *BERTTensorizerBase.Config* = *BERTTensorizer.Config*(**columns=['text2'], max_seq_len=128**)

**labels:** *NumericLabelTensorizer.Config* = *NumericLabelTensorizer.Config*()

**num_tokens:** *NtokensTensorizer.Config* = *NtokensTensorizer.Config*(**names=['tokens1', 'tokens2'], indexes=[2,**

**Default JSON**

```
{
    "tokens1": {
        "BERTTensorizer": {
            "is_input": true,
            "columns": [
                "text1"
            ],
            "tokenizer": {
                "WordPieceTokenizer": {
                    "basic_tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    },
                    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                }
            },
            "base_tokenizer": null,
            "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt",
            "max_seq_len": 128
        }
    },
    "tokens2": {
        "BERTTensorizer": {
            "is_input": true,
            "columns": [
                "text2"
            ],
```

(continues on next page)

(continued from previous page)

```
            "tokenizer": {
                "WordPieceTokenizer": {
                    "basic_tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    },
                    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                }
            },
            "base_tokenizer": null,
            "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt",
            "max_seq_len": 128
        }
    },
    "labels": {
        "is_input": false,
        "column": "label",
        "rescale_range": null
    },
    "num_tokens": {
        "is_input": false,
        "names": [
            "tokens1",
            "tokens2"
        ],
        "indexes": [
            2,
            2
        ]
    }
}
```

### NewBertRegressionModel.Config

**Component:** *NewBertRegressionModel*

**class** NewBertRegressionModel.**Config**

**Bases:** NewBertModel.Config

**All Attributes (including base classes)**

**inputs:** *InputConfig* = *InputConfig*()

**encoder:** *TransformerSentenceEncoderBase.Config* = *HuggingFaceBertSentenceEncoder.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *RegressionOutputLayer.Config* = *RegressionOutputLayer.Config*()

**Subclasses**

- RoBERTaRegression.Config

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "BERTTensorizer": {
                "is_input": true,
                "columns": [
                    "text1",
                    "text2"
                ],
                "tokenizer": {
                    "WordPieceTokenizer": {
                        "basic_tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        },
                        "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/vocab.txt",
                "max_seq_len": 128
            }
        },
        "labels": {
            "is_input": false,
            "column": "label",
            "rescale_range": null
        }
    },
    "encoder": {
        "HuggingFaceBertSentenceEncoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/",
            "load_weights": true
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
```

(continues on next page)

(continued from previous page)

```
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {},
        "squash_to_unit_range": false
    }
}
```

### decoders

### decoder_base

### DecoderBase.Config

**Component:** *DecoderBase*

**class** DecoderBase.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Subclasses**

- IntentSlotModelDecoder.Config
- MLPDecoder.Config
- MLPDecoderQueryResponse.Config
- MLPDecoderTwoTower.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### intent_slot_model_decoder

#### IntentSlotModelDecoder.Config

**Component:** *IntentSlotModelDecoder*

**class** IntentSlotModelDecoder.**Config**

**Bases:** DecoderBase.Config

Configuration class for *IntentSlotModelDecoder*.

**use_doc_probs_in_word**

Whether to use intent probabilities for predicting slots.

**Type** bool

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**use_doc_probs_in_word: bool = False**

**doc_decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**word_decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "use_doc_probs_in_word": false,
    "doc_decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "word_decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
```

(continues on next page)

(continued from previous page)

```
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    }
}
```

### mlp_decoder

### MLPDecoder.Config

**Component:** *MLPDecoder*

**class** MLPDecoder.**Config**

**Bases:** DecoderBase.Config

Configuration class for *MLPDecoder*.

**hidden_dims**

Dimensions of the outputs of hidden layers..

**Type** List[int]

**temperature**

Scales logits by this value (before the softmax

**Type** float

**operation) during test-time only. Temperature scaling has no effect on the top prediction but changes the shape of the posterior distribution, which can be useful for a range of tasks**

**Type** e.g., model calibration

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**hidden_dims: list[int] = []**

**out_dim: Optional[int] = None**

**layer_norm: bool = False**

**dropout: float = 0.0**

**bias: bool = True**

**activation: Activation = <Activation.RELU: 'relu'>**

**temperature: float = 1.0**

**spectral_normalization: bool = False**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "hidden_dims": [],
    "out_dim": null,
    "layer_norm": false,
    "dropout": 0.0,
    "bias": true,
    "activation": "relu",
    "temperature": 1.0,
    "spectral_normalization": false
}
```

### mlp_decoder_query_response

#### MLPDecoderQueryResponse.Config

**Component:** *MLPDecoderQueryResponse*

**class** MLPDecoderQueryResponse.**Config**

**Bases:** DecoderBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**hidden_dims: list[int] = []**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "hidden_dims": []
}
```

### mlp_decoder_two_tower

#### MLPDecoderTwoTower.Config

**Component:** *MLPDecoderTwoTower*

**class** MLPDecoderTwoTower.**Config**

**Bases:** DecoderBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**right_hidden_dims: list[int] = `[]`**

**left_hidden_dims: list[int] = `[]`**

**hidden_dims: list[int] = `[]`**

**layer_norm: bool = `False`**

**dropout: float = `0.0`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "right_hidden_dims": [],
    "left_hidden_dims": [],
    "hidden_dims": [],
    "layer_norm": false,
    "dropout": 0.0
}
```

### disjoint_multitask_model

### DisjointMultitaskModel.Config

**Component:** *`DisjointMultitaskModel`*

**class** `DisjointMultitaskModel.`**`Config`**

**Bases:** `Model.Config`

**All Attributes (including base classes)**

**inputs:** ***ModelInput*** **=** ***ModelInput*****()**

**Subclasses**

- `NewDisjointMultitaskModel.Config`

**Default JSON**

```
{
    "inputs": {}
}
```

### NewDisjointMultitaskModel.Config

**Component:** *`NewDisjointMultitaskModel`*

**class** NewDisjointMultitaskModel.**Config**

**Bases:** DisjointMultitaskModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**Default JSON**

```
{
    "inputs": {}
}
```

## doc_model

### ByteModelInput

**class** pytext.models.doc_model.**ByteModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**dense:** Optional[*FloatListTensorizer.Config*] = **None**

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**token_bytes:** *ByteTokenTensorizer.Config* = *ByteTokenTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
```

(continues on next page)

(continued from previous page)

```
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    },
    "token_bytes": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "max_seq_len": null,
        "max_byte_len": 15,
        "offset_for_non_padding": 0,
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false
    }
}
```

### ByteTokensDocumentModel.Config

**Component:** *ByteTokensDocumentModel*

**class** ByteTokensDocumentModel.**Config**

**Bases:** DocModel.Config

**All Attributes (including base classes)**

**inputs:** *ByteModelInput* = *ByteModelInput*()

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** Union[*PureDocAttention.Config*, *BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*, *DeepCNNRe*

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**byte_embedding:** *CharacterEmbedding.Config* = *CharacterEmbedding.Config*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
```

(continues on next page)

(continued from previous page)

```
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        },
        "token_bytes": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "max_seq_len": null,
            "max_byte_len": 15,
            "offset_for_non_padding": 0,
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
```

(continues on next page)

(continued from previous page)

```
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
```

(continues on next page)

(continued from previous page)

```
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    },
    "byte_embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "sparse": false,
        "cnn": {
            "kernel_num": 100,
            "kernel_sizes": [
                3,
                4
            ],
            "weight_norm": false,
            "dilated": false,
            "causal": false
        },
        "highway_layers": 0,
        "projection_dim": null,
        "export_input_names": [
            "char_vals"
        ],
        "vocab_from_train_data": true,
        "max_word_length": 20,
        "min_freq": 1
    }
}
```

### DocModel.Config

**Component:** *DocModel*

**class** DocModel.**Config**

**Bases:** Model.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** Union[*PureDocAttention.Config*, *BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*, *DeepCNNRe*

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

### Subclasses

- ByteTokensDocumentModel.Config
- DocRegressionModel.Config
- PersonalizedDocModel.Config
- SeqNNModel.Config

### Default JSON

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
```

(continues on next page)

(continued from previous page)

```
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
```

(continues on next page)

(continued from previous page)

```
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    }
}
```

### DocRegressionModel.Config

**Component:** *DocRegressionModel*

**class** DocRegressionModel.**Config**

**Bases:** DocModel.Config

**All Attributes (including base classes)**

**inputs:** *RegressionModelInput* = *RegressionModelInput*()

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** Union[*PureDocAttention.Config*, *BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*, *DeepCNNRe*

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *RegressionOutputLayer.Config* = *RegressionOutputLayer.Config*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
```

(continues on next page)

(continued from previous page)

```
        },
        "dense": null,
        "labels": {
            "is_input": false,
            "column": "label",
            "rescale_range": null
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
```

(continues on next page)

(continued from previous page)

```
                }
            },
            "mlp_decoder": null
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {},
        "squash_to_unit_range": false
    }
}
```

### ModelInput

**class** pytext.models.doc_model.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**dense:** Optional[*FloatListTensorizer.Config*] = **None**

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
```

(continues on next page)

(continued from previous page)

```
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    }
}
```

### PersonalizedDocModel.Config

**Component:** *PersonalizedDocModel*

**class** PersonalizedDocModel.**Config**

**Bases:** DocModel.Config

**All Attributes (including base classes)**

**inputs:** *PersonalizedModelInput* = *PersonalizedModelInput*()

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation: Union[***PureDocAttention.Config***,** *BiLSTMDocAttention.Config***,** *DocNNRepresentation.Config***,** *DeepCNNRe*

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**user_embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
```

(continues on next page)

(continued from previous page)

```
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        },
        "uid": {
            "is_input": true,
            "column": "uid",
            "allow_unknown": true
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
```

(continues on next page)

(continued from previous page)

```
        "delimiter": " "
    },
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    },
    "user_embedding": {
        "load_path": null,
        "save_path": null,
```

(continues on next page)

(continued from previous page)

```
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    }
}
```

### PersonalizedModelInput

**class** pytext.models.doc_model.**PersonalizedModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**dense: Optional[***FloatListTensorizer.Config***]** = **None**

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**uid: Optional[***UidTensorizer.Config***]** = *UidTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
```

(continues on next page)

(continued from previous page)

```
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    },
    "uid": {
        "is_input": true,
        "column": "uid",
        "allow_unknown": true
    }
}
```

### RegressionModelInput

**class** pytext.models.doc_model.**RegressionModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**dense:** Optional[*FloatListTensorizer.Config*] = **None**

**labels:** *NumericLabelTensorizer.Config* = *NumericLabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
```

(continues on next page)

(continued from previous page)

```
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "dense": null,
    "labels": {
        "is_input": false,
        "column": "label",
        "rescale_range": null
    }
}
```

### embeddings

### char_embedding

### CharacterEmbedding.Config

**Component:** *CharacterEmbedding*

**class** CharacterEmbedding.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**embed_dim: int = 100**

**sparse: bool = False**

**cnn: *CNNParams* = *CNNParams*()**

**highway_layers: int = 0**

**projection_dim: Optional[int] = None**

**export_input_names: list[str] = ['char_vals']**

**vocab_from_train_data: bool = True**

**max_word_length: int = 20**

**min_freq: int = 1**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "embed_dim": 100,
    "sparse": false,
    "cnn": {
        "kernel_num": 100,
        "kernel_sizes": [
            3,
            4
        ],
        "weight_norm": false,
        "dilated": false,
        "causal": false
    },
    "highway_layers": 0,
    "projection_dim": null,
    "export_input_names": [
        "char_vals"
    ],
    "vocab_from_train_data": true,
    "max_word_length": 20,
    "min_freq": 1
}
```

### contextual_token_embedding

### ContextualTokenEmbedding.Config

**Component:** *ContextualTokenEmbedding*

**class** ContextualTokenEmbedding.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**embed_dim: int = 0**

**model_paths: Optional[dict[str, str]] = None**

**export_input_names: list[str] = ['contextual_token_embedding']**

**downsample_dim: Optional[int] = None**

**Default JSON**

```
{
    "embed_dim": 0,
    "model_paths": null,
    "export_input_names": [
        "contextual_token_embedding"
    ],
    "downsample_dim": null
}
```

### dict_embedding

### DictEmbedding.Config

**Component:** *DictEmbedding*

**class** DictEmbedding.**Config**
**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**embed_dim: int = 100**

**sparse: bool = False**

**pooling: PoolingType = <PoolingType.MEAN: 'mean'>**

**export_input_names: list[str] = ['dict_vals', 'dict_weights', 'dict_lens']**

**vocab_from_train_data: bool = True**

**mobile: bool = False**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "embed_dim": 100,
    "sparse": false,
    "pooling": "mean",
    "export_input_names": [
        "dict_vals",
        "dict_weights",
        "dict_lens"
    ],
    "vocab_from_train_data": true,
    "mobile": false
}
```

### embedding_base

### EmbeddingBase.Config

**Component:** *EmbeddingBase*

**class** EmbeddingBase.**Config**
**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Subclasses**

- EmbeddingList.Config
- ScriptableEmbeddingList.Config
- WordSeqEmbedding.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### embedding_list

### EmbeddingList.Config

**Component:** *EmbeddingList*

**class** EmbeddingList.**Config**

**Bases:** EmbeddingBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### mlp_embedding

### MLPEmbedding.Config

**Component:** *MLPEmbedding*

**class** MLPEmbedding.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**embed_dim: int = 100**

**embedding_init_strategy: EmbedInitStrategy = <EmbedInitStrategy.RANDOM: 'random'>**

**embedding_init_range: Optional[list[float]] = None**

**embeddding_init_std: Optional[float] = 0.02**

**export_input_names: list[str] = ['mlp_vals']**

**mlp_layer_dims: list[int] = []**

**cpu_only: bool = False**

**skip_header: bool = True**

**delimiter: str = ' '**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "embed_dim": 100,
    "embedding_init_strategy": "random",
    "embedding_init_range": null,
    "embeddding_init_std": 0.02,
    "export_input_names": [
        "mlp_vals"
    ],
    "mlp_layer_dims": [],
    "cpu_only": false,
    "skip_header": true,
    "delimiter": " "
}
```

### scriptable_embedding_list

### ScriptableEmbeddingList.Config

**Component:** *ScriptableEmbeddingList*

**class** ScriptableEmbeddingList.**Config**

**Bases:** EmbeddingBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### word_embedding

### WordEmbedding.Config

**Component:** *WordEmbedding*

**class** WordEmbedding.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**embed_dim: int = 100**

**embedding_init_strategy: EmbedInitStrategy = <EmbedInitStrategy.RANDOM: 'random'>**

**embedding_init_range: Optional[list[float]] = None**

**embeddding_init_std: Optional[float] = 0.02**

**export_input_names: list[str] = ['tokens_vals']**

**pretrained_embeddings_path: str = ''**

**vocab_file: str = ''**

**vocab_size: int = 0**

**vocab_from_train_data: bool = True**

**vocab_from_all_data: bool = False**

**vocab_from_pretrained_embeddings: bool = False**

**lowercase_tokens: bool = True**

**min_freq: int = 1**

**mlp_layer_dims: Optional[list[int]] = []**

**padding_idx: Optional[int] = None**

**cpu_only: bool = False**

**skip_header: bool = True**

**delimiter: str = ' '**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "embed_dim": 100,
    "embedding_init_strategy": "random",
    "embedding_init_range": null,
    "embeddding_init_std": 0.02,
    "export_input_names": [
        "tokens_vals"
    ],
    "pretrained_embeddings_path": "",
    "vocab_file": "",
    "vocab_size": 0,
    "vocab_from_train_data": true,
    "vocab_from_all_data": false,
    "vocab_from_pretrained_embeddings": false,
    "lowercase_tokens": true,
    "min_freq": 1,
    "mlp_layer_dims": [],
    "padding_idx": null,
    "cpu_only": false,
    "skip_header": true,
    "delimiter": " "
}
```

### word_seq_embedding

### WordSeqEmbedding.Config

**Component:** *WordSeqEmbedding*

**class** WordSeqEmbedding.**Config**

**Bases:** EmbeddingBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**word_embed_dim: int = 100**

**embedding_init_strategy: EmbedInitStrategy = <EmbedInitStrategy.RANDOM: 'random'>**

**embedding_init_range: Optional[list[float]] = None**

**embeddding_init_std: Optional[float] = 0.02**

**padding_idx: Optional[int] = None**

**lstm: *BiLSTM.Config* = *BiLSTM.Config*()**

**pretrained_embeddings_path: str = ''**

**vocab_size: int = 0** If *pretrained_embeddings_path* and *vocab_from_pretrained_embeddings* are set, only the first *vocab_size* tokens in the file will be added to the vocab.

**lowercase_tokens: bool = True**

**skip_header: bool = True**

**delimiter: str = ' '**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "word_embed_dim": 100,
    "embedding_init_strategy": "random",
    "embedding_init_range": null,
    "embeddding_init_std": 0.02,
    "padding_idx": null,
    "lstm": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true,
        "pack_sequence": true,
        "disable_sort_in_jit": false
    },
    "pretrained_embeddings_path": "",
    "vocab_size": 0,
    "lowercase_tokens": true,
    "skip_header": true,
    "delimiter": " "
}
```

### ensembles

### bagging_doc_ensemble

### BaggingDocEnsembleModel.Config

**Component:** *BaggingDocEnsembleModel*

**class** BaggingDocEnsembleModel.**Config**

**Bases:** EnsembleModel.Config

Configuration class for *NewBaggingDocEnsemble*. These attributes are used by *Ensemble.from_config()* to construct instance of *NewBaggingDocEnsemble*.

**models**
List of document classification model configurations.

**Type** List[NewDocModel.Config]

**All Attributes (including base classes)**

**models: list[*DocModel.Config*]**

**sample_rate: float = `1.0`**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### bagging_intent_slot_ensemble

### BaggingIntentSlotEnsembleModel.Config

**Component:** `BaggingIntentSlotEnsembleModel`

**class** `BaggingIntentSlotEnsembleModel.`**`Config`**
**Bases:** `EnsembleModel.Config`

Configuration class for *BaggingIntentSlotEnsemble*. These attributes are used by *Ensemble.from_config()* to construct instance of *BaggingIntentSlotEnsemble*.

**models**
List of intent-slot model configurations.

**Type** List[IntentSlotModel.Config]

**output_layer**
Output layer of intent-slot model responsible for computing loss and predictions.

**Type** IntentSlotOutputLayer

**All Attributes (including base classes)**

**models: list[*IntentSlotModel.Config*]**

**sample_rate: float = `1.0`**

**use_crf: bool = `False`**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### ensemble

### EnsembleModel.Config

**Component:** `EnsembleModel`

**class** EnsembleModel.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**models: list[Any]**

**sample_rate: float = 1.0**

**Subclasses**

- BaggingDocEnsembleModel.Config
- BaggingIntentSlotEnsembleModel.Config

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### joint_model

### IntentSlotModel.Config

**Component:** *IntentSlotModel*

**class** IntentSlotModel.**Config**

**Bases:** Model.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**word_embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation: Union[***BiLSTMDocSlotAttention.Config***,** *JointCNNRepresentation.Config***,** *SharedCNNRepresentation.Confi*

**output_layer:** *IntentSlotOutputLayer.Config* = *IntentSlotOutputLayer.Config*()

**decoder:** *IntentSlotModelDecoder.Config* = *IntentSlotModelDecoder.Config*()

**default_doc_loss_weight: float = 0.2**

**default_word_loss_weight: float = 0.5**

**Subclasses**

- ContextualIntentSlotModel.Config

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
```

(continues on next page)

(continued from previous page)

```
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "word_labels": {
            "is_input": false,
            "slot_column": "slots",
            "text_column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "allow_unknown": true
        },
        "doc_labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": true,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        },
        "doc_weight": null,
        "word_weight": null
    },
    "word_embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
```

(continues on next page)

(continued from previous page)

```
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "BiLSTMDocSlotAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "BiLSTM": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
                    "num_layers": 1,
                    "bidirectional": true,
                    "pack_sequence": true,
                    "disable_sort_in_jit": false
                }
            },
            "pooling": null,
            "slot_attention": null,
            "doc_mlp_layers": 0,
            "word_mlp_layers": 0
        }
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "doc_output": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        },
        "word_output": {
            "WordTaggingOutputLayer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
```

(continues on next page)

(continued from previous page)

```
                "loss": {
                    "CrossEntropyLoss": {}
                },
                "label_weights": {},
                "ignore_pad_in_loss": true
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "use_doc_probs_in_word": false,
        "doc_decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "word_decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        }
    },
    "default_doc_loss_weight": 0.2,
    "default_word_loss_weight": 0.5
}
```

### ModelInput

**class** pytext.models.joint_model.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**word_labels:** *SlotLabelTensorizer.Config* = *SlotLabelTensorizer.Config*(**allow_unknown=True**)

**doc_labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*(**allow_unknown=True**)

**doc_weight:** Optional[*FloatTensorizer.Config*] = **None**

**word_weight:** Optional[*FloatTensorizer.Config*] = **None**

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "word_labels": {
        "is_input": false,
        "slot_column": "slots",
        "text_column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "allow_unknown": true
    },
    "doc_labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": true,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    },
    "doc_weight": null,
    "word_weight": null
}
```

### language_models

### lmlstm

### LMLSTM.Config

**Component:** *LMLSTM*

**class** LMLSTM.**Config**

**Bases:** BaseModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** Union[*BiLSTM.Config*, *DeepCNNRepresentation.Config*] = *BiLSTM.Config*(bidirectional=False)

**decoder:** Optional[*MLPDecoder.Config*] = *MLPDecoder.Config*()

**output_layer:** *LMOutputLayer.Config* = *LMOutputLayer.Config*()

**tied_weights:** bool = False

**stateful:** bool = False

**caffe2_format:** ExporterType = <ExporterType.PREDICTOR: 'predictor'>

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": true,
            "add_eos_token": true,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
```

(continues on next page)

(continued from previous page)

```
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "BiLSTM": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm_dim": 32,
            "num_layers": 1,
            "bidirectional": false,
            "pack_sequence": true,
            "disable_sort_in_jit": false
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {}
```

(continues on next page)

(continued from previous page)

```
    },
    "tied_weights": false,
    "stateful": false,
    "caffe2_format": "predictor"
}
```

### ModelInput

**class** pytext.models.language_models.lmlstm.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens: Optional[*TokenTensorizer.Config*] = *TokenTensorizer.Config*(add_bos_token=True, add_eos_token=True)**

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": true,
        "add_eos_token": true,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    }
}
```

### masked_lm

### InputConfig

**class** pytext.models.masked_lm.**InputConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**tokens: *BERTTensorizerBase.Config* = *BERTTensorizerBase.Config*(max_seq_len=128)**

**Default JSON**

```
{
    "tokens": {
        "BERTTensorizerBase": {
            "is_input": true,
            "columns": [
                "text"
            ],
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "base_tokenizer": null,
            "vocab_file": "",
            "max_seq_len": 128
        }
    }
}
```

### MaskedLanguageModel.Config

**Component:** *MaskedLanguageModel*

**class** MaskedLanguageModel.**Config**

**Bases:** BaseModel.Config

**All Attributes (including base classes)**

**inputs:** *InputConfig* = *InputConfig*()

**encoder:** *TransformerSentenceEncoderBase.Config* = *TransformerSentenceEncoder.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *LMOutputLayer.Config* = *LMOutputLayer.Config*()

**mask_prob:** float = 0.15

**mask_bos:** bool = **False**

**masking_strategy:** MaskingStrategy = **<MaskingStrategy.RANDOM: 'random'>**

**tie_weights:** bool = **True**

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "BERTTensorizerBase": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
```

(continues on next page)

(continued from previous page)

```
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "",
                "max_seq_len": 128
            }
        }
    },
    "encoder": {
        "TransformerSentenceEncoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "dropout": 0.1,
            "attention_dropout": 0.1,
            "activation_dropout": 0.1,
            "ffn_embedding_dim": 3072,
            "num_encoder_layers": 6,
            "num_attention_heads": 8,
            "num_segments": 2,
            "use_position_embeddings": true,
            "offset_positions_by_padding": true,
            "apply_bert_init": true,
            "encoder_normalize_before": true,
            "activation_fn": "relu",
            "max_seq_len": 128,
            "multilingual": false,
            "freeze_embeddings": false,
            "n_trans_layers_to_freeze": 0,
            "use_torchscript": false,
            "use_bias_finetuning": false
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
```

(continues on next page)

(continued from previous page)

```
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {}
    },
    "mask_prob": 0.15,
    "mask_bos": false,
    "masking_strategy": "random",
    "tie_weights": true
}
```

### model

### BaseModel.Config

**Component:** *BaseModel*

**class** BaseModel.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**Subclasses**

- BertPairwiseModel.Config
- NewBertModel.Config
- _EncoderBaseModel.Config
- _EncoderPairwiseModel.Config
- BertPairwiseRegressionModel.Config
- NewBertRegressionModel.Config
- DisjointMultitaskModel.Config
- NewDisjointMultitaskModel.Config
- ByteTokensDocumentModel.Config
- DocModel.Config
- DocRegressionModel.Config
- PersonalizedDocModel.Config
- IntentSlotModel.Config
- LMLSTM.Config
- MaskedLanguageModel.Config
- Model.Config
- BasePairwiseModel.Config
- PairwiseModel.Config
- BertSquadQAModel.Config

- DrQAModel.Config
- QueryDocPairwiseRankingModel.Config
- RoBERTa.Config
- RoBERTaR3F.Config
- RoBERTaRegression.Config
- RoBERTaWordTaggingModel.Config
- SELFIE.Config
- ContextualIntentSlotModel.Config
- Seq2SeqModel.Config
- SeqNNModel.Config
- TwoTowerClassificationModel.Config
- WordTaggingLiteModel.Config
- WordTaggingModel.Config

**Default JSON**

```
{
    "inputs": {}
}
```

### Model.Config

**Component:** *Model*

**class** Model.**Config**

**Bases:** BaseModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**Subclasses**

- DisjointMultitaskModel.Config
- NewDisjointMultitaskModel.Config
- ByteTokensDocumentModel.Config
- DocModel.Config
- DocRegressionModel.Config
- PersonalizedDocModel.Config
- IntentSlotModel.Config
- ContextualIntentSlotModel.Config
- Seq2SeqModel.Config
- SeqNNModel.Config
- WordTaggingLiteModel.Config

- WordTaggingModel.Config

**Default JSON**

```
{
    "inputs": {}
}
```

### ModelInput

**class** pytext.models.model.**ModelInput**

**Bases:** *ModelInputBase*

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

### module

### Module.Config

**Component:** *Module*

**class** Module.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Subclasses**

- *FeatureConfig*
- *BatcherSchedulerConfig*
- *ExponentialBatcherSchedulerConfig*
- DecoderBase.Config
- IntentSlotModelDecoder.Config
- MLPDecoder.Config
- MLPDecoderQueryResponse.Config
- MLPDecoderTwoTower.Config
- CharacterEmbedding.Config
- DictEmbedding.Config
- EmbeddingBase.Config
- EmbeddingList.Config

- MLPEmbedding.Config
- ScriptableEmbeddingList.Config
- WordEmbedding.Config
- WordSeqEmbedding.Config
- DenseRetrievalOutputLayer.Config
- PairwiseCosineDistanceOutputLayer.Config
- BinaryClassificationOutputLayer.Config
- ClassificationOutputLayer.Config
- MultiLabelOutputLayer.Config
- MulticlassOutputLayer.Config
- PairwiseCosineRegressionOutputLayer.Config
- RegressionOutputLayer.Config
- IntentSlotOutputLayer.Config
- LMOutputLayer.Config
- OutputLayerBase.Config
- PairwiseRankingOutputLayer.Config
- SquadOutputLayer.Config
- CRFOutputLayer.Config
- WordTaggingOutputLayer.Config
- DotProductSelfAttention.Config
- MultiplicativeAttention.Config
- SequenceAlignedAttention.Config
- AugmentedLSTM.Config
- BiLSTM.Config
- BiLSTMDocAttention.Config
- BiLSTMDocSlotAttention.Config
- BiLSTMSlotAttention.Config
- BSeqCNNRepresentation.Config
- ContextualIntentSlotRepresentation.Config
- DeepCNNRepresentation.Config
- DocNNRepresentation.Config
- HuggingFaceBertSentenceEncoder.Config
- HuggingFaceElectraSentenceEncoder.Config
- JointCNNRepresentation.Config
- SharedCNNRepresentation.Config
- OrderedNeuronLSTM.Config

- OrderedNeuronLSTMLayer.Config
- PassThroughRepresentation.Config
- LastTimestepPool.Config
- MaxPool.Config
- MeanPool.Config
- NoPool.Config
- PureDocAttention.Config
- RepresentationBase.Config
- SeqRepresentation.Config
- SparseTransformerSentenceEncoder.Config
- StackedBidirectionalRNN.Config
- TransformerSentenceEncoder.Config
- TransformerSentenceEncoderBase.Config
- RoBERTaEncoder.Config
- RoBERTaEncoderBase.Config
- RoBERTaEncoderJit.Config
- PyTextIncrementalDecoderComponent.Config
- PyTextSeq2SeqModule.Config
- DecoderWithLinearOutputProjection.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### output_layers

### distance_output_layer

### DenseRetrievalOutputLayer.Config

**Component:** *DenseRetrievalOutputLayer*

**class** DenseRetrievalOutputLayer.**Config**

**Bases:** PairwiseCosineDistanceOutputLayer.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*BinaryCrossEntropyLoss.Config*, *CosineEmbeddingLoss.Config*, *MAELoss.Config*, *MSELoss.Config*, *NLLLoss.C***

**score_threshold: float = 0.9**

**score_type: OutputScore = <OutputScore.norm_cosine: 2>**

**label_weights: Optional[dict[str, float]] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CosineEmbeddingLoss": {
            "margin": 0.0
        }
    },
    "score_threshold": 0.9,
    "score_type": 2,
    "label_weights": null
}
```

### PairwiseCosineDistanceOutputLayer.Config

**Component:** *PairwiseCosineDistanceOutputLayer*

**class** PairwiseCosineDistanceOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*BinaryCrossEntropyLoss.Config*, *CosineEmbeddingLoss.Config*, *MAELoss.Config*, *MSELoss.Config*, *NLLLoss.C***

**score_threshold: float = 0.9**

**score_type: OutputScore = <OutputScore.norm_cosine: 2>**

**label_weights: Optional[dict[str, float]] = None**

**Subclasses**

- DenseRetrievalOutputLayer.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CosineEmbeddingLoss": {
            "margin": 0.0
        }
    },
    "score_threshold": 0.9,
    "score_type": 2,
    "label_weights": null
}
```

### doc_classification_output_layer

#### BinaryClassificationOutputLayer.Config

**Component:** *BinaryClassificationOutputLayer*

**class** BinaryClassificationOutputLayer.**Config**

**Bases:** ClassificationOutputLayer.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*CrossEntropyLoss.Config*, *BinaryCrossEntropyLoss.Config*, *BinaryCrossEntropyWithLogitsLoss.Config*, *MultiLa***

**label_weights: Optional[dict[str, float]] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    },
    "label_weights": null
}
```

#### ClassificationOutputLayer.Config

**Component:** *ClassificationOutputLayer*

**class** ClassificationOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*CrossEntropyLoss.Config*, *BinaryCrossEntropyLoss.Config*, *BinaryCrossEntropyWithLogitsLoss.Config*, *MultiLa***

**label_weights: Optional[dict[str, float]] = None**

**Subclasses**

- BinaryClassificationOutputLayer.Config
- MultiLabelOutputLayer.Config
- MulticlassOutputLayer.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    },
    "label_weights": null
}
```

### MultiLabelOutputLayer.Config

**Component:** *MultiLabelOutputLayer*

**class** MultiLabelOutputLayer.**Config**

**Bases:** ClassificationOutputLayer.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*CrossEntropyLoss.Config*, *BinaryCrossEntropyLoss.Config*, *BinaryCrossEntropyWithLogitsLoss.Config*, *MultiLa***

**label_weights: Optional[dict[str, float]] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    },
    "label_weights": null
}
```

### MulticlassOutputLayer.Config

**Component:** *MulticlassOutputLayer*

**class** MulticlassOutputLayer.**Config**

**Bases:** ClassificationOutputLayer.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*CrossEntropyLoss.Config*, *BinaryCrossEntropyLoss.Config*, *BinaryCrossEntropyWithLogitsLoss.Config*, *MultiLa***

**label_weights: Optional[dict[str, float]] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    },
    "label_weights": null
}
```

### doc_regression_output_layer

### PairwiseCosineRegressionOutputLayer.Config

**Component:** *PairwiseCosineRegressionOutputLayer*

**class** PairwiseCosineRegressionOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*MSELoss.Config*, *MAELoss.Config*] = *MSELoss.Config*()**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "MSELoss": {}
    }
}
```

### RegressionOutputLayer.Config

**Component:** *RegressionOutputLayer*

**class** RegressionOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: *MSELoss.Config* = *MSELoss.Config*()**

**squash_to_unit_range: bool = False**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {},
    "squash_to_unit_range": false
}
```

### intent_slot_output_layer

### IntentSlotOutputLayer.Config

**Component:** *IntentSlotOutputLayer*

**class** IntentSlotOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**doc_output:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**word_output: Union[***WordTaggingOutputLayer.Config***, *CRFOutputLayer.Config*] = *WordTaggingOutputLayer.Config*()**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "doc_output": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    },
    "word_output": {
        "WordTaggingOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": {},
            "ignore_pad_in_loss": true
        }
    }
}
```

### lm_output_layer

### LMOutputLayer.Config

**Component:** *LMOutputLayer*

**class** LMOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: *CrossEntropyLoss.Config* = *CrossEntropyLoss.Config*()**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {}
}
```

### output_layer_base

### OutputLayerBase.Config

**Component:** *OutputLayerBase*

**class** OutputLayerBase.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Subclasses**

- DenseRetrievalOutputLayer.Config
- PairwiseCosineDistanceOutputLayer.Config
- BinaryClassificationOutputLayer.Config
- ClassificationOutputLayer.Config
- MultiLabelOutputLayer.Config
- MulticlassOutputLayer.Config
- PairwiseCosineRegressionOutputLayer.Config
- RegressionOutputLayer.Config
- IntentSlotOutputLayer.Config
- LMOutputLayer.Config
- PairwiseRankingOutputLayer.Config
- SquadOutputLayer.Config
- CRFOutputLayer.Config

- WordTaggingOutputLayer.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### pairwise_ranking_output_layer

#### PairwiseRankingOutputLayer.Config

**Component:** *PairwiseRankingOutputLayer*

**class** PairwiseRankingOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss:** *PairwiseRankingLoss.Config* = *PairwiseRankingLoss.Config*()

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "margin": 1.0
    }
}
```

### squad_output_layer

#### SquadOutputLayer.Config

**Component:** *SquadOutputLayer*

**class** SquadOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**loss: Union[*CrossEntropyLoss.Config*, *KLDivergenceCELoss.Config*] = *CrossEntropyLoss.Config*()**

**ignore_impossible: bool = `True`**

**pos_loss_weight: float = `0.5`**

**has_answer_loss_weight: float = `0.5`**

**false_label: str = `'False'`**

**max_answer_len: int = `30`**

**hard_weight: float = `0.0`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    },
    "ignore_impossible": true,
    "pos_loss_weight": 0.5,
    "has_answer_loss_weight": 0.5,
    "false_label": "False",
    "max_answer_len": 30,
    "hard_weight": 0.0
}
```

### word_tagging_output_layer

### CRFOutputLayer.Config

**Component:** *`CRFOutputLayer`*

**class** `CRFOutputLayer.`**`Config`**

**Bases:** `OutputLayerBase.Config`

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
```

(continues on next page)

(continued from previous page)

```
    "shared_module_key": null
}
```

### WordTaggingOutputLayer.Config

**Component:** *WordTaggingOutputLayer*

**class** WordTaggingOutputLayer.**Config**

**Bases:** OutputLayerBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**loss: Union[*CrossEntropyLoss.Config*, *BinaryCrossEntropyLoss.Config*, *AUCPRHingeLoss.Config*, *KLDivergenceBCELoss.C***

**label_weights: dict[str, float] = { }**

**ignore_pad_in_loss: Optional[bool] = True**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    },
    "label_weights": {},
    "ignore_pad_in_loss": true
}
```

### pair_classification_model

### BasePairwiseModel.Config

**Component:** *BasePairwiseModel*

**class** BasePairwiseModel.**Config**

**Bases:** BaseModel.Config

**All Attributes (including base classes)**

**inputs: *ModelInput* = *ModelInput*()**

**decoder: *MLPDecoder.Config* = *MLPDecoder.Config*()**

**output_layer: Union[*ClassificationOutputLayer.Config*, *PairwiseCosineDistanceOutputLayer.Config*] = *ClassificationOutput***

**encode_relations: bool = True**

**Subclasses**

- BertPairwiseModel.Config
- _EncoderPairwiseModel.Config
- BertPairwiseRegressionModel.Config
- PairwiseModel.Config
- QueryDocPairwiseRankingModel.Config

**Default JSON**

```
{
    "inputs": {},
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "ClassificationOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        }
    },
    "encode_relations": true
}
```

### ModelInput

**class** pytext.models.pair_classification_model.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens1:** *TokenTensorizer.Config* = *TokenTensorizer.Config*(**column='text1'**)

**tokens2:** *TokenTensorizer.Config* = *TokenTensorizer.Config*(**column='text2'**)

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens1": {
        "is_input": true,
        "column": "text1",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "tokens2": {
        "is_input": true,
        "column": "text2",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    }
}
```

### PairwiseModel.Config

**Component:** *PairwiseModel*

**class** PairwiseModel.**Config**

**Bases:** BasePairwiseModel.Config

**encode_relations**

if *false*, return the concatenation of the two representations; if *true*, also concatenate their pairwise absolute difference and pairwise elementwise product (à la arXiv:1705.02364). Default: *true*.

**Type** bool

**tied_representation**

whether to use the same representation, with tied weights, for all the input subrepresentations. Default: *true*.

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** Union[*ClassificationOutputLayer.Config*, *PairwiseCosineDistanceOutputLayer.Config*] = *ClassificationOutput*

**encode_relations:** bool = **True**

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** Union[*BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*] = *BiLSTMDocAttention.Config*()

**shared_representations:** bool = **True**

**Subclasses**

- QueryDocPairwiseRankingModel.Config

**Default JSON**

```
{
    "inputs": {
        "tokens1": {
            "is_input": true,
            "column": "text1",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
```

(continues on next page)

(continued from previous page)

```
            },
            "vocab_file_delimiter": " "
        },
        "tokens2": {
            "is_input": true,
            "column": "text2",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "ClassificationOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
```

(continues on next page)

(continued from previous page)

```
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        }
    },
    "encode_relations": true,
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
```

(continues on next page)

(continued from previous page)

```
            },
            "mlp_decoder": null
        }
    },
    "shared_representations": true
}
```

### qna

### bert_squad_qa

### BertSquadQAModel.Config

**Component:** *BertSquadQAModel*

**class** BertSquadQAModel.**Config**

**Bases:** NewBertModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**encoder:** *TransformerSentenceEncoderBase.Config* = *HuggingFaceBertSentenceEncoder.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *SquadOutputLayer.Config* = *SquadOutputLayer.Config*()

**pos_decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*(out_dim=2)

**has_ans_decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*(out_dim=2)

**is_kd:** bool = **False**

**Default JSON**

```
{
    "inputs": {
        "squad_input": {
            "SquadForBERTTensorizer": {
                "is_input": true,
                "columns": [
                    "question",
                    "doc"
                ],
                "tokenizer": {
                    "WordPieceTokenizer": {
                        "basic_tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        },
                        "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                    }
                },
                "base_tokenizer": null,
```

(continues on next page)

(continued from previous page)

```
                "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/vocab.txt",
                "max_seq_len": 256,
                "answers_column": "answers",
                "answer_starts_column": "answer_starts"
            }
        },
        "has_answer": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "has_answer",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "encoder": {
        "HuggingFaceBertSentenceEncoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/",
            "load_weights": true
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
```

(continues on next page)

(continued from previous page)

```
        },
        "ignore_impossible": true,
        "pos_loss_weight": 0.5,
        "has_answer_loss_weight": 0.5,
        "false_label": "False",
        "max_answer_len": 30,
        "hard_weight": 0.0
    },
    "pos_decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": 2,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "has_ans_decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": 2,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "is_kd": false
}
```

### ModelInput

**class** pytext.models.qna.bert_squad_qa.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**squad_input: Union[*SquadForBERTTensorizer.Config*, *SquadForRoBERTaTensorizer.Config*] = *SquadForBERTTensorizer.C***

**has_answer: *LabelTensorizer.Config* = *LabelTensorizer.Config*(column='has_answer')**

**Default JSON**

```
{
    "squad_input": {
        "SquadForBERTTensorizer": {
```

(continues on next page)

(continued from previous page)

```
            "is_input": true,
            "columns": [
                "question",
                "doc"
            ],
            "tokenizer": {
                "WordPieceTokenizer": {
                    "basic_tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    },
                    "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                }
            },
            "base_tokenizer": null,
            "vocab_file": "manifold://nlp_technologies/tree/huggingface-models/bert-
↪base-uncased/vocab.txt",
            "max_seq_len": 256,
            "answers_column": "answers",
            "answer_starts_column": "answer_starts"
        }
    },
    "has_answer": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "has_answer",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    }
}
```

### dr_qa

### DrQAModel.Config

**Component:** *DrQAModel*

**class** DrQAModel.**Config**

**Bases:** BaseModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**dropout: float** = 0.4

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*(**embed_dim=300, pretrained_embeddings_path='/mnt/v**

**ques_rnn:** *StackedBidirectionalRNN.Config* = *StackedBidirectionalRNN.Config*(**dropout=0.4**)

**doc_rnn:** *StackedBidirectionalRNN.Config* = *StackedBidirectionalRNN.Config*(**dropout=0.4**)

**output_layer:** *SquadOutputLayer.Config* = *SquadOutputLayer.Config*()

**is_kd: bool** = **False**

**Default JSON**

```
{
    "inputs": {
        "squad_input": {
            "SquadTensorizer": {
                "is_input": true,
                "column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\W+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " ",
                "doc_column": "doc",
                "ques_column": "question",
                "answers_column": "answers",
                "answer_starts_column": "answer_starts",
                "max_ques_seq_len": 64,
                "max_doc_seq_len": 256
            }
        },
        "has_answer": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "has_answer",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "dropout": 0.4,
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 300,
```

(continues on next page)

(continued from previous page)

```
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "/mnt/vol/pytext/users/kushall/pretrained/glove.
↪840B.300d.txt",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": true,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "ques_rnn": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_size": 32,
        "num_layers": 1,
        "dropout": 0.4,
        "bidirectional": true,
        "rnn_type": "lstm",
        "concat_layers": true
    },
    "doc_rnn": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_size": 32,
        "num_layers": 1,
        "dropout": 0.4,
        "bidirectional": true,
        "rnn_type": "lstm",
        "concat_layers": true
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "ignore_impossible": true,
        "pos_loss_weight": 0.5,
        "has_answer_loss_weight": 0.5,
        "false_label": "False",
```

(continues on next page)

(continued from previous page)

```
        "max_answer_len": 30,
        "hard_weight": 0.0
    },
    "is_kd": false
}
```

### ModelInput

**class** pytext.models.qna.dr_qa.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**squad_input:** *SquadTensorizer.Config* = *SquadTensorizer.Config*()

**has_answer:** *LabelTensorizer.Config* = *LabelTensorizer.Config*(column='has_answer')

**Default JSON**

```
{
    "squad_input": {
        "SquadTensorizer": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\W+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " ",
            "doc_column": "doc",
            "ques_column": "question",
            "answers_column": "answers",
            "answer_starts_column": "answer_starts",
            "max_ques_seq_len": 64,
            "max_doc_seq_len": 256
        }
    },
    "has_answer": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "has_answer",
            "allow_unknown": false,
            "pad_in_vocab": false,
```

(continues on next page)

(continued from previous page)

```
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    }
}
```

### query_document_pairwise_ranking_model

#### ModelInput

**class** pytext.models.query_document_pairwise_ranking_model.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**pos_response:** *TokenTensorizer.Config* = *TokenTensorizer.Config*(**column='pos_response'**)

**neg_response:** *TokenTensorizer.Config* = *TokenTensorizer.Config*(**column='neg_response'**)

**query:** *TokenTensorizer.Config* = *TokenTensorizer.Config*(**column='query'**)

**Default JSON**

```
{
    "pos_response": {
        "is_input": true,
        "column": "pos_response",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "neg_response": {
        "is_input": true,
        "column": "neg_response",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
```

(continues on next page)

(continued from previous page)

```
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "query": {
        "is_input": true,
        "column": "query",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    }
}
```

### QueryDocPairwiseRankingModel.Config

**Component:** *QueryDocPairwiseRankingModel*

**class** QueryDocPairwiseRankingModel.**Config**

**Bases:** PairwiseModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**decoder:** *MLPDecoderQueryResponse.Config* = *MLPDecoderQueryResponse.Config*()

**output_layer:** *PairwiseRankingOutputLayer.Config* = *PairwiseRankingOutputLayer.Config*()

**encode_relations:** bool = **True**

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** Union[*BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*] = *BiLSTMDocAttention.Config*()

**shared_representations: bool = `True`**

**decoder_output_dim: int = `64`**

**Default JSON**

```
{
    "inputs": {
        "pos_response": {
            "is_input": true,
            "column": "pos_response",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "neg_response": {
            "is_input": true,
            "column": "neg_response",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "query": {
            "is_input": true,
            "column": "query",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
```

(continues on next page)

(continued from previous page)

```
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": []
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "margin": 1.0
        }
    },
    "encode_relations": true,
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
```

(continues on next page)

(continued from previous page)

```
        "delimiter": " "
    },
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    },
    "shared_representations": true,
    "decoder_output_dim": 64
}
```

### r3f_models

### R3FConfigOptions

**class** pytext.models.r3f_models.**R3FConfigOptions**

**Bases:** *ConfigBase*

Configuration options for models using R3F

**All Attributes (including base classes)**

**r3f_lambda_by_loss: dict[str, float] = { }**

**r3f_default_lambda: float = `0.5`**

**eps: float = `1e-05`**

**noise_type: R3FNoiseType = `<R3FNoiseType.UNIFORM: 'uniform'>`**

**Default JSON**

```
{
    "r3f_lambda_by_loss": {},
```

(continues on next page)

(continued from previous page)

```
    "r3f_default_lambda": 0.5,
    "eps": 1e-05,
    "noise_type": "uniform"
}
```

### representations

### attention

### DotProductSelfAttention.Config

**Component:** *DotProductSelfAttention*

**class** DotProductSelfAttention.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**input_dim: int = 32**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "input_dim": 32
}
```

### MultiplicativeAttention.Config

**Component:** *MultiplicativeAttention*

**class** MultiplicativeAttention.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**p_hidden_dim: int = 32**

**q_hidden_dim: int = 32**

**normalize: bool = `False`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "p_hidden_dim": 32,
    "q_hidden_dim": 32,
    "normalize": false
}
```

### SequenceAlignedAttention.Config

**Component:** *SequenceAlignedAttention*

**class** SequenceAlignedAttention.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**proj_dim: int = `32`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "proj_dim": 32
}
```

### augmented_lstm

### AugmentedLSTM.Config

**Component:** *AugmentedLSTM*

**class** AugmentedLSTM.**Config**

**Bases:** RepresentationBase.Config, *ConfigBase*

Configuration class for *AugmentedLSTM*.

**dropout**

Variational dropout probability to use. Defaults to 0.0.

**Type** float

**lstm_dim**
Number of features in the hidden state of the LSTM. Defaults to 32.

**Type** int

**num_layers**
Number of recurrent layers. Eg. setting *num_layers=2* would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in the outputs of the first LSTM and computing the final result. Defaults to 1.

**Type** int

**bidirectional**
If *True*, becomes a bidirectional LSTM. Defaults to *True*.

**Type** bool

**use_highway**
If *True* we append a highway network to the outputs of the LSTM.

**Type** bool

**use_bias**
If *True* we use a bias in our LSTM calculations, otherwise we don't.

**Type** bool

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**dropout: float = `0.0`**

**lstm_dim: int = `32`**

**use_highway: bool = `True`**

**bidirectional: bool = `False`**

**num_layers: int = `1`**

**use_bias: bool = `False`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "dropout": 0.0,
    "lstm_dim": 32,
    "use_highway": true,
    "bidirectional": false,
    "num_layers": 1,
    "use_bias": false
}
```

### biLstm

### BiLSTM.Config

**Component:** *BiLSTM*

**class** BiLSTM.**Config**

**Bases:** RepresentationBase.Config, *ConfigBase*

Configuration class for *BiLSTM*.

**dropout**

Dropout probability to use. Defaults to 0.4.

**Type** float

**lstm_dim**

Number of features in the hidden state of the LSTM. Defaults to 32.

**Type** int

**num_layers**

Number of recurrent layers. Eg. setting *num_layers=2* would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in the outputs of the first LSTM and computing the final result. Defaults to 1.

**Type** int

**bidirectional**

If *True*, becomes a bidirectional LSTM. Defaults to *True*.

**Type** bool

**disable_sort_in_jit**

If *True*, disable sort in pack_padded_sequence to allow inference on GPU. Defaults to *False*.

**Type** bool

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**dropout: float = 0.4**

**lstm_dim: int = 32**

**num_layers: int = 1**

**bidirectional: bool = True**

**pack_sequence: bool = True**

**disable_sort_in_jit: bool = False**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "dropout": 0.4,
    "lstm_dim": 32,
    "num_layers": 1,
    "bidirectional": true,
    "pack_sequence": true,
    "disable_sort_in_jit": false
}
```

### bilstm_doc_attention

### BiLSTMDocAttention.Config

**Component:** *BiLSTMDocAttention*

**class** BiLSTMDocAttention.**Config**

**Bases:** RepresentationBase.Config

Configuration class for *BiLSTM*.

**dropout**

Dropout probability to use. Defaults to 0.4.

**Type** float

**lstm**

Config for the BiLSTM.

**Type** BiLSTM.Config

**pooling**

Config for the underlying pooling module.

**Type** ConfigBase

**mlp_decoder**

Config for the non-linear projection module.

**Type** MLPDecoder.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**dropout: float = 0.4**

**lstm:** *BiLSTM.Config* = *BiLSTM.Config*()

**pooling: Union[***SelfAttention.Config***,** *MaxPool.Config***,** *MeanPool.Config***,** *NoPool.Config***,** *LastTimestepPool.Config***] =** *SelfAt*

**mlp_decoder: Optional[***MLPDecoder.Config***] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "dropout": 0.4,
    "lstm": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true,
        "pack_sequence": true,
        "disable_sort_in_jit": false
    },
    "pooling": {
        "SelfAttention": {
            "attn_dimension": 64,
            "dropout": 0.4
        }
    },
    "mlp_decoder": null
}
```

### bilstm_doc_slot_attention

### BiLSTMDocSlotAttention.Config

**Component:** *BiLSTMDocSlotAttention*

**class** BiLSTMDocSlotAttention.**Config**

**Bases:** RepresentationBase.Config, *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**dropout: float = 0.4**

**lstm: Union[*BiLSTM.Config*, *OrderedNeuronLSTM.Config*, *AugmentedLSTM.Config*] = *BiLSTM.Config*()**

**pooling: Union[*SelfAttention.Config*, *MaxPool.Config*, *MeanPool.Config*, NoneType] = None**

**slot_attention: Optional[*SlotAttention.Config*] = None**

**doc_mlp_layers: int = 0**

**word_mlp_layers: int = 0**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "dropout": 0.4,
    "lstm": {
        "BiLSTM": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm_dim": 32,
            "num_layers": 1,
            "bidirectional": true,
            "pack_sequence": true,
            "disable_sort_in_jit": false
        }
    },
    "pooling": null,
    "slot_attention": null,
    "doc_mlp_layers": 0,
    "word_mlp_layers": 0
}
```

### bilstm_slot_attn

### BiLSTMSlotAttention.Config

**Component:** *BiLSTMSlotAttention*

**class** BiLSTMSlotAttention.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**dropout: float = 0.4**

**lstm:** *BiLSTM.Config* **=** *BiLSTM.Config*()

**slot_attention:** *SlotAttention.Config* **=** *SlotAttention.Config*()

**mlp_decoder: Optional[***MLPDecoder.Config***] = None**

**Default JSON**

```
{
    "load_path": null,
```

(continues on next page)

(continued from previous page)

```
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "dropout": 0.4,
    "lstm": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true,
        "pack_sequence": true,
        "disable_sort_in_jit": false
    },
    "slot_attention": {
        "attn_dimension": 64,
        "attention_type": "no_attention"
    },
    "mlp_decoder": null
}
```

### biseqcnn

### BSeqCNNRepresentation.Config

**Component:** *BSeqCNNRepresentation*

**class** BSeqCNNRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**cnn: *CNNParams* = *CNNParams*()**

**fwd_bwd_context_len: int = 5**

**surrounding_context_len: int = 2**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "cnn": {
        "kernel_num": 100,
        "kernel_sizes": [
```

(continues on next page)

(continued from previous page)

```
            3,
            4
        ],
        "weight_norm": false,
        "dilated": false,
        "causal": false
    },
    "fwd_bwd_context_len": 5,
    "surrounding_context_len": 2
}
```

### contextual_intent_slot_rep

### ContextualIntentSlotRepresentation.Config

**Component:** *ContextualIntentSlotRepresentation*

**class** ContextualIntentSlotRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**sen_representation:** *DocNNRepresentation.Config* **=** *DocNNRepresentation.Config*()

**seq_representation:** *DocNNRepresentation.Config* **=** *DocNNRepresentation.Config*()

**joint_representation: Union[***BiLSTMDocSlotAttention.Config***,** *JointCNNRepresentation.Config***] =** *BiLSTMDocSlotAttentio*

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "sen_representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "cnn": {
            "kernel_num": 100,
            "kernel_sizes": [
                3,
                4
            ],
            "weight_norm": false,
            "dilated": false,
```

(continues on next page)

(continued from previous page)

```
            "causal": false
        },
        "pooling": "max"
    },
    "seq_representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "cnn": {
            "kernel_num": 100,
            "kernel_sizes": [
                3,
                4
            ],
            "weight_norm": false,
            "dilated": false,
            "causal": false
        },
        "pooling": "max"
    },
    "joint_representation": {
        "BiLSTMDocSlotAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "BiLSTM": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
                    "num_layers": 1,
                    "bidirectional": true,
                    "pack_sequence": true,
                    "disable_sort_in_jit": false
                }
            },
            "pooling": null,
            "slot_attention": null,
            "doc_mlp_layers": 0,
            "word_mlp_layers": 0
        }
    }
}
```

### deepcnn

### DeepCNNRepresentation.Config

**Component:** *DeepCNNRepresentation*

**class** DeepCNNRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**cnn:** *CNNParams* = *CNNParams*()

**dropout: float = 0.3**

**activation: Activation = <Activation.GLU: 'glu'>**

**separable: bool = False**

**bottleneck: int = 0**

**pooling_type: PoolingType = <PoolingType.NONE: 'none'>**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "cnn": {
        "kernel_num": 100,
        "kernel_sizes": [
            3,
            4
        ],
        "weight_norm": false,
        "dilated": false,
        "causal": false
    },
    "dropout": 0.3,
    "activation": "glu",
    "separable": false,
    "bottleneck": 0,
    "pooling_type": "none"
}
```

### docnn

### DocNNRepresentation.Config

**Component:** *DocNNRepresentation*

**class** DocNNRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**dropout: float = `0.4`**

**cnn: *CNNParams* = *CNNParams*()**

**pooling: PoolingType = `<PoolingType.MAX: 'max'>`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "dropout": 0.4,
    "cnn": {
        "kernel_num": 100,
        "kernel_sizes": [
            3,
            4
        ],
        "weight_norm": false,
        "dilated": false,
        "causal": false
    },
    "pooling": "max"
}
```

### huggingface_bert_sentence_encoder

### HuggingFaceBertSentenceEncoder.Config

**Component:** *HuggingFaceBertSentenceEncoder*

**class** `HuggingFaceBertSentenceEncoder.`**`Config`**

**Bases:** `TransformerSentenceEncoderBase.Config`, *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**output_dropout: float = `0.4`**

**embedding_dim: int = `768`**

**pooling: PoolingMethod = `<PoolingMethod.CLS_TOKEN: 'cls_token'>`**

**export: bool = `False`**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**bert_cpt_dir: str = 'manifold://nlp_technologies/tree/huggingface-models/bert-base-uncased/'**

**load_weights: bool = True**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false,
    "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/bert-base-
↪uncased/",
    "load_weights": true
}
```

### huggingface_electra_sentence_encoder

### HuggingFaceElectraSentenceEncoder.Config

**Component:** *HuggingFaceElectraSentenceEncoder*

**class** HuggingFaceElectraSentenceEncoder.**Config**

**Bases:** TransformerSentenceEncoderBase.Config, *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**output_dropout: float = 0.4**

**embedding_dim: int = 768**

**pooling: PoolingMethod = <PoolingMethod.CLS_TOKEN: 'cls_token'>**

**export: bool = False**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**electra_cpt_dir: str = '/mnt/vol/nlp_technologies/electra/electra-base-discriminator'**

**load_weights: bool = True**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false,
    "electra_cpt_dir": "/mnt/vol/nlp_technologies/electra/electra-base-discriminator",
    "load_weights": true
}
```

### jointcnn_rep

#### JointCNNRepresentation.Config

**Component:** *JointCNNRepresentation*

**class** JointCNNRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**doc_representation:** *DocNNRepresentation.Config* **=** *DocNNRepresentation.Config*()

**word_representation: Union[***BSeqCNNRepresentation.Config***,** *DeepCNNRepresentation.Config***] =** *BSeqCNNRepresentation*

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "doc_representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "cnn": {
            "kernel_num": 100,
            "kernel_sizes": [
                3,
                4
```

(continues on next page)

(continued from previous page)

```
            ],
            "weight_norm": false,
            "dilated": false,
            "causal": false
        },
        "pooling": "max"
    },
    "word_representation": {
        "BSeqCNNRepresentation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "cnn": {
                "kernel_num": 100,
                "kernel_sizes": [
                    3,
                    4
                ],
                "weight_norm": false,
                "dilated": false,
                "causal": false
            },
            "fwd_bwd_context_len": 5,
            "surrounding_context_len": 2
        }
    }
}
```

### SharedCNNRepresentation.Config

**Component:** *SharedCNNRepresentation*

**class** SharedCNNRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**word_representation: Union[*BSeqCNNRepresentation.Config*, *DeepCNNRepresentation.Config*] = *DeepCNNRepresentation***

**pooling_type: PoolingType = <PoolingType.MAX: 'max'>**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
```

(continues on next page)

(continued from previous page)

```
    "word_representation": {
        "DeepCNNRepresentation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "cnn": {
                "kernel_num": 100,
                "kernel_sizes": [
                    3,
                    4
                ],
                "weight_norm": false,
                "dilated": false,
                "causal": false
            },
            "dropout": 0.3,
            "activation": "glu",
            "separable": false,
            "bottleneck": 0,
            "pooling_type": "none"
        }
    },
    "pooling_type": "max"
}
```

### ordered_neuron_lstm

### OrderedNeuronLSTM.Config

**Component:** *OrderedNeuronLSTM*

**class** OrderedNeuronLSTM.**Config**

**Bases:** RepresentationBase.Config, *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**dropout: float = 0.4**

**lstm_dim: int = 32**

**num_layers: int = 1**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
```

(continues on next page)

(continued from previous page)

```
    "dropout": 0.4,
    "lstm_dim": 32,
    "num_layers": 1
}
```

### OrderedNeuronLSTMLayer.Config

**Component:** *OrderedNeuronLSTMLayer*

**class** OrderedNeuronLSTMLayer.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### pass_through

### PassThroughRepresentation.Config

**Component:** *PassThroughRepresentation*

**class** PassThroughRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### pooling

#### BoundaryPool.Config

**Component:** *BoundaryPool*

**class** BoundaryPool.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**boundary_type: str = 'first'**

**Default JSON**

```
{
    "boundary_type": "first"
}
```

#### LastTimestepPool.Config

**Component:** *LastTimestepPool*

**class** LastTimestepPool.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

#### MaxPool.Config

**Component:** *MaxPool*

**class** MaxPool.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### MeanPool.Config

**Component:** *MeanPool*

**class** MeanPool.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### NoPool.Config

**Component:** *NoPool*

**class** NoPool.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
```

(continues on next page)

(continued from previous page)

```
    "shared_module_key": null
}
```

### SelfAttention.Config

**Component:** *SelfAttention*

**class** SelfAttention.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**attn_dimension: int = 64**

**dropout: float = 0.4**

**Default JSON**

```
{
    "attn_dimension": 64,
    "dropout": 0.4
}
```

### pure_doc_attention

### PureDocAttention.Config

**Component:** *PureDocAttention*

**class** PureDocAttention.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**dropout: float = 0.4**

**pooling: Union[*SelfAttention.Config*, *MaxPool.Config*, *MeanPool.Config*, *NoPool.Config*, *BoundaryPool.Config*] = *SelfAttent***

**mlp_decoder: Optional[*MLPDecoder.Config*] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "dropout": 0.4,
```

(continues on next page)

(continued from previous page)

```
    "pooling": {
        "SelfAttention": {
            "attn_dimension": 64,
            "dropout": 0.4
        }
    },
    "mlp_decoder": null
}
```

### representation_base

#### RepresentationBase.Config

**Component:** *RepresentationBase*

**class** RepresentationBase.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Subclasses**

- AugmentedLSTM.Config
- BiLSTM.Config
- BiLSTMDocAttention.Config
- BiLSTMDocSlotAttention.Config
- BiLSTMSlotAttention.Config
- BSeqCNNRepresentation.Config
- ContextualIntentSlotRepresentation.Config
- DeepCNNRepresentation.Config
- DocNNRepresentation.Config
- HuggingFaceBertSentenceEncoder.Config
- HuggingFaceElectraSentenceEncoder.Config
- JointCNNRepresentation.Config
- SharedCNNRepresentation.Config
- OrderedNeuronLSTM.Config
- PassThroughRepresentation.Config
- PureDocAttention.Config
- SeqRepresentation.Config

- SparseTransformerSentenceEncoder.Config
- TransformerSentenceEncoder.Config
- TransformerSentenceEncoderBase.Config
- RoBERTaEncoder.Config
- RoBERTaEncoderBase.Config
- RoBERTaEncoderJit.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### seq_rep

### SeqRepresentation.Config

**Component:** *SeqRepresentation*

**class** SeqRepresentation.**Config**

**Bases:** RepresentationBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**doc_representation:** *DocNNRepresentation.Config* **=** *DocNNRepresentation.Config*()

**seq_representation: Union[***BiLSTMDocAttention.Config***,** *DocNNRepresentation.Config***] =** *BiLSTMDocAttention.Config*()

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "doc_representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "cnn": {
            "kernel_num": 100,
            "kernel_sizes": [
```

(continues on next page)

(continued from previous page)

```
                3,
                4
            ],
            "weight_norm": false,
            "dilated": false,
            "causal": false
        },
        "pooling": "max"
    },
    "seq_representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    }
}
```

### slot_attention

### SlotAttention.Config

**Component:** *SlotAttention*

**class** SlotAttention.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**attn_dimension: int = 64**

**attention_type: SlotAttentionType = <SlotAttentionType.NO_ATTENTION: 'no_attention'>**

**Default JSON**

```
{
    "attn_dimension": 64,
    "attention_type": "no_attention"
}
```

### sparse_transformer_sentence_encoder

### SparseTransformerSentenceEncoder.Config

**Component:** *SparseTransformerSentenceEncoder*

**class** SparseTransformerSentenceEncoder.**Config**

**Bases:** TransformerSentenceEncoder.Config, *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**output_dropout: float = 0.4**

**embedding_dim: int = 768**

**pooling: PoolingMethod = <PoolingMethod.CLS_TOKEN: 'cls_token'>**

**export: bool = False**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**dropout: float = 0.1**

**attention_dropout: float = 0.1**

**activation_dropout: float = 0.1**

**ffn_embedding_dim: int = 3072**

**num_encoder_layers: int = 6**

**num_attention_heads: int = 8**

**num_segments: int = 2**

**use_position_embeddings: bool = True**

**offset_positions_by_padding: bool = True**

**apply_bert_init: bool = True**

**encoder_normalize_before: bool = True**

**activation_fn: str = 'relu'**

**max_seq_len: int = 128**

**multilingual: bool = False**

**freeze_embeddings: bool = False**

**n_trans_layers_to_freeze: int = 0**

**use_torchscript: bool = `False`**

**use_bias_finetuning: bool = `False`**

**project_representation: bool = `False`**

**is_bidirectional: bool = `True`**

**stride: int = 32**

**expressivity: int = 8**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false,
    "dropout": 0.1,
    "attention_dropout": 0.1,
    "activation_dropout": 0.1,
    "ffn_embedding_dim": 3072,
    "num_encoder_layers": 6,
    "num_attention_heads": 8,
    "num_segments": 2,
    "use_position_embeddings": true,
    "offset_positions_by_padding": true,
    "apply_bert_init": true,
    "encoder_normalize_before": true,
    "activation_fn": "relu",
    "max_seq_len": 128,
    "multilingual": false,
    "freeze_embeddings": false,
    "n_trans_layers_to_freeze": 0,
    "use_torchscript": false,
    "use_bias_finetuning": false,
    "project_representation": false,
    "is_bidirectional": true,
    "stride": 32,
    "expressivity": 8
}
```

### stacked_bidirectional_rnn

### StackedBidirectionalRNN.Config

**Component:** *StackedBidirectionalRNN*

**class** StackedBidirectionalRNN.**Config**

**Bases:** Module.Config

Configuration class for *StackedBidirectionalRNN*.

**hidden_size**
Number of features in the hidden state of the RNN. Defaults to 32.

**Type** int

**num_layers**
Number of recurrent layers. Eg. setting *num_layers=2* would mean stacking two RNNs together to form a stacked RNN, with the second RNN taking in the outputs of the first RNN and computing the final result. Defaults to 1.

**Type** int

**dropout**
Dropout probability to use. Defaults to 0.4.

**Type** float

**bidirectional**
If *True*, becomes a bidirectional RNN. Defaults to *True*.

**Type** bool

**rnn_type**
Which RNN type to use. Options: "rnn", "lstm", "gru".

**Type** str

**concat_layers**
Whether to concatenate the outputs of each layer of stacked RNN.

**Type** bool

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**hidden_size: int = `32`**

**num_layers: int = `1`**

**dropout: float = `0.0`**

**bidirectional: bool = `True`**

**rnn_type: RnnType = `<RnnType.LSTM: 'lstm'>`**

**concat_layers: bool = `True`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "hidden_size": 32,
    "num_layers": 1,
    "dropout": 0.0,
```

(continues on next page)

(continued from previous page)

```
    "bidirectional": true,
    "rnn_type": "lstm",
    "concat_layers": true
}
```

### transformer

### representation

### TransformerRepresentation.Config

**Component:** *TransformerRepresentation*

**class** TransformerRepresentation.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**num_layers: int = 3**

**num_attention_heads: int = 4**

**ffnn_embed_dim: int = 32**

**dropout: float = 0.0**

**Default JSON**

```
{
    "num_layers": 3,
    "num_attention_heads": 4,
    "ffnn_embed_dim": 32,
    "dropout": 0.0
}
```

### transformer_sentence_encoder

### TransformerSentenceEncoder.Config

**Component:** *TransformerSentenceEncoder*

**class** TransformerSentenceEncoder.**Config**

**Bases:** TransformerSentenceEncoderBase.Config, *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**output_dropout: float = 0.4**

**embedding_dim: int = 768**

**pooling: PoolingMethod = <PoolingMethod.CLS_TOKEN: 'cls_token'>**

**export: bool = False**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**dropout: float = 0.1**

**attention_dropout: float = 0.1**

**activation_dropout: float = 0.1**

**ffn_embedding_dim: int = 3072**

**num_encoder_layers: int = 6**

**num_attention_heads: int = 8**

**num_segments: int = 2**

**use_position_embeddings: bool = True**

**offset_positions_by_padding: bool = True**

**apply_bert_init: bool = True**

**encoder_normalize_before: bool = True**

**activation_fn: str = 'relu'**

**max_seq_len: int = 128**

**multilingual: bool = False**

**freeze_embeddings: bool = False**

**n_trans_layers_to_freeze: int = 0**

**use_torchscript: bool = False**

**use_bias_finetuning: bool = False**

**Subclasses**

- `SparseTransformerSentenceEncoder.Config`

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false,
    "dropout": 0.1,
    "attention_dropout": 0.1,
    "activation_dropout": 0.1,
    "ffn_embedding_dim": 3072,
    "num_encoder_layers": 6,
    "num_attention_heads": 8,
```

(continues on next page)

(continued from previous page)

```
    "num_segments": 2,
    "use_position_embeddings": true,
    "offset_positions_by_padding": true,
    "apply_bert_init": true,
    "encoder_normalize_before": true,
    "activation_fn": "relu",
    "max_seq_len": 128,
    "multilingual": false,
    "freeze_embeddings": false,
    "n_trans_layers_to_freeze": 0,
    "use_torchscript": false,
    "use_bias_finetuning": false
}
```

### transformer_sentence_encoder_base

### TransformerSentenceEncoderBase.Config

**Component:** *TransformerSentenceEncoderBase*

**class** TransformerSentenceEncoderBase.**Config**

**Bases:** RepresentationBase.Config, *ConfigBase*

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**output_dropout: float = 0.4**

**embedding_dim: int = 768**

**pooling: PoolingMethod = <PoolingMethod.CLS_TOKEN: 'cls_token'>**

**export: bool = False**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**Subclasses**

- HuggingFaceBertSentenceEncoder.Config
- HuggingFaceElectraSentenceEncoder.Config
- SparseTransformerSentenceEncoder.Config
- TransformerSentenceEncoder.Config
- RoBERTaEncoder.Config
- RoBERTaEncoderBase.Config
- RoBERTaEncoderJit.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false
}
```

### roberta

### InputConfig

**class** pytext.models.roberta.**InputConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**tokens:** *RoBERTaTensorizer.Config* = *RoBERTaTensorizer.Config*()

**dense: Optional[***FloatListTensorizer.Config***]** = **None**

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "RoBERTaTensorizer": {
            "is_input": true,
            "columns": [
                "text"
            ],
            "tokenizer": {
                "GPT2BPETokenizer": {
                    "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                    "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/
↪bpe/gpt2/vocab.bpe",
                    "lowercase": false
                }
            },
            "base_tokenizer": null,
            "vocab_file": "gpt2_bpe_dict",
            "max_seq_len": 256,
            "add_selfie_token": false
        }
    },
    "dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
```

(continues on next page)

(continued from previous page)

```
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    }
}
```

### RegressionModelInput

**class** pytext.models.roberta.**RegressionModelInput**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**tokens:** *RoBERTaTensorizer.Config* = *RoBERTaTensorizer.Config*()

**labels:** *NumericLabelTensorizer.Config* = *NumericLabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "RoBERTaTensorizer": {
            "is_input": true,
            "columns": [
                "text"
            ],
            "tokenizer": {
                "GPT2BPETokenizer": {
                    "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                    "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/
↪bpe/gpt2/vocab.bpe",
                    "lowercase": false
                }
            },
            "base_tokenizer": null,
            "vocab_file": "gpt2_bpe_dict",
            "max_seq_len": 256,
            "add_selfie_token": false
        }
    },
    "labels": {
        "is_input": false,
        "column": "label",
        "rescale_range": null
    }
}
```

### RoBERTa.Config

**Component:** *RoBERTa*

**class** RoBERTa.**Config**

**Bases:** NewBertModel.Config

**All Attributes (including base classes)**

**inputs:** *InputConfig* = *InputConfig*()

**encoder:** *RoBERTaEncoderBase.Config* = *RoBERTaEncoderJit.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**Subclasses**

- RoBERTaR3F.Config
- SELFIE.Config

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "RoBERTaTensorizer": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "GPT2BPETokenizer": {
                        "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                        "bpe_vocab_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/vocab.bpe",
                        "lowercase": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "gpt2_bpe_dict",
                "max_seq_len": 256,
                "add_selfie_token": false
            }
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "encoder": {
        "RoBERTaEncoderJit": {
            "load_path": null,
            "save_path": null,
```

(continues on next page)

(continued from previous page)

```
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "pretrained_encoder": {
                "load_path": "public",
                "save_path": null,
                "freeze": false,
                "shared_module_key": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    }
}
```

### RoBERTaEncoder.Config

**Component:** *RoBERTaEncoder*

**class** RoBERTaEncoder.**Config**

**Bases:** RoBERTaEncoderBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**output_dropout: float = 0.4**

**embedding_dim: int = 768**

**pooling: PoolingMethod = <PoolingMethod.CLS_TOKEN: 'cls_token'>**

**export: bool = False**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**vocab_size: int = 50265**

**num_encoder_layers: int = 12**

**num_attention_heads: int = 12**

**model_path: str = 'manifold://pytext_training/tree/static/models/roberta_base_torch.pt'**

**is_finetuned: bool = False**

**max_seq_len: int = 514**

**use_bias_finetuning: bool = False**

**use_linformer_encoder: bool = False**

**linformer_compressed_ratio: int = 4**

**linformer_quantize: bool = False**

**export_encoder: bool = False**

**variable_size_embedding: bool = True**

**use_selfie_encoder: bool = False**

**transformer_layer_to_keep: Optional[int] = None**

**attention_heads_to_keep_per_layer: Optional[int] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false,
    "vocab_size": 50265,
    "num_encoder_layers": 12,
    "num_attention_heads": 12,
    "model_path": "manifold://pytext_training/tree/static/models/roberta_base_torch.pt
↪",
    "is_finetuned": false,
    "max_seq_len": 514,
    "use_bias_finetuning": false,
    "use_linformer_encoder": false,
    "linformer_compressed_ratio": 4,
```

(continues on next page)

(continued from previous page)

```
    "linformer_quantize": false,
    "export_encoder": false,
    "variable_size_embedding": true,
    "use_selfie_encoder": false,
    "transformer_layer_to_keep": null,
    "attention_heads_to_keep_per_layer": null
}
```

### RoBERTaEncoderBase.Config

**Component:** *RoBERTaEncoderBase*

**class** RoBERTaEncoderBase.**Config**

**Bases:** TransformerSentenceEncoderBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**output_dropout: float = 0.4**

**embedding_dim: int = 768**

**pooling: PoolingMethod = <PoolingMethod.CLS_TOKEN: 'cls_token'>**

**export: bool = False**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**Subclasses**

- RoBERTaEncoder.Config
- RoBERTaEncoderJit.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false
}
```

### RoBERTaEncoderJit.Config

**Component:** *RoBERTaEncoderJit*

**class** RoBERTaEncoderJit.**Config**

**Bases:** RoBERTaEncoderBase.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**output_dropout: float = 0.4**

**embedding_dim: int = 768**

**pooling: PoolingMethod = <PoolingMethod.CLS_TOKEN: 'cls_token'>**

**export: bool = False**

**projection_dim: int = 0**

**normalize_output_rep: bool = False**

**pretrained_encoder:** *Module.Config* = *Module.Config*(**load_path='public'**)

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "output_dropout": 0.4,
    "embedding_dim": 768,
    "pooling": "cls_token",
    "export": false,
    "projection_dim": 0,
    "normalize_output_rep": false,
    "pretrained_encoder": {
        "load_path": "public",
        "save_path": null,
        "freeze": false,
        "shared_module_key": null
    }
}
```

### RoBERTaR3F.Config

**Component:** *RoBERTaR3F*

**class** RoBERTaR3F.**Config**

**Bases:** RoBERTa.Config

**All Attributes (including base classes)**

**inputs:** *InputConfig* = *InputConfig*()

**encoder:** *RoBERTaEncoderBase.Config* = *RoBERTaEncoderJit.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**r3f_options:** *R3FConfigOptions* = *R3FConfigOptions*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "RoBERTaTensorizer": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "GPT2BPETokenizer": {
                        "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                        "bpe_vocab_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/vocab.bpe",
                        "lowercase": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "gpt2_bpe_dict",
                "max_seq_len": 256,
                "add_selfie_token": false
            }
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "encoder": {
        "RoBERTaEncoderJit": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "pretrained_encoder": {
```

(continues on next page)

(continued from previous page)

```
                "load_path": "public",
                "save_path": null,
                "freeze": false,
                "shared_module_key": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    },
    "r3f_options": {
        "r3f_lambda_by_loss": {},
        "r3f_default_lambda": 0.5,
        "eps": 1e-05,
        "noise_type": "uniform"
    }
}
```

### RoBERTaRegression.Config

**Component:** *RoBERTaRegression*

**class** RoBERTaRegression.**Config**

**Bases:** NewBertRegressionModel.Config

**All Attributes (including base classes)**

**inputs:** *RegressionModelInput* = *RegressionModelInput*()

**encoder:** *RoBERTaEncoderBase.Config* = *RoBERTaEncoderJit.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *RegressionOutputLayer.Config* = *RegressionOutputLayer.Config*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "RoBERTaTensorizer": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "GPT2BPETokenizer": {
                        "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                        "bpe_vocab_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/vocab.bpe",
                        "lowercase": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "gpt2_bpe_dict",
                "max_seq_len": 256,
                "add_selfie_token": false
            }
        },
        "labels": {
            "is_input": false,
            "column": "label",
            "rescale_range": null
        }
    },
    "encoder": {
        "RoBERTaEncoderJit": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "pretrained_encoder": {
                "load_path": "public",
                "save_path": null,
                "freeze": false,
                "shared_module_key": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
```

(continues on next page)

(continued from previous page)

```
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {},
        "squash_to_unit_range": false
    }
}
```

### RoBERTaWordTaggingModel.Config

**Component:** *RoBERTaWordTaggingModel*

**class** RoBERTaWordTaggingModel.**Config**

**Bases:** BaseModel.Config

**All Attributes (including base classes)**

**inputs:** *WordTaggingInputConfig* = *WordTaggingInputConfig*()

**encoder:** *RoBERTaEncoderBase.Config* = *RoBERTaEncoderJit.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *WordTaggingOutputLayer.Config* = *WordTaggingOutputLayer.Config*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "columns": [
                "text"
            ],
            "tokenizer": {
                "GPT2BPETokenizer": {
                    "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                    "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/
↪bpe/gpt2/vocab.bpe",
                    "lowercase": false
                }
            },
            "base_tokenizer": null,
            "vocab_file": "gpt2_bpe_dict",
            "max_seq_len": 256,
            "add_selfie_token": false,
            "labels_columns": [
                "label"
```

(continues on next page)

(continued from previous page)

```
            ],
            "labels": []
        }
    },
    "encoder": {
        "RoBERTaEncoderJit": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "pretrained_encoder": {
                "load_path": "public",
                "save_path": null,
                "freeze": false,
                "shared_module_key": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": {},
        "ignore_pad_in_loss": true
    }
}
```

### SELFIE.Config

**Component:** *SELFIE*

**class** SELFIE.**Config**

**Bases:** `RoBERTa.Config`

**All Attributes (including base classes)**

**inputs:** *InputConfig* = *InputConfig*()

**encoder:** *RoBERTaEncoderBase.Config* = *RoBERTaEncoderJit.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**use_selfie:** bool = `True`

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "RoBERTaTensorizer": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "GPT2BPETokenizer": {
                        "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                        "bpe_vocab_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/vocab.bpe",
                        "lowercase": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "gpt2_bpe_dict",
                "max_seq_len": 256,
                "add_selfie_token": false
            }
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "encoder": {
        "RoBERTaEncoderJit": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
```

(continues on next page)

(continued from previous page)

```
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "pretrained_encoder": {
                "load_path": "public",
                "save_path": null,
                "freeze": false,
                "shared_module_key": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    },
    "use_selfie": true
}
```

### WordTaggingInputConfig

**class** pytext.models.roberta.**WordTaggingInputConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**tokens:** *RoBERTaTokenLevelTensorizer.Config* = *RoBERTaTokenLevelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "columns": [
            "text"
        ],
        "tokenizer": {
```

(continues on next page)

(continued from previous page)

```
            "GPT2BPETokenizer": {
                "bpe_encoder_path": "manifold://pytext_training/tree/static/vocabs/
↪bpe/gpt2/encoder.json",
                "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/bpe/
↪gpt2/vocab.bpe",
                "lowercase": false
            }
        },
        "base_tokenizer": null,
        "vocab_file": "gpt2_bpe_dict",
        "max_seq_len": 256,
        "add_selfie_token": false,
        "labels_columns": [
            "label"
        ],
        "labels": []
    }
}
```

### semantic_parsers

### rnng

### rnng_parser

### AblationParams

**class** pytext.models.semantic_parsers.rnng.rnng_parser.**AblationParams**

**Bases:** *ConfigBase*

Ablation parameters.

**use_buffer**

whether to use the buffer LSTM

**Type** bool

**use_stack**

whether to use the stack LSTM

**Type** bool

**use_action**

whether to use the action LSTM

**Type** bool

**use_last_open_NT_feature**

whether to use the last open non-terminal as a 1-hot feature when computing representation for the action classifier

**Type** bool

**All Attributes (including base classes)**

**use_buffer: bool = True**

**use_stack: bool = True**

**use_action: bool = `True`**

**use_last_open_NT_feature: bool = `False`**

**Default JSON**

```
{
    "use_buffer": true,
    "use_stack": true,
    "use_action": true,
    "use_last_open_NT_feature": false
}
```

### ModelInput

**class** `pytext.models.semantic_parsers.rnng.rnng_parser.`**`ModelInput`**

**Bases:** `ModelInput`

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*(**column='tokenized_text'**)

**actions:** *AnnotationNumberizer.Config* = *AnnotationNumberizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "tokenized_text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "actions": {
        "is_input": true,
        "column": "seqlogical"
    }
}
```

### RNNGConstraints

**class** pytext.models.semantic_parsers.rnng.rnng_parser.**RNNGConstraints**

**Bases:** *ConfigBase*

Constraints when computing valid actions.

**intent_slot_nesting**

for the intent slot models, the top level non-terminal has to be an intent, an intent can only have slot non-terminals as children and vice-versa.

**Type** bool

**ignore_loss_for_unsupported**

if the data has "unsupported" label, that is if the label has a substring "unsupported" in it, do not compute loss

**Type** bool

**no_slots_inside_unsupported**

if the data has "unsupported" label, that is if the label has a substring "unsupported" in it, do not predict slots inside this label.

**Type** bool

**All Attributes (including base classes)**

**intent_slot_nesting: bool = `True`**

**ignore_loss_for_unsupported: bool = `False`**

**no_slots_inside_unsupported: bool = `True`**

**Default JSON**

```
{
    "intent_slot_nesting": true,
    "ignore_loss_for_unsupported": false,
    "no_slots_inside_unsupported": true
}
```

### RNNGParser.Config

**Component:** *RNNGParser*

**class** RNNGParser.**Config**

**Bases:** RNNGParserBase.Config

**All Attributes (including base classes)**

**version: int = `2`**

**lstm: *BiLSTM.Config* = *BiLSTM.Config*()**

**ablation: *AblationParams* = *AblationParams*()**

**constraints: *RNNGConstraints* = *RNNGConstraints*()**

**max_open_NT: int = `10`**

**dropout: float = `0.1`**

**beam_size: int = `1`**

**top_k: int = 1**

**compositional_type: CompositionalType = <CompositionalType.BLSTM: 'blstm'>**

**inputs:** *ModelInput* **=** *ModelInput*()

**embedding:** *WordEmbedding.Config* **=** *WordEmbedding.Config*()

**Default JSON**

```
{
    "version": 2,
    "lstm": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true,
        "pack_sequence": true,
        "disable_sort_in_jit": false
    },
    "ablation": {
        "use_buffer": true,
        "use_stack": true,
        "use_action": true,
        "use_last_open_NT_feature": false
    },
    "constraints": {
        "intent_slot_nesting": true,
        "ignore_loss_for_unsupported": false,
        "no_slots_inside_unsupported": true
    },
    "max_open_NT": 10,
    "dropout": 0.1,
    "beam_size": 1,
    "top_k": 1,
    "compositional_type": "blstm",
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "tokenized_text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
```

(continues on next page)

(continued from previous page)

```
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "actions": {
            "is_input": true,
            "column": "seqlogical"
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    }
}
```

### RNNGParserBase.Config

**Component:** *RNNGParserBase*

**class** RNNGParserBase.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**version: int = 2**

**lstm: *BiLSTM.Config* = *BiLSTM.Config*()**

**ablation: *AblationParams* = *AblationParams*()**

**constraints: *RNNGConstraints* = *RNNGConstraints*()**

**max_open_NT: int = 10**

**dropout: float = 0.1**

**beam_size: int = 1**

**top_k: int = 1**

**compositional_type: CompositionalType = <CompositionalType.BLSTM: 'blstm'>**

**Subclasses**

- RNNGParser.Config

**Default JSON**

```
{
    "version": 2,
    "lstm": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true,
        "pack_sequence": true,
        "disable_sort_in_jit": false
    },
    "ablation": {
        "use_buffer": true,
        "use_stack": true,
        "use_action": true,
        "use_last_open_NT_feature": false
    },
    "constraints": {
        "intent_slot_nesting": true,
        "ignore_loss_for_unsupported": false,
        "no_slots_inside_unsupported": true
    },
    "max_open_NT": 10,
    "dropout": 0.1,
    "beam_size": 1,
    "top_k": 1,
    "compositional_type": "blstm"
}
```

### seq_models

### attention

### MultiheadAttention.Config

**Component:** *MultiheadAttention*

**class** MultiheadAttention.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**dropout: float = 0.0**

**kdim: Optional[int] = None**

**vdim: Optional[int] = None**

**bias: bool = True**

**Default JSON**

```
{
    "dropout": 0.0,
    "kdim": null,
    "vdim": null,
    "bias": true
}
```

### base

#### PyTextIncrementalDecoderComponent.Config

**Component:** *PyTextIncrementalDecoderComponent*

**class** PyTextIncrementalDecoderComponent.**Config**

**Bases:** PyTextSeq2SeqModule.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

#### PyTextSeq2SeqModule.Config

**Component:** *PyTextSeq2SeqModule*

**class** PyTextSeq2SeqModule.**Config**

**Bases:** Module.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = None**

**save_path: Optional[str] = None**

**freeze: bool = False**

**shared_module_key: Optional[str] = None**

**Subclasses**

- PyTextIncrementalDecoderComponent.Config

- DecoderWithLinearOutputProjection.Config

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### contextual_intent_slot

### ContextualIntentSlotModel.Config

**Component:** *ContextualIntentSlotModel*

**class** ContextualIntentSlotModel.**Config**

**Bases:** IntentSlotModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**word_embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** *ContextualIntentSlotRepresentation.Config* = *ContextualIntentSlotRepresentation.Config*()

**output_layer:** *IntentSlotOutputLayer.Config* = *IntentSlotOutputLayer.Config*()

**decoder:** *IntentSlotModelDecoder.Config* = *IntentSlotModelDecoder.Config*()

**default_doc_loss_weight:** float = 0.2

**default_word_loss_weight:** float = 0.5

**seq_embedding:** Optional[*WordEmbedding.Config*] = *WordEmbedding.Config*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
```

(continues on next page)

(continued from previous page)

```
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "word_labels": {
            "is_input": false,
            "slot_column": "slots",
            "text_column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "allow_unknown": true
        },
        "doc_labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": true,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        },
        "doc_weight": null,
        "word_weight": null,
        "seq_tokens": {
            "is_input": true,
            "column": "text_seq",
            "max_seq_len": null,
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "add_bol_token": false,
            "add_eol_token": false,
            "use_eol_token_for_bol": false,
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "max_turn": 50
        }
    },
    "word_embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
```

(continues on next page)

(continued from previous page)

```
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "sen_representation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "cnn": {
                "kernel_num": 100,
                "kernel_sizes": [
                    3,
                    4
                ],
                "weight_norm": false,
                "dilated": false,
                "causal": false
            },
            "pooling": "max"
        },
        "seq_representation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "cnn": {
                "kernel_num": 100,
                "kernel_sizes": [
                    3,
                    4
                ],
                "weight_norm": false,
                "dilated": false,
```

(continues on next page)

(continued from previous page)

```
                "causal": false
            },
            "pooling": "max"
        },
        "joint_representation": {
            "BiLSTMDocSlotAttention": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "BiLSTM": {
                        "load_path": null,
                        "save_path": null,
                        "freeze": false,
                        "shared_module_key": null,
                        "dropout": 0.4,
                        "lstm_dim": 32,
                        "num_layers": 1,
                        "bidirectional": true,
                        "pack_sequence": true,
                        "disable_sort_in_jit": false
                    }
                },
                "pooling": null,
                "slot_attention": null,
                "doc_mlp_layers": 0,
                "word_mlp_layers": 0
            }
        }
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "doc_output": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        },
        "word_output": {
            "WordTaggingOutputLayer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {
                    "CrossEntropyLoss": {}
                },
                "label_weights": {},
```

(continues on next page)

(continued from previous page)

```
                "ignore_pad_in_loss": true
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "use_doc_probs_in_word": false,
        "doc_decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "word_decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        }
    },
    "default_doc_loss_weight": 0.2,
    "default_word_loss_weight": 0.5,
    "seq_embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
```

(continues on next page)

(continued from previous page)

```
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    }
}
```

### ModelInput

**class** pytext.models.seq_models.contextual_intent_slot.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**word_labels:** *SlotLabelTensorizer.Config* = *SlotLabelTensorizer.Config*(allow_unknown=`True`)

**doc_labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*(allow_unknown=`True`)

**doc_weight:** Optional[*FloatTensorizer.Config*] = `None`

**word_weight:** Optional[*FloatTensorizer.Config*] = `None`

**seq_tokens:** Optional[*SeqTokenTensorizer.Config*] = *SeqTokenTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "word_labels": {
        "is_input": false,
```

(continues on next page)

(continued from previous page)

```
        "slot_column": "slots",
        "text_column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "allow_unknown": true
    },
    "doc_labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": true,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    },
    "doc_weight": null,
    "word_weight": null,
    "seq_tokens": {
        "is_input": true,
        "column": "text_seq",
        "max_seq_len": null,
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "add_bol_token": false,
        "add_eol_token": false,
        "use_eol_token_for_bol": false,
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "max_turn": 50
    }
}
```

### rnn_decoder

### DecoderWithLinearOutputProjection.Config

**Component:** *DecoderWithLinearOutputProjection*

**class** DecoderWithLinearOutputProjection.**Config**

**Bases:** PyTextSeq2SeqModule.Config

**All Attributes (including base classes)**

**load_path: Optional[str] = `None`**

**save_path: Optional[str] = `None`**

**freeze: bool = `False`**

**shared_module_key: Optional[str] = `None`**

**Default JSON**

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null
}
```

### RNNDecoder.Config

**Component:** *RNNDecoder*

**class** RNNDecoder.**Config**

**Bases:** RNNDecoderBase.Config

**All Attributes (including base classes)**

**encoder_hidden_dim: int = `512`**

**embed_dim: int = `512`**

**hidden_dim: int = `512`**

**out_embed_dim: int = `512`**

**cell_type: str = `'lstm'`**

**num_layers: int = `1`**

**dropout_in: float = `0.1`**

**dropout_out: float = `0.1`**

**attention_type: str = `'dot'`**

**attention_heads: int = `8`**

**first_layer_attention: bool = `False`**

**averaging_encoder: bool = `False`**

**Default JSON**

```
{
    "encoder_hidden_dim": 512,
    "embed_dim": 512,
    "hidden_dim": 512,
    "out_embed_dim": 512,
    "cell_type": "lstm",
    "num_layers": 1,
    "dropout_in": 0.1,
    "dropout_out": 0.1,
    "attention_type": "dot",
    "attention_heads": 8,
```

(continues on next page)

(continued from previous page)

```
    "first_layer_attention": false,
    "averaging_encoder": false
}
```

### RNNDecoderBase.Config

**Component:** *RNNDecoderBase*

**class** RNNDecoderBase.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**encoder_hidden_dim: int = 512**

**embed_dim: int = 512**

**hidden_dim: int = 512**

**out_embed_dim: int = 512**

**cell_type: str = 'lstm'**

**num_layers: int = 1**

**dropout_in: float = 0.1**

**dropout_out: float = 0.1**

**attention_type: str = 'dot'**

**attention_heads: int = 8**

**first_layer_attention: bool = False**

**averaging_encoder: bool = False**

**Subclasses**

- RNNDecoder.Config

**Default JSON**

```
{
    "encoder_hidden_dim": 512,
    "embed_dim": 512,
    "hidden_dim": 512,
    "out_embed_dim": 512,
    "cell_type": "lstm",
    "num_layers": 1,
    "dropout_in": 0.1,
    "dropout_out": 0.1,
    "attention_type": "dot",
    "attention_heads": 8,
    "first_layer_attention": false,
    "averaging_encoder": false
}
```

### rnn_encoder

#### LSTMSequenceEncoder.Config

**Component:** *LSTMSequenceEncoder*

**class** LSTMSequenceEncoder.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**embed_dim: int = 512**

**hidden_dim: int = 512**

**num_layers: int = 1**

**dropout_in: float = 0.1**

**dropout_out: float = 0.1**

**bidirectional: bool = False**

**Default JSON**

```
{
    "embed_dim": 512,
    "hidden_dim": 512,
    "num_layers": 1,
    "dropout_in": 0.1,
    "dropout_out": 0.1,
    "bidirectional": false
}
```

### rnn_encoder_decoder

#### RNNModel.Config

**Component:** *RNNModel*

**class** RNNModel.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**encoder:** *LSTMSequenceEncoder.Config* = *LSTMSequenceEncoder.Config*()

**decoder:** *RNNDecoder.Config* = *RNNDecoder.Config*()

**Default JSON**

```
{
    "encoder": {
        "embed_dim": 512,
        "hidden_dim": 512,
        "num_layers": 1,
        "dropout_in": 0.1,
        "dropout_out": 0.1,
        "bidirectional": false
```

(continues on next page)

(continued from previous page)

```
    },
    "decoder": {
        "encoder_hidden_dim": 512,
        "embed_dim": 512,
        "hidden_dim": 512,
        "out_embed_dim": 512,
        "cell_type": "lstm",
        "num_layers": 1,
        "dropout_in": 0.1,
        "dropout_out": 0.1,
        "attention_type": "dot",
        "attention_heads": 8,
        "first_layer_attention": false,
        "averaging_encoder": false
    }
}
```

### seq2seq_model

#### ModelInput

**class** pytext.models.seq_models.seq2seq_model.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**src_seq_tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**trg_seq_tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**dict_feat:** Optional[*GazetteerTensorizer.Config*] = **None**

**contextual_token_embedding:** Optional[*ByteTokenTensorizer.Config*] = **None**

**Default JSON**

```
{
    "src_seq_tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
```

(continues on next page)

(continued from previous page)

```
        },
        "vocab_file_delimiter": " "
    },
    "trg_seq_tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "dict_feat": null,
    "contextual_token_embedding": null
}
```

### Seq2SeqModel.Config

**Component:** *Seq2SeqModel*

**class** Seq2SeqModel.**Config**

**Bases:** Model.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**encoder_decoder:** *RNNModel.Config* = *RNNModel.Config*()

**source_embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**target_embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**dict_embedding:** Optional[*DictEmbedding.Config*] = **None**

**contextual_token_embedding:** Optional[*ContextualTokenEmbedding.Config*] = **None**

**output_layer:** *Seq2SeqOutputLayer.Config* = *Seq2SeqOutputLayer.Config*()

**sequence_generator:** *ScriptedSequenceGenerator.Config* = *ScriptedSequenceGenerator.Config*()

**Default JSON**

```
{
    "inputs": {
        "src_seq_tokens": {
```

(continues on next page)

(continued from previous page)

```
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "trg_seq_tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "dict_feat": null,
        "contextual_token_embedding": null
    },
    "encoder_decoder": {
        "encoder": {
            "embed_dim": 512,
            "hidden_dim": 512,
            "num_layers": 1,
            "dropout_in": 0.1,
            "dropout_out": 0.1,
            "bidirectional": false
        },
        "decoder": {
            "encoder_hidden_dim": 512,
```

(continues on next page)

(continued from previous page)

```
            "embed_dim": 512,
            "hidden_dim": 512,
            "out_embed_dim": 512,
            "cell_type": "lstm",
            "num_layers": 1,
            "dropout_in": 0.1,
            "dropout_out": 0.1,
            "attention_type": "dot",
            "attention_heads": 8,
            "first_layer_attention": false,
            "averaging_encoder": false
        }
    },
    "source_embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "target_embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
```

(continues on next page)

(continued from previous page)

```
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "dict_embedding": null,
    "contextual_token_embedding": null,
    "output_layer": {
        "loss": {
            "CrossEntropyLoss": {}
        }
    },
    "sequence_generator": {
        "beam_size": 2,
        "targetlen_cap": 100,
        "targetlen_a": 0,
        "targetlen_b": 2,
        "targetlen_c": 2,
        "quantize": true,
        "length_penalty": 0.25,
        "nbest": 2,
        "stop_at_eos": true,
        "record_attention": false
    }
}
```

### seq2seq_output_layer

### Seq2SeqOutputLayer.Config

**Component:** *Seq2SeqOutputLayer*

**class** Seq2SeqOutputLayer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**loss: Union[*CrossEntropyLoss.Config*, *LabelSmoothedCrossEntropyLoss.Config*, *NLLLoss.Config*] = *CrossEntropyLoss.Conf***

**Default JSON**

```
{
    "loss": {
        "CrossEntropyLoss": {}
    }
}
```

### seqnn

### ModelInput

**class** pytext.models.seq_models.seqnn.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *SeqTokenTensorizer.Config* = *SeqTokenTensorizer.Config*()

**dense:** Optional[*FloatListTensorizer.Config*] = **None**

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text_seq",
        "max_seq_len": null,
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "add_bol_token": false,
        "add_eol_token": false,
        "use_eol_token_for_bol": false,
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "max_turn": 50
    },
    "dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    }
}
```

### SeqNNModel.Config

**Component:** *SeqNNModel*

**class** SeqNNModel.**Config**

**Bases:** DocModel.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** *SeqRepresentation.Config* = *SeqRepresentation.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text_seq",
            "max_seq_len": null,
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "add_bol_token": false,
            "add_eol_token": false,
            "use_eol_token_for_bol": false,
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "max_turn": 50
        },
        "dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
```

(continues on next page)

(continued from previous page)

```
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "doc_representation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "cnn": {
                "kernel_num": 100,
                "kernel_sizes": [
                    3,
                    4
                ],
                "weight_norm": false,
                "dilated": false,
                "causal": false
            },
            "pooling": "max"
        },
        "seq_representation": {
            "BiLSTMDocAttention": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
                    "num_layers": 1,
                    "bidirectional": true,
                    "pack_sequence": true,
                    "disable_sort_in_jit": false
                },
                "pooling": {
                    "SelfAttention": {
                        "attn_dimension": 64,
                        "dropout": 0.4
```

(continues on next page)

(continued from previous page)

```
                    }
                },
                "mlp_decoder": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    }
}
```

### SeqNNModel_Deprecated.Config

**Component:** *SeqNNModel_Deprecated*

**class** SeqNNModel_Deprecated.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**representation:** *SeqRepresentation.Config* = *SeqRepresentation.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**Default JSON**

```
{
    "representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "doc_representation": {
            "load_path": null,
```

(continues on next page)

(continued from previous page)

```
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "cnn": {
                "kernel_num": 100,
                "kernel_sizes": [
                    3,
                    4
                ],
                "weight_norm": false,
                "dilated": false,
                "causal": false
            },
            "pooling": "max"
        },
        "seq_representation": {
            "BiLSTMDocAttention": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
                    "num_layers": 1,
                    "bidirectional": true,
                    "pack_sequence": true,
                    "disable_sort_in_jit": false
                },
                "pooling": {
                    "SelfAttention": {
                        "attn_dimension": 64,
                        "dropout": 0.4
                    }
                },
                "mlp_decoder": null
            }
        }
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    },
    "decoder": {
        "load_path": null,
```

(continues on next page)

(continued from previous page)

```
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    }
}
```

### two_tower_classification_model

#### InputConfig

**class** pytext.models.two_tower_classification_model.**InputConfig**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**right_tokens:** *RoBERTaTensorizer.Config* = *RoBERTaTensorizer.Config*()

**left_tokens:** *RoBERTaTensorizer.Config* = *RoBERTaTensorizer.Config*()

**right_dense:** Optional[*FloatListTensorizer.Config*] = `None`

**left_dense:** Optional[*FloatListTensorizer.Config*] = `None`

**labels:** *LabelTensorizer.Config* = *LabelTensorizer.Config*()

**Default JSON**

```
{
    "right_tokens": {
        "RoBERTaTensorizer": {
            "is_input": true,
            "columns": [
                "text"
            ],
            "tokenizer": {
                "GPT2BPETokenizer": {
                    "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                    "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/
↪bpe/gpt2/vocab.bpe",
                    "lowercase": false
                }
            },
            "base_tokenizer": null,
            "vocab_file": "gpt2_bpe_dict",
            "max_seq_len": 256,
            "add_selfie_token": false
        }
    },
```

(continues on next page)

(continued from previous page)

```
    "left_tokens": {
        "RoBERTaTensorizer": {
            "is_input": true,
            "columns": [
                "text"
            ],
            "tokenizer": {
                "GPT2BPETokenizer": {
                    "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                    "bpe_vocab_path": "manifold://pytext_training/tree/static/vocabs/
↪bpe/gpt2/vocab.bpe",
                    "lowercase": false
                }
            },
            "base_tokenizer": null,
            "vocab_file": "gpt2_bpe_dict",
            "max_seq_len": 256,
            "add_selfie_token": false
        }
    },
    "right_dense": null,
    "left_dense": null,
    "labels": {
        "LabelTensorizer": {
            "is_input": false,
            "column": "label",
            "allow_unknown": false,
            "pad_in_vocab": false,
            "label_vocab": null,
            "label_vocab_file": null,
            "add_labels": null
        }
    }
}
```

### TwoTowerClassificationModel.Config

**Component:** *TwoTowerClassificationModel*

**class** TwoTowerClassificationModel.**Config**

**Bases:** BaseModel.Config

**All Attributes (including base classes)**

**inputs:** *InputConfig* = *InputConfig*()

**right_encoder:** *RoBERTaEncoderBase.Config* = *RoBERTaEncoder.Config*()

**left_encoder:** *RoBERTaEncoderBase.Config* = *RoBERTaEncoder.Config*()

**decoder:** *MLPDecoderTwoTower.Config* = *MLPDecoderTwoTower.Config*()

**output_layer:** *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

**Default JSON**

```
{
    "inputs": {
        "right_tokens": {
            "RoBERTaTensorizer": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "GPT2BPETokenizer": {
                        "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                        "bpe_vocab_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/vocab.bpe",
                        "lowercase": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "gpt2_bpe_dict",
                "max_seq_len": 256,
                "add_selfie_token": false
            }
        },
        "left_tokens": {
            "RoBERTaTensorizer": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "GPT2BPETokenizer": {
                        "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                        "bpe_vocab_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/vocab.bpe",
                        "lowercase": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "gpt2_bpe_dict",
                "max_seq_len": 256,
                "add_selfie_token": false
            }
        },
        "right_dense": null,
        "left_dense": null,
        "labels": {
            "LabelTensorizer": {
                "is_input": false,
                "column": "label",
                "allow_unknown": false,
                "pad_in_vocab": false,
                "label_vocab": null,
                "label_vocab_file": null,
                "add_labels": null
            }
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "right_encoder": {
        "RoBERTaEncoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "vocab_size": 50265,
            "num_encoder_layers": 12,
            "num_attention_heads": 12,
            "model_path": "manifold://pytext_training/tree/static/models/roberta_base_
↪torch.pt",
            "is_finetuned": false,
            "max_seq_len": 514,
            "use_bias_finetuning": false,
            "use_linformer_encoder": false,
            "linformer_compressed_ratio": 4,
            "linformer_quantize": false,
            "export_encoder": false,
            "variable_size_embedding": true,
            "use_selfie_encoder": false,
            "transformer_layer_to_keep": null,
            "attention_heads_to_keep_per_layer": null
        }
    },
    "left_encoder": {
        "RoBERTaEncoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "output_dropout": 0.4,
            "embedding_dim": 768,
            "pooling": "cls_token",
            "export": false,
            "projection_dim": 0,
            "normalize_output_rep": false,
            "vocab_size": 50265,
            "num_encoder_layers": 12,
            "num_attention_heads": 12,
            "model_path": "manifold://pytext_training/tree/static/models/roberta_base_
↪torch.pt",
            "is_finetuned": false,
            "max_seq_len": 514,
            "use_bias_finetuning": false,
            "use_linformer_encoder": false,
            "linformer_compressed_ratio": 4,
            "linformer_quantize": false,
            "export_encoder": false,
            "variable_size_embedding": true,
            "use_selfie_encoder": false,
```

(continues on next page)

(continued from previous page)

```
            "transformer_layer_to_keep": null,
            "attention_heads_to_keep_per_layer": null
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "right_hidden_dims": [],
        "left_hidden_dims": [],
        "hidden_dims": [],
        "layer_norm": false,
        "dropout": 0.0
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        },
        "label_weights": null
    }
}
```

### word_model

### ByteModelInput

**class** pytext.models.word_model.**ByteModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**token_bytes:** *ByteTokenTensorizer.Config* = *ByteTokenTensorizer.Config*()

**labels:** *SlotLabelTensorizer.Config* = *SlotLabelTensorizer.Config*()

**Default JSON**

```
{
    "token_bytes": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "max_seq_len": null,
        "max_byte_len": 15,
        "offset_for_non_padding": 0,
```

(continues on next page)

(continued from previous page)

```
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false
    },
    "labels": {
        "is_input": false,
        "slot_column": "slots",
        "text_column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "allow_unknown": false
    }
}
```

### ModelInput

**class** pytext.models.word_model.**ModelInput**

**Bases:** ModelInput

**All Attributes (including base classes)**

**tokens:** *TokenTensorizer.Config* = *TokenTensorizer.Config*()

**labels:** *SlotLabelTensorizer.Config* = *SlotLabelTensorizer.Config*()

**Default JSON**

```
{
    "tokens": {
        "is_input": true,
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null,
        "vocab": {
            "build_from_data": true,
            "size_from_data": 0,
            "min_counts": 0,
            "vocab_files": []
        },
        "vocab_file_delimiter": " "
    },
    "labels": {
```

(continues on next page)

(continued from previous page)

```
        "is_input": false,
        "slot_column": "slots",
        "text_column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true,
                "use_byte_offsets": false
            }
        },
        "allow_unknown": false
    }
}
```

### WordTaggingLiteModel.Config

**Component:** *WordTaggingLiteModel*

**class** WordTaggingLiteModel.**Config**

**Bases:** WordTaggingModel.Config

**All Attributes (including base classes)**

**inputs:** *ByteModelInput* = *ByteModelInput*()

**embedding:** *CharacterEmbedding.Config* = *CharacterEmbedding.Config*()

**representation:** Union[*BiLSTMSlotAttention.Config*, *BSeqCNNRepresentation.Config*, *PassThroughRepresentation.Config*,

**output_layer:** Union[*WordTaggingOutputLayer.Config*, *CRFOutputLayer.Config*] = *WordTaggingOutputLayer.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**Default JSON**

```
{
    "inputs": {
        "token_bytes": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "max_seq_len": null,
            "max_byte_len": 15,
            "offset_for_non_padding": 0,
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false
        },
        "labels": {
```

(continues on next page)

(continued from previous page)

```
            "is_input": false,
            "slot_column": "slots",
            "text_column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "allow_unknown": false
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "sparse": false,
        "cnn": {
            "kernel_num": 100,
            "kernel_sizes": [
                3,
                4
            ],
            "weight_norm": false,
            "dilated": false,
            "causal": false
        },
        "highway_layers": 0,
        "projection_dim": null,
        "export_input_names": [
            "char_vals"
        ],
        "vocab_from_train_data": true,
        "max_word_length": 20,
        "min_freq": 1
    },
    "representation": {
        "PassThroughRepresentation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null
        }
    },
    "output_layer": {
        "WordTaggingOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": {},
```

(continues on next page)

(continued from previous page)

```
            "ignore_pad_in_loss": true
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    }
}
```

### WordTaggingModel.Config

**Component:** *WordTaggingModel*

**class** WordTaggingModel.**Config**

**Bases:** Model.Config

**All Attributes (including base classes)**

**inputs:** *ModelInput* = *ModelInput*()

**embedding:** *WordEmbedding.Config* = *WordEmbedding.Config*()

**representation:** Union[*BiLSTMSlotAttention.Config*, *BSeqCNNRepresentation.Config*, *PassThroughRepresentation.Config*,

**output_layer:** Union[*WordTaggingOutputLayer.Config*, *CRFOutputLayer.Config*] = *WordTaggingOutputLayer.Config*()

**decoder:** *MLPDecoder.Config* = *MLPDecoder.Config*()

**Subclasses**

- WordTaggingLiteModel.Config

**Default JSON**

```
{
    "inputs": {
        "tokens": {
            "is_input": true,
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
```

(continues on next page)

(continued from previous page)

```
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null,
            "vocab": {
                "build_from_data": true,
                "size_from_data": 0,
                "min_counts": 0,
                "vocab_files": []
            },
            "vocab_file_delimiter": " "
        },
        "labels": {
            "is_input": false,
            "slot_column": "slots",
            "text_column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true,
                    "use_byte_offsets": false
                }
            },
            "allow_unknown": false
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "embeddding_init_std": 0.02,
        "export_input_names": [
            "tokens_vals"
        ],
        "pretrained_embeddings_path": "",
        "vocab_file": "",
        "vocab_size": 0,
        "vocab_from_train_data": true,
        "vocab_from_all_data": false,
        "vocab_from_pretrained_embeddings": false,
        "lowercase_tokens": true,
        "min_freq": 1,
        "mlp_layer_dims": [],
        "padding_idx": null,
        "cpu_only": false,
        "skip_header": true,
        "delimiter": " "
    },
    "representation": {
        "PassThroughRepresentation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
```

(continues on next page)

(continued from previous page)

```
            "shared_module_key": null
        }
    },
    "output_layer": {
        "WordTaggingOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": {},
            "ignore_pad_in_loss": true
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    }
}
```

### 1.21.7 optimizer

#### adabelief

#### AdaBelief.Config

**Component:** *AdaBelief*

**class** AdaBelief.**Config**

**Bases:** `Optimizer.Config`

**All Attributes (including base classes)**

**lr: float =** `0.001`

**beta_1: float =** `0.9`

**beta_2: float =** `0.999`

**eps: float =** `1e-08`

**weight_decay: float =** `0`

**amsgrad: bool =** `False`

**weight_decouple: bool =** `True`

**fixed_decay: bool = `True`**

**rectify: bool = `False`**

**Default JSON**

```
{
    "lr": 0.001,
    "beta_1": 0.9,
    "beta_2": 0.999,
    "eps": 1e-08,
    "weight_decay": 0,
    "amsgrad": false,
    "weight_decouple": true,
    "fixed_decay": true,
    "rectify": false
}
```

### fp16_optimizer

#### FP16Optimizer.Config

**Component:** *FP16Optimizer*

**class** FP16Optimizer.**Config**

**Bases:** `Optimizer.Config`

**All Attributes (including base classes)**

**Subclasses**

- `FP16OptimizerApex.Config`
- `FP16OptimizerFairseq.Config`
- `MemoryEfficientFP16OptimizerFairseq.Config`

**Default JSON**

```
{}
```

#### FP16OptimizerApex.Config

**Component:** *FP16OptimizerApex*

**class** FP16OptimizerApex.**Config**

**Bases:** `FP16Optimizer.Config`

**All Attributes (including base classes)**

**opt_level: str = `'O2'`**

**init_loss_scale: Optional[int] = `None`**

**min_loss_scale: Optional[float] = `None`**

**Default JSON**

```
{
    "opt_level": "O2",
    "init_loss_scale": null,
    "min_loss_scale": null
}
```

### FP16OptimizerFairseq.Config

**Component:** *FP16OptimizerFairseq*

**class** FP16OptimizerFairseq.**Config**

**Bases:** FP16Optimizer.Config

**All Attributes (including base classes)**

**init_loss_scale: int = 128**

**scale_window: Optional[int] = None**

**scale_tolerance: float = 0.0**

**threshold_loss_scale: Optional[float] = None**

**min_loss_scale: float = 0.0001**

**Default JSON**

```
{
    "init_loss_scale": 128,
    "scale_window": null,
    "scale_tolerance": 0.0,
    "threshold_loss_scale": null,
    "min_loss_scale": 0.0001
}
```

### MemoryEfficientFP16OptimizerFairseq.Config

**Component:** *MemoryEfficientFP16OptimizerFairseq*

**class** MemoryEfficientFP16OptimizerFairseq.**Config**

**Bases:** FP16Optimizer.Config

**All Attributes (including base classes)**

**init_loss_scale: int = 128**

**scale_window: Optional[int] = None**

**scale_tolerance: float = 0.0**

**threshold_loss_scale: Optional[float] = None**

**min_loss_scale: float = 0.0001**

**Default JSON**

```
{
    "init_loss_scale": 128,
    "scale_window": null,
```

(continues on next page)

(continued from previous page)

```
    "scale_tolerance": 0.0,
    "threshold_loss_scale": null,
    "min_loss_scale": 0.0001
}
```

### lamb

#### Lamb.Config

**Component:** *Lamb*

**class** Lamb.**Config**

**Bases:** Optimizer.Config

**All Attributes (including base classes)**

**lr: float = 0.001**

**weight_decay: float = 1e-05**

**eps: float = 1e-08**

**min_trust: Optional[float] = None**

**Default JSON**

```
{
    "lr": 0.001,
    "weight_decay": 1e-05,
    "eps": 1e-08,
    "min_trust": null
}
```

### madgrad

#### MADGRAD.Config

**Component:** *MADGRAD*

**class** MADGRAD.**Config**

**Bases:** Optimizer.Config

**All Attributes (including base classes)**

**lr: float = 0.001**

**eps: float = 1e-06**

**momentum: float = 0.9**

**weight_decay: float = 0.0**

**Default JSON**

```
{
    "lr": 0.001,
    "eps": 1e-06,
```

(continues on next page)

(continued from previous page)

```
    "momentum": 0.9,
    "weight_decay": 0.0
}
```

### optimizers

#### Adagrad.Config

**Component:** *Adagrad*

**class** Adagrad.**Config**

**Bases:** Optimizer.Config

**All Attributes (including base classes)**

**lr: float = 0.01**

**weight_decay: float = 1e-05**

**Default JSON**

```
{
    "lr": 0.01,
    "weight_decay": 1e-05
}
```

#### Adam.Config

**Component:** *Adam*

**class** Adam.**Config**

**Bases:** Optimizer.Config

**All Attributes (including base classes)**

**lr: float = 0.001**

**weight_decay: float = 1e-05**

**eps: float = 1e-08**

**Default JSON**

```
{
    "lr": 0.001,
    "weight_decay": 1e-05,
    "eps": 1e-08
}
```

#### AdamW.Config

**Component:** *AdamW*

**class** AdamW.**Config**

**Bases:** Optimizer.Config

**All Attributes (including base classes)**

**lr: float = `0.001`**

**weight_decay: float = `0.01`**

**eps: float = `1e-08`**

**Default JSON**

```
{
    "lr": 0.001,
    "weight_decay": 0.01,
    "eps": 1e-08
}
```

### Optimizer.Config

**Component:** *Optimizer*

**class** Optimizer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Subclasses**

- AdaBelief.Config
- FP16Optimizer.Config
- FP16OptimizerApex.Config
- FP16OptimizerFairseq.Config
- MemoryEfficientFP16OptimizerFairseq.Config
- Lamb.Config
- MADGRAD.Config
- Adagrad.Config
- Adam.Config
- AdamW.Config
- SGD.Config
- RAdam.Config
- StochasticWeightAveraging.Config

**Default JSON**

```
{}
```

### SGD.Config

**Component:** *SGD*

**class** SGD.**Config**

**Bases:** Optimizer.Config

**All Attributes (including base classes)**

**lr: float = `0.001`**

**momentum: float = `0.0`**

**Default JSON**

```
{
    "lr": 0.001,
    "momentum": 0.0
}
```

### privacy_engine

#### PrivacyEngine.Config

**Component:** *PrivacyEngine*

**class** PrivacyEngine.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**noise_multiplier: float**

**max_grad_norm: float**

**batch_size: float**

**dataset_size: float**

**target_delta: Optional[float] = `1e-06`**

**alphas: Optional[list[float]] = `[1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3,`**

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### radam

#### RAdam.Config

**Component:** *RAdam*

**class** RAdam.**Config**

**Bases:** `Optimizer.Config`

**All Attributes (including base classes)**

**lr: float = `0.001`**

**weight_decay: float = `1e-05`**

**eps: float = `1e-08`**

**Default JSON**

```
{
    "lr": 0.001,
    "weight_decay": 1e-05,
    "eps": 1e-08
}
```

### scheduler

#### BatchScheduler.Config

**Component:** *BatchScheduler*

**class** BatchScheduler.**Config**

**Bases:** Scheduler.Config

**All Attributes (including base classes)**

**Subclasses**

- PolynomialDecayScheduler.Config
- SchedulerWithWarmup.Config
- WarmupScheduler.Config

**Default JSON**

```
{}
```

#### CosineAnnealingLR.Config

**Component:** *CosineAnnealingLR*

**class** CosineAnnealingLR.**Config**

**Bases:** Scheduler.Config

**All Attributes (including base classes)**

**t_max: int = 1000** Maximum number of iterations.

**eta_min: float = 0** Minimum learning rate

**Default JSON**

```
{
    "t_max": 1000,
    "eta_min": 0
}
```

#### CyclicLR.Config

**Component:** *CyclicLR*

**class** CyclicLR.**Config**

**Bases:** Scheduler.Config

**All Attributes (including base classes)**

**base_lr: float = `0.001`**

**max_lr: float = `0.002`**

**step_size_up: int = `2000`**

**step_size_down: Optional[int] = `None`**

**mode: str = `'triangular'`**

**gamma: float = `1.0`**

**scale_mode: str = `'cycle'`**

**cycle_momentum: bool = `True`**

**base_momentum: float = `0.8`**

**max_momentum: float = `0.9`**

**last_epoch: int = `-1`**

**Default JSON**

```
{
    "base_lr": 0.001,
    "max_lr": 0.002,
    "step_size_up": 2000,
    "step_size_down": null,
    "mode": "triangular",
    "gamma": 1.0,
    "scale_mode": "cycle",
    "cycle_momentum": true,
    "base_momentum": 0.8,
    "max_momentum": 0.9,
    "last_epoch": -1
}
```

### ExponentialLR.Config

**Component:** *ExponentialLR*

**class** ExponentialLR.**Config**

**Bases:** `Scheduler.Config`

**All Attributes (including base classes)**

**gamma: float = `0.1`** Multiplicative factor of learning rate decay.

**Default JSON**

```
{
    "gamma": 0.1
}
```

### LmFineTuning.Config

**Component:** *LmFineTuning*

**class** LmFineTuning.**Config**

**Bases:** `Scheduler.Config`

**All Attributes (including base classes)**

**cut_frac: float = `0.1`** The fraction of iterations we increase the learning rate. Default 0.1

**ratio: int = `32`** How much smaller the lowest LR is from the maximum LR eta_max.

**non_pretrained_param_groups: int = `2`** Number of param_groups, starting from the end, that were not pretrained. The default value is 2, since the base Model class supplies to the optimizer typically one param_group from the embedding and one param_group from its other components.

**lm_lr_multiplier: float = `1.0`** Factor to multiply lr for all pretrained layers by.

**lm_use_per_layer_lr: bool = `False`** Whether to make each pretrained layer's lr one-half as large as the next (higher) layer.

**lm_gradual_unfreezing: bool = `True`** Whether to unfreeze layers one by one (per epoch).

**last_epoch: int = `-1`** Though the name is *last_epoch*, it means *last batch update*. last_batch_update: = current_epoch_number * num_batches_per_epoch + batch_id after each batch update, it will increment 1

**Default JSON**

```
{
    "cut_frac": 0.1,
    "ratio": 32,
    "non_pretrained_param_groups": 2,
    "lm_lr_multiplier": 1.0,
    "lm_use_per_layer_lr": false,
    "lm_gradual_unfreezing": true,
    "last_epoch": -1
}
```

### PolynomialDecayScheduler.Config

**Component:** *PolynomialDecayScheduler*

**class** `PolynomialDecayScheduler.`**`Config`**

**Bases:** `BatchScheduler.Config`

**All Attributes (including base classes)**

**warmup_steps: int = `0`** number of training steps over which to increase learning rate

**total_steps: int** number of training steps for learning rate decay

**end_learning_rate: float** end learning rate after *total_steps* of training

**power: float = `1.0`** power used for polynomial decay calculation

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### ReduceLROnPlateau.Config

**Component:** *ReduceLROnPlateau*

**class** `ReduceLROnPlateau.`**`Config`**

**Bases:** `Scheduler.Config`

**All Attributes (including base classes)**

**lower_is_better: bool = `True`** This indicates the desirable direction in which we would like the training to proceed. If set to true, learning rate will be reduce when quantity being monitored stops going down

**factor: float = `0.1`** Factor by which the learning rate will be reduced. new_lr = lr * factor

**patience: int = `5`** Number of epochs with no improvement after which learning rate will be reduced

**min_lr: float = `0`** Lower bound on the learning rate of all param groups

**threshold: float = `0.0001`** Threshold for measuring the new optimum, to only focus on significant changes.

**threshold_is_absolute: bool = `True`** One of rel, abs. In rel mode, dynamic_threshold = best * ( 1 + threshold ) in 'max' mode or best * ( 1 - threshold ) in min mode. In abs mode, dynamic_threshold = best + threshold in max mode or best - threshold in min mode.

**cooldown: int = `0`** Number of epochs to wait before resuming normal operation after lr has been reduced.

**Default JSON**

```
{
    "lower_is_better": true,
    "factor": 0.1,
    "patience": 5,
    "min_lr": 0,
    "threshold": 0.0001,
    "threshold_is_absolute": true,
    "cooldown": 0
}
```

### Scheduler.Config

**Component:** *Scheduler*

**class** `Scheduler.`**`Config`**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Subclasses**

- `BatchScheduler.Config`
- `CosineAnnealingLR.Config`
- `CyclicLR.Config`
- `ExponentialLR.Config`
- `LmFineTuning.Config`
- `PolynomialDecayScheduler.Config`
- `ReduceLROnPlateau.Config`
- `SchedulerWithWarmup.Config`
- `StepLR.Config`
- `WarmupScheduler.Config`

**Default JSON**

```
{}
```

### SchedulerWithWarmup.Config

**Component:** *SchedulerWithWarmup*

**class** SchedulerWithWarmup.**Config**

**Bases:** BatchScheduler.Config

**All Attributes (including base classes)**

**warmup_scheduler:** *WarmupScheduler.Config* = *WarmupScheduler.Config*()

**scheduler:** Union[*ExponentialLR.Config*, *CosineAnnealingLR.Config*, *ReduceLROnPlateau.Config*, *LmFineTuning.Config*,

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### StepLR.Config

**Component:** *StepLR*

**class** StepLR.**Config**

**Bases:** Scheduler.Config

**All Attributes (including base classes)**

**step_size: int = 30** Period of learning rate decay.

**gamma: float = 0.1** Multiplicative factor of learning rate decay.

**Default JSON**

```
{
    "step_size": 30,
    "gamma": 0.1
}
```

### WarmupScheduler.Config

**Component:** *WarmupScheduler*

**class** WarmupScheduler.**Config**

**Bases:** BatchScheduler.Config

**All Attributes (including base classes)**

**warmup_steps: int = 10000** number of training steps over which to increase learning rate

**inverse_sqrt_decay: bool = False** whether to perform inverse sqrt decay after the warmup phase

**Default JSON**

```
{
    "warmup_steps": 10000,
    "inverse_sqrt_decay": false
}
```

### sparsifiers

### blockwise_sparsifier

### BlockwiseMagnitudeSparsifier.Config

**Component:** *BlockwiseMagnitudeSparsifier*

**class** BlockwiseMagnitudeSparsifier.**Config**

**Bases:** L0_projection_sparsifier.Config

**All Attributes (including base classes)**

**sparsity: float = 0.9**

**starting_epoch: int = 2**

**frequency: int = 1**

**layerwise_pruning: bool = True**

**accumulate_mask: bool = False**

**block_size: int = 16**

**columnwise_blocking: bool = False**

**Default JSON**

```
{
    "sparsity": 0.9,
    "starting_epoch": 2,
    "frequency": 1,
    "layerwise_pruning": true,
    "accumulate_mask": false,
    "block_size": 16,
    "columnwise_blocking": false
}
```

### sparsifier

### CRF_L1_SoftThresholding.Config

**Component:** *CRF_L1_SoftThresholding*

**class** CRF_L1_SoftThresholding.**Config**

**Bases:** CRF_SparsifierBase.Config

**All Attributes (including base classes)**

**starting_epoch: int = 1**

**frequency: int = 1**

**lambda_l1: float = 0.001**

**Default JSON**

```
{
    "starting_epoch": 1,
    "frequency": 1,
    "lambda_l1": 0.001
}
```

### CRF_MagnitudeThresholding.Config

**Component:** *CRF_MagnitudeThresholding*

**class** CRF_MagnitudeThresholding.**Config**

**Bases:** CRF_SparsifierBase.Config

**All Attributes (including base classes)**

**starting_epoch: int = 1**

**frequency: int = 1**

**sparsity: float = 0.9**

**grouping: str = 'row'**

**Default JSON**

```
{
    "starting_epoch": 1,
    "frequency": 1,
    "sparsity": 0.9,
    "grouping": "row"
}
```

### CRF_SparsifierBase.Config

**Component:** *CRF_SparsifierBase*

**class** CRF_SparsifierBase.**Config**

**Bases:** Sparsifier.Config

**All Attributes (including base classes)**

**starting_epoch: int = 1**

**frequency: int = 1**

**Subclasses**

- CRF_L1_SoftThresholding.Config
- CRF_MagnitudeThresholding.Config

**Default JSON**

```
{
    "starting_epoch": 1,
    "frequency": 1
}
```

### L0_projection_sparsifier.Config

**Component:** *L0_projection_sparsifier*

**class** L0_projection_sparsifier.**Config**

**Bases:** Sparsifier.Config

**All Attributes (including base classes)**

**sparsity: float =** `0.9`

**starting_epoch: int =** `2`

**frequency: int =** `1`

**layerwise_pruning: bool =** `True`

**accumulate_mask: bool =** `False`

**Subclasses**

- BlockwiseMagnitudeSparsifier.Config

**Default JSON**

```
{
    "sparsity": 0.9,
    "starting_epoch": 2,
    "frequency": 1,
    "layerwise_pruning": true,
    "accumulate_mask": false
}
```

### SensitivityAnalysisSparsifier.Config

**Component:** *SensitivityAnalysisSparsifier*

**class** SensitivityAnalysisSparsifier.**Config**

**Bases:** Sparsifier.Config

**All Attributes (including base classes)**

**pre_train_model_path: str =** `''`

**analyzed_sparsity: float =** `0.8`

**max_analysis_batches: int =** `0`

**max_skipped_weight: int =** `0`

**pre_analysis_path: str =** `''`

**sparsity: float =** `0.8`

**iterative_pruning: bool =** `True`

**pruning_iterations: int = 2**

**start_sparsity_ratio: float = 0.5**

**Default JSON**

```
{
    "pre_train_model_path": "",
    "analyzed_sparsity": 0.8,
    "max_analysis_batches": 0,
    "max_skipped_weight": 0,
    "pre_analysis_path": "",
    "sparsity": 0.8,
    "iterative_pruning": true,
    "pruning_iterations": 2,
    "start_sparsity_ratio": 0.5
}
```

### Sparsifier.Config

**Component:** *Sparsifier*

**class** Sparsifier.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**Subclasses**

- BlockwiseMagnitudeSparsifier.Config
- CRF_L1_SoftThresholding.Config
- CRF_MagnitudeThresholding.Config
- CRF_SparsifierBase.Config
- L0_projection_sparsifier.Config
- SensitivityAnalysisSparsifier.Config

**Default JSON**

```
{}
```

## swa

### StochasticWeightAveraging.Config

**Component:** *StochasticWeightAveraging*

**class** StochasticWeightAveraging.**Config**

**Bases:** Optimizer.Config

**All Attributes (including base classes)**

**optimizer: Union[***SGD.Config***, *Adam.Config*, *AdamW.Config*, *Adagrad.Config*, *RAdam.Config*, *Lamb.Config*, *MADGRAD.C*

**start: int = 10**

**frequency: int = 5**

**swa_learning_rate: Optional[float] = 0.05**

**Default JSON**

```
{
    "optimizer": {
        "SGD": {
            "lr": 0.001,
            "momentum": 0.0
        }
    },
    "start": 10,
    "frequency": 5,
    "swa_learning_rate": 0.05
}
```

### 1.21.8 task

#### disjoint_multitask

#### DisjointMultitask.Config

**Component:** *DisjointMultitask*

**class** DisjointMultitask.**Config**

**Bases:** TaskBase.Config

**All Attributes (including base classes)**

**features:** *FeatureConfig* = *FeatureConfig*()

**featurizer:** *Featurizer.Config* = *SimpleFeaturizer.Config*()

**data_handler:** *DisjointMultitaskDataHandler.Config* = *DisjointMultitaskDataHandler.Config*()

**trainer:** *Trainer.Config* = *Trainer.Config*()

**exporter: Optional[***ModelExporter.Config***] = None**

**tasks: dict[str, *Task_Deprecated.Config*]**

**task_weights: dict[str, float] = { }**

**target_task_name: Optional[str] = None**

**metric_reporter:** *DisjointMultitaskMetricReporter.Config* = *DisjointMultitaskMetricReporter.Config*()

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

#### NewDisjointMultitask.Config

**Component:** *NewDisjointMultitask*

**class** NewDisjointMultitask.**Config**

**Bases:** _NewTask.Config

**All Attributes (including base classes)**

**data:** *DisjointMultitaskData.Config* = *DisjointMultitaskData.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**tasks: dict[str,** *NewTask.Config***] = { }**

**task_weights: dict[str, float] = { }**

**target_task_name: Optional[str] = None**

**metric_reporter:** *DisjointMultitaskMetricReporter.Config* = *DisjointMultitaskMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "sampler": {
            "RoundRobinBatchSampler": {
                "iter_to_set_epoch": ""
            }
        },
        "test_key": null
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
```

(continues on next page)

(continued from previous page)

```
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "tasks": {},
    "task_weights": {},
    "target_task_name": null,
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false,
        "use_subtask_select_metric": false
    }
}
```

### new_task

### NewTask.Config

**Component:** *NewTask*

**class** NewTask.**Config**

**Bases:** _NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**model: Optional[*BaseModel.Config*]**

**Subclasses**

- BertPairRegressionTask.Config
- DocumentClassificationTask.Config
- DocumentRegressionTask.Config
- EnsembleTask.Config
- IntentSlotTask.Config
- LMTask.Config
- MaskedLMTask.Config
- NewBertClassificationTask.Config
- NewBertPairClassificationTask.Config
- PairwiseClassificationForDenseRetrievalTask.Config
- PairwiseClassificationTask.Config
- PairwiseRegressionTask.Config

- QueryDocumentPairwiseRankingTask.Config
- RoBERTaNERTask.Config
- SemanticParsingTask.Config
- SeqNNTask.Config
- SequenceLabelingTask.Config
- SquadQATask.Config
- WordTaggingTask.Config

**Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### _NewTask.Config

**Component:** _NewTask

**class** _NewTask.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**Subclasses**

- NewDisjointMultitask.Config
- NewTask.Config
- BertPairRegressionTask.Config
- DocumentClassificationTask.Config
- DocumentRegressionTask.Config
- EnsembleTask.Config
- IntentSlotTask.Config
- LMTask.Config
- MaskedLMTask.Config
- NewBertClassificationTask.Config
- NewBertPairClassificationTask.Config
- PairwiseClassificationForDenseRetrievalTask.Config
- PairwiseClassificationTask.Config
- PairwiseRegressionTask.Config
- QueryDocumentPairwiseRankingTask.Config
- RoBERTaNERTask.Config

- SemanticParsingTask.Config
- SeqNNTask.Config
- SequenceLabelingTask.Config
- SquadQATask.Config
- WordTaggingTask.Config

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
```

(continues on next page)

(continued from previous page)

```
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null
}
```

### task

#### TaskBase.Config

**Component:** *TaskBase*

**class** TaskBase.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**features:** *FeatureConfig* = *FeatureConfig*()

**featurizer:** *Featurizer.Config* = *SimpleFeaturizer.Config*()

**data_handler:** *DataHandler.Config*

**trainer:** *Trainer.Config* = *Trainer.Config*()

**exporter:** Optional[*ModelExporter.Config*] = **None**

**Subclasses**

- DisjointMultitask.Config
- Task_Deprecated.Config

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

#### Task_Deprecated.Config

**Component:** *Task_Deprecated*

**class** Task_Deprecated.**Config**

**Bases:** TaskBase.Config

**All Attributes (including base classes)**

**features:** *FeatureConfig* = *FeatureConfig*()

**featurizer:** *Featurizer.Config* = *SimpleFeaturizer.Config*()

**data_handler:** *DataHandler.Config*

**trainer:** *Trainer.Config* = *Trainer.Config*()

**exporter:** Optional[*ModelExporter.Config*] = `None`

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### tasks

### BertPairRegressionTask.Config

**Component:** `BertPairRegressionTask`

**class** `BertPairRegressionTask.`**`Config`**

**Bases:** `DocumentRegressionTask.Config`

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = `None`

**model:** *NewBertRegressionModel.Config* = *NewBertRegressionModel.Config*()

**metric_reporter:** *RegressionMetricReporter.Config* = *RegressionMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
```

(continues on next page)

(continued from previous page)

```
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "tokens": {
                "BERTTensorizer": {
                    "is_input": true,
                    "columns": [
                        "text1",
                        "text2"
                    ],
                    "tokenizer": {
                        "WordPieceTokenizer": {
                            "basic_tokenizer": {
                                "split_regex": "\\s+",
                                "lowercase": true,
                                "use_byte_offsets": false
                            },
                            "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                        }
```

(continues on next page)

(continued from previous page)

```
                    },
                    "base_tokenizer": null,
                    "vocab_file": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt",
                    "max_seq_len": 128
                }
            },
            "labels": {
                "is_input": false,
                "column": "label",
                "rescale_range": null
            }
        },
        "encoder": {
            "HuggingFaceBertSentenceEncoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "output_dropout": 0.4,
                "embedding_dim": 768,
                "pooling": "cls_token",
                "export": false,
                "projection_dim": 0,
                "normalize_output_rep": false,
                "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/",
                "load_weights": true
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {},
            "squash_to_unit_range": false
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
```

(continues on next page)

(continued from previous page)

```
        "log_gradient": false
    }
}
```

### DocumentClassificationTask.Config

**Component:** *DocumentClassificationTask*

**class** DocumentClassificationTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** *BaseModel.Config* = *DocModel.Config*()

**metric_reporter:** Union[*ClassificationMetricReporter.Config*, *PureLossMetricReporter.Config*] = *ClassificationMetricReporte*

**Subclasses**

- NewBertClassificationTask.Config
- NewBertPairClassificationTask.Config

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "DocModel": {
            "inputs": {
                "tokens": {
                    "is_input": true,
                    "column": "text",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "add_bos_token": false,
                    "add_eos_token": false,
                    "use_eos_token_for_bos": false,
                    "max_seq_len": null,
                    "vocab": {
                        "build_from_data": true,
                        "size_from_data": 0,
                        "min_counts": 0,
```

(continues on next page)

(continued from previous page)

```
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "dense": null,
            "labels": {
                "LabelTensorizer": {
                    "is_input": false,
                    "column": "label",
                    "allow_unknown": false,
                    "pad_in_vocab": false,
                    "label_vocab": null,
                    "label_vocab_file": null,
                    "add_labels": null
                }
            }
        },
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "representation": {
            "BiLSTMDocAttention": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
```

(continues on next page)

(continued from previous page)

```
                        "num_layers": 1,
                        "bidirectional": true,
                        "pack_sequence": true,
                        "disable_sort_in_jit": false
                    },
                    "pooling": {
                        "SelfAttention": {
                            "attn_dimension": 64,
                            "dropout": 0.4
                        }
                    },
                    "mlp_decoder": null
                }
            },
            "decoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "hidden_dims": [],
                "out_dim": null,
                "layer_norm": false,
                "dropout": 0.0,
                "bias": true,
                "activation": "relu",
                "temperature": 1.0,
                "spectral_normalization": false
            },
            "output_layer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {
                    "CrossEntropyLoss": {}
                },
                "label_weights": null
            }
        }
    },
    "metric_reporter": {
        "ClassificationMetricReporter": {
            "output_path": "/tmp/test_out.txt",
            "pep_format": false,
            "student_column_names": [],
            "log_gradient": false,
            "model_select_metric": "accuracy",
            "target_label": null,
            "text_column_names": [
                "text"
            ],
            "additional_column_names": [],
            "recall_at_precision_thresholds": [
                0.2,
                0.4,
                0.6,
                0.8,
```

(continues on next page)

(continued from previous page)

```
                0.9
            ],
            "is_memory_efficient": false
        }
    }
}
```

### DocumentRegressionTask.Config

**Component:** *DocumentRegressionTask*

**class** DocumentRegressionTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** *BaseModel.Config* = *DocRegressionModel.Config*()

**metric_reporter:** *RegressionMetricReporter.Config* = *RegressionMetricReporter.Config*()

**Subclasses**

- BertPairRegressionTask.Config

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
```

(continues on next page)

(continued from previous page)

```
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "DocRegressionModel": {
            "inputs": {
                "tokens": {
                    "is_input": true,
                    "column": "text",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "add_bos_token": false,
                    "add_eos_token": false,
                    "use_eos_token_for_bos": false,
                    "max_seq_len": null,
                    "vocab": {
                        "build_from_data": true,
                        "size_from_data": 0,
```

(continues on next page)

(continued from previous page)

```
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "dense": null,
            "labels": {
                "is_input": false,
                "column": "label",
                "rescale_range": null
            }
        },
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "representation": {
            "BiLSTMDocAttention": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
                    "num_layers": 1,
                    "bidirectional": true,
                    "pack_sequence": true,
                    "disable_sort_in_jit": false
                },
```

(continues on next page)

(continued from previous page)

```
                    "pooling": {
                        "SelfAttention": {
                            "attn_dimension": 64,
                            "dropout": 0.4
                        }
                    },
                    "mlp_decoder": null
                }
            },
            "decoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "hidden_dims": [],
                "out_dim": null,
                "layer_norm": false,
                "dropout": 0.0,
                "bias": true,
                "activation": "relu",
                "temperature": 1.0,
                "spectral_normalization": false
            },
            "output_layer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {},
                "squash_to_unit_range": false
            }
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false
    }
}
```

### EnsembleTask.Config

**Component:** *EnsembleTask*

**class** EnsembleTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *EnsembleTrainer.Config* = *EnsembleTrainer.Config*()

**use_elastic:** Optional[bool] = None

**model:** *EnsembleModel.Config*

**metric_reporter: Union[*ClassificationMetricReporter.Config*, *IntentSlotMetricReporter.Config*] = *ClassificationMetricReport***

> **Warning:** This config has parameters with no default values. We aren't yet able to generate functional JSON for it.

### IntentSlotTask.Config

**Component:** `IntentSlotTask`

**class** `IntentSlotTask.`**`Config`**

**Bases:** `NewTask.Config`

**All Attributes (including base classes)**

**data: *Data.Config* = *Data.Config*()**

**trainer: *TaskTrainer.Config* = *TaskTrainer.Config*()**

**use_elastic: Optional[bool] = `None`**

**model: *IntentSlotModel.Config* = *IntentSlotModel.Config*()**

**metric_reporter: *IntentSlotMetricReporter.Config* = *IntentSlotMetricReporter.Config*()**

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
```

(continues on next page)

(continued from previous page)

```
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "IntentSlotModel": {
            "inputs": {
                "tokens": {
                    "is_input": true,
                    "column": "text",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "add_bos_token": false,
                    "add_eos_token": false,
                    "use_eos_token_for_bos": false,
                    "max_seq_len": null,
                    "vocab": {
                        "build_from_data": true,
                        "size_from_data": 0,
                        "min_counts": 0,
                        "vocab_files": []
                    },
                    "vocab_file_delimiter": " "
                },
```

(continues on next page)

(continued from previous page)

```
                "word_labels": {
                    "is_input": false,
                    "slot_column": "slots",
                    "text_column": "text",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "allow_unknown": true
                },
                "doc_labels": {
                    "LabelTensorizer": {
                        "is_input": false,
                        "column": "label",
                        "allow_unknown": true,
                        "pad_in_vocab": false,
                        "label_vocab": null,
                        "label_vocab_file": null,
                        "add_labels": null
                    }
                },
                "doc_weight": null,
                "word_weight": null
            },
            "word_embedding": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "embed_dim": 100,
                "embedding_init_strategy": "random",
                "embedding_init_range": null,
                "embeddding_init_std": 0.02,
                "export_input_names": [
                    "tokens_vals"
                ],
                "pretrained_embeddings_path": "",
                "vocab_file": "",
                "vocab_size": 0,
                "vocab_from_train_data": true,
                "vocab_from_all_data": false,
                "vocab_from_pretrained_embeddings": false,
                "lowercase_tokens": true,
                "min_freq": 1,
                "mlp_layer_dims": [],
                "padding_idx": null,
                "cpu_only": false,
                "skip_header": true,
                "delimiter": " "
            },
            "representation": {
                "BiLSTMDocSlotAttention": {
                    "load_path": null,
                    "save_path": null,
```

(continues on next page)

(continued from previous page)

```
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "BiLSTM": {
                        "load_path": null,
                        "save_path": null,
                        "freeze": false,
                        "shared_module_key": null,
                        "dropout": 0.4,
                        "lstm_dim": 32,
                        "num_layers": 1,
                        "bidirectional": true,
                        "pack_sequence": true,
                        "disable_sort_in_jit": false
                    }
                },
                "pooling": null,
                "slot_attention": null,
                "doc_mlp_layers": 0,
                "word_mlp_layers": 0
            }
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "doc_output": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {
                    "CrossEntropyLoss": {}
                },
                "label_weights": null
            },
            "word_output": {
                "WordTaggingOutputLayer": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "loss": {
                        "CrossEntropyLoss": {}
                    },
                    "label_weights": {},
                    "ignore_pad_in_loss": true
                }
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
```

(continues on next page)

(continued from previous page)

```
                "use_doc_probs_in_word": false,
                "doc_decoder": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "hidden_dims": [],
                    "out_dim": null,
                    "layer_norm": false,
                    "dropout": 0.0,
                    "bias": true,
                    "activation": "relu",
                    "temperature": 1.0,
                    "spectral_normalization": false
                },
                "word_decoder": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "hidden_dims": [],
                    "out_dim": null,
                    "layer_norm": false,
                    "dropout": 0.0,
                    "bias": true,
                    "activation": "relu",
                    "temperature": 1.0,
                    "spectral_normalization": false
                }
            },
            "default_doc_loss_weight": 0.2,
            "default_word_loss_weight": 0.5
        }
    },
    "metric_reporter": {
        "IntentSlotMetricReporter": {
            "output_path": "/tmp/test_out.txt",
            "pep_format": false,
            "student_column_names": [],
            "log_gradient": false
        }
    }
}
```

### LMTask.Config

**Component:** *LMTask*

**class** LMTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**model:** *LMLSTM.Config* = *LMLSTM.Config*()

**metric_reporter:** *LanguageModelMetricReporter.Config* = *LanguageModelMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
```

(continues on next page)

(continued from previous page)

```
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "tokens": {
                "is_input": true,
                "column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": true,
                "add_eos_token": true,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            }
        },
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
```

(continues on next page)

(continued from previous page)

```
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "representation": {
            "BiLSTM": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": false,
                "pack_sequence": true,
                "disable_sort_in_jit": false
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {}
        },
        "tied_weights": false,
        "stateful": false,
        "caffe2_format": "predictor"
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false,
        "aggregate_metrics": true,
        "perplexity_type": "median"
    }
}
```

### MaskedLMTask.Config

**Component:** *MaskedLMTask*

**class** MaskedLMTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *PackedLMData.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** *MaskedLanguageModel.Config* = *MaskedLanguageModel.Config*()

**metric_reporter:** *MaskedLMMetricReporter.Config* = *MaskedLMMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "PackedLMData": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true,
            "max_seq_len": 128
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
```

(continues on next page)

(continued from previous page)

```
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "tokens": {
                "BERTTensorizerBase": {
                    "is_input": true,
                    "columns": [
                        "text"
                    ],
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "base_tokenizer": null,
                    "vocab_file": "",
                    "max_seq_len": 128
                }
            }
        },
        "encoder": {
            "TransformerSentenceEncoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "output_dropout": 0.4,
                "embedding_dim": 768,
                "pooling": "cls_token",
                "export": false,
                "projection_dim": 0,
```

(continues on next page)

(continued from previous page)

```
                "normalize_output_rep": false,
                "dropout": 0.1,
                "attention_dropout": 0.1,
                "activation_dropout": 0.1,
                "ffn_embedding_dim": 3072,
                "num_encoder_layers": 6,
                "num_attention_heads": 8,
                "num_segments": 2,
                "use_position_embeddings": true,
                "offset_positions_by_padding": true,
                "apply_bert_init": true,
                "encoder_normalize_before": true,
                "activation_fn": "relu",
                "max_seq_len": 128,
                "multilingual": false,
                "freeze_embeddings": false,
                "n_trans_layers_to_freeze": 0,
                "use_torchscript": false,
                "use_bias_finetuning": false
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {}
        },
        "mask_prob": 0.15,
        "mask_bos": false,
        "masking_strategy": "random",
        "tie_weights": true
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false,
        "aggregate_metrics": true,
        "perplexity_type": "median"
    }
}
```

### NewBertClassificationTask.Config

**Component:** *NewBertClassificationTask*

**class** NewBertClassificationTask.**Config**

**Bases:** DocumentClassificationTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** *NewBertModel.Config* = *NewBertModel.Config*()

**metric_reporter:** Union[*ClassificationMetricReporter.Config*, *PureLossMetricReporter.Config*] = *ClassificationMetricReporte*

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
```

(continues on next page)

(continued from previous page)

```
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "tokens": {
                "BERTTensorizer": {
                    "is_input": true,
                    "columns": [
                        "text"
                    ],
                    "tokenizer": {
                        "WordPieceTokenizer": {
                            "basic_tokenizer": {
                                "split_regex": "\\s+",
                                "lowercase": true,
                                "use_byte_offsets": false
                            },
                            "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                        }
                    },
                    "base_tokenizer": null,
                    "vocab_file": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt",
                    "max_seq_len": 128
                }
            },
            "dense": null,
            "labels": {
                "LabelTensorizer": {
                    "is_input": false,
                    "column": "label",
                    "allow_unknown": false,
                    "pad_in_vocab": false,
```

(continues on next page)

(continued from previous page)

```
                    "label_vocab": null,
                    "label_vocab_file": null,
                    "add_labels": null
                }
            },
            "num_tokens": {
                "is_input": false,
                "names": [
                    "tokens"
                ],
                "indexes": [
                    2
                ]
            }
        },
        "encoder": {
            "HuggingFaceBertSentenceEncoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "output_dropout": 0.4,
                "embedding_dim": 768,
                "pooling": "cls_token",
                "export": false,
                "projection_dim": 0,
                "normalize_output_rep": false,
                "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/",
                "load_weights": true
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        }
    },
```

(continues on next page)

(continued from previous page)

```
    "metric_reporter": {
        "ClassificationMetricReporter": {
            "output_path": "/tmp/test_out.txt",
            "pep_format": false,
            "student_column_names": [],
            "log_gradient": false,
            "model_select_metric": "accuracy",
            "target_label": null,
            "text_column_names": [
                "text"
            ],
            "additional_column_names": [],
            "recall_at_precision_thresholds": [
                0.2,
                0.4,
                0.6,
                0.8,
                0.9
            ],
            "is_memory_efficient": false
        }
    }
}
```

### NewBertPairClassificationTask.Config

**Component:** *NewBertPairClassificationTask*

**class** NewBertPairClassificationTask.**Config**

**Bases:** DocumentClassificationTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**model:** *NewBertModel.Config* = *NewBertModel.Config*(**inputs=***BertModelInput*(**tokens=***BERTTensorizer.Config*(**columns=[**

**metric_reporter:** *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*(**text_column_names=['text1**

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
```

(continues on next page)

(continued from previous page)

```
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "tokens": {
                "BERTTensorizer": {
```

(continues on next page)

(continued from previous page)

```
                    "is_input": true,
                    "columns": [
                        "text1",
                        "text2"
                    ],
                    "tokenizer": {
                        "WordPieceTokenizer": {
                            "basic_tokenizer": {
                                "split_regex": "\\s+",
                                "lowercase": true,
                                "use_byte_offsets": false
                            },
                            "wordpiece_vocab_path": "manifold://nlp_technologies/tree/
↪huggingface-models/bert-base-uncased/vocab.txt"
                        }
                    },
                    "base_tokenizer": null,
                    "vocab_file": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt",
                    "max_seq_len": 128
                }
            },
            "dense": null,
            "labels": {
                "LabelTensorizer": {
                    "is_input": false,
                    "column": "label",
                    "allow_unknown": false,
                    "pad_in_vocab": false,
                    "label_vocab": null,
                    "label_vocab_file": null,
                    "add_labels": null
                }
            },
            "num_tokens": {
                "is_input": false,
                "names": [
                    "tokens"
                ],
                "indexes": [
                    2
                ]
            }
        },
        "encoder": {
            "HuggingFaceBertSentenceEncoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "output_dropout": 0.4,
                "embedding_dim": 768,
                "pooling": "cls_token",
                "export": false,
                "projection_dim": 0,
                "normalize_output_rep": false,
                "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-models/
↪bert-base-uncased/",
```

(continues on next page)

(continued from previous page)

```
                "load_weights": true
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        }
    },
    "metric_reporter": {
        "ClassificationMetricReporter": {
            "output_path": "/tmp/test_out.txt",
            "pep_format": false,
            "student_column_names": [],
            "log_gradient": false,
            "model_select_metric": "accuracy",
            "target_label": null,
            "text_column_names": [
                "text1",
                "text2"
            ],
            "additional_column_names": [],
            "recall_at_precision_thresholds": [
                0.2,
                0.4,
                0.6,
                0.8,
                0.9
            ],
            "is_memory_efficient": false
        }
    }
}
```

### PairwiseClassificationForDenseRetrievalTask.Config

**Component:** *PairwiseClassificationForDenseRetrievalTask*

**class** PairwiseClassificationForDenseRetrievalTask.**Config**

**Bases:** PairwiseClassificationTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = `None`

**model:** *BasePairwiseModel.Config* = *PairwiseModel.Config*()

**metric_reporter:** *DenseRetrievalMetricReporter.Config* = *DenseRetrievalMetricReporter.Config*()

**trace_both_encoders:** bool = `True`

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
```

(continues on next page)

(continued from previous page)

```
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "PairwiseModel": {
            "inputs": {
                "tokens1": {
                    "is_input": true,
                    "column": "text1",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "add_bos_token": false,
                    "add_eos_token": false,
                    "use_eos_token_for_bos": false,
                    "max_seq_len": null,
                    "vocab": {
                        "build_from_data": true,
                        "size_from_data": 0,
                        "min_counts": 0,
                        "vocab_files": []
                    },
                    "vocab_file_delimiter": " "
                },
                "tokens2": {
                    "is_input": true,
                    "column": "text2",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "add_bos_token": false,
```

(continues on next page)

(continued from previous page)

```
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "labels": {
                "LabelTensorizer": {
                    "is_input": false,
                    "column": "label",
                    "allow_unknown": false,
                    "pad_in_vocab": false,
                    "label_vocab": null,
                    "label_vocab_file": null,
                    "add_labels": null
                }
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "output_layer": {
            "ClassificationOutputLayer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {
                    "CrossEntropyLoss": {}
                },
                "label_weights": null
            }
        },
        "encode_relations": true,
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
```

(continues on next page)

(continued from previous page)

```
                "embedding_init_range": null,
                "embeddding_init_std": 0.02,
                "export_input_names": [
                    "tokens_vals"
                ],
                "pretrained_embeddings_path": "",
                "vocab_file": "",
                "vocab_size": 0,
                "vocab_from_train_data": true,
                "vocab_from_all_data": false,
                "vocab_from_pretrained_embeddings": false,
                "lowercase_tokens": true,
                "min_freq": 1,
                "mlp_layer_dims": [],
                "padding_idx": null,
                "cpu_only": false,
                "skip_header": true,
                "delimiter": " "
            },
            "representation": {
                "BiLSTMDocAttention": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm": {
                        "load_path": null,
                        "save_path": null,
                        "freeze": false,
                        "shared_module_key": null,
                        "dropout": 0.4,
                        "lstm_dim": 32,
                        "num_layers": 1,
                        "bidirectional": true,
                        "pack_sequence": true,
                        "disable_sort_in_jit": false
                    },
                    "pooling": {
                        "SelfAttention": {
                            "attn_dimension": 64,
                            "dropout": 0.4
                        }
                    },
                    "mlp_decoder": null
                }
            },
            "shared_representations": true
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false,
        "text_column_names": [
            "question",
```

(continues on next page)

(continued from previous page)

```
            "positive_ctx",
            "negative_ctxs"
        ],
        "model_select_metric": "accuracy",
        "task_batch_size": 0,
        "num_negative_ctxs": 0
    },
    "trace_both_encoders": true
}
```

### PairwiseClassificationTask.Config

**Component:** *PairwiseClassificationTask*

**class** PairwiseClassificationTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**model:** *BasePairwiseModel.Config* = *PairwiseModel.Config*()

**metric_reporter:** *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*(**text_column_names=['text1**

**trace_both_encoders: bool = True**

**Subclasses**

- PairwiseClassificationForDenseRetrievalTask.Config
- PairwiseRegressionTask.Config

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
```

(continues on next page)

(continued from previous page)

```
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "PairwiseModel": {
            "inputs": {
                "tokens1": {
                    "is_input": true,
                    "column": "text1",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
```

(continues on next page)

(continued from previous page)

```
                    "add_bos_token": false,
                    "add_eos_token": false,
                    "use_eos_token_for_bos": false,
                    "max_seq_len": null,
                    "vocab": {
                        "build_from_data": true,
                        "size_from_data": 0,
                        "min_counts": 0,
                        "vocab_files": []
                    },
                    "vocab_file_delimiter": " "
                },
                "tokens2": {
                    "is_input": true,
                    "column": "text2",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true,
                            "use_byte_offsets": false
                        }
                    },
                    "add_bos_token": false,
                    "add_eos_token": false,
                    "use_eos_token_for_bos": false,
                    "max_seq_len": null,
                    "vocab": {
                        "build_from_data": true,
                        "size_from_data": 0,
                        "min_counts": 0,
                        "vocab_files": []
                    },
                    "vocab_file_delimiter": " "
                },
                "labels": {
                    "LabelTensorizer": {
                        "is_input": false,
                        "column": "label",
                        "allow_unknown": false,
                        "pad_in_vocab": false,
                        "label_vocab": null,
                        "label_vocab_file": null,
                        "add_labels": null
                    }
                }
            },
            "decoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "hidden_dims": [],
                "out_dim": null,
                "layer_norm": false,
                "dropout": 0.0,
                "bias": true,
                "activation": "relu",
```

(continues on next page)

(continued from previous page)

```
                "temperature": 1.0,
                "spectral_normalization": false
            },
            "output_layer": {
                "ClassificationOutputLayer": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "loss": {
                        "CrossEntropyLoss": {}
                    },
                    "label_weights": null
                }
            },
            "encode_relations": true,
            "embedding": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "embed_dim": 100,
                "embedding_init_strategy": "random",
                "embedding_init_range": null,
                "embeddding_init_std": 0.02,
                "export_input_names": [
                    "tokens_vals"
                ],
                "pretrained_embeddings_path": "",
                "vocab_file": "",
                "vocab_size": 0,
                "vocab_from_train_data": true,
                "vocab_from_all_data": false,
                "vocab_from_pretrained_embeddings": false,
                "lowercase_tokens": true,
                "min_freq": 1,
                "mlp_layer_dims": [],
                "padding_idx": null,
                "cpu_only": false,
                "skip_header": true,
                "delimiter": " "
            },
            "representation": {
                "BiLSTMDocAttention": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm": {
                        "load_path": null,
                        "save_path": null,
                        "freeze": false,
                        "shared_module_key": null,
                        "dropout": 0.4,
                        "lstm_dim": 32,
                        "num_layers": 1,
```

(continues on next page)

(continued from previous page)

```
                        "bidirectional": true,
                        "pack_sequence": true,
                        "disable_sort_in_jit": false
                    },
                    "pooling": {
                        "SelfAttention": {
                            "attn_dimension": 64,
                            "dropout": 0.4
                        }
                    },
                    "mlp_decoder": null
                }
            },
            "shared_representations": true
        }
    },
    "metric_reporter": {
        "ClassificationMetricReporter": {
            "output_path": "/tmp/test_out.txt",
            "pep_format": false,
            "student_column_names": [],
            "log_gradient": false,
            "model_select_metric": "accuracy",
            "target_label": null,
            "text_column_names": [
                "text1",
                "text2"
            ],
            "additional_column_names": [],
            "recall_at_precision_thresholds": [
                0.2,
                0.4,
                0.6,
                0.8,
                0.9
            ],
            "is_memory_efficient": false
        }
    },
    "trace_both_encoders": true
}
```

### PairwiseRegressionTask.Config

**Component:** *PairwiseRegressionTask*

**class** PairwiseRegressionTask.**Config**

**Bases:** PairwiseClassificationTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = None

**model:** *BasePairwiseModel.Config* = *BertPairwiseRegressionModel.Config*()

**metric_reporter:** *RegressionMetricReporter.Config* = *RegressionMetricReporter.Config*()

**trace_both_encoders: bool = `True`**

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
```

(continues on next page)

(continued from previous page)

```
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "BertPairwiseRegressionModel": {
            "inputs": {
                "tokens1": {
                    "BERTTensorizer": {
                        "is_input": true,
                        "columns": [
                            "text1"
                        ],
                        "tokenizer": {
                            "WordPieceTokenizer": {
                                "basic_tokenizer": {
                                    "split_regex": "\\s+",
                                    "lowercase": true,
                                    "use_byte_offsets": false
                                },
                                "wordpiece_vocab_path": "manifold://nlp_technologies/
↪tree/huggingface-models/bert-base-uncased/vocab.txt"
                            }
                        },
                        "base_tokenizer": null,
                        "vocab_file": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt",
                        "max_seq_len": 128
                    }
                },
                "tokens2": {
                    "BERTTensorizer": {
                        "is_input": true,
                        "columns": [
                            "text2"
                        ],
                        "tokenizer": {
                            "WordPieceTokenizer": {
                                "basic_tokenizer": {
                                    "split_regex": "\\s+",
                                    "lowercase": true,
                                    "use_byte_offsets": false
                                },
                                "wordpiece_vocab_path": "manifold://nlp_technologies/
↪tree/huggingface-models/bert-base-uncased/vocab.txt"
                            }
                        },
                        "base_tokenizer": null,
                        "vocab_file": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/vocab.txt",
                        "max_seq_len": 128
```

(continues on next page)

(continued from previous page)

```
                    }
                },
                "labels": {
                    "is_input": false,
                    "column": "label",
                    "rescale_range": null
                },
                "num_tokens": {
                    "is_input": false,
                    "names": [
                        "tokens1",
                        "tokens2"
                    ],
                    "indexes": [
                        2,
                        2
                    ]
                }
            },
            "decoder": null,
            "output_layer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {
                    "MSELoss": {}
                }
            },
            "encode_relations": false,
            "encoder": {
                "HuggingFaceBertSentenceEncoder": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "output_dropout": 0.4,
                    "embedding_dim": 768,
                    "pooling": "cls_token",
                    "export": false,
                    "projection_dim": 0,
                    "normalize_output_rep": false,
                    "bert_cpt_dir": "manifold://nlp_technologies/tree/huggingface-
↪models/bert-base-uncased/",
                    "load_weights": true
                }
            },
            "shared_encoder": true
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false
    },
    "trace_both_encoders": true
```

(continues on next page)

(continued from previous page)

```
}
```

### QueryDocumentPairwiseRankingTask.Config

**Component:** *QueryDocumentPairwiseRankingTask*

**class** QueryDocumentPairwiseRankingTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** *QueryDocPairwiseRankingModel.Config* = *QueryDocPairwiseRankingModel.Config*()

**metric_reporter:** *PairwiseRankingMetricReporter.Config* = *PairwiseRankingMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
```

(continues on next page)

(continued from previous page)

```
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "pos_response": {
                "is_input": true,
                "column": "pos_response",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "neg_response": {
                "is_input": true,
                "column": "neg_response",
```

(continues on next page)

(continued from previous page)

```
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "query": {
                "is_input": true,
                "column": "query",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": []
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "margin": 1.0
            }
```

(continues on next page)

(continued from previous page)

```
        },
        "encode_relations": true,
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "representation": {
            "BiLSTMDocAttention": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
                    "num_layers": 1,
                    "bidirectional": true,
                    "pack_sequence": true,
                    "disable_sort_in_jit": false
                },
                "pooling": {
                    "SelfAttention": {
                        "attn_dimension": 64,
                        "dropout": 0.4
                    }
                },
                "mlp_decoder": null
            }
        },
        "shared_representations": true,
```

(continues on next page)

(continued from previous page)

```
        "decoder_output_dim": 64
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false
    }
}
```

### RoBERTaNERTask.Config

**Component:** *RoBERTaNERTask*

**class** RoBERTaNERTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**model:** *RoBERTaWordTaggingModel.Config* = *RoBERTaWordTaggingModel.Config*()

**metric_reporter:** *NERMetricReporter.Config* = *NERMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "tokens": {
                "is_input": true,
                "columns": [
                    "text"
                ],
                "tokenizer": {
                    "GPT2BPETokenizer": {
                        "bpe_encoder_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/encoder.json",
                        "bpe_vocab_path": "manifold://pytext_training/tree/static/
↪vocabs/bpe/gpt2/vocab.bpe",
                        "lowercase": false
                    }
                },
                "base_tokenizer": null,
                "vocab_file": "gpt2_bpe_dict",
                "max_seq_len": 256,
                "add_selfie_token": false,
                "labels_columns": [
```

(continues on next page)

(continued from previous page)

```
                    "label"
                ],
                "labels": []
            }
        },
        "encoder": {
            "RoBERTaEncoderJit": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "output_dropout": 0.4,
                "embedding_dim": 768,
                "pooling": "cls_token",
                "export": false,
                "projection_dim": 0,
                "normalize_output_rep": false,
                "pretrained_encoder": {
                    "load_path": "public",
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null
                }
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null,
            "layer_norm": false,
            "dropout": 0.0,
            "bias": true,
            "activation": "relu",
            "temperature": 1.0,
            "spectral_normalization": false
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": {},
            "ignore_pad_in_loss": true
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false
    }
```

(continues on next page)

(continued from previous page)

```
}
```

### SemanticParsingTask.Config

**Component:** *SemanticParsingTask*

**class** SemanticParsingTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *HogwildTrainer.Config* = *HogwildTrainer.Config*()

**use_elastic: Optional[bool] = None**

**model:** *RNNGParser.Config* = *RNNGParser.Config*()

**metric_reporter:** *CompositionalMetricReporter.Config* = *CompositionalMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "real_trainer": {
            "TaskTrainer": {
                "epochs": 10,
                "early_stop_after": 0,
                "max_clip_norm": null,
                "report_train_metrics": true,
```

(continues on next page)

(continued from previous page)

```
                "target_time_limit_seconds": null,
                "do_eval": true,
                "load_best_model_after_train": true,
                "num_samples_to_log_progress": 1000,
                "num_accumulated_batches": 1,
                "num_batches_per_epoch": null,
                "optimizer": {
                    "Adam": {
                        "lr": 0.001,
                        "weight_decay": 1e-05,
                        "eps": 1e-08
                    }
                },
                "scheduler": null,
                "sparsifier": null,
                "fp16_args": {
                    "FP16OptimizerFairseq": {
                        "init_loss_scale": 128,
                        "scale_window": null,
                        "scale_tolerance": 0.0,
                        "threshold_loss_scale": null,
                        "min_loss_scale": 0.0001
                    }
                },
                "privacy_engine": null,
                "use_tensorboard": false
            }
        },
        "num_workers": 1
    },
    "use_elastic": null,
    "model": {
        "version": 2,
        "lstm": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm_dim": 32,
            "num_layers": 1,
            "bidirectional": true,
            "pack_sequence": true,
            "disable_sort_in_jit": false
        },
        "ablation": {
            "use_buffer": true,
            "use_stack": true,
            "use_action": true,
            "use_last_open_NT_feature": false
        },
        "constraints": {
            "intent_slot_nesting": true,
            "ignore_loss_for_unsupported": false,
            "no_slots_inside_unsupported": true
        },
        "max_open_NT": 10,
```

(continues on next page)

(continued from previous page)

```
        "dropout": 0.1,
        "beam_size": 1,
        "top_k": 1,
        "compositional_type": "blstm",
        "inputs": {
            "tokens": {
                "is_input": true,
                "column": "tokenized_text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "actions": {
                "is_input": true,
                "column": "seqlogical"
            }
        },
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
```

(continues on next page)

(continued from previous page)

```
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false,
        "text_column_name": "tokenized_text"
    }
}
```

### SeqNNTask.Config

**Component:** *SeqNNTask*

**class** SeqNNTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic: Optional[bool] = None**

**model:** *SeqNNModel.Config* = *SeqNNModel.Config*()

**metric_reporter:** *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*(**text_column_names=['text_**

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
```

(continues on next page)

(continued from previous page)

```
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "inputs": {
            "tokens": {
                "is_input": true,
                "column": "text_seq",
                "max_seq_len": null,
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "add_bol_token": false,
                "add_eol_token": false,
                "use_eol_token_for_bol": false,
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
```

(continues on next page)

(continued from previous page)

```
                },
                "max_turn": 50
            },
            "dense": null,
            "labels": {
                "LabelTensorizer": {
                    "is_input": false,
                    "column": "label",
                    "allow_unknown": false,
                    "pad_in_vocab": false,
                    "label_vocab": null,
                    "label_vocab_file": null,
                    "add_labels": null
                }
            }
        },
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "representation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "doc_representation": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "cnn": {
                    "kernel_num": 100,
                    "kernel_sizes": [
                        3,
```

(continues on next page)

(continued from previous page)

```
                    4
                ],
                "weight_norm": false,
                "dilated": false,
                "causal": false
            },
            "pooling": "max"
        },
        "seq_representation": {
            "BiLSTMDocAttention": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "dropout": 0.4,
                    "lstm_dim": 32,
                    "num_layers": 1,
                    "bidirectional": true,
                    "pack_sequence": true,
                    "disable_sort_in_jit": false
                },
                "pooling": {
                    "SelfAttention": {
                        "attn_dimension": 64,
                        "dropout": 0.4
                    }
                },
                "mlp_decoder": null
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null,
        "layer_norm": false,
        "dropout": 0.0,
        "bias": true,
        "activation": "relu",
        "temperature": 1.0,
        "spectral_normalization": false
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
```

(continues on next page)

(continued from previous page)

```
                "CrossEntropyLoss": {}
            },
            "label_weights": null
        }
    },
    "metric_reporter": {
        "ClassificationMetricReporter": {
            "output_path": "/tmp/test_out.txt",
            "pep_format": false,
            "student_column_names": [],
            "log_gradient": false,
            "model_select_metric": "accuracy",
            "target_label": null,
            "text_column_names": [
                "text_seq"
            ],
            "additional_column_names": [],
            "recall_at_precision_thresholds": [
                0.2,
                0.4,
                0.6,
                0.8,
                0.9
            ],
            "is_memory_efficient": false
        }
    }
}
```

### SequenceLabelingTask.Config

**Component:** *SequenceLabelingTask*

**class** SequenceLabelingTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** *Seq2SeqModel.Config* = *Seq2SeqModel.Config*()

**metric_reporter:** *Seq2SeqCompositionalMetricReporter.Config* = *Seq2SeqCompositionalMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
```

(continues on next page)

(continued from previous page)

```
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
```

(continues on next page)

(continued from previous page)

```
    "model": {
        "inputs": {
            "src_seq_tokens": {
                "is_input": true,
                "column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "trg_seq_tokens": {
                "is_input": true,
                "column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "dict_feat": null,
            "contextual_token_embedding": null
        },
        "encoder_decoder": {
            "encoder": {
                "embed_dim": 512,
                "hidden_dim": 512,
                "num_layers": 1,
                "dropout_in": 0.1,
                "dropout_out": 0.1,
                "bidirectional": false
```

(continues on next page)

(continued from previous page)

```
            },
            "decoder": {
                "encoder_hidden_dim": 512,
                "embed_dim": 512,
                "hidden_dim": 512,
                "out_embed_dim": 512,
                "cell_type": "lstm",
                "num_layers": 1,
                "dropout_in": 0.1,
                "dropout_out": 0.1,
                "attention_type": "dot",
                "attention_heads": 8,
                "first_layer_attention": false,
                "averaging_encoder": false
            }
        },
        "source_embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "target_embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
```

(continues on next page)

(continued from previous page)

```
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "dict_embedding": null,
        "contextual_token_embedding": null,
        "output_layer": {
            "loss": {
                "CrossEntropyLoss": {}
            }
        },
        "sequence_generator": {
            "beam_size": 2,
            "targetlen_cap": 100,
            "targetlen_a": 0,
            "targetlen_b": 2,
            "targetlen_c": 2,
            "quantize": true,
            "length_penalty": 0.25,
            "nbest": 2,
            "stop_at_eos": true,
            "record_attention": false
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false,
        "accept_flat_intents_slots": false
    }
}
```

### SquadQATask.Config

**Component:** *SquadQATask*

**class** SquadQATask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** Union[*BertSquadQAModel.Config*, *DrQAModel.Config*] = *DrQAModel.Config*()

**metric_reporter:** *SquadMetricReporter.Config* = *SquadMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
```

(continues on next page)

(continued from previous page)

```
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "DrQAModel": {
            "inputs": {
                "squad_input": {
                    "SquadTensorizer": {
                        "is_input": true,
                        "column": "text",
                        "tokenizer": {
                            "Tokenizer": {
                                "split_regex": "\\W+",
                                "lowercase": true,
                                "use_byte_offsets": false
                            }
                        },
                        "add_bos_token": false,
                        "add_eos_token": false,
                        "use_eos_token_for_bos": false,
                        "max_seq_len": null,
                        "vocab": {
                            "build_from_data": true,
                            "size_from_data": 0,
                            "min_counts": 0,
                            "vocab_files": []
                        },
                        "vocab_file_delimiter": " ",
                        "doc_column": "doc",
                        "ques_column": "question",
                        "answers_column": "answers",
                        "answer_starts_column": "answer_starts",
                        "max_ques_seq_len": 64,
                        "max_doc_seq_len": 256
                    }
                },
                "has_answer": {
                    "LabelTensorizer": {
                        "is_input": false,
                        "column": "has_answer",
                        "allow_unknown": false,
                        "pad_in_vocab": false,
                        "label_vocab": null,
                        "label_vocab_file": null,
                        "add_labels": null
                    }
                }
            },
            "dropout": 0.4,
            "embedding": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
```

(continues on next page)

(continued from previous page)

```
                "shared_module_key": null,
                "embed_dim": 300,
                "embedding_init_strategy": "random",
                "embedding_init_range": null,
                "embeddding_init_std": 0.02,
                "export_input_names": [
                    "tokens_vals"
                ],
                "pretrained_embeddings_path": "/mnt/vol/pytext/users/kushall/
↪pretrained/glove.840B.300d.txt",
                "vocab_file": "",
                "vocab_size": 0,
                "vocab_from_train_data": true,
                "vocab_from_all_data": false,
                "vocab_from_pretrained_embeddings": true,
                "lowercase_tokens": true,
                "min_freq": 1,
                "mlp_layer_dims": [],
                "padding_idx": null,
                "cpu_only": false,
                "skip_header": true,
                "delimiter": " "
            },
            "ques_rnn": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "hidden_size": 32,
                "num_layers": 1,
                "dropout": 0.4,
                "bidirectional": true,
                "rnn_type": "lstm",
                "concat_layers": true
            },
            "doc_rnn": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "hidden_size": 32,
                "num_layers": 1,
                "dropout": 0.4,
                "bidirectional": true,
                "rnn_type": "lstm",
                "concat_layers": true
            },
            "output_layer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {
                    "CrossEntropyLoss": {}
                },
                "ignore_impossible": true,
                "pos_loss_weight": 0.5,
```

(continues on next page)

(continued from previous page)

```
                "has_answer_loss_weight": 0.5,
                "false_label": "False",
                "max_answer_len": 30,
                "hard_weight": 0.0
            },
            "is_kd": false
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false,
        "n_best_size": 5,
        "max_answer_length": 16,
        "ignore_impossible": true,
        "false_label": "False"
    }
}
```

### WordTaggingTask.Config

**Component:** *WordTaggingTask*

**class** WordTaggingTask.**Config**

**Bases:** NewTask.Config

**All Attributes (including base classes)**

**data:** *Data.Config* = *Data.Config*()

**trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**use_elastic:** Optional[bool] = **None**

**model:** *WordTaggingModel.Config* = *WordTaggingModel.Config*()

**metric_reporter:** *SequenceTaggingMetricReporter.Config* = *SequenceTaggingMetricReporter.Config*()

**Default JSON**

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t",
                    "quoted": false,
                    "drop_incomplete_rows": false
                }
            },
```

(continues on next page)

(continued from previous page)

```
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 1000,
                    "num_shuffled_pools": 1
                }
            },
            "sort_key": null,
            "in_memory": true
        }
    },
    "trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "use_elastic": null,
    "model": {
        "WordTaggingModel": {
            "inputs": {
                "tokens": {
                    "is_input": true,
                    "column": "text",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
```

(continues on next page)

(continued from previous page)

```
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null,
                "vocab": {
                    "build_from_data": true,
                    "size_from_data": 0,
                    "min_counts": 0,
                    "vocab_files": []
                },
                "vocab_file_delimiter": " "
            },
            "labels": {
                "is_input": false,
                "slot_column": "slots",
                "text_column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true,
                        "use_byte_offsets": false
                    }
                },
                "allow_unknown": false
            }
        },
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "embeddding_init_std": 0.02,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
            "vocab_size": 0,
            "vocab_from_train_data": true,
            "vocab_from_all_data": false,
            "vocab_from_pretrained_embeddings": false,
            "lowercase_tokens": true,
            "min_freq": 1,
            "mlp_layer_dims": [],
            "padding_idx": null,
            "cpu_only": false,
            "skip_header": true,
            "delimiter": " "
        },
        "representation": {
```

(continues on next page)

(continued from previous page)

```
                "PassThroughRepresentation": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null
                }
            },
            "output_layer": {
                "WordTaggingOutputLayer": {
                    "load_path": null,
                    "save_path": null,
                    "freeze": false,
                    "shared_module_key": null,
                    "loss": {
                        "CrossEntropyLoss": {}
                    },
                    "label_weights": {},
                    "ignore_pad_in_loss": true
                }
            },
            "decoder": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "hidden_dims": [],
                "out_dim": null,
                "layer_norm": false,
                "dropout": 0.0,
                "bias": true,
                "activation": "relu",
                "temperature": 1.0,
                "spectral_normalization": false
            }
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "pep_format": false,
        "student_column_names": [],
        "log_gradient": false
    }
}
```

### 1.21.9 torchscript

#### seq2seq

#### scripted_seq2seq_generator

#### ScriptedSequenceGenerator.Config

**Component:** *ScriptedSequenceGenerator*

**class** ScriptedSequenceGenerator.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**beam_size: int = 2**

**targetlen_cap: int = 100**

**targetlen_a: float = 0**

**targetlen_b: float = 2**

**targetlen_c: float = 2**

**quantize: bool = True**

**length_penalty: float = 0.25**

**nbest: int = 2**

**stop_at_eos: bool = True**

**record_attention: bool = False**

**Default JSON**

```
{
    "beam_size": 2,
    "targetlen_cap": 100,
    "targetlen_a": 0,
    "targetlen_b": 2,
    "targetlen_c": 2,
    "quantize": true,
    "length_penalty": 0.25,
    "nbest": 2,
    "stop_at_eos": true,
    "record_attention": false
}
```

### 1.21.10 trainers

#### ensemble_trainer

#### EnsembleTrainer.Config

**Component:** *EnsembleTrainer*

**class** EnsembleTrainer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**real_trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**Default JSON**

```
{
    "real_trainer": {
        "TaskTrainer": {
            "epochs": 10,
```

(continues on next page)

(continued from previous page)

```
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    }
}
```

### hogwild_trainer

### HogwildTrainer.Config

**Component:** *HogwildTrainer*

**class** HogwildTrainer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**real_trainer:** *TaskTrainer.Config* = *TaskTrainer.Config*()

**num_workers: int = 1**

**Default JSON**

```
{
    "real_trainer": {
        "TaskTrainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
```

(continues on next page)

(continued from previous page)

```
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "load_best_model_after_train": true,
            "num_samples_to_log_progress": 1000,
            "num_accumulated_batches": 1,
            "num_batches_per_epoch": null,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05,
                    "eps": 1e-08
                }
            },
            "scheduler": null,
            "sparsifier": null,
            "fp16_args": {
                "FP16OptimizerFairseq": {
                    "init_loss_scale": 128,
                    "scale_window": null,
                    "scale_tolerance": 0.0,
                    "threshold_loss_scale": null,
                    "min_loss_scale": 0.0001
                }
            },
            "privacy_engine": null,
            "use_tensorboard": false
        }
    },
    "num_workers": 1
}
```

### HogwildTrainer_Deprecated.Config

**Component:** *HogwildTrainer_Deprecated*

**class** HogwildTrainer_Deprecated.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**real_trainer:** *Trainer.Config* = *Trainer.Config*()

**num_workers: int = 1**

**Default JSON**

```
{
    "real_trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "load_best_model_after_train": true,
```

(continues on next page)

(continued from previous page)

```
        "num_samples_to_log_progress": 1000,
        "num_accumulated_batches": 1,
        "num_batches_per_epoch": null,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05,
                "eps": 1e-08
            }
        },
        "scheduler": null,
        "sparsifier": null,
        "fp16_args": {
            "FP16OptimizerFairseq": {
                "init_loss_scale": 128,
                "scale_window": null,
                "scale_tolerance": 0.0,
                "threshold_loss_scale": null,
                "min_loss_scale": 0.0001
            }
        },
        "privacy_engine": null,
        "use_tensorboard": false
    },
    "num_workers": 1
}
```

### trainer

#### TaskTrainer.Config

**Component:** *TaskTrainer*

**class** TaskTrainer.**Config**

**Bases:** Trainer.Config

Make mypy happy

**All Attributes (including base classes)**

**epochs: int = 10**

**early_stop_after: int = 0**

**max_clip_norm: Optional[float] = None**

**report_train_metrics: bool = True**

**target_time_limit_seconds: Optional[int] = None**

**do_eval: bool = True**

**load_best_model_after_train: bool = True**

**num_samples_to_log_progress: int = 1000**

**num_accumulated_batches: int = 1**

**num_batches_per_epoch: Optional[int] = None**

**optimizer:** *Optimizer.Config* = *Adam.Config*()

**scheduler:** Optional[*Scheduler.Config*] = `None`

**sparsifier:** Optional[*Sparsifier.Config*] = `None`

**fp16_args:** *FP16Optimizer.Config* = *FP16OptimizerFairseq.Config*()

**privacy_engine:** Optional[*PrivacyEngine.Config*] = `None`

**use_tensorboard:** bool = `False`

**Default JSON**

```
{
    "epochs": 10,
    "early_stop_after": 0,
    "max_clip_norm": null,
    "report_train_metrics": true,
    "target_time_limit_seconds": null,
    "do_eval": true,
    "load_best_model_after_train": true,
    "num_samples_to_log_progress": 1000,
    "num_accumulated_batches": 1,
    "num_batches_per_epoch": null,
    "optimizer": {
        "Adam": {
            "lr": 0.001,
            "weight_decay": 1e-05,
            "eps": 1e-08
        }
    },
    "scheduler": null,
    "sparsifier": null,
    "fp16_args": {
        "FP16OptimizerFairseq": {
            "init_loss_scale": 128,
            "scale_window": null,
            "scale_tolerance": 0.0,
            "threshold_loss_scale": null,
            "min_loss_scale": 0.0001
        }
    },
    "privacy_engine": null,
    "use_tensorboard": false
}
```

### Trainer.Config

**Component:** *Trainer*

**class** Trainer.**Config**

**Bases:** *ConfigBase*

**All Attributes (including base classes)**

**epochs:** int = `10` Training epochs

**early_stop_after:** int = `0` Stop after how many epochs when the eval metric is not improving

**max_clip_norm:** Optional[float] = `None` Clip gradient norm if set

**report_train_metrics: bool = `True`** Whether metrics on training data should be computed and reported.

**target_time_limit_seconds: Optional[int] = `None`** Target time limit for training, default (None) to no time limit.

**do_eval: bool = `True`** Whether to do evaluation and model selection based on it.

**load_best_model_after_train: bool = `True`**

**num_samples_to_log_progress: int = `1000`** Number of samples for logging training progress.

**num_accumulated_batches: int = `1`** Number of forward & backward per batch before update gradients, the actual_batch_size = batch_size x num_accumulated_batches

**num_batches_per_epoch: Optional[int] = `None`** Define epoch as a fixed number of batches. Subsequent epochs will continue to iterate through the data, cycling through it when they reach the end. If not set, use exactly one pass through the dataset as one epoch. This configuration only affects the train epochs, test and eval will always test their entire datasets.

**optimizer: *Optimizer.Config* = *Adam.Config*()** config for optimizer, used in parameter update

**scheduler: Optional[*Scheduler.Config*] = `None`**

**sparsifier: Optional[*Sparsifier.Config*] = `None`**

**fp16_args: *FP16Optimizer.Config* = *FP16OptimizerFairseq.Config*()** Define arguments for fp16 training. A fp16_optimizer will be created and wraps the original optimizer, which will scale loss during backward and master weight will be maintained on original optimizer. https://arxiv.org/abs/1710.03740

**privacy_engine: Optional[*PrivacyEngine.Config*] = `None`**

**use_tensorboard: bool = `False`**

**Subclasses**

- `TaskTrainer.Config`

**Default JSON**

```
{
    "epochs": 10,
    "early_stop_after": 0,
    "max_clip_norm": null,
    "report_train_metrics": true,
    "target_time_limit_seconds": null,
    "do_eval": true,
    "load_best_model_after_train": true,
    "num_samples_to_log_progress": 1000,
    "num_accumulated_batches": 1,
    "num_batches_per_epoch": null,
    "optimizer": {
        "Adam": {
            "lr": 0.001,
            "weight_decay": 1e-05,
            "eps": 1e-08
        }
    },
    "scheduler": null,
    "sparsifier": null,
    "fp16_args": {
        "FP16OptimizerFairseq": {
            "init_loss_scale": 128,
```

(continues on next page)

(continued from previous page)

```
            "scale_window": null,
            "scale_tolerance": 0.0,
            "threshold_loss_scale": null,
            "min_loss_scale": 0.0001
        }
    },
    "privacy_engine": null,
    "use_tensorboard": false
}
```

#### TrainerBase.Config

**Component:** *TrainerBase*

**class** TrainerBase.**Config**

**Bases:** Component.Config

**All Attributes (including base classes)**

**Default JSON**

```
{}
```

## 1.22 pytext package

### 1.22.1 Subpackages

#### pytext.common package

#### Submodules

#### pytext.common.constants module

**class** pytext.common.constants.**BatchContext**

Bases: object

**IGNORE_LOSS = 'ignore_loss'**

**INDEX = 'row_index'**

**TASK_NAME = 'task_name'**

**class** pytext.common.constants.**DFColumn**

Bases: object

**ALIGNMENT = 'alignment'**

**CONTEXT_SEQUENCE = 'context_sequence'**

**DENSE_FEAT = 'dense_feat'**

**DICT_FEAT = 'dict_feat'**

**DOC_LABEL = 'doc_label'**

DOC_WEIGHT = 'doc_weight'

LANGUAGE_ID = 'lang'

MODEL_FEATS = 'model_feats'

RAW_FEATS = 'raw_feats'

SEQLOGICAL = 'seqlogical'

SOURCE_FEATS = 'source_feats'

SOURCE_SEQUENCE = 'source_sequence'

TARGET_LABELS = 'target_labels'

TARGET_LOGITS = 'target_logits'

TARGET_PROBS = 'target_probs'

TARGET_SEQUENCE = 'target_sequence'

TARGET_TOKENS = 'target_tokens'

TOKEN_RANGE = 'token_range'

UTTERANCE = 'text'

WORD_LABEL = 'word_label'

WORD_WEIGHT = 'word_weight'

**class** pytext.common.constants.**DatasetFieldName**

Bases: object

CHAR_FIELD = 'char_feat'

CONTEXTUAL_TOKEN_EMBEDDING = 'contextual_token_embedding'

DENSE_FIELD = 'dense_feat'

DICT_FIELD = 'dict_feat'

DOC_LABEL_FIELD = 'doc_label'

DOC_WEIGHT_FIELD = 'doc_weight'

LANGUAGE_ID_FIELD = 'lang'

NUM_TOKENS = 'num_tokens'

RAW_DICT_FIELD = 'sparsefeat'

RAW_SEQUENCE = 'raw_sequence'

RAW_WORD_LABEL = 'raw_word_label'

SEQ_FIELD = 'seq_word_feat'

SEQ_LENS = 'seq_lens'

SOURCE_SEQ_FIELD = 'source_sequence'

TARGET_SEQ_FIELD = 'target_sequence'

TARGET_SEQ_LENS = 'target_seq_lens'

TEXT_FIELD = 'word_feat'

TOKENS = 'tokens'

**TOKEN_INDICES = 'token_indices'**

**TOKEN_RANGE = 'token_range'**

**UTTERANCE_FIELD = 'utterance'**

**WORD_LABEL_FIELD = 'word_label'**

**WORD_WEIGHT_FIELD = 'word_weight'**

**class** pytext.common.constants.**PackageFileName**

Bases: object

**RAW_EMBED = 'pretrained_embed_raw'**

**SERIALIZED_EMBED = 'pretrained_embed_pt_serialized'**

**class** pytext.common.constants.**Padding**

Bases: object

**DEFAULT_LABEL_PAD_IDX = -1**

**WORD_LABEL_PAD = 'PAD_LABEL'**

**WORD_LABEL_PAD_IDX = 0**

**class** pytext.common.constants.**RawExampleFieldName**

Bases: object

**ROW_INDEX = 'row_index'**

**class** pytext.common.constants.**SpecialTokens**

Bases: object

**BOL = '__BEGIN_OF_LIST__'**

**BOS = '__BEGIN_OF_SENTENCE__'**

**BYTE_BOS = '^'**

**BYTE_EOS = '#'**

**BYTE_SPACE = ' '**

**EOL = '__END_OF_LIST__'**

**EOS = '__END_OF_SENTENCE__'**

**MASK = '__MASK__'**

**PAD = '__PAD__'**

**SELFIE_RAW_IMAGE = '__RAW_IMAGE__'**

**UNK = '__UNKNOWN__'**

**class** pytext.common.constants.**Stage**

Bases: enum.Enum

An enumeration.

**EVAL = 'Evaluation'**

**OTHERS = 'Others'**

**TEST = 'Test'**

**TRAIN = 'Training'**

**class** pytext.common.constants.**Token**(*input_str*)
Bases: str

**class** pytext.common.constants.**VocabMeta**
Bases: object

**EOS_SEQ = '</s_seq>'**

**EOS_TOKEN = '</s>'**

**INIT_SEQ = '<s_seq>'**

**INIT_TOKEN = '<s>'**

**PAD_SEQ = '<pad_seq>'**

**PAD_TOKEN = '<pad>'**

**UNK_NUM_TOKEN = '<unk>-NUM'**

**UNK_TOKEN = '<unk>'**

### pytext.common.utils module

pytext.common.utils.**eprint**(**args*, ***kwargs*)

### Module contents

### pytext.config package

### Submodules

### pytext.config.component module

**class** pytext.config.component.**Component**(*config=None*, **args*, ***kwargs*)
Bases: object

**classmethod from_config**(*config*, **args*, ***kwargs*)

**class** pytext.config.component.**ComponentMeta**
Bases: type

**class** pytext.config.component.**ComponentType**
Bases: enum.Enum

An enumeration.

**BATCHER = 'batcher'**

**BATCH_SAMPLER = 'batch_sampler'**

**COLUMN = 'column'**

**DATA_HANDLER = 'data_handler'**

**DATA_SOURCE = 'data_source'**

**DATA_TYPE = 'data_type'**

**EXPORTER = 'exporter'**

**FEATURIZER = 'featurizer'**

**LOSS = 'loss'**

**MASKING_FUNCTION = 'masking_function'**

**METRIC_REPORTER = 'metric_reporter'**

**MODEL = 'model'**

**MODEL2 = 'model2'**

**MODULE = 'module'**

**OPTIMIZER = 'optimizer'**

**PREDICTOR = 'predictor'**

**PRIVACY_ENGINE = 'privacy_engine'**

**SCHEDULER = 'scheduler'**

**SPARSIFIER = 'sparsifier'**

**TASK = 'task'**

**TENSORIZER = 'tensorizer'**

**TOKENIZER = 'tokenizer'**

**TRAINER = 'trainer'**

**class** pytext.config.component.**Registry**

Bases: object

**classmethod add**(*component_type: pytext.config.component.ComponentType, cls_to_add: Type[CT_co], config_cls: Type[CT_co]*)

**classmethod configs**(*component_type: pytext.config.component.ComponentType*) → Tuple[Type[CT_co], ...]

**classmethod get**(*component_type: pytext.config.component.ComponentType, config_cls: Type[CT_co]*) → Type[CT_co]

**classmethod subconfigs**(*config_cls: Type[CT_co]*) → Tuple[Type[CT_co], ...]

**classmethod values**(*component_type: pytext.config.component.ComponentType*) → Tuple[Type[CT_co], ...]

**exception** pytext.config.component.**RegistryError**

Bases: Exception

pytext.config.component.**create_component**(*component_type: pytext.config.component.ComponentType, config: Any, *args, **kwargs*)

pytext.config.component.**create_data_handler**(*data_handler_config, *args, **kwargs*)

pytext.config.component.**create_exporter**(*exporter_config, *args, **kwargs*)

pytext.config.component.**create_featurizer**(*featurizer_config, *args, **kwargs*)

pytext.config.component.**create_loss**(*loss_config, *args, **kwargs*)

pytext.config.component.**create_metric_reporter**(*module_config, *args, **kwargs*)

pytext.config.component.**create_model**(*model_config, *args, **kwargs*)

pytext.config.component.**create_optimizer**(*optimizer_config, model: torch.nn.modules.module.Module, *args, **kwargs*)

pytext.config.component.**create_predictor**(*predictor_config*, **args*, ***kwargs*)

pytext.config.component.**create_privacy_engine**(*privacy_engine_config*, **args*, ***kwargs*)

pytext.config.component.**create_scheduler**(*scheduler_config*, *optimizer*, **args*, ***kwargs*)

pytext.config.component.**create_sparsifier**(*sparsifier_config*, **args*, ***kwargs*)

pytext.config.component.**create_trainer**(*trainer_config*, *model: torch.nn.modules.module.Module*, **args*, ***kwargs*)

pytext.config.component.**get_component_name**(*obj*)
Return the human-readable name of the class of *obj*. Document the type of a config field and can be used as a Union value in a json config.

pytext.config.component.**register_tasks**(*task_cls: Union[Type[CT_co], List[Type[CT_co]]]*)
Task classes are already added to registry during declaration, pass them as parameters here just to make sure they're imported

### pytext.config.config_adapter module

pytext.config.config_adapter.**create_parameter**(*config*, *path_str*, *value*)

pytext.config.config_adapter.**delete_parameter**(*config*, *path_str*)

pytext.config.config_adapter.**deprecate**(*json_config*, *t*)

pytext.config.config_adapter.**doc_model_deprecated**(*json_config*)
Rename DocModel to DocModel_Deprecated.

pytext.config.config_adapter.**downgrade_one_version**(*json_config*)

pytext.config.config_adapter.**ensemble_task_deprecated**(*json_config*)
Rename tasks with new API consistently

pytext.config.config_adapter.**find_dicts_containing_key**(*json_config*, *key*)

pytext.config.config_adapter.**find_parameter**(*config*, *path_str*)

pytext.config.config_adapter.**fix_fl_local_optimizer_and_trainer**(*json_config*)
a) Change FL local optimizer from optimizer:{SGD:{lr=0.1, momentum=0.2}} to optimizer:{lr=0.1, momentum=0.2} b) Replace trainer:{FLSyncTrainer:{foo}} by trainer:{fl_trainer:{foo, type:SyncTrainer}} Same for FLAsyncTrainer

pytext.config.config_adapter.**flatten_deprecated_ensemble_config**(*json_config*)

pytext.config.config_adapter.**get_json_config_iterator**(*json_config*, *lookup_key*)

pytext.config.config_adapter.**get_name_from_options**(*export_config*)
Reverse engineer which model is which based on recognized export configurations. If the export configurations don't adhere to the set of recognized backends, then set target name to unknown

pytext.config.config_adapter.**is_type_specifier**(*json_dict*)
If a config object is a class, it might have a level which is a type specifier, with one key corresponding to the name of whichever type it is. These types should not be explicitly named in the path.

pytext.config.config_adapter.**lm_model_deprecated**(*json_config*)
Rename LM model to _Deprecated (LMTask is already deprecated in v5)

pytext.config.config_adapter.**migrate_to_new_data_handler**(*task*, *columns*)

pytext.config.config_adapter.**move_epoch_size**(*json_config*)

pytext.config.config_adapter.**new_tasks_rename**(*json_config*)
Rename tasks with new API consistently

pytext.config.config_adapter.**old_tasks_deprecated**(*json_config*)
Rename tasks with data_handler config to _Deprecated

pytext.config.config_adapter.**register_adapter**(*from_version*)

pytext.config.config_adapter.**register_down_grade_adapter**(*from_version*)

pytext.config.config_adapter.**remove_docclassificationtask_deprecated**(*json_config*)

pytext.config.config_adapter.**remove_lmtask_deprecated**(*json_config*)

pytext.config.config_adapter.**rename**(*json_config*, *old_name*, *new_name*)

pytext.config.config_adapter.**rename_bitransformer_inputs**(*json_config*)
In "BiTransformer" model, rename input "characters" -> "bytes" and update subfields.

pytext.config.config_adapter.**rename_fl_task**(*json_config*)

pytext.config.config_adapter.**rename_parameter**(*config*, *old_path*, *new_path*, *transform=<function <lambda>>*)
A powerful tool for writing config adapters, this allows you to specify a JSON-style path for an old and new config parameter. For instance

rename_parameter(config, "task.data.epoch_size", "task.trainer.batches_per_epoch")

will look through the config for task.data.epoch_size, including moving through explicitly specified types. If it's specified, it will delete the value and set it in task.trainer.num_batches_per_epoch instead, creating trainer as an empty dictionary if necessary.

pytext.config.config_adapter.**rename_tensorizer_vocab_params**(*json_config*)

pytext.config.config_adapter.**update_to_version**(*json_config*, *expected_version=26*)

pytext.config.config_adapter.**upgrade_export_config**(*json_config*)
Upgrade model export related config fields to the new "export" section.

pytext.config.config_adapter.**upgrade_if_xlm**(*json_config*)
Make *XLMModel* Union changes for encoder and tokens config. Since they are now unions, insert the old class into the config if no class name is mentioned.

pytext.config.config_adapter.**upgrade_one_version**(*json_config*)

pytext.config.config_adapter.**upgrade_padding**(*json_config*)
Upgrade config option padding_control to seq_padding_control.

pytext.config.config_adapter.**upgrade_to_latest**(*json_config*)

pytext.config.config_adapter.**v0_to_v1**(*json_config*)

pytext.config.config_adapter.**v12_to_v13**(*json_config*)
remove_output_encoded_layers(json_config)

pytext.config.config_adapter.**v1_to_v2**(*json_config*)

pytext.config.config_adapter.**v22_to_v23**(*json_config*)
Upgrade by adding read_chunk_size option

pytext.config.config_adapter.**v23_to_v22**(*json_config*)
Upgrade by removing read_chunk_size option

pytext.config.config_adapter.**v23_to_v24**(*json_config*)
No-op since export_list is optional

pytext.config.config_adapter.**v24_to_v23**(*json_config*)
Downgrade by removing export_list option

pytext.config.config_adapter.**v24_to_v25**(*json_config*)
Upgrade by adding max_input_text_length option and default to None

pytext.config.config_adapter.**v25_to_v24**(*json_config*)
Downgrade by removing max_input_text_length option for SentencePieceTokenizer

pytext.config.config_adapter.**v25_to_v26**(*json_config*)

pytext.config.config_adapter.**v26_to_v25**(*json_config*)
Downgrade by removing target option from all exports in export_list

pytext.config.config_adapter.**v2_to_v3**(*json_config*)
Optimizer and Scheduler configs used to be part of the task config, they now live in the trainer's config.

pytext.config.config_adapter.**v3_to_v4**(*json_config*)
Key for provding the path for contextual token embedding has changed from *pretrained_model_embedding* to *contextual_token_embedding. This affects the 'features* section of the config.

pytext.config.config_adapter.**v6_to_v7**(*json_config*)
Make *LabelTensorizer* expansible. If the *labels* field should be an instance of *LabelTensorizer*, convert it to'{LabelTensorizer: labels}'.

### pytext.config.contextual_intent_slot module

**class** pytext.config.contextual_intent_slot.**ExtraField**
Bases: object

**DOC_WEIGHT = 'doc_weight'**

**RAW_WORD_LABEL = 'raw_word_label'**

**TOKEN_RANGE = 'token_range'**

**UTTERANCE = 'utterance'**

**WORD_WEIGHT = 'word_weight'**

**class** pytext.config.contextual_intent_slot.**ModelInput**
Bases: object

**CHAR = 'char_feat'**

**CONTEXTUAL_TOKEN_EMBEDDING = 'contextual_token_embedding'**

**DENSE = 'dense_feat'**

**DICT = 'dict_feat'**

**SEQ = 'seq_word_feat'**

**TEXT = 'word_feat'**

**class** pytext.config.contextual_intent_slot.**ModelInputConfig**(***kwargs*)
Bases: pytext.config.module_config.Module.Config

**char_feat = None**

**contextual_token_embedding = None**

**dense_feat = None**

**dict_feat = None**

**seq_word_feat = <pytext.config.field_config.WordFeatConfig object>**

**word_feat = <pytext.config.field_config.WordFeatConfig object>**

### pytext.config.doc_classification module

**class** pytext.config.doc_classification.**ExtraField**

Bases: object

**RAW_TEXT = 'text'**

**class** pytext.config.doc_classification.**ModelInput**

Bases: object

**CHAR_FEAT = 'char_feat'**

**CONTEXTUAL_TOKEN_EMBEDDING = 'contextual_token_embedding'**

**DENSE_FEAT = 'dense_feat'**

**DICT_FEAT = 'dict_feat'**

**SEQ_LENS = 'seq_lens'**

**WORD_FEAT = 'word_feat'**

**class** pytext.config.doc_classification.**ModelInputConfig**(***kwargs*)

Bases: pytext.config.module_config.Module.Config

**char_feat = None**

**contextual_token_embedding = None**

**dense_feat = None**

**dict_feat = None**

**word_feat = <pytext.config.field_config.WordFeatConfig object>**

### pytext.config.field_config module

pytext.config.field_config.**CharFeatConfig**

alias of *pytext.config.field_config.CharFeatConfig*

pytext.config.field_config.**ContextualTokenEmbeddingConfig**

alias of *pytext.config.field_config.ContextualTokenEmbeddingConfig*

pytext.config.field_config.**DictFeatConfig**

alias of *pytext.config.field_config.DictFeatConfig*

**class** pytext.config.field_config.**DocLabelConfig**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

**export_output_names = ['doc_scores']**

**label_weights = {}**

**target_prob = False**

**class** pytext.config.field_config.**EmbedInitStrategy**

Bases: enum.Enum

An enumeration.

**RANDOM = 'random'**

**ZERO = 'zero'**

**class** pytext.config.field_config.**FeatureConfig**(***kwargs*)
Bases: pytext.config.module_config.Module.Config

**char_feat = None**

**contextual_token_embedding = None**

**dense_feat = None**

**dict_feat = None**

**seq_word_feat = None**

**word_feat = <pytext.config.field_config.WordFeatConfig object>**

**class** pytext.config.field_config.**FloatVectorConfig**(***kwargs*)
Bases: *pytext.config.pytext_config.ConfigBase*

**dim = 0**

**dim_error_check = False**

**export_input_names = ['float_vec_vals']**

pytext.config.field_config.**MLPFeatConfig**
alias of *pytext.config.field_config.MLPFeatConfig*

**class** pytext.config.field_config.**Target**
Bases: object

**DOC_LABEL = 'doc_label'**

**TARGET_LABEL_FIELD = 'target_label'**

**TARGET_LOGITS_FIELD = 'target_logit'**

**TARGET_PROB_FIELD = 'target_prob'**

pytext.config.field_config.**WordFeatConfig**
alias of *pytext.config.field_config.WordFeatConfig*

**class** pytext.config.field_config.**WordLabelConfig**(***kwargs*)
Bases: *pytext.config.pytext_config.ConfigBase*

**export_output_names = ['word_scores']**

**use_bio_labels = False**

### pytext.config.module_config module

**class** pytext.config.module_config.**Activation**
Bases: enum.Enum

An enumeration.

**GELU = 'gelu'**

**GLU = 'glu'**

**LEAKYRELU = 'leakyrelu'**

**RELU = 'relu'**

**TANH = 'tanh'**

**class** pytext.config.module_config.**CNNParams**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

**causal = False**

**dilated = False**

**kernel_num = 100**

**kernel_sizes = [3, 4]**

**weight_norm = False**

**class** pytext.config.module_config.**ExporterType**

Bases: enum.Enum

An enumeration.

**INIT_PREDICT = 'init_predict'**

**PREDICTOR = 'predictor'**

pytext.config.module_config.**ModuleConfig**

alias of *pytext.config.module_config.ModuleConfig*

**class** pytext.config.module_config.**PerplexityType**

Bases: enum.Enum

An enumeration.

**EOS = 'eos'**

**MAX = 'max'**

**MEAN = 'mean'**

**MEDIAN = 'median'**

**MIN = 'min'**

**class** pytext.config.module_config.**PoolingType**

Bases: enum.Enum

An enumeration.

**LOGSUMEXP = 'logsumexp'**

**MAX = 'max'**

**MEAN = 'mean'**

**NONE = 'none'**

**class** pytext.config.module_config.**SlotAttentionType**

Bases: enum.Enum

An enumeration.

**CONCAT = 'concat'**

**DOT = 'dot'**

**MULTIPLY = 'multiply'**

**NO_ATTENTION = 'no_attention'**

### pytext.config.pair_classification module

**class** pytext.config.pair_classification.**ExtraField**
Bases: object

**UTTERANCE_PAIR = 'utterance'**

**class** pytext.config.pair_classification.**ModelInput**
Bases: object

**TEXT1 = 'text1'**

**TEXT2 = 'text2'**

**class** pytext.config.pair_classification.**ModelInputConfig**(***kwargs*)
Bases: pytext.config.module_config.Module.Config

**text1 = <pytext.config.field_config.WordFeatConfig object>**

**text2 = <pytext.config.field_config.WordFeatConfig object>**

### pytext.config.pytext_config module

**class** pytext.config.pytext_config.**ConfigBase**(***kwargs*)
Bases: object

**items**()

**class** pytext.config.pytext_config.**ConfigBaseMeta**
Bases: type

**annotations_and_defaults**()

**class** pytext.config.pytext_config.**ExportConfig**(***kwargs*)
Bases: *pytext.config.pytext_config.ConfigBase*

**accelerate = []**

**batch_padding_control = None**

**export_caffe2_path = None**

**export_lite_path = None**

**export_onnx_path = '/tmp/model.onnx'**

**export_torchscript_path = None**

**inference_interface = None**

**seq_padding_control = None**

**target = ''**

**torchscript_quantize = False**

**exception** pytext.config.pytext_config.**InvalidMethodInvocation**(*message*)
Bases: Exception

**class** pytext.config.pytext_config.**LogitsConfig**(***kwargs*)
Bases: *pytext.config.pytext_config.TestConfig*

**batch_size = 16**

**dump_raw_input = False**

**fp16 = False**

**gpus = 1**

**ndigits_precision = 0**

**output_columns = None**

**use_gzip = False**

**class** pytext.config.pytext_config.**PlaceHolder**

Bases: object

**class** pytext.config.pytext_config.**PyTextConfig**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

**accelerate**

**auto_resume_from_snapshot = False**

**batch_padding_control**

**debug_path = '/tmp/model.debug'**

**distributed_world_size = 1**

**export = <pytext.config.pytext_config.ExportConfig object>**

**export_caffe2_path**

**export_check**(*method_name*)

**export_list = []**

**export_onnx_path**

**export_torchscript_path**

**get_export_accelerate**(*index*)

**get_export_batch_padding_control**(*index*)

**get_export_caffe2_path**(*index*)

**get_export_inference_interface**(*index*)

**get_export_onnx_path**(*index*)

**get_export_seq_padding_control**(*index*)

**get_export_target**(*index*)

**get_export_torchscript_path**(*index*)

**get_export_torchscript_quantize**(*index*)

**get_first_config**()

**gpu_streams_for_distributed_training = 1**

**include_dirs = None**

**inference_interface**

**load_snapshot_path = ''**

**modules_save_dir = ''**

**random_seed = 0**

Seed value to seed torch, python, and numpy random generators.

**read_chunk_size = 1000000000**

**report_eval_results = False**

**report_test_results = True**

**save_all_checkpoints = False**

**save_module_checkpoints = False**

**save_snapshot_path = '/tmp/model.pt'**

**seq_padding_control**

**set_export_accelerate**(*acc*, *index*)

**set_export_batch_padding_control**(*batch_padding_control*, *index*)

**set_export_caffe2_path**(*p*, *index*)

**set_export_inference_interface**(*inference_interface*, *index*)

**set_export_onnx_path**(*p*, *index*)

**set_export_seq_padding_control**(*seq_padding_control*, *index*)

**set_export_target**(*tgt*, *index*)

**set_export_torchscript_path**(*p*, *index*)

**set_export_torchscript_quantize**(*quantize*, *index*)

**target**

**test_out_path = '/tmp/test_out.txt'**

**torchscript_quantize**

**use_config_from_snapshot = True**

**use_cuda_for_testing = True**

**use_cuda_if_available = True**

**use_deterministic_cudnn = False**

Whether to allow CuDNN to behave deterministically.

**use_fp16 = False**

**use_tensorboard = True**

**class** pytext.config.pytext_config.**TestConfig**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

**field_names = None**

Field names for the TSV. If this is not set, the first line of each file will be assumed to be a header containing the field names.

**test_out_path = ''**

**test_path = 'test.tsv'**

**use_cuda_if_available = True**

**use_fp16 = False**

**use_tensorboard = True**

### pytext.config.query_document_pairwise_ranking module

**class** pytext.config.query_document_pairwise_ranking.**ModelInput**
Bases: object

**NEG_RESPONSE = 'neg_response'**

**POS_RESPONSE = 'pos_response'**

**QUERY = 'query'**

**class** pytext.config.query_document_pairwise_ranking.**ModelInputConfig**(***kwargs*)
Bases: pytext.config.module_config.Module.Config

**neg_response = <pytext.config.field_config.WordFeatConfig object>**

**pos_response = <pytext.config.field_config.WordFeatConfig object>**

**query = <pytext.config.field_config.WordFeatConfig object>**

### pytext.config.serialize module

**exception** pytext.config.serialize.**ConfigParseError**
Bases: Exception

**exception** pytext.config.serialize.**EnumTypeError**
Bases: *pytext.config.serialize.ConfigParseError*

**exception** pytext.config.serialize.**IncorrectTypeError**
Bases: Exception

**exception** pytext.config.serialize.**MissingValueError**
Bases: *pytext.config.serialize.ConfigParseError*

**exception** pytext.config.serialize.**UnionTypeError**
Bases: *pytext.config.serialize.ConfigParseError*

**exception** pytext.config.serialize.**ValueSerializationError**
Bases: Exception

pytext.config.serialize.**build_subclass_dict**(*subclasses*)

pytext.config.serialize.**component_config_type_from_type_name**(*cls*, *type_name: str*) → Type[CT_co]

pytext.config.serialize.**config_from_json**(*cls*, *json_obj*, *ignore_fields=()*)

pytext.config.serialize.**config_to_json**(*cls*, *config_obj*)

pytext.config.serialize.**parse_config**(*config_json*)
Parse PyTextConfig object from parameter string or parameter file

pytext.config.serialize.**pytext_config_from_json**(*json_obj*, *ignore_fields=()*, *auto_upgrade=True*)

### pytext.config.utils module

pytext.config.utils.**cast_str**(*to_type*, *value*)

pytext.config.utils.**find_param**(*root*, *suffix*, *parent=''*)
Recursively look at all fields in config to find where *suffix* would fit. This is used to change configs so that they don't use default values. Return the list of field paths matching.

pytext.config.utils.**is_component_class**(*obj*)

pytext.config.utils.**replace_param**(*root*, *path_list*, *value*)

pytext.config.utils.**resolve_optional**(*type_v*)
Deal with Optional implemented as Union[type, None]

### Module contents

### pytext.data package

### Subpackages

### pytext.data.data_structures package

### Submodules

### pytext.data.data_structures.annotation module

**class** pytext.data.data_structures.annotation.**Annotation**(*annotation_string: str, utterance: str = '', brackets: str = '[]', combination_labels: bool = True, add_dict_feat: bool = False, accept_flat_intents_slots: bool = False*)
Bases: object

**build_tree**(*accept_flat_intents_slots: bool = False*)

**class** pytext.data.data_structures.annotation.**Intent**(*label*)
Bases: *pytext.data.data_structures.annotation.Node*

**validate_node**(**args*)

**class** pytext.data.data_structures.annotation.**Node**(*label*)
Bases: object

**children_flat_str_spans**()

**flat_str**()

**get_info**()

**get_token_indices**()

**get_token_span**()
0 indexed Like array slicing: For the first 3 tokens, returns 0, 3

**list_ancestors**()

**list_nonTerminals**()
Returns all Intent and Slot nodes subordinate to this node

**list_terminals**()
Returns all Token nodes

**list_tokens**()

**validate_node**(**args*)

**class** pytext.data.data_structures.annotation.**Node_Info**(*node*)
Bases: object

This class extracts the essential information for a mode, for use in rules.

**get_parent**(*node*)

**get_same_span**(*node*)

**class** pytext.data.data_structures.annotation.**Root**
Bases: *pytext.data.data_structures.annotation.Node*

**validate_node**(**args*)

**class** pytext.data.data_structures.annotation.**Slot**(*label*)
Bases: *pytext.data.data_structures.annotation.Node*

**validate_node**(*allow_empty_slots=True*, **args*)

**class** pytext.data.data_structures.annotation.**Token**(*label*, *index*)
Bases: *pytext.data.data_structures.annotation.Node*

**remove**()
Removes this token from the tree

**validate_node**(**args*)

**class** pytext.data.data_structures.annotation.**Token_Info**(*node*)
Bases: object

This class extracts the essential information for a token for use in rules.

**get_parent**(*node*)

**class** pytext.data.data_structures.annotation.**Tree**(*root: pytext.data.data_structures.annotation.Root*, *combination_labels: bool*, *utterance: str = ''*, *validate_tree: bool = True*)
Bases: object

**depth**()

**flat_str**()

**list_tokens**()

**lotv_str**()
LOTV – Limited Output Token Vocabulary We map the terminal tokens in the input to a constant output (SEQLOGICAL_LOTV_TOKEN) to make the parsing task easier for models where the decoding is decoupled from the input (e.g. seq2seq). This way, the model can focus on learning to predict the parse tree, rather than waste effort learning to replicate terminal tokens.

**print_tree**()

**recursive_validation**(*node*, **args*)

**to_actions**()

**validate_tree**(*allow_empty_slots=True*)
This is a method for checking that roots/intents/slots are nested correctly. Root( Intent( Slot( Intent( Slot, etc.) ) ) )

**class** pytext.data.data_structures.annotation.**TreeBuilder**(*combination_labels: bool = True*)

Bases: object

**finalize_tree**(*validate_tree=True*)

**update_tree**(*action*, *label*)

pytext.data.data_structures.annotation.**escape_brackets**(*string: str*) → str

pytext.data.data_structures.annotation.**is_intent_nonterminal**(*node_label: str*) → bool

pytext.data.data_structures.annotation.**is_slot_nonterminal**(*node_label: str*) → bool

pytext.data.data_structures.annotation.**is_unsupported**(*node_label: str*) → bool

pytext.data.data_structures.annotation.**is_valid_nonterminal**(*node_label: str*) → bool

pytext.data.data_structures.annotation.**list_from_actions**(*tokens_str: List[str], actions_vocab: List[str], actions_indices: List[int]*)

### pytext.data.data_structures.node module

**class** pytext.data.data_structures.node.**Node**(*label: str, span: pytext.data.data_structures.node.Span, children: Optional[AbstractSet[Node]] = None, text: str = None*)

Bases: object

Node in an intent-slot tree, representing either an intent or a slot.

**label**
Label of the node.

**Type** str

**span**
Span of the node.

**Type** Span

**children**
Children of the node.

**Type** set of *Node*

**children**

**get_depth**() → int

**label**

**span**

**text**

**class** pytext.data.data_structures.node.**Span**

Bases: tuple

Span of a node in an intent-slot tree.

**start**

Start position of the node.

**end**

End position of the node (exclusive).

**end**

Alias for field number 1

**start**

Alias for field number 0

### Module contents

### pytext.data.featurizer package

### Submodules

### pytext.data.featurizer.featurizer module

**class** pytext.data.featurizer.featurizer.**Featurizer**(*config*, *feature_config: pytext.config.field_config.FeatureConfig*)

Bases: *pytext.config.component.Component*

Featurizer is tasked with performing data preprocessing that should be shared between training and inference, namely, tokenization and gazetteer features alignment.

This is an interface whose featurize() method must be implemented so that the implemented interface can be used with the appropriate data handler.

**featurize**(*input_record: pytext.data.featurizer.featurizer.InputRecord*) → pytext.data.featurizer.featurizer.OutputRecord

**featurize_batch**(*input_record_list: Sequence[pytext.data.featurizer.featurizer.InputRecord]*) → Sequence[pytext.data.featurizer.featurizer.OutputRecord]

Featurize a batch of instances/examples.

**classmethod from_config**(*config*, *feature_config: pytext.config.field_config.FeatureConfig*)

**get_sentence_markers**(*locale=None*)

**class** pytext.data.featurizer.featurizer.**InputRecord**

Bases: tuple

Input data contract between Featurizer and DataHandler.

**locale**

Alias for field number 2

**raw_gazetteer_feats**

Alias for field number 1

**raw_text**

Alias for field number 0

**class** pytext.data.featurizer.featurizer.**OutputRecord**

Bases: tuple

Output data contract between Featurizer and DataHandler.

**characters**
Alias for field number 5

**contextual_token_embedding**
Alias for field number 6

**dense_feats**
Alias for field number 7

**gazetteer_feat_lengths**
Alias for field number 3

**gazetteer_feat_weights**
Alias for field number 4

**gazetteer_feats**
Alias for field number 2

**token_ranges**
Alias for field number 1

**tokens**
Alias for field number 0

### pytext.data.featurizer.simple_featurizer module

**class** pytext.data.featurizer.simple_featurizer.**SimpleFeaturizer**(*config, feature_config: pytext.config.field_config.FeatureConfig*)

Bases: *pytext.data.featurizer.featurizer.Featurizer*

Simple featurizer for basic tokenization and gazetteer feature alignment.

**featurize**(*input_record: pytext.data.featurizer.featurizer.InputRecord*) → pytext.data.featurizer.featurizer.OutputRecord
Featurize one instance/example only.

**featurize_batch**(*input_records: Sequence[pytext.data.featurizer.featurizer.InputRecord]*) → Sequence[pytext.data.featurizer.featurizer.OutputRecord]
Featurize a batch of instances/examples.

**get_sentence_markers**(*locale=None*)

**tokenize**(*input_record: pytext.data.featurizer.featurizer.InputRecord*) → pytext.data.featurizer.featurizer.OutputRecord
Tokenize one instance/example only.

**tokenize_batch**(*input_records: Sequence[pytext.data.featurizer.featurizer.InputRecord]*) → Sequence[pytext.data.featurizer.featurizer.OutputRecord]

### Module contents

**class** pytext.data.featurizer.**Featurizer**(*config*, *feature_config: pytext.config.field_config.FeatureConfig*)

Bases: *pytext.config.component.Component*

Featurizer is tasked with performing data preprocessing that should be shared between training and inference, namely, tokenization and gazetteer features alignment.

This is an interface whose featurize() method must be implemented so that the implemented interface can be used with the appropriate data handler.

**featurize**(*input_record: pytext.data.featurizer.featurizer.InputRecord*) → pytext.data.featurizer.featurizer.OutputRecord

**featurize_batch**(*input_record_list: Sequence[pytext.data.featurizer.featurizer.InputRecord]*) → Sequence[pytext.data.featurizer.featurizer.OutputRecord]

Featurize a batch of instances/examples.

**classmethod from_config**(*config*, *feature_config: pytext.config.field_config.FeatureConfig*)

**get_sentence_markers**(*locale=None*)

**class** pytext.data.featurizer.**InputRecord**

Bases: tuple

Input data contract between Featurizer and DataHandler.

**locale**

Alias for field number 2

**raw_gazetteer_feats**

Alias for field number 1

**raw_text**

Alias for field number 0

**class** pytext.data.featurizer.**OutputRecord**

Bases: tuple

Output data contract between Featurizer and DataHandler.

**characters**

Alias for field number 5

**contextual_token_embedding**

Alias for field number 6

**dense_feats**

Alias for field number 7

**gazetteer_feat_lengths**

Alias for field number 3

**gazetteer_feat_weights**

Alias for field number 4

**gazetteer_feats**

Alias for field number 2

**token_ranges**

Alias for field number 1

**tokens**

Alias for field number 0

**class** pytext.data.featurizer.**SimpleFeaturizer**(*config*, *feature_config: pytext.config.field_config.FeatureConfig*)

Bases: *pytext.data.featurizer.featurizer.Featurizer*

Simple featurizer for basic tokenization and gazetteer feature alignment.

**featurize**(*input_record: pytext.data.featurizer.featurizer.InputRecord*) → pytext.data.featurizer.featurizer.OutputRecord

Featurize one instance/example only.

**featurize_batch**(*input_records: Sequence[pytext.data.featurizer.featurizer.InputRecord]*) → Sequence[pytext.data.featurizer.featurizer.OutputRecord]

Featurize a batch of instances/examples.

**get_sentence_markers**(*locale=None*)

**tokenize**(*input_record: pytext.data.featurizer.featurizer.InputRecord*) → pytext.data.featurizer.featurizer.OutputRecord

Tokenize one instance/example only.

**tokenize_batch**(*input_records: Sequence[pytext.data.featurizer.featurizer.InputRecord]*) → Sequence[pytext.data.featurizer.featurizer.OutputRecord]

## pytext.data.sources package

### Submodules

### pytext.data.sources.conllu module

**class** pytext.data.sources.conllu.**CoNLLUNERDataSource**(*language=None*, *train_file=None*, *test_file=None*, *eval_file=None*, *field_names=None*, *delimiter='t'*, ***kwargs*)

Bases: *pytext.data.sources.conllu.CoNLLUPOSDataSource*

Reads an empty line separated data (word label). This data source supports datasets for NER tasks

**class** pytext.data.sources.conllu.**CoNLLUNERFile**(*file*, *delim*, *lang*)

Bases: object

**class** pytext.data.sources.conllu.**CoNLLUPOSDataSource**(*language=None*, *train_file=None*, *test_file=None*, *eval_file=None*, *field_names=None*, *delimiter='t'*, ***kwargs*)

Bases: *pytext.data.sources.data_source.RootDataSource*

DataSource which loads data from CoNLL-U file.

**classmethod from_config**(*config: pytext.data.sources.conllu.CoNLLUPOSDataSource.Config*, *schema: Dict[str, Type[CT_co]], **kwargs*)

**raw_eval_data_generator**()

Returns a generator that yields the EVAL data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_test_data_generator**()
Returns a generator that yields the TEST data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_train_data_generator**()
Returns a generator that yields the TRAIN data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

### pytext.data.sources.data_source module

**class** pytext.data.sources.data_source.**DataSource**(*schema: Dict[str, Type[CT_co]]*)
Bases: *pytext.config.component.Component*

Data sources are simple components that stream data from somewhere using Python's iteration interface. It should expose 3 iterators, "train", "test", and "eval". Each of these should be able to be iterated over any number of times, and iterating over it should yield dictionaries whose values are deserialized python types.

Simply, these data sources exist as an interface to read through datasets in a pythonic way, with pythonic types, and abstract away the form that they are stored in.

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.data_source.**GeneratorIterator**(*generator, *args, **kwargs*)
Bases: object

Create an object which can be iterated over multiple times from a generator call. Each iteration will call the generator and allow iterating over it. This is unsafe to use on generators which have side effects, such as file readers; it's up to the callers to safely manage these scenarios.

**class** pytext.data.sources.data_source.**GeneratorMethodProperty**(*generator*)
Bases: object

Identify a generator method as a property. This will allow instances to iterate over the property multiple times, and not consume the generator. It accomplishes this by wrapping the generator and creating multiple generator instances if iterated over multiple times.

**class** pytext.data.sources.data_source.**RawExample**
Bases: dict

A wrapper class for a single example row with a dict interface. This is here for any logic we want row objects to have that dicts don't do.

**class** pytext.data.sources.data_source.**RootDataSource**(*schema: Dict[str, Type[CT_co]], column_mapping: Dict[str, str] = ()*)
Bases: *pytext.data.sources.data_source.DataSource*

A data source which actually loads data from a location. This data source needs to be responsible for converting types based on a schema, because it should be the only part of the system that actually needs to understand details about the underlying storage system.

RootDataSource presents a simpler abstraction than DataSource where the rows are automatically converted to the right DataTypes.

A RootDataSource should implement *raw_train_data_generator*, *raw_test_data_generator*, and *raw_eval_data_generator*. These functions should yield dictionaries of raw objects which the loading system can convert using the schema loading functions.

**DATA_SOURCE_TYPES = {<class 'str'>: <function load_text>, typing.Any: <function load**

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**load**(*value*, *schema_type*)

**raw_eval_data_generator**()
Returns a generator that yields the EVAL data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_test_data_generator**()
Returns a generator that yields the TEST data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_train_data_generator**()
Returns a generator that yields the TRAIN data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**classmethod register_type**(*type*)

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.data_source.**RowShardedDataSource**(*data_source: pytext.data.sources.data_source.DataSource*, *rank=0*, *world_size=1*)

Bases: *pytext.data.sources.data_source.ShardedDataSource*

Shards a given datasource by row.

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

**train_unsharded = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.data_source.**SafeFileWrapper**(**args*, ***kwargs*)

Bases: object

A simple wrapper class for files which allows filedescriptors to be managed with normal Python ref counts. Without using this, if you create a file in a from_config you will see a warning along the lines of "ResourceWarning: self._file is acquired but not always released" this is because we're opening a file not in a context manager (with statement). We want to do it this way because it lets us pass a file object to the DataSource, rather than a filename. This exposes a ton more flexibility and testability, passing filenames is one of the paths towards pain.

However, we don't have a clear resource management system set up for configuration. from_config functions are the tool that we have to allow objects to specify how they should be created from a configuration, which generally should only happen from the command line, whereas in eg. a notebook you should build the objects with constructors directly. If building from constructors, you can just open a file and pass it, but from_config here needs to create a file object from a configured filename. Python files don't close automatically, so you also need a system that will close them when the python interpreter shuts down. If you don't, it will print a resource warning at runtime, as the interpreter manually closes the filehandles (although modern OSs are pretty okay with having open file handles, it's hard for me to justify exactly why Python is so strict about this; I think one of the main reasons you might actually care is if you have a writeable file handle it might not have flushed properly when the C runtime exits, but Python doesn't actually distinguish between writeable and non-writeable file handles).

This class is a wrapper that creates a system for (sort-of) safely closing the file handles before the runtime exits. It does this by closing the file when the object's deleter is called. Although the python standard doesn't actually make any guarantees about when deleters are called, CPython is reference counted and so as an mplementation detail will call a deleter whenever the last reference to it is removed, which generally will happen to all objects created during program execution as long as there aren't reference cycles (I don't actually know off-hand whether the cycle collection is run before shutdown, and anyway the cycles would have to include objects that the runtime itself maintains pointers to, which seems like you'd have to work hard to do and wouldn't do accidentally). This isn't true for other python systems like PyPy or Jython which use generational garbage collection and so don't actually always call destructors before the system shuts down, but again this is only really relevant for mutable files.

An alternative implementation would be to build a resource management system into PyText, something like a function that we use for opening system resources that registers the resources and then we make sure are all closed before system shutdown. That would probably technically be the right solution, but I didn't really think of that first and also it's a bit longer to implement.

If you are seeing resource warnings on your system, please file a github issue.

**class** pytext.data.sources.data_source.**ShardedDataSource**(*schema: Dict[str, Type[CT_co]]*)

Bases: *pytext.data.sources.data_source.DataSource*

Base class for sharded data sources.

pytext.data.sources.data_source.**generator_property**

alias of *pytext.data.sources.data_source.GeneratorMethodProperty*

pytext.data.sources.data_source.**load_float**(*f*)

pytext.data.sources.data_source.**load_float_list**(*s*)

pytext.data.sources.data_source.**load_int**(*x*)

pytext.data.sources.data_source.**load_json**(*s*)

pytext.data.sources.data_source.**load_json_string**(*s*)

pytext.data.sources.data_source.**load_slots**(*s*)

pytext.data.sources.data_source.**load_text**(*s*)

### pytext.data.sources.dense_retrieval module

**class** pytext.data.sources.dense_retrieval.**DenseRetrievalDataSource**(*schema, train_filename=None, test_filename=None, eval_filename=None, num_negative_ctxs=1, use_title=True, use_cache=False*)

Bases: *pytext.data.sources.data_source.DataSource*

Data source for DPR (https://github.com/facebookresearch/DPR).

Expects multiline json for lazy loading and improved memory usage. The original DPR files can be converted to multiline json using *jq -c .[]*

**DEFAULT_SCHEMA = {'negative_ctxs': typing.List[str], 'positive_ctx': <class 'str'>,**

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**classmethod from_config**(*config: pytext.data.sources.dense_retrieval.DenseRetrievalDataSource.Config, schema={'negative_ctxs': typing.List[str], 'positive_ctx': <class 'str'>, 'question': <class 'str'>}*)

**process_file**(*fname*, *is_train*)

**read_file**(*fname*)

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

pytext.data.sources.dense_retrieval.**combine_title_text_id**(*ctx*, *use_title*)

### pytext.data.sources.pandas module

**class** pytext.data.sources.pandas.**PandasDataSource**(*train_df: Optional[pandas.core.frame.DataFrame] = None, eval_df: Optional[pandas.core.frame.DataFrame] = None, test_df: Optional[pandas.core.frame.DataFrame] = None, **kwargs*)

Bases: *pytext.data.sources.data_source.RootDataSource*

DataSource which loads data from a pandas DataFrame.

**Inputs:** train_df: DataFrame for training

eval_df: DataFrame for evalu

test_df: DataFrame for test

schema: same as base DataSource, define the list of output values with their types

column_mapping: maps the column names in DataFrame to the name defined in schema

**classmethod from_config**(*config: pytext.data.sources.pandas.PandasDataSource.Config, schema: Dict[str, Type[CT_co]]*)

**raw_eval_data_generator**()

Returns a generator that yields the EVAL data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**static raw_generator**(*df: Optional[pandas.core.frame.DataFrame]*)

**raw_test_data_generator**()

Returns a generator that yields the TEST data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_train_data_generator**()

Returns a generator that yields the TRAIN data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**class** pytext.data.sources.pandas.**SessionPandasDataSource**(*schema: Dict[str, Type[CT_co]], id_col: str, train_df: Optional[pandas.core.frame.DataFrame] = None, eval_df: Optional[pandas.core.frame.DataFrame] = None, test_df: Optional[pandas.core.frame.DataFrame] = None, column_mapping: Dict[str, str] = ()*)

Bases: *pytext.data.sources.pandas.PandasDataSource*, *pytext.data.sources.session.SessionDataSource*

### pytext.data.sources.session module

**class** pytext.data.sources.session.**SessionDataSource**(*id_col, **kwargs*)

Bases: *pytext.data.sources.data_source.RootDataSource*

Data source for session based data, the input data is organized in sessions, each session may have multiple rows. The first column is always the session id. Raw input rows are consolidated by session id and returned as one session per example

**merge_session**(*session*)

### pytext.data.sources.squad module

**class** pytext.data.sources.squad.**SquadDataSource**(*train_filename=None, test_filename=None, eval_filename=None, ignore_impossible=True, max_character_length=1048576, min_overlap=0.1, delimiter='t', quoted=False, schema={'answer_ends': typing.List[int], 'answer_starts': typing.List[int], 'answers': typing.List[str], 'doc': <class 'str'>, 'has_answer': <class 'str'>, 'id': <class 'int'>, 'question': <class 'str'>}*)

Bases: *pytext.data.sources.data_source.DataSource*

Download data from https://rajpurkar.github.io/SQuAD-explorer/ Will return tuples of (doc, question, answer, answer_start, has_answer)

**DEFAULT_SCHEMA = {'answer_ends': typing.List[int], 'answer_starts': typing.List[int]**

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**classmethod from_config**(*config: pytext.data.sources.squad.SquadDataSource.Config, schema={'answer_ends': typing.List[int], 'answer_starts': typing.List[int], 'answers': typing.List[str], 'doc': <class 'str'>, 'has_answer': <class 'str'>, 'id': <class 'int'>, 'question': <class 'str'>}*)

**process_file**(*fname*)

**process_squad_json**(*fname*)

**process_squad_tsv**(*fname*)

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.squad.**SquadDataSourceForKD**(***kwargs*)

Bases: *pytext.data.sources.squad.SquadDataSource*

Squad-like data along with soft labels (logits). Will return tuples of ( doc, question, answer, answer_start, has_answer, start_logits, end_logits, has_answer_logits, pad_mask, segment_labels )

**process_squad_tsv**(*fname*)

### pytext.data.sources.tsv module

**class** pytext.data.sources.tsv.**BlockShardedTSV**(*file, field_names=None, delimiter='t', quoted=False, block_id=0, num_blocks=1, drop_incomplete_rows=False*)

Bases: object

Take a TSV file, split into N pieces (by byte location) and return an iterator on one of the pieces. The pieces are equal by byte size, not by number of rows. Thus, care needs to be taken when using this for distributed training, otherwise number of batches for different workers might be different.

**class** pytext.data.sources.tsv.**BlockShardedTSVDataSource**(*rank=0, world_size=1, **kwargs*)

Bases: *pytext.data.sources.tsv.TSVDataSource*, *pytext.data.sources.data_source.ShardedDataSource*

**train_unsharded = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.tsv.**MultilingualTSVDataSource**(*train_file=None, test_file=None, eval_file=None, field_names=None, delimiter='t', data_source_languages={'eval': ['en'], 'test': ['en'], 'train': ['en']}, language_columns=['language'], **kwargs*)

Bases: *pytext.data.sources.tsv.TSVDataSource*

Data Source for multi-lingual data. The input data can have multiple text fields and each field can either have the same language or different languages. The *data_source_languages* dict contains the language information for each text field and this should match the number of language identifiers specified in *language_columns*.

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.tsv.**SessionTSVDataSource**(*train_file=None, test_file=None, eval_file=None, field_names=None, **kwargs*)

Bases: *pytext.data.sources.tsv.TSVDataSource*, *pytext.data.sources.session.SessionDataSource*

**class** pytext.data.sources.tsv.**TSV**(*file, field_names=None, delimiter='t', quoted=False, drop_incomplete_rows=False*)

Bases: object

**class** pytext.data.sources.tsv.**TSVDataSource**(*train_file=None, test_file=None, eval_file=None, field_names=None, delimiter='t', quoted=False, drop_incomplete_rows=False, **kwargs*)

Bases: *pytext.data.sources.data_source.RootDataSource*

DataSource which loads data from TSV sources. Uses python's csv library.

**classmethod from_config**(*config: pytext.data.sources.tsv.TSVDataSource.Config, schema: Dict[str, Type[CT_co]], **kwargs*)

**raw_eval_data_generator**()

Returns a generator that yields the EVAL data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_test_data_generator**()

Returns a generator that yields the TEST data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_train_data_generator**()

Returns a generator that yields the TRAIN data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

### Module contents

**class** pytext.data.sources.**DataSource**(*schema: Dict[str, Type[CT_co]]*)

Bases: *pytext.config.component.Component*

Data sources are simple components that stream data from somewhere using Python's iteration interface. It should expose 3 iterators, "train", "test", and "eval". Each of these should be able to be iterated over any number of times, and iterating over it should yield dictionaries whose values are deserialized python types.

Simply, these data sources exist as an interface to read through datasets in a pythonic way, with pythonic types, and abstract away the form that they are stored in.

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.**RawExample**

Bases: dict

A wrapper class for a single example row with a dict interface. This is here for any logic we want row objects to have that dicts don't do.

**class** pytext.data.sources.**SquadDataSource**(*train_filename=None, test_filename=None, eval_filename=None, ignore_impossible=True, max_character_length=1048576, min_overlap=0.1, delimiter='t', quoted=False, schema={'answer_ends': typing.List[int], 'answer_starts': typing.List[int], 'answers': typing.List[str], 'doc': <class 'str'>, 'has_answer': <class 'str'>, 'id': <class 'int'>, 'question': <class 'str'>}*)

Bases: *pytext.data.sources.data_source.DataSource*

Download data from https://rajpurkar.github.io/SQuAD-explorer/ Will return tuples of (doc, question, answer, answer_start, has_answer)

**DEFAULT_SCHEMA = {'answer_ends': typing.List[int], 'answer_starts': typing.List[int]**

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**classmethod from_config**(*config: pytext.data.sources.squad.SquadDataSource.Config, schema={'answer_ends': typing.List[int], 'answer_starts': typing.List[int], 'answers': typing.List[str], 'doc': <class 'str'>, 'has_answer': <class 'str'>, 'id': <class 'int'>, 'question': <class 'str'>}*)

**process_file**(*fname*)

**process_squad_json**(*fname*)

**process_squad_tsv**(*fname*)

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

**class** pytext.data.sources.**TSVDataSource**(*train_file=None, test_file=None, eval_file=None, field_names=None, delimiter='t', quoted=False, drop_incomplete_rows=False, **kwargs*)

Bases: *pytext.data.sources.data_source.RootDataSource*

DataSource which loads data from TSV sources. Uses python's csv library.

**classmethod from_config**(*config: pytext.data.sources.tsv.TSVDataSource.Config, schema: Dict[str, Type[CT_co]], **kwargs*)

**raw_eval_data_generator**()

Returns a generator that yields the EVAL data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_test_data_generator**()

Returns a generator that yields the TEST data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_train_data_generator**()

Returns a generator that yields the TRAIN data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**class** pytext.data.sources.**PandasDataSource**(*train_df: Optional[pandas.core.frame.DataFrame] = None, eval_df: Optional[pandas.core.frame.DataFrame] = None, test_df: Optional[pandas.core.frame.DataFrame] = None, **kwargs*)

Bases: *pytext.data.sources.data_source.RootDataSource*

DataSource which loads data from a pandas DataFrame.

**Inputs:** train_df: DataFrame for training

eval_df: DataFrame for evalu

test_df: DataFrame for test

schema: same as base DataSource, define the list of output values with their types

column_mapping: maps the column names in DataFrame to the name defined in schema

**classmethod from_config**(*config: pytext.data.sources.pandas.PandasDataSource.Config, schema: Dict[str, Type[CT_co]]*)

**raw_eval_data_generator**()

Returns a generator that yields the EVAL data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**static raw_generator**(*df: Optional[pandas.core.frame.DataFrame]*)

**raw_test_data_generator**()

Returns a generator that yields the TEST data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**raw_train_data_generator**()

Returns a generator that yields the TRAIN data one item at a time in a dictionary where each key is a field and the value is of the raw type from the source. DataSources need to implement this.

**class** pytext.data.sources.**CoNLLUNERDataSource**(*language=None, train_file=None, test_file=None, eval_file=None, field_names=None, delimiter='t', **kwargs*)

Bases: *pytext.data.sources.conllu.CoNLLUPOSDataSource*

Reads an empty line separated data (word label). This data source supports datasets for NER tasks

**class** pytext.data.sources.**DenseRetrievalDataSource**(*schema, train_filename=None, test_filename=None, eval_filename=None, num_negative_ctxs=1, use_title=True, use_cache=False*)

Bases: *pytext.data.sources.data_source.DataSource*

Data source for DPR (https://github.com/facebookresearch/DPR).

Expects multiline json for lazy loading and improved memory usage. The original DPR files can be converted to multiline json using *jq -c .[]*

**DEFAULT_SCHEMA = {'negative_ctxs': typing.List[str], 'positive_ctx': <class 'str'>,**

**eval = <pytext.data.sources.data_source.GeneratorIterator object>**

**classmethod from_config**(*config: pytext.data.sources.dense_retrieval.DenseRetrievalDataSource.Config, schema={'negative_ctxs': typing.List[str], 'positive_ctx': <class 'str'>, 'question': <class 'str'>}*)

**process_file**(*fname, is_train*)

**read_file**(*fname*)

**test = <pytext.data.sources.data_source.GeneratorIterator object>**

**train = <pytext.data.sources.data_source.GeneratorIterator object>**

### pytext.data.test package

#### Submodules

#### pytext.data.test.batch_sampler_test module

**class** pytext.data.test.batch_sampler_test.**BatchSamplerTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

**setUp**()

Hook method for setting up the test fixture before exercising it.

**test_alternate_prob_batch_sampler**()

**test_eval_batch_sampler**()

**test_prob_batch_sampler**()

**test_round_robin_batch_sampler**()

#### pytext.data.test.data_test module

**class** pytext.data.test.data_test.**BatcherTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

**test_batcher**()

**test_pooling_batcher**()

**class** pytext.data.test.data_test.**DataTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

**setUp**()

Hook method for setting up the test fixture before exercising it.

**test_create_batches**()

**test_create_batches_different_tensorizers**()

**test_create_batches_with_cache**()

**test_create_data_no_batcher_provided**()

**test_data_initializes_tensorsizers**()

**test_data_iterate_multiple_times**()

**test_fp16_padding**()

**test_sort**()

**class** pytext.data.test.data_test.**RawExampleTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

**test_raw_example_hashable**()

### pytext.data.test.dynamic_pooling_batcher_test module

**class** pytext.data.test.dynamic_pooling_batcher_test.**DynamicPoolingBatcherTest**(*methodName='run*

Bases: unittest.case.TestCase

**end_of_scheduler**()

**test_batch_size_greater_than_data**()

**test_exponential_scheduler**()

**test_linear_scheduler**()

**test_step_size**()

### pytext.data.test.mask_tensorizers_test module

**class** pytext.data.test.mask_tensorizers_test.**MaskTensorizersTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

**setUp**()

Hook method for setting up the test fixture before exercising it.

**test_basic_tree_masking**()

**test_mask_all**()

**test_mask_at_depth_k**()

**test_mask_no_op**()

**test_mask_random**()

**test_tree_mask_with_bos_eos**()

### pytext.data.test.pandas_data_source_test module

**class** pytext.data.test.pandas_data_source_test.**PandasDataSourceTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

**test_create_data_source**()

**test_create_from_config**()

**test_empty_data**()

### pytext.data.test.round_robin_batchiterator_test module

**class** pytext.data.test.round_robin_batchiterator_test.**RoundRobinBatchIteratorTest**(*methodNam*

Bases: unittest.case.TestCase

**test_batch_iterator**()

### pytext.data.test.simple_featurizer_test module

**class** pytext.data.test.simple_featurizer_test.**SimpleFeaturizerTest**(*methodName='runTest'*)

Bases: unittest.case.TestCase

**setUp**()
Hook method for setting up the test fixture before exercising it.

**test_convert_to_bytes**()

**test_split_with_regex**()

**test_tokenize**()

**test_tokenize_add_sentence_markers**()

**test_tokenize_dont_lowercase**()

### pytext.data.test.tensorizers_test module

**class** pytext.data.test.tensorizers_test.**BERTTensorizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_bert_pair_tensorizer**()

**test_bert_tensorizer**()

**class** pytext.data.test.tensorizers_test.**CharacterVocabTokenTensorizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_character_vocab_token_tensorizer**()

**class** pytext.data.test.tensorizers_test.**ListTensorizersTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**setUp**()
Hook method for setting up the test fixture before exercising it.

**test_create_label_list_tensors**()

**test_initialize_list_tensorizers**()

**test_label_list_tensors_no_pad_in_vocab**()

**test_label_list_tensors_pad_missing**()

**class** pytext.data.test.tensorizers_test.**LookupTokensTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_lookup_tokens**()

**class** pytext.data.test.tensorizers_test.**RobertaTensorizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_roberta_tensorizer**()

**class** pytext.data.test.tensorizers_test.**SquadForBERTTensorizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_squad_tensorizer**()

**class** pytext.data.test.tensorizers_test.**SquadForRobertaTensorizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_squad_roberta_tensorizer**()

**class** pytext.data.test.tensorizers_test.**SquadTensorizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**setUp**()
Hook method for setting up the test fixture before exercising it.

**test_initialize**()

**test_numberize_with_alphanumeric**()

**test_numberize_with_wordpiece**()

**test_tsv_numberize_with_alphanumeric**()

**class** pytext.data.test.tensorizers_test.**String2DListTensorizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**expected_numberized = [([[2, 3], [4, 5, 6, 3]], [2, 4], 2), ([[2, 3], [4, 5, 6, 0, 0]]**

**expected_tensorized = (tensor([[[2, 3, 1, 1, 1], [4, 5, 6, 3, 1]], [[2, 3, 1, 1, 1], [**

**init_rows = [{'text': [['Move', 'fast'], ['And', 'break', 'things', 'fast']]}]**

**test_original**()

**test_rows = [{'text': [['Move', 'fast'], ['And', 'break', 'things', 'fast']]}, {'text**

**test_torchscriptified**()

**class** pytext.data.test.tensorizers_test.**TensorizersTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**setUp**()
Hook method for setting up the test fixture before exercising it.

**test_annotation_num**()

**test_byte_tensors_error_code**()

**test_create_byte_tensors**()

**test_create_byte_token_tensors**()

**test_create_float_list_seq_tensor**()

**test_create_float_list_tensor**()

**test_create_label_tensors**()

**test_create_label_tensors_add_labels**()

**test_create_label_tensors_label_vocab**()

**test_create_normalized_float_list_tensor**()

**test_float_1D_list_tensorizer**()

**test_float_list_seq_tensor_prepare_input**()

**test_float_list_seq_torchscriptify**()

**test_float_list_tensor_prepare_input**()

**test_gazetteer_tensor**()

**test_gazetteer_tensor_bad_json**()

**test_initialize_label_tensorizer**()

**test_initialize_tensorizers**()

**test_initialize_token_tensorizer**()

**test_integer_1D_list_tensorizer**()

**test_numberize_with_script_token_tensorizer**()

**test_numberize_with_token_tensorizer**()

**test_seq_tensor**()

**test_seq_tensor_max_turn**()

**test_seq_tensor_pad_batch**()

**test_seq_tensor_with_bos_eos_eol_bol**()

**test_tensorize_with_script_token_tensorizer**()

### pytext.data.test.tokenizers_test module

**class** pytext.data.test.tokenizers_test.**GPT2BPETest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_gpt2_bpe_tokenizer**()

**class** pytext.data.test.tokenizers_test.**SentencePieceTokenizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_input_text_truncation**()

**test_tokenize**()

**class** pytext.data.test.tokenizers_test.**TokenizeTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_split_with_regex**()

**test_tokenize**()

**test_tokenize_dont_lowercase**()

**test_tokenize_no_byte_offsets**()

**test_tokenize_use_byte_offsets**()

**class** pytext.data.test.tokenizers_test.**WordpieceTokenizerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_wordpiece_tokenizer**()

### pytext.data.test.tsv_data_source_test module

**class** pytext.data.test.tsv_data_source_test.**BlockShardedTSVDataSourceTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_bad_quoting**()
The text column of the first row of this file opens a quote but does not close it.

**test_quoting**()
The text column of the first row of this file opens a quote but does not close it.

**class** pytext.data.test.tsv_data_source_test.**SessionTSVDataSourceTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**setUp**()
Hook method for setting up the test fixture before exercising it.

**test_read_session_data**()

**class** pytext.data.test.tsv_data_source_test.**TSVDataSourceTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**setUp**()
Hook method for setting up the test fixture before exercising it.

**test_bad_quoting**()
The text column of the first row of this file opens a quote but does not close it.

**test_csv**()

**test_iterate_training_data_multiple_times**()

**test_quoting**()
The text column of the first row of this file opens a quote but does not close it.

**test_read_data_source**()

**test_read_data_source_with_column_remapping**()

**test_read_data_source_with_utf8_issues**()

**test_read_eval_data_source**()

**test_read_test_data_source**()

### pytext.data.test.utils_test module

**class** pytext.data.test.utils_test.**PaddingTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**testPadding**()

**testPaddingProvideShape**()

**class** pytext.data.test.utils_test.**TargetTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_align_target_label**()

**class** pytext.data.test.utils_test.**VocabularyTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**testBuildVocabulary**()

### Module contents

### pytext.data.tokenizers package

### Submodules

### pytext.data.tokenizers.tokenizer module

**class** pytext.data.tokenizers.tokenizer.**BERTInitialTokenizer**(*basic_tokenizer*)
Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*

Basic initial tokenization for BERT. This is run prior to word piece, does white space tokenization in addition to lower-casing and accent removal if specified.

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.BERTInitialTokenizer.Config*)

**tokenize**(*text*)
Tokenizes a piece of text.

**class** pytext.data.tokenizers.tokenizer.**CppProcessorMixin**
Bases: object

Cpp processors like SentencePiece don't pickle well; reload them.

**class** pytext.data.tokenizers.tokenizer.**DoNothingTokenizer**
Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*

Tokenizer that takes a list of strings and converts to a list of Tokens. Useful in cases where tokenizer is run before-hand

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.DoNothingTokenizer.Config*)

**tokenize**(*tokens: Union[List[str], str]*) → List[pytext.data.tokenizers.tokenizer.Token]

**torchscriptify**()

**class** pytext.data.tokenizers.tokenizer.**GPT2BPETokenizer**(*bpe: fairseq.data.encoders.gpt2_bpe_utils.Encoder, lowercase: bool = False*)
Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*

Tokenizer for gpt-2 and RoBERTa.

**decode**(*sentence: str*)

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.GPT2BPETokenizer.Config*)

**tokenize**(*input_str: str*) → List[pytext.data.tokenizers.tokenizer.Token]

**class** pytext.data.tokenizers.tokenizer.**PickleableGPT2BPEEncoder**(*encoder, bpe_merges, errors='replace'*)
Bases: fairseq.data.encoders.gpt2_bpe_utils.Encoder

Fairseq's encoder stores the regex module as a local reference on its encoders, which means they can't be saved via pickle.dumps or torch.save. This modified their save/load logic doesn't store the module, and restores the reference after re-inflating.

**class** pytext.data.tokenizers.tokenizer.**SentencePieceTokenizer**(*sp_model_path: str = '', max_input_text_length: Optional[int] = None*)
Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*, *pytext.data.tokenizers.tokenizer.CppProcessorMixin*

Sentence piece tokenizer.

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.SentencePieceTokenizer.Config*)

**tokenize**(*input_str: str*) → List[pytext.data.tokenizers.tokenizer.Token]

**torchscriptify**()

**class** pytext.data.tokenizers.tokenizer.**Token**(*value, start, end*)
Bases: tuple

**end**
Alias for field number 2

**start**
Alias for field number 1

**value**
Alias for field number 0

**class** pytext.data.tokenizers.tokenizer.**Tokenizer**(*split_regex='\s+', lowercase=True, use_byte_offsets=False*)

Bases: *pytext.config.component.Component*

A simple regex-splitting tokenizer.

**decode**(*sentence: str*)

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.Tokenizer.Config*)

**tokenize**(*input: str*) → List[pytext.data.tokenizers.tokenizer.Token]

**torchscriptify**()

**class** pytext.data.tokenizers.tokenizer.**WordPieceTokenizer**(*wordpiece_vocab, basic_tokenizer, wordpiece_tokenizer*)

Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*

Word piece tokenizer for BERT models.

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.WordPieceTokenizer.Config*)

**static load_vocab**(*vocab_file*)
Loads a vocabulary file into a dictionary.

**tokenize**(*input_str: str*) → List[pytext.data.tokenizers.tokenizer.Token]

### Module contents

**class** pytext.data.tokenizers.**GPT2BPETokenizer**(*bpe: fairseq.data.encoders.gpt2_bpe_utils.Encoder, lowercase: bool = False*)

Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*

Tokenizer for gpt-2 and RoBERTa.

**decode**(*sentence: str*)

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.GPT2BPETokenizer.Config*)

**tokenize**(*input_str: str*) → List[pytext.data.tokenizers.tokenizer.Token]

**class** pytext.data.tokenizers.**Token**(*value, start, end*)

Bases: tuple

**end**
Alias for field number 2

**start**
Alias for field number 1

**value**
Alias for field number 0

**class** pytext.data.tokenizers.**Tokenizer**(*split_regex='\s+', lowercase=True, use_byte_offsets=False*)

Bases: *pytext.config.component.Component*

A simple regex-splitting tokenizer.

**decode**(*sentence: str*)

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.Tokenizer.Config*)

**tokenize**(*input: str*) → List[pytext.data.tokenizers.tokenizer.Token]

**torchscriptify**()

**class** pytext.data.tokenizers.**DoNothingTokenizer**

Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*

Tokenizer that takes a list of strings and converts to a list of Tokens. Useful in cases where tokenizer is run before-hand

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.DoNothingTokenizer.Config*)

**tokenize**(*tokens: Union[List[str], str]*) → List[pytext.data.tokenizers.tokenizer.Token]

**torchscriptify**()

**class** pytext.data.tokenizers.**WordPieceTokenizer**(*wordpiece_vocab, basic_tokenizer, wordpiece_tokenizer*)

Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*

Word piece tokenizer for BERT models.

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.WordPieceTokenizer.Config*)

**static load_vocab**(*vocab_file*)

Loads a vocabulary file into a dictionary.

**tokenize**(*input_str: str*) → List[pytext.data.tokenizers.tokenizer.Token]

**class** pytext.data.tokenizers.**CppProcessorMixin**

Bases: object

Cpp processors like SentencePiece don't pickle well; reload them.

**class** pytext.data.tokenizers.**SentencePieceTokenizer**(*sp_model_path: str = '', max_input_text_length: Optional[int] = None*)

Bases: *pytext.data.tokenizers.tokenizer.Tokenizer*, *pytext.data.tokenizers.tokenizer.CppProcessorMixin*

Sentence piece tokenizer.

**classmethod from_config**(*config: pytext.data.tokenizers.tokenizer.SentencePieceTokenizer.Config*)

**tokenize**(*input_str: str*) → List[pytext.data.tokenizers.tokenizer.Token]

**torchscriptify**()

## Submodules

### pytext.data.batch_sampler module

**class** pytext.data.batch_sampler.**AlternatingRandomizedBatchSampler**(*unnormalized_iterator_probs: Dict[str, float], second_unnormalized_iterator_probs: Dict[str, float]*)

Bases: *pytext.data.batch_sampler.RandomizedBatchSampler*

This sampler takes in a dictionary of iterators and returns batches alternating between keys and probabilities specified by *unnormalized_iterator_probs* and 'second_unnormalized_iterator_probs', This is used for example in XLM pre-training where we alternate between MLM and TLM batches.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.data.batch_sampler.AlternatingRandomizedBatchSampler.Config*)

**class** pytext.data.batch_sampler.**BaseBatchSampler**

Bases: *pytext.config.component.Component*

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.config.component.Component.Config*)

**class** pytext.data.batch_sampler.**EvalBatchSampler**

Bases: *pytext.data.batch_sampler.BaseBatchSampler*

This sampler takes in a dictionary of Iterators and returns batches associated with each key in the dictionary. It guarentees that we will see each batch associated with each key exactly once in the epoch.

#### Example

Iterator 1: [A, B, C, D], Iterator 2: [a, b]

Output: [A, B, C, D, a, b]

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

Loop through each key in the input dict and generate batches from the iterator associated with that key.

**Parameters** **iterators** – Dictionary of iterators

**class** pytext.data.batch_sampler.**NaturalBatchSampler**(*dataset_counts: Dict[str, int]*)

Bases: *pytext.data.batch_sampler.RandomizedBatchSampler*

This sampler iterates over all the datasets, sampling according to the weighted number of samples in each dataset.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.data.batch_sampler.NaturalBatchSampler.Config*)

**class** pytext.data.batch_sampler.**RandomizedBatchSampler**(*unnormalized_iterator_probs: Dict[str, float]*)

Bases: *pytext.data.batch_sampler.BaseBatchSampler*

This sampler takes in a dictionary of iterators and returns batches according to the specified probabilities by *unnormalized_iterator_probs*. We cycle through the iterators (restarting any that "run out") indefinitely. Set batches_per_epoch in Trainer.Config.

### Example

Iterator A: [A, B, C, D], Iterator B: [a, b]

batches_per_epoch = 3, unnormalized_iterator_probs = {"A": 0, "B": 1} Epoch 1 = [a, b, a] Epoch 2 = [b, a, b]

**Parameters** **unnormalized_iterator_probs** (*Dict[str, float]*) – Iterator sampling probabilities. The keys should be the same as the keys of the underlying iterators, and the values will be normalized to sum to 1.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.data.batch_sampler.RandomizedBatchSampler.Config*)

**class** pytext.data.batch_sampler.**RoundRobinBatchSampler**(*iter_to_set_epoch: Optional[str] = None*)

Bases: *pytext.data.batch_sampler.BaseBatchSampler*

This sampler takes a dictionary of Iterators and returns batches in a round robin fashion till a the end of one of the iterators is reached. The end is specified by *iter_to_set_epoch*.

If *iter_to_set_epoch* is set, cycle batches from each iterator until one epoch of the target iterator is fulfilled. Iterators with fewer batches than the target iterator are repeated, so they never run out.

If *iter_to_set_epoch* is None, cycle over batches from each iterator until the shortest iterator completes one epoch.

### Example

Iterator 1: [A, B, C, D], Iterator 2: [a, b]

iter_to_set_epoch = "Iterator 1" Output: [A, a, B, b, C, a, D, b]

iter_to_set_epoch = None Output: [A, a, B, b]

**Parameters** **iter_to_set_epoch** (*Optional[str]*) – Name of iterator to define epoch size. If this is not set, epoch size defaults to the length of the shortest iterator.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

Loop through each key in the input dict and generate batches from the iterator associated with that key until the target iterator reaches its end.

**Parameters** **iterators** – Dictionary of iterators

**classmethod from_config**(*config: pytext.data.batch_sampler.RoundRobinBatchSampler.Config*)

pytext.data.batch_sampler.**extract_iterator_properties**(*input_iterator_probs: Dict[str, float]*)

Helper function for RandomizedBatchSampler and AlternatingRandomizedBatchSampler to generate iterator properties: iterator_names and iterator_probs.

pytext.data.batch_sampler.**select_key_and_batch**(*iterator_names: Dict[str, str], iterator_probs: Dict[str, float], iter_dict: Dict[str, collections.abc.Iterator], iterators: Dict[str, collections.abc.Iterator]*)

Helper function for RandomizedBatchSampler and AlternatingRandomizedBatchSampler to select a key from iterator_names using iterator_probs and return a batch for the selected key using iter_dict and iterators.

### pytext.data.bert_tensorizer module

**class** pytext.data.bert_tensorizer.**BERTTensorizer**(*columns: List[str] = ['text'], vocab: pytext.data.utils.Vocabulary = None, tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, max_seq_len: int = 256, **kwargs*)

Bases: *pytext.data.bert_tensorizer.BERTTensorizerBase*

Tensorizer for BERT tasks. Works for single sentence, sentence pair, triples etc.

**classmethod from_config**(*config: pytext.data.bert_tensorizer.BERTTensorizer.Config, **kwargs*)

from_config parses the config associated with the tensorizer and creates both the tokenizer and the Vocabulary object. The extra arguments passed as kwargs allow us to reuse thie function with variable number of arguments (eg: for classes which derive from this class).

**class** pytext.data.bert_tensorizer.**BERTTensorizerBase**(*columns: List[str] = ['text'], vocab: pytext.data.utils.Vocabulary = None, tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, max_seq_len: int = 256, base_tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None*)

Bases: *pytext.data.tensorizers.Tensorizer*

Base Tensorizer class for all BERT style models including XLM, RoBERTa and XLM-R.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**initialize**(*vocab_builder=None, from_scratch=True*)

The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row: Dict[KT, VT]*) → Tuple[Any, ...]

This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model.

**sort_key**(*row*)

**tensorize**(*batch*) → Tuple[torch.Tensor, ...]
Convert instance level vectors into batch level tensors.

**tensorizer_script_impl = None**

**class** pytext.data.bert_tensorizer.**BERTTensorizerBaseScriptImpl**(*tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer, vocab: pytext.data.utils.Vocabulary, max_seq_len: int*)

Bases: *pytext.data.tensorizers.TensorizerScriptImpl*

**forward**(*inputs: pytext.torchscript.utils.ScriptBatchInput*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]
Wire up tokenize(), numberize() and tensorize() functions for data processing. When export to TorchScript, the wrapper module should choose to use texts or pre_tokenized based on the TorchScript tokenizer implementation (e.g use external tokenizer such as Yoda or not).

**numberize**(*per_sentence_tokens: List[List[Tuple[str, int, int]]]*) → Tuple[List[int], List[int], int, List[int]]
This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model.

**Parameters**

- **per_sentence_tokens** – list of tokens per sentence level in one row,
- **token represented by token string, start and end indices.** (*each*) –

**Returns** List[int], a list of token ids, concatenate all sentences token ids. segment_labels: List[int], denotes each token belong to which sentence. seq_len: int, tokens length positions: List[int], token positions

**Return type** tokens

**tensorize**(*tokens_2d: List[List[int]], segment_labels_2d: List[List[int]], seq_lens_1d: List[int], positions_2d: List[List[int]]*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]
Convert instance level vectors into batch level tensors.

**tokenize**(*row_text: Optional[List[str]], row_pre_tokenized: Optional[List[List[str]]]*) → List[List[Tuple[str, int, int]]]
This function convert raw inputs into tokens, each token is represented by token(str), start and end indices in the raw inputs. There are two possible inputs to this function depends if the tokenized in implemented in TorchScript or not.

Case 1: Tokenizer has a full TorchScript implementation, the input will be a list of sentences (in most case it is single sentence or a pair).

Case 2: Tokenizer have partial or no TorchScript implementation, in most case, the tokenizer will be host in Yoda, the input will be a list of pre-processed tokens.

**Returns** tokens per sentence level, each token is represented by token(str), start and end indices.

**Return type** per_sentence_tokens

**torchscriptify**()

**class** pytext.data.bert_tensorizer.**BERTTensorizerScriptImpl**(*tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer, vocab: pytext.data.utils.Vocabulary, max_seq_len: int*)

Bases: *pytext.data.bert_tensorizer.BERTTensorizerBaseScriptImpl*

pytext.data.bert_tensorizer.**build_fairseq_vocab**(*vocab_file: str, dictionary_class: fairseq.data.dictionary.Dictionary = <class 'fairseq.data.dictionary.Dictionary'>, special_token_replacements: Dict[str, pytext.common.constants.Token] = None, max_vocab: int = -1, min_count: int = -1, tokens_to_add: Optional[List[str]] = None*) → pytext.data.utils.Vocabulary

Function builds a PyText vocabulary for models pre-trained using Fairseq modules. The dictionary class can take any Fairseq Dictionary class and is used to load the vocab file.

### pytext.data.data module

**class** pytext.data.data.**BatchData**(*raw_data, numberized*)

Bases: tuple

**numberized**

Alias for field number 1

**raw_data**

Alias for field number 0

**class** pytext.data.data.**Batcher**(*train_batch_size=16, eval_batch_size=16, test_batch_size=16*)

Bases: *pytext.config.component.Component*

Batcher designed to batch rows of data, before padding.

**batchify**(*iterable: Iterable[pytext.data.sources.data_source.RawExample], sort_key=None, stage=<Stage.TRAIN: 'Training'>*)

Group rows by batch_size. Assume iterable of dicts, yield dict of lists. The last batch will be of length len(iterable) % batch_size.

**classmethod from_config**(*config: pytext.data.data.Batcher.Config*)

**class** pytext.data.data.**Data**(*data_source: pytext.data.sources.data_source.DataSource, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], batcher: pytext.data.data.Batcher = None, sort_key: Optional[str] = None, in_memory: Optional[bool] = True, init_tensorizers: Optional[bool] = True, init_tensorizers_from_scratch: Optional[bool] = True*)

Bases: *pytext.config.component.Component*

Data is an abstraction that handles all of the following:

- Initialize model metadata parameters
- Create batches of tensors for model training or prediction

It can accomplish these in any way it needs to. The base implementation utilizes *pytext.data.sources.DataSource*, and sends batches to *pytext.data.tensorizers.Tensorizer* to create tensors.

The *tensorizers* dict passed to the initializer should be considered something like a signature for the model. Each batch should be a dictionary with the same keys as the *tensorizers* dict, and values should be tensors arranged in the way specified by that tensorizer. The tensorizers dict doubles as a simple baseline implementation of that same signature, but subclasses of Data can override the implementation using other methods. This value is how the model specifies what inputs it's looking for.

**add_row_indices**(*rows*)

**batches**(*stage: pytext.common.constants.Stage*, *data_source=None*, *load_early=False*)

Create batches of tensors to pass to model train_batch. This function yields dictionaries that mirror the *tensorizers* dict passed to *__init__*, ie. the keys will be the same, and the tensors will be the shape expected from the respective tensorizers.

*stage* is used to determine which data source is used to create batches. if data_source is provided, it is used instead of the configured data_sorce this is to allow setting a different data_source for testing a model.

Passing in *load_early* = True disables loading all data in memory and using PoolingBatcher, so that we get the first batch as quickly as possible.

**cache**(*numberized_rows*, *stage*)

**classmethod from_config**(*config: pytext.data.data.Data.Config, schema: Dict[str, Type[CT_co]], tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], rank=0, world_size=1, init_tensorizers=True, **kwargs*)

**numberize_rows**(*rows*)

**class** pytext.data.data.**PoolingBatcher**(*train_batch_size=16, eval_batch_size=16, test_batch_size=16, pool_num_batches=1000, num_shuffled_pools=1*)

Bases: *pytext.data.data.Batcher*

Batcher that shuffles and (if requested) sorts data.

**Rationale**

There is a trade-off between having batches of data that are truly randomly shuffled, and batches of data that are efficiently padded. If we wanted to maximise the efficiency of padding (i.e. minimise the amount of padding that is needed), we would have to enforce that all inputs of a similar length appear in the same batch. This however would lead to a dramatic decrease in the randomness of batches. On the other end of the spectrum, if we wanted to maximise randomness, we would often end up with inputs of wildly different lengths in the same batch, which would lead to a lot of padding.

**Operation**

This batcher uses a multi-staged approach.

1. It first loads a number of "pools" of data, and shuffles them (this is controlled by *num_shuffled_pools*).
2. It then splits up the shuffled data sequentially into individual pools, and the examples within each pool are sorted (if requested).
3. Finally, each pool is split up sequentially into batches, and yielded. If sorting was requested in step #2, the order in which the batches are yielded is randomised.

The size of a pool is expressed as a multiple of the batch size, and is controlled by *pool_num_batches*.

**Examples**

Assuming sorting is enabled, with the default settings of *pool_num_batches: 1000* and *num_shuffled_pools: 1*, a pool of *1k * batch_size* examples is loaded, sorted by length, and split up into 1k batches. These batches are then yielded in random order. Once they run out, a new pool is loaded, and the process is repeated. An

advantage of this approach is that padding will be somewhat reduced. A disadvantage is that, for every epoch, the first 1k batches will be always the same (albeit in a different order).

On the other hand, specifying *pool_num_batches: 1000* and *num_shuffled_pools: 1000* would achieve the following: *1k * 1k * batch_size* examples are loaded, and shuffled. These are then split up into pools of size *1k * batch_size*, which are then sorted internally, split into individual batches, and yielded in random order. Compared to the previous example, we no longer have the problem that the first 1k batches are always the same in each epoch, but we've had to load in memory 1M examples.

**batchify**(*iterable: Iterable[pytext.data.sources.data_source.RawExample], sort_key=None, stage=<Stage.TRAIN: 'Training'>*)

From an iterable of dicts, yield dicts of lists:

1. Load *num_shuffled_pools* pools of data, and shuffle them.
2. Load a pool (*batch_size * pool_num_batches* examples).
3. Sort rows, if necessary.
4. Shuffle the order in which the batches are returned, if necessary.

**classmethod from_config**(*config: pytext.data.data.PoolingBatcher.Config*)

**get_batch_size**(*stage: pytext.common.constants.Stage*) → int

**class** pytext.data.data.**RowData**(*raw_data, numberized*)

Bases: `tuple`

**numberized**

Alias for field number 1

**raw_data**

Alias for field number 0

pytext.data.data.**generator_iterator**(*fn*)

Turn a generator into a GeneratorIterator-wrapped function. Effectively this allows iterating over a generator multiple times by recording the call arguments, and calling the generator with them anew each item __iter__ is called on the returned object.

pytext.data.data.**pad_and_tensorize_batches**(*tensorizers, batches*)

pytext.data.data.**zip_dicts**(*dicts*)

### pytext.data.data_handler module

**class** pytext.data.data_handler.**BatchIterator**(*batches, processor, include_input=True, include_target=True, include_context=True, is_train=True, num_batches=0*)

Bases: `object`

BatchIterator is a wrapper of TorchText. Iterator that provide flexibility to map batched data to a tuple of (input, target, context) and other additional steps such as dealing with distributed training.

**Parameters**

- **batches** (*Iterator[TorchText.Batch]*) – iterator of TorchText.Batch, which shuffles/batches the data in __iter__ and return a batch of data in __next__
- **processor** – function to run after getting batched data from TorchText.Iterator, the function should define a way to map to data into (input, target, context)
- **include_input** (*bool*) – if input data should be returned, default is true

- **include_target** (*bool*) – if target data should be returned, default is true
- **include_context** (*bool*) – if context data should be returned, default is true
- **is_train** (*bool*) – if the batch data is for training
- **num_batches** (*int*) – total batches to generate, this param if for distributed training due to a limitation in PyTorch's distributed training backend that enforces all the parallel workers to have the same number of batches we workaround it by adding dummy batches at the end

**class** pytext.data.data_handler.**CommonMetadata**

Bases: object

**class** pytext.data.data_handler.**DataHandler**(*raw_columns: List[str], labels: Dict[str, pytext.fields.field.Field], features: Dict[str, pytext.fields.field.Field], featurizer: pytext.data.featurizer.featurizer.Featurizer, extra_fields: Dict[str, pytext.fields.field.Field] = None, text_feature_name: str = 'word_feat', shuffle: bool = True, sort_within_batch: bool = True, train_path: str = 'train.tsv', eval_path: str = 'eval.tsv', test_path: str = 'test.tsv', train_batch_size: int = 128, eval_batch_size: int = 128, test_batch_size: int = 128, max_seq_len: int = -1, pass_index: bool = True, column_mapping: Dict[str, str] = None, **kwargs*)

Bases: *pytext.config.component.Component*

DataHandler is the central place to prepare data for model training/testing. The class is responsible of:

- Define pipeline to process data and generate batch of tensors to be consumed by model. Each batch is a (input, target, extra_data) tuple, in which input can be feed directly into model.
- Initialize global context, such as build vocab, load pretrained embeddings. Store the context as metadata, and provide function to serialize/deserialize the metadata

The data processing pipeline contains the following steps:

- Read data from file into a list of raw data examples
- Convert each row of row data to a TorchText Example. This logic happens in process_row function and will:
  - Invoke featurizer, which contains data processing steps to apply for both training and inference time, e.g: tokenization
  - Use the raw data and results from featurizer to do any preprocessing
- Generate a TorchText.Dataset that contains the list of Example, the Dataset also has a list of TorchText.Field, which defines how to do padding and numericalization while batching data.
- Return a BatchIterator which will give a tuple of (input, target, context) tensors for each iteration. By default the tensors have a 1:1 mapping to the TorchText.Field fields, but this behavior can be overwritten by _input_from_batch, _target_from_batch, _context_from_batch functions.

**raw_columns**

columns to read from data source. The order should match the data stored in that file.

**Type** List[str]

**featurizer**

perform data preprocessing that should be shared between training and inference

**Type** Featurizer

**features**
a dict of name -> field that used to process data as model input

**Type** Dict[str, Field]

**labels**
a dict of name -> field that used to process data as training target

**Type** Dict[str, Field]

**extra_fields**
fields that process any extra data used neither as model input nor target. This is None by default

**Type** Dict[str, Field]

**text_feature_name**
name of the text field, used to define the default sort key of data

**Type** str

**shuffle**
if the dataset should be shuffled, true by default

**Type** bool

**sort_within_batch**
if data within same batch should be sorted, true by default

**Type** bool

**train_path**
path of training data file

**Type** str

**eval_path**
path of evaluation data file

**Type** str

**test_path**
path of test data file

**Type** str

**train_batch_size**
training batch size, 128 by default

**Type** int

**eval_batch_size**
evaluation batch size, 128 by default

**Type** int

**test_batch_size**
test batch size, 128 by default

**Type** int

**max_seq_len**
maximum length of tokens to keep in sequence

**Type** int

**pass_index**
if the original index of data in the batch should be passed along to downstream steps, default is true

**Type** bool

**gen_dataset**(*data: Iterable[Dict[str, Any]], include_label_fields: bool = True, shard_range: Tuple[int, int] = None*) → torchtext.legacy.data.dataset.Dataset
Generate torchtext Dataset from raw in memory data. :returns: dataset (TorchText.Dataset)

**gen_dataset_from_path**(*path: str, rank: int = 0, world_size: int = 1, include_label_fields: bool = True, use_cache: bool = True*) → torchtext.legacy.data.dataset.Dataset
Generate a dataset from file :returns: dataset (TorchText.Dataset)

**get_eval_iter**()

**get_predict_iter**(*data: Iterable[Dict[str, Any]], batch_size: Optional[int] = None*)

**get_test_iter**()

**get_test_iter_from_path**(*test_path: str, batch_size: int*) → pytext.data.data_handler.BatchIterator

**get_test_iter_from_raw_data**(*test_data: List[Dict[str, Any]], batch_size: int*) → pytext.data.data_handler.BatchIterator

**get_train_iter**(*rank: int = 0, world_size: int = 1*)

**get_train_iter_from_path**(*train_path: str, batch_size: int, rank: int = 0, world_size: int = 1*) → pytext.data.data_handler.BatchIterator
Generate data batch iterator for training data. See *_get_train_iter()* for details

**Parameters**

- **train_path** (*str*) – file path of training data
- **batch_size** (*int*) – batch size
- **rank** (*int*) – used for distributed training, the rank of current Gpu, don't set it to anything but 0 for non-distributed training
- **world_size** (*int*) – used for distributed training, total number of Gpu

**get_train_iter_from_raw_data**(*train_data: List[Dict[str, Any]], batch_size: int, rank: int = 0, world_size: int = 1*) → pytext.data.data_handler.BatchIterator

**init_feature_metadata**(*train_data: torchtext.legacy.data.dataset.Dataset, eval_data: torchtext.legacy.data.dataset.Dataset, test_data: torchtext.legacy.data.dataset.Dataset*)

**init_metadata**()
Initialize metadata using data from configured path

**init_metadata_from_path**(*train_path, eval_path, test_path*)
Initialize metadata using data from file

**init_metadata_from_raw_data**(**data*)
Initialize metadata using in memory data

**init_target_metadata**(*train_data: torchtext.legacy.data.dataset.Dataset, eval_data: torchtext.legacy.data.dataset.Dataset, test_data: torchtext.legacy.data.dataset.Dataset*)

**load_metadata**(*metadata: pytext.data.data_handler.CommonMetadata*)
Load previously saved metadata

**load_vocab**(*vocab_file*, *vocab_size*, *lowercase_tokens: bool = False*)

Loads items into a set from a file containing one item per line. Items are added to the set from top of the file to bottom. So, the items in the file should be ordered by a preference (if any), e.g., it makes sense to order tokens in descending order of frequency in corpus.

**Parameters**

- **vocab_file** (*str*) – vocab file to load
- **vocab_size** (*int*) – maximum tokens to load, will only load the first n if the actual vocab size is larger than this parameter
- **lowercase_tokens** (*bool*) – if the tokens should be lowercased

**metadata_to_save**()

Save metadata, pretrained_embeds_weight should be excluded

**preprocess**(*data: Iterable[Dict[str, Any]]*)

preprocess the raw data to create TorchText.Example, this is the second step in whole processing pipeline :returns: data (Generator[Dict[str, Any]])

**preprocess_row**(*row_data: Dict[str, Any]*) → Dict[str, Any]

preprocess steps for a single input row, sub class should override it

**read_from_file**(*file_name: str, columns_to_use: Union[Dict[str, int], List[str]]*) → Generator[Dict[KT, VT], None, None]

Read data from csv file. Input file format is required to be tab-separated columns

**Parameters**

- **file_name** (*str*) – csv file name
- **columns_to_use** (*Union[Dict[str, int], List[str]]*) – either a list of column names or a dict of column name -> column index in the file

**sort_key**(*example: torchtext.legacy.data.example.Example*) → Any

How to sort data in every batch, default behavior is by the length of input text :param example: one torchtext example :type example: Example

### pytext.data.dense_retrieval_tensorizer module

**class** pytext.data.dense_retrieval_tensorizer.**BERTContextTensorizerForDenseRetrieval**(*columns: List[str] = ['text'], vocab: pytext.data.t = None, tokenizer: pytext.data.t = None, max_seq_ int = 256, **kwargs*

Bases: *pytext.data.bert_tensorizer.BERTTensorizer*

Methods numberize() and tensorize() implement https://fburl.com/an4fv7m1.

**numberize**(*row: Dict[KT, VT]*) → Tuple[Any, ...]

This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model. It works off of one sample.

**tensorize**(*batch*)

Works off of a batch that's numerized.

**class** pytext.data.dense_retrieval_tensorizer.**PositiveLabelTensorizerForDenseRetrieval**(*label_c str = 'label', allow_un bool = False, pad_in bool = False, label_vo Optional[ = None, label_vo Optional[ = None, is_inpu bool = False, add_la Optional[ = None*)

Bases: *pytext.data.tensorizers.LabelTensorizer*

**numberize**(*row: Dict[KT, VT]*)

Numberize labels.

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.dense_retrieval_tensorizer.**RoBERTaContextTensorizerForDenseRetrieval**(*colum List[ = ['text vo-cab: Op-tiona = None to-k-enize Op-tiona = None max_ int = 256*)

Bases: *pytext.data.dense_retrieval_tensorizer.BERTContextTensorizerForDenseRetrieval*, *pytext.data.roberta_tensorizer.RoBERTaTensorizer*

**classmethod from_config**(*config: pytext.data.dense_retrieval_tensorizer.RoBERTaContextTensorizerForDenseRetrieval*

from_config parses the config associated with the tensorizer and creates both the tokenizer and the Vocabulary object. The extra arguments passed as kwargs allow us to reuse thie function with variable number of arguments (eg: for classes which derive from this class).

### pytext.data.disjoint_multitask_data module

**class** pytext.data.disjoint_multitask_data.**DisjointMultitaskData**(*data_dict: Dict[str, pytext.data.data.Data], samplers: Dict[pytext.common.constants.Stage, pytext.data.batch_sampler.BaseBatchSam test_key: str = None, task_key: str = 'task_name'*)

Bases: *pytext.data.data.Data*

Wrapper for doing multitask training using multiple data objects. Takes a dictionary of data objects, does round robin over their iterators using BatchSampler.

**Parameters**

- **config** (*Config*) – Configuration object of type DisjointMultitaskData.Config.
- **data_dict** (*Dict[str, Data]*) – Data objects to do roundrobin over.
- ***args** (*type*) – Extra arguments to be passed down to sub data handlers.
- ****kwargs** (*type*) – Extra arguments to be passed down to sub data handlers.

**data_dict**
Data handlers to do roundrobin over.

**Type** type

**batches**(*stage: pytext.common.constants.Stage*, *data_source=None*, *load_early=False*)
Yield batches from each task, sampled according to a given sampler. This batcher additionally exposes a task name in the batch to allow the model to filter examples to the appropriate tasks.

**classmethod from_config**(*config: pytext.data.disjoint_multitask_data.DisjointMultitaskData.Config, data_dict: Dict[str, pytext.data.data.Data], task_key: str = 'task_name', rank=0, world_size=1, init_tensorizers=True*)

## pytext.data.disjoint_multitask_data_handler module

**class** pytext.data.disjoint_multitask_data_handler.**DisjointMultitaskDataHandler**(*config: pytext.data.disjoint_ data_handlers: Dict[str, pytext.data.data_ha tar- get_task_name: Op- tional[str] = None, *args, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

Wrapper for doing multitask training using multiple data handlers. Takes a dictionary of data handlers, does round robin over their iterators using RoundRobinBatchIterator.

**Parameters**

- **config** (*Config*) – Configuration object of type DisjointMultitaskDataHandler.Config.
- **data_handlers** (*Dict[str, DataHandler]*) – Data handlers to do roundrobin over.
- **target_task_name** (*Optional[str]*) – Used to select best epoch, and set batch_per_epoch.
- ***args** (*type*) – Extra arguments to be passed down to sub data handlers.
- ****kwargs** (*type*) – Extra arguments to be passed down to sub data handlers.

**data_handlers**
Data handlers to do roundrobin over.

**Type** type

**target_task_name**
Used to select best epoch, and set batch_per_epoch.

**Type** type

**upsample**
If upsample, keep cycling over each iterator in round-robin. Iterators with less batches will get more

passes. If False, we do single pass over each iterator, the ones which run out will sit idle. This is used for evaluation. Default True.

**Type** bool

**get_eval_iter**() → pytext.data.data_handler.BatchIterator

**get_test_iter**() → pytext.data.data_handler.BatchIterator

**get_train_iter**(*rank: int = 0, world_size: int = 1*) → Tuple[pytext.data.data_handler.BatchIterator, ...]

**init_metadata**()
Initialize metadata using data from configured path

**load_metadata**(*metadata*)
Load previously saved metadata

**metadata_to_save**()
Save metadata, pretrained_embeds_weight should be excluded

**class** pytext.data.disjoint_multitask_data_handler.**RoundRobinBatchIterator**(*iterators: Dict[str, pytext.data.data_handler.B upsample: bool = True, iter_to_set_epoch: Optional[str] = None*)

Bases: *pytext.data.data_handler.BatchIterator*

We take a dictionary of BatchIterators and do round robin over them in a cycle. The below describes the behavior for one epoch, with the example

Iterator 1: [A, B, C, D], Iterator 2: [a, b]

**If *upsample* is True:** If *iter_to_set_epoch* is set, cycle batches from each iterator until one epoch of the target iterator is fulfilled. Iterators with fewer batches than the target iterator are repeated, so they never run out.

iter_to_set_epoch = "Iterator 1" Output: [A, a, B, b, C, a, D, b]

If *iter_to_set_epoch* is None, cycle over batches from each iterator until the shortest iterator completes one epoch.

Output: [A, a, B, b]

**If *upsample* is False:** Iterate over batches from one epoch of each iterator, with the order among iterators uniformly shuffled.

Possible output: [a, A, B, C, b, D]

**Parameters**

- **iterators** (*Dict[str, BatchIterator]*) – Iterators to do roundrobin over.

- **upsample** (*bool*) – If upsample, keep cycling over each iterator in round-robin. Iterators with less batches will get more passes. If False, we do single pass over each iterator, in random order. Evaluation will use upsample=False. Default True.
- **iter_to_set_epoch** (*Optional[str]*) – Name of iterator to define epoch size. If upsample is True and this is not set, epoch size defaults to the length of the shortest iterator. If upsample is False, this argument is not used.

**iterators**
Iterators to do roundrobin over.

**Type** Dict[str, BatchIterator]

**upsample**
Whether to upsample iterators with fewer batches.

**Type** bool

**iter_to_set_epoch**
Name of iterator to define epoch size.

**Type** str

**classmethod cycle**(*iterator*)

### pytext.data.dynamic_pooling_batcher module

**class** pytext.data.dynamic_pooling_batcher.**BatcherSchedulerConfig**(***kwargs*)
Bases: pytext.config.module_config.Module.Config

**end_batch_size = 256**

**epoch_period = 10**

**start_batch_size = 32**

**step_size = 1**

**class** pytext.data.dynamic_pooling_batcher.**DynamicPoolingBatcher**(*train_batch_size=16, eval_batch_size=16, test_batch_size=16, pool_num_batches=1000, num_shuffled_pools=1, sched-uler_config=<pytext.data.dynamic_poo object>*)
Bases: *pytext.data.data.PoolingBatcher*

Allows dynamic batch training, extends pooling batcher with a scheduler config, which specifies how batch size should increase

**batchify**(*iterable: Iterable[pytext.data.sources.data_source.RawExample], sort_key=None, stage=<Stage.TRAIN: 'Training'>*)
From an iterable of dicts, yield dicts of lists:

1. Load *num_shuffled_pools* pools of data, and shuffle them.
2. Load a pool (*batch_size* * *pool_num_batches* examples).
3. Sort rows, if necessary.
4. Shuffle the order in which the batches are returned, if necessary.

**compute_dynamic_batch_size**(*curr_epoch: int, scheduler_config: pytext.data.dynamic_pooling_batcher.BatcherSchedulerConfig, curr_steps: int*) → int

**finished_dynamic**() → bool

**classmethod from_config**(*config: pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher.Config*)

**get_batch_size**(*stage: pytext.common.constants.Stage*) → int

**step_epoch**()

**class** pytext.data.dynamic_pooling_batcher.**ExponentialBatcherSchedulerConfig**(***kwargs*)

Bases: *pytext.data.dynamic_pooling_batcher.BatcherSchedulerConfig*

**gamma = 5**

**class** pytext.data.dynamic_pooling_batcher.**ExponentialDynamicPoolingBatcher**(**args, **kwargs*)

Bases: *pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher*

Exponential Dynamic Batch Scheduler: scales up batch size by a factor of gamma

**compute_dynamic_batch_size**(*curr_epoch: int, scheduler_config: pytext.data.dynamic_pooling_batcher.ExponentialBatcherSchedulerConfig, curr_steps: int*) → int

**finished_dynamic**() → bool

**get_max_steps**()

**class** pytext.data.dynamic_pooling_batcher.**LinearDynamicPoolingBatcher**(*train_batch_size=16, eval_batch_size=16, test_batch_size=16, pool_num_batches=1000, num_shuffled_pools=1, scheduler_config=<pytext.data.dyna object>*)

Bases: *pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher*

Linear Dynamic Batch Scheduler: scales up batch size linearly

**compute_dynamic_batch_size**(*curr_epoch: int, scheduler_config: pytext.data.dynamic_pooling_batcher.BatcherSchedulerConfig, curr_steps: int*) → int

### pytext.data.masked_tensorizer module

**class** pytext.data.masked_tensorizer.**MaskedTokenTensorizer**(*text_column, mask, tokenizer=None, add_bos_token=False, add_eos_token=False, use_eos_token_for_bos=False, max_seq_len=None, vocab_config=None, vocab=None, vocab_file_delimiter=' ', is_input=True*)

Bases: *pytext.data.tensorizers.TokenTensorizer*

**classmethod from_config**(*config: pytext.data.masked_tensorizer.MaskedTokenTensorizer.Config*)

**mask_and_tensorize**(*batch*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

### pytext.data.masked_util module

**class** pytext.data.masked_util.**MaskEverything**(*use_bos*, *use_eos*)
Bases: *pytext.data.masked_util.MaskingFunction*

**gen_masked_source_target**(*tokens*, *vocab: pytext.data.utils.Vocabulary*)

**gen_masked_tree**(*node*, *mask_token*, *depth=1*)

**class** pytext.data.masked_util.**MaskedVocabBuilder**(*delimiter=' '*)
Bases: *pytext.data.utils.VocabBuilder*

**class** pytext.data.masked_util.**MaskingFunction**(*use_bos*, *use_eos*)
Bases: *pytext.config.component.Component*

**classmethod from_config**(*config*, *use_bos*, *use_eos*)

**gen_masked_source_target**(*tokens*, **args*, ***kwargs*)

**should_mask**(**args*, ***kwargs*) → bool

**class** pytext.data.masked_util.**NoOpMaskingFunction**(*seed: Optional[int], minimum_masks: int, use_bos: bool, use_eos: bool*)
Bases: *pytext.data.masked_util.MaskingFunction*

**classmethod from_config**(*config: pytext.data.masked_util.NoOpMaskingFunction.Config*, *use_bos: bool*, *use_eos: bool*)

**gen_masked_source_target**(*tokens: List[int], vocab: pytext.data.utils.Vocabulary*)

**class** pytext.data.masked_util.**RandomizedMaskingFunction**(*seed: Optional[int], minimum_masks: int, use_bos: bool, use_eos: bool*)
Bases: *pytext.data.masked_util.MaskingFunction*

**classmethod from_config**(*config: pytext.data.masked_util.RandomizedMaskingFunction.Config*, *use_bos: bool*, *use_eos: bool*)

**gen_masked_source_target**(*tokens: List[int], vocab: pytext.data.utils.Vocabulary*)

**class** pytext.data.masked_util.**TreeMask**(*accept_flat_intents_slots*, *factor*, *use_bos*, *use_eos*)
Bases: *pytext.data.masked_util.MaskingFunction*

**clean_eos_bos**(*tokens*)

**classmethod from_config**(*config*, *use_bos*, *use_eos*)

**gen_masked_source_target**(*tokens: List[int], vocab: pytext.data.utils.Vocabulary*)

**gen_masked_tree**(*node*, *mask_token*, *depth=1*)

**should_mask**(*depth=1*)

### pytext.data.packed_lm_data module

**class** pytext.data.packed_lm_data.**PackedLMData**(*data_source: pytext.data.sources.data_source.DataSource, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], batcher: pytext.data.data.Batcher = None, max_seq_len: int = 128, sort_key: Optional[str] = None, language: Optional[str] = None, in_memory: Optional[bool] = False, init_tensorizers: Optional[bool] = True*)

Bases: *pytext.data.data.Data*

Special purpose Data object which assumes a single text tensorizer. Packs tokens into a square batch with no padding. Used for LM training. The object also takes in an optional language argument which is used for cross-lingual LM training.

**classmethod from_config**(*config: pytext.data.packed_lm_data.PackedLMData.Config, schema: Dict[str, Type[CT_co]], tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], language: Optional[str] = None, rank: int = 0, world_size: int = 1, init_tensorizers: Optional[bool] = True*)

**numberize_rows**(*rows*)

### pytext.data.roberta_tensorizer module

**class** pytext.data.roberta_tensorizer.**RoBERTaTensorizer**(*columns: List[str] = ['text'], vocab: pytext.data.utils.Vocabulary = None, tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, max_seq_len: int = 256, **kwargs*)

Bases: *pytext.data.bert_tensorizer.BERTTensorizerBase*

**classmethod from_config**(*config: pytext.data.roberta_tensorizer.RoBERTaTensorizer.Config, **kwargs*)

**class** pytext.data.roberta_tensorizer.**RoBERTaTokenLevelTensorizer**(*columns, tokenizer=None, vocab=None, max_seq_len=256, labels_columns=['label'], labels=[]*)

Bases: *pytext.data.roberta_tensorizer.RoBERTaTensorizer*

Tensorizer for token level classification tasks such as NER, POS etc using RoBERTa. Here each token has an associated label and the tensorizer should output a label tensor as well. The input for this tensorizer comes from the CoNLLUNERDataSource data source.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.roberta_tensorizer.RoBERTaTokenLevelTensorizer.Config*)

**numberize**(*row: Dict[KT, VT]*) → Tuple[Any, ...]
Numberize both the tokens and labels. Since we break up tokens, the label for anything other than the first sub-word is assigned the padding idx.

**tensorize**(*batch*) → Tuple[torch.Tensor, ...]
Convert instance level vectors into batch level tensors.

**torchscriptify**()

### pytext.data.squad_for_bert_tensorizer module

**class** pytext.data.squad_for_bert_tensorizer.**SquadForBERTTensorizer**(*answers_column: str = 'answers', answer_starts_column: str = 'answer_starts', **kwargs*)

Bases: *pytext.data.bert_tensorizer.BERTTensorizer*

Produces BERT inputs and answer spans for Squad.

**SPAN_PAD_IDX = -100**

**classmethod from_config**(*config: pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizer.Config, **kwargs*)
from_config parses the config associated with the tensorizer and creates both the tokenizer and the Vocabulary object. The extra arguments passed as kwargs allow us to reuse thie function with variable number of arguments (eg: for classes which derive from this class).

**numberize**(*row*)
This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model.

**tensorize**(*batch*)
Convert instance level vectors into batch level tensors.

**class** pytext.data.squad_for_bert_tensorizer.**SquadForBERTTensorizerForKD**(*start_logits_column='start_ end_logits_column='end_l has_answer_logits_column pad_mask_column='pad_m segment_labels_column='segm **kwargs*)

Bases: *pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizer*

**classmethod from_config**(*config: pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizerForKD.Config, **kwargs*)
from_config parses the config associated with the tensorizer and creates both the tokenizer and the Vocabulary object. The extra arguments passed as kwargs allow us to reuse thie function with variable number of arguments (eg: for classes which derive from this class).

**numberize**(*row*)
This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model.

**tensorize**(*batch*)
Convert instance level vectors into batch level tensors.

**class** pytext.data.squad_for_bert_tensorizer.**SquadForRoBERTaTensorizer**(*answers_column: str = 'answers', answer_starts_column: str = 'answer_starts', **kwargs*)

Bases: pytext.data.roberta_tensorizer.RoBERTaTensorizer, pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizer

Produces RoBERTa inputs and answer spans for Squad.

**classmethod from_config**(*config: pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizer.Config, **kwargs*)

from_config parses the config associated with the tensorizer and creates both the tokenizer and the Vocabulary object. The extra arguments passed as kwargs allow us to reuse thie function with variable number of arguments (eg: for classes which derive from this class).

**torchscriptify**()

**class** pytext.data.squad_for_bert_tensorizer.**SquadForRoBERTaTensorizerForKD**(*start_logits_column='s end_logits_column='e has_answer_logits_col pad_mask_column='pa segment_labels_column=' **kwargs*)

Bases: pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizer

**classmethod from_config**(*config: pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizerForKD.Config, **kwargs*)

from_config parses the config associated with the tensorizer and creates both the tokenizer and the Vocabulary object. The extra arguments passed as kwargs allow us to reuse thie function with variable number of arguments (eg: for classes which derive from this class).

**numberize**(*row*)

This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model.

**tensorize**(*batch*)

Convert instance level vectors into batch level tensors.

### pytext.data.squad_tensorizer module

**class** pytext.data.squad_tensorizer.**SquadTensorizer**(*doc_tensorizer: pytext.data.tensorizers.TokenTensorizer, ques_tensorizer: pytext.data.tensorizers.TokenTensorizer, doc_column: str = 'doc', ques_column: str = 'question', answers_column: str = 'answers', answer_starts_column: str = 'answer_starts', **kwargs*)

Bases: pytext.data.tensorizers.TokenTensorizer

Produces inputs and answer spans for Squad.

**SPAN_PAD_IDX = -100**

**classmethod from_config**(*config: pytext.data.squad_tensorizer.SquadTensorizer.Config, **kwargs*)

**initialize**(*vocab_builder=None, from_scratch=True*)
Build vocabulary based on training corpus.

**numberize**(*row*)
Tokenize, look up in vocabulary.

**sort_key**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.squad_tensorizer.**SquadTensorizerForKD**(*start_logits_column='start_logits', end_logits_column='end_logits', has_answer_logits_column='has_answer_logits', pad_mask_column='pad_mask', segment_labels_column='segment_labels', **kwargs*)

Bases: *pytext.data.squad_tensorizer.SquadTensorizer*

**classmethod from_config**(*config: pytext.data.squad_tensorizer.SquadTensorizerForKD.Config, **kwargs*)

**numberize**(*row*)
Tokenize, look up in vocabulary.

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

### pytext.data.tensorizers module

**class** pytext.data.tensorizers.**AnnotationNumberizer**(*column: str = 'seqlogical', vocab=None, is_input: bool = True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Not really a Tensorizer (since it does not create tensors) but technically serves the same function. This class parses Annotations in the format below and extracts the actions (type List[List[int]])

```
[IN:GET_ESTIMATED_DURATION How long will it take to [SL:METHOD_TRAVEL
drive ] from [SL:SOURCE Chicago ] to [SL:DESTINATION Mississippi ] ]
```

Extraction algorithm is handled by Annotation class. We only care about the list of actions, which before vocab index lookups would look like:

```
[
    IN:GET_ESTIMATED_DURATION, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
    SL:METHOD_TRAVEL, SHIFT, REDUCE,
    SHIFT,
    SL:SOURCE, SHIFT, REDUCE,
    SHIFT,
    SL:DESTINATION, SHIFT, REDUCE,
]
```

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.AnnotationNumberizer.Config*)

**initialize**(*vocab_builder=None*, *from_scratch=True*)
Build vocabulary based on training corpus.

**numberize**(*row*)
Tokenize, look up in vocabulary.

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**ByteTensorizer**(*text_column*, *lower=True*, *max_seq_len=None*, *add_bos_token=False*, *add_eos_token=False*, *use_eos_token_for_bos=False*, *is_input=True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Turn characters into sequence of int8 bytes. One character will have one or more bytes depending on it's encoding

**NUM = 256**

**PAD_BYTE = 0**

**UNK_BYTE = 0**

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.ByteTensorizer.Config*)

**numberize**(*row*)
Convert text to characters.

**sort_key**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**ByteTokenTensorizer**(*text_column*, *tokenizer=None*, *max_seq_len=None*, *max_byte_len=15*, *offset_for_non_padding=0*, *add_bos_token=False*, *add_eos_token=False*, *use_eos_token_for_bos=False*, *is_input=True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Turn words into 2-dimensional tensors of int8 bytes. Words are padded to *max_byte_len*. Also computes sequence lengths (1-D tensor) and token lengths (2-D tensor). 0 is the pad byte.

**NUM_BYTES = 256**

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.ByteTokenTensorizer.Config*)

**numberize**(*row*)
Convert text to bytes, pad batch.

**sort_key**(*row*)

**tensorize**(*batch*, *pad_token=0*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**CharacterTokenTensorizer**(*max_char_length: int = 20*, ***kwargs*)

Bases: *pytext.data.tensorizers.TokenTensorizer*

Turn words into 2-dimensional tensors of ints based on their ascii values. Words are padded to the maximum word length (also capped at *max_char_length*). Sequence lengths are the length of each token, 0 for pad token.

**initialize**(*from_scratch=True*)
The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row*)
Convert text to characters, pad batch.

**sort_key**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**CharacterVocabTokenTensorizer**(*text_column*, *tokenizer=None*, *add_bos_token=False*, *add_eos_token=False*, *use_eos_token_for_bos=False*, *max_seq_len=None*, *vocab_config=None*, *vocab=None*, *vocab_file_delimiter=' '*, *is_input=True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Turn words into 2-dimensional tensors of ints based on the char vocab. Words are padded to the maximum word length (also capped at *max_char_length*). Sequence lengths are the length of each token.

The difference with pytext.data.tensorizers.CharacterTokenTensorizer is that the CharacterTokenTensorizer uses the ascii value and does not require to build a vocab. Here we tensorize based on the vocab.

**character_tokenize**(*tokens: List[pytext.data.tokenizers.tokenizer.Token]*)

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.CharacterVocabTokenTensorizer.Config*)

**initialize**(*vocab_builder=None*, *from_scratch=True*)

Build vocabulary based on training corpus.

**numberize**(*row*)

Tokenize, look up in vocabulary.

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**class** pytext.data.tensorizers.**CharacterVocabTokenTensorizerScriptImpl**(*add_bos_token: bool, add_eos_token: bool, use_eos_token_for_bos: bool, max_seq_len: int, vocab: pytext.data.utils.Vocabulary, tokenizer: Optional[pytext.data.tokenizers.to*

Bases: *pytext.data.tensorizers.TensorizerScriptImpl*

**forward**(*inputs: pytext.torchscript.utils.ScriptBatchInput*) → Tuple[torch.Tensor, torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_texts_by_index**(*texts: Optional[List[List[str]]], index: int*) → Optional[str]

**get_tokens_by_index**(*tokens: Optional[List[List[List[str]]]], index: int*) → Optional[List[str]]

**numberize**(*char_tokens: List[List[str]], char_tokens_lengths: List[int]*) → Tuple[List[List[int]], List[int]]

This functions will receive the outputs from function: tokenize() or will be called directly from PyTextTensorizer function: numberize().

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**tensorize**(*tokens: List[List[List[int]]], tokens_lengths: List[List[int]]*) → Tuple[torch.Tensor, torch.Tensor]

This functions will receive a list(e.g a batch) of outputs from function numberize(), padding and convert to output tensors.

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**tokenize**(*row_text: Optional[str] = None, row_pre_tokenized: Optional[List[str]] = None*) → Tuple[List[List[str]], List[int]]

This functions will receive the inputs from Clients, usually there are two possible inputs 1) a row of texts: List[str] 2) a row of pre-processed tokens: List[List[str]]

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**class** pytext.data.tensorizers.**Float1DListTensorizer**(*config: pytext.data.tensorizers.Float1DListTensorizer.Config, **kwargs*)

Bases: *pytext.data.tensorizers.Tensorizer*

Tensorizes the 1d list of floats – List[float] TODO: Even though very similar, 'FloatListTensorizer' currently does not support this vanilla case for tensorization of List[float]. In future, if 'FloatListTensorizer' accommodates this case, we do not need this separate tensorizer.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.Float1DListTensorizer.Config, **kwargs*)

**initialize**(*from_scratch=True*)

The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row*)

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**class** pytext.data.tensorizers.**FloatListSeqTensorizer**(*column: str, error_check: bool, dim: Optional[int], pad_token: float = -1.0, is_input: bool = True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Numberize numeric labels.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.FloatListSeqTensorizer.Config*)

**numberize**(*row*)

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**class** pytext.data.tensorizers.**FloatListTensorizer**(*column: str, error_check: bool, dim: Optional[int], normalize: bool, is_input: bool = True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Numberize numeric labels.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.FloatListTensorizer.Config*)

**initialize**()

The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row*)

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**FloatTensorizer**(*column: str, is_input: bool = True*)

Bases: *pytext.data.tensorizers.Tensorizer*

A tensorizer for reading in scalars from the data.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.FloatTensorizer.Config*)

**numberize**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**GazetteerTensorizer**(*text_column: str = 'text', dict_column: str = 'dict', tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, is_input: bool = True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Create 3 tensors for dict features.

- idx: index of feature in token order.
- weights: weight of feature in token order.
- lens: number of features per token.

For each input token, there will be the same number of *idx* and *weights* entries. (equal to the max number of features any token has in this row). The values in *lens* will tell how many of these features are actually used per token.

Input format for the dict column is json and should be a list of dictionaries containing the "features" and their weight for each relevant "tokenIdx". Example:

```
text: "Order coffee from Starbucks please"
dict: [
    {"tokenIdx": 1, "features": {"drink/beverage": 0.8, "music/song": 0.2}},
    {"tokenIdx": 3, "features": {"store/coffee_shop": 1.0}}
]
```

if we assume this vocab

```
vocab = {
    UNK: 0, PAD: 1,
    "drink/beverage": 2, "music/song": 3, "store/coffee_shop": 4
}
```

this example will result in those tensors:

```
idx =     [1,   1,   2,   3,   1,   1,   4,   1,   1,   1]
weights = [0.0, 0.0, 0.8, 0.2, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
lens =    [1,        2,        1,        1,        1]
```

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.GazetteerTensorizer.Config*)

**initialize**(*from_scratch=True*)
Look through the dataset for all dict features to create vocab.

**numberize**(*row*)
Numberize dict features. Fill in for tokens with no features with PAD and weight 0.0. All tokens need to have at least one entry. Tokens with more than one feature will have multiple idx and weight added in sequence.

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**Integer1DListTensorizer**(*config: pytext.data.tensorizers.Integer1DListTensorizer.Config, **kwargs*)

Bases: *pytext.data.tensorizers.Tensorizer*

Tensorizes the 1d list of integers – List[int]

**SPAN_PAD_IDX = 0**

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.Integer1DListTensorizer.Config, **kwargs*)

**initialize**(*from_scratch=True*)
The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**class** pytext.data.tensorizers.**LabelListRankTensorizer**(**args, pad_missing: bool = False, **kwargs*)

Bases: *pytext.data.tensorizers.LabelTensorizer*

LabelListRankTensorizer takes a list of a single array with [[labelA, rankA], [labelB, rankB], ...] as input and generate a tuple of tensors (label_idx, list_length). Example: Input: ["["weather","1"]","["business","1"]"] Output of size len(vocab) {"timer", "weather", "business"} => [0, 1, 1]. This would suggest both labels are of equal rank.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute.

We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.LabelListRankTensorizer.Config*)

**initialize**(*from_scratch=True*)
Look through the dataset for all labels and create a vocab map for them.

**numberize**(*row*)
Numberize labels.

**sort_key**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**LabelListTensorizer**(**args, pad_missing: bool = False, **kwargs*)

Bases: pytext.data.tensorizers.LabelTensorizer

LabelListTensorizer takes a list of labels as input and generate a tuple of tensors (label_idx, list_length).

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.LabelListTensorizer.Config*)

**numberize**(*row*)
Numberize labels.

**sort_key**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**LabelTensorizer**(*label_column: str = 'label', allow_unknown: bool = False, pad_in_vocab: bool = False, label_vocab: Optional[List[str]] = None, label_vocab_file: Optional[str] = None, is_input: bool = False, add_labels: Optional[List[str]] = None*)

Bases: pytext.data.tensorizers.Tensorizer

Numberize labels. Label can be used as either input or target.

NB: if the labels are used as targets for binary classification with a loss such as cosine distance, the order of the *label_vocab does* matter, and it should be *[negative_class, positive_class]*.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.LabelTensorizer.Config*)

**initialize**(*from_scratch=True*)
Look through the dataset for all labels and create a vocab map for them.

**numberize**(*row*)
Numberize labels.

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**MetricTensorizer**(*names: List[str], indexes: List[int], is_input: bool = False*)

Bases: *pytext.data.tensorizers.Tensorizer*

A tensorizer which use other tensorizers' numerized data. Used mostly for metric reporting.

**classmethod from_config**(*config: pytext.data.tensorizers.MetricTensorizer.Config*)

**numberize**(*row*)

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**NtokensTensorizer**(*names: List[str], indexes: List[int], is_input: bool = False*)

Bases: *pytext.data.tensorizers.MetricTensorizer*

A tensorizer which will reference another tensorizer's numerized data to calculate the num tokens. Used for calculating tokens per second.

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**NumericLabelTensorizer**(*label_column: str = 'label', rescale_range: Optional[List[float]] = None, is_input: bool = False*)

Bases: *pytext.data.tensorizers.Tensorizer*

Numberize numeric labels.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.NumericLabelTensorizer.Config*)

**numberize**(*row*)

Numberize labels.

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**SeqTokenTensorizer**(*column: str = 'text_seq', tokenizer=None, add_bos_token: bool = False, add_eos_token: bool = False, use_eos_token_for_bos: bool = False, add_bol_token: bool = False, add_eol_token: bool = False, use_eol_token_for_bol: bool = False, max_seq_len=None, vocab=None, is_input: bool = True, max_turn=50*)

Bases: *pytext.data.tensorizers.Tensorizer*

Tensorize a sequence of sentences. The input is a list of strings, like this one:

```
["where do you wanna meet?", "MPK"]
```

if we assume this vocab

```
vocab  {
  UNK: 0, PAD: 1,
  'where': 2, 'do': 3, 'you': 4, 'wanna': 5, 'meet?': 6, 'mpk': 7
}
```

this example will result in those tensors:

```
idx = [[2, 3, 4, 5, 6], [7, 1, 1, 1, 1]]
sentence_len = [5, 1]
seq_len = [2]
```

If you're using BOS, EOS, BOL and EOL, the vocab will look like this

```
vocab  {
  UNK: 0, PAD: 1,  BOS: 2, EOS: 3, BOL: 4, EOL: 5
  'where': 6, 'do': 7, 'you': 8, 'wanna': 9, 'meet?': 10, 'mpk': 11
}
```

this example will result in those tensors:

```
idx = [
    [2,  4, 3, 1, 1,  1, 1],
    [2,  6, 7, 8, 9, 10, 3],
    [2, 11, 3, 1, 1,  1, 1],
    [2,  5, 3, 1, 1,  1, 1]
]
sentence_len = [3, 8, 3, 3]
seq_len = [4]
```

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.SeqTokenTensorizer.Config*)

**initialize**(*vocab_builder=None*, *from_scratch=True*)
Build vocabulary based on training corpus.

**numberize**(*row*)
Tokenize, look up in vocabulary.

**prepare_input**(*row*)
Tokenize, return tokenized_texts in raw text

**sort_key**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**SlotLabelTensorizer**(*slot_column: str = 'slots', text_column: str = 'text', tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, allow_unknown: bool = False, is_input: bool = False*)

Bases: `pytext.data.tensorizers.Tensorizer`

Numberize word/slot labels.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.SlotLabelTensorizer.Config*)

**initialize**(*from_scratch=True*)
Look through the dataset for all labels and create a vocab map for them.

**numberize**(*row*)
Turn slot labels and text into a list of token labels with the same length as the number of tokens in the text.

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**SlotLabelTensorizerExpansible**(*slot_column: str = 'slots', text_column: str = 'text', tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, allow_unknown: bool = False, is_input: bool = False*)

Bases: *pytext.data.tensorizers.SlotLabelTensorizer*

Create a base SlotLabelTensorizer to support selecting different types in ModelInput.

**class** pytext.data.tensorizers.**SoftLabelTensorizer**(*label_column: str = 'label', allow_unknown: bool = False, pad_in_vocab: bool = False, label_vocab: Optional[List[str]] = None, probs_column: str = 'target_probs', logits_column: str = 'target_logits', labels_column: str = 'target_labels', label_vocab_file: Optional[str] = None, is_input: bool = False*)

Bases: *pytext.data.tensorizers.LabelTensorizer*

Handles numberizing labels for knowledge distillation. This still requires the same label column as *LabelTensorizer* for the "true" label, but also processes soft "probabilistic" labels generated from a teacher model, via three new columns.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.SoftLabelTensorizer.Config*)

**numberize**(*row*)
Numberize hard and soft labels

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**String2DListTensorizer**(*column, vocab_config=None, vocab=None, vocab_file_delimiter=' ', is_input=True*)

Bases: *pytext.data.tensorizers.Tensorizer*

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.String2DListTensorizer.Config*)

**initialize**(*from_scratch=True*)
The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**class** pytext.data.tensorizers.**String2DListTensorizerScriptImpl**(*vocab: pytext.data.utils.Vocabulary*)

Bases: *pytext.data.tensorizers.TensorizerScriptImpl*

**forward**(*inputs: List[List[List[str]]]*) → Tuple[torch.Tensor, torch.Tensor]
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**numberize**(*tokens: List[List[str]]*) → Tuple[List[List[int]], List[int], int]
This functions will receive the outputs from function: tokenize() or will be called directly from PyTextTensorizer function: numberize().

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**tensorize**(*tokens_3d: List[List[List[int]]], seq_lens_2d: List[List[int]], seq_lens_1d: List[int]*) → Tuple[torch.Tensor, torch.Tensor]
This functions will receive a list(e.g a batch) of outputs from function numberize(), padding and convert to output tensors.

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**class** pytext.data.tensorizers.**Tensorizer**(*is_input: bool = True*)

Bases: *pytext.config.component.Component*

Tensorizers are a component that converts from batches of *pytext.data.type.DataType* instances to tensors. These tensors will eventually be inputs to the model, but the model is aware of the tensorizers and can arrange the tensors they create to conform to its model.

Tensorizers have an initialize function. This function allows the tensorizer to read through the training dataset to build up any data that it needs for creating the model. Commonly this is valuable for things like inferring a vocabulary from the training set, or learning the entire set of training labels, or slot labels, etc.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.Tensorizer.Config*)

**initialize**(*from_scratch=True*)

The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row*)

**prepare_input**(*row*)

Return preprocessed input tensors/blob for caffe2 prediction net.

**sort_key**(*row*)

**stringify**(*token_indices*)

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**torchscriptify**()

**class** pytext.data.tensorizers.**TensorizerScriptImpl**

Bases: torch.nn.modules.module.Module

**batch_size**(*inputs: pytext.torchscript.utils.ScriptBatchInput*) → int

**get_texts_by_index**(*texts: Optional[List[List[str]]], index: int*) → Optional[List[str]]

**get_tokens_by_index**(*tokens: Optional[List[List[List[str]]]], index: int*) → Optional[List[List[str]]]

**numberize**(**args*, ***kwargs*)
This functions will receive the outputs from function: tokenize() or will be called directly from PyTextTensorizer function: numberize().

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**row_size**(*inputs: pytext.torchscript.utils.ScriptBatchInput*) → int

**set_device**(*device: str*)

**set_padding_control**(*dimension: str, padding_control: Optional[List[int]]*)
This functions will be called to set a padding style. None - No padding List: first element 0, round seq length to the smallest list element larger than inputs

**tensorize**(**args*, ***kwargs*)
This functions will receive a list(e.g a batch) of outputs from function numberize(), padding and convert to output tensors.

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**tensorize_wrapper**(**args*, ***kwargs*)
This functions will receive a list(e.g a batch) of outputs from function numberize(), padding and convert to output tensors.

It will be called in PyText Tensorizer during training time, this function is not torchscriptiable because it depends on cuda.device().

**tokenize**(**args*, ***kwargs*)
This functions will receive the inputs from Clients, usually there are two possible inputs 1) a row of texts: List[str] 2) a row of pre-processed tokens: List[List[str]]

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**torchscriptify**()

**class** pytext.data.tensorizers.**TokenTensorizer**(*text_column*, *tokenizer=None*, *add_bos_token=False*, *add_eos_token=False*, *use_eos_token_for_bos=False*, *max_seq_len=None*, *vocab_config=None*, *vocab=None*, *vocab_file_delimiter=' '*, *is_input=True*)

Bases: *pytext.data.tensorizers.Tensorizer*

Convert text to a list of tokens. Do this based on a tokenizer configuration, and build a vocabulary for numberization. Finally, pad the batch to create a square tensor of the correct size.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.TokenTensorizer.Config*)

**initialize**(*vocab_builder=None*, *from_scratch=True*)
Build vocabulary based on training corpus.

**numberize**(*row*)
Tokenize, look up in vocabulary.

**prepare_input**(*row*)
Tokenize, look up in vocabulary, return tokenized_texts in raw text

**sort_key**(*row*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**UidTensorizer**(*uid_column: str = 'uid', allow_unknown: bool = True, is_input: bool = True*)
Bases: pytext.data.tensorizers.Tensorizer

Numberize user IDs which can be either strings or tensors.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.UidTensorizer.Config*)

**initialize**(*from_scratch=True*)
Look through the dataset for all uids and create a vocab map for them.

**numberize**(*row*)
Numberize uids.

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**class** pytext.data.tensorizers.**VocabConfig**(***kwargs*)
Bases: pytext.config.component.Component.Config

**build_from_data = True**
Whether to add tokens from training data to vocab.

**min_counts = 0**
Add *min_counts* filter out tokens in training data that with count smaller than min_counts.

**size_from_data = 0**
Add *size_from_data* most frequent tokens in training data to vocab (if this is 0, add all tokens from training data).

**vocab_files = []**

**class** pytext.data.tensorizers.**VocabFileConfig**(***kwargs*)
Bases: pytext.config.component.Component.Config

**filepath = ''**
File containing tokens to add to vocab (first whitespace-separated entry per line)

**lowercase_tokens = False**
Whether to lowercase each of the tokens in the file

**size_limit = 0**
The max number of tokens to add to vocab

**skip_header_line = False**
Whether to skip the first line of the file (e.g. if it is a header line)

pytext.data.tensorizers.**initialize_tensorizers**(*tensorizers, data_source, from_scratch=True*)
A utility function to stream a data source to the initialize functions of a dict of tensorizers.

pytext.data.tensorizers.**lookup_tokens**(*text: str = None, pre_tokenized: List[pytext.data.tokenizers.tokenizer.Token] = None, tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, vocab: pytext.data.utils.Vocabulary = None, bos_token: Optional[str] = None, eos_token: Optional[str] = None, pad_token: str = '__PAD__', use_eos_token_for_bos: bool = False, max_seq_len: int = 1073741824*)

pytext.data.tensorizers.**to_device**(*tensorizer_script_impl, device*)

pytext.data.tensorizers.**tokenize**(*text: str = None, pre_tokenized: List[pytext.data.tokenizers.tokenizer.Token] = None, tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, bos_token: Optional[str] = None, eos_token: Optional[str] = None, pad_token: str = '__PAD__', use_eos_token_for_bos: bool = False, max_seq_len: int = 1073741824*)

### pytext.data.token_tensorizer module

**class** pytext.data.token_tensorizer.**ScriptBasedTokenTensorizer**(*text_column, tokenizer=None, add_bos_token=False, add_eos_token=False, use_eos_token_for_bos=False, max_seq_len=None, vocab_config=None, vocab=None, vocab_file_delimiter=' ', is_input=True*)

Bases: *pytext.data.tensorizers.Tensorizer*

An Implementation of TokenTensorizer that uses a TorchScript module in the background and is hence torchscriptifiable.

Note that unlike the original TokenTensorizer, this version cannot deal with arbitrarily nested lists of tokens.

**column_schema**

Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.token_tensorizer.ScriptBasedTokenTensorizer.Config*)

**initialize**(*vocab_builder=None, from_scratch=True*)

Build vocabulary based on training corpus.

**numberize**(*row*)

Tokenize and look up in vocabulary.

A few notable things:

1) We're using the non-torchsciptified tokenizer here. This allows us to use non-torchscriptifiable tokenizers if we don't intend to torchscriptify this module.

2) When using the ScriptImpl to do the lookup, it takes care of the BOS / EOS stuff there. Hence we don't need to do that with the tokenizer.

3) The tokenize function from tensorizer.py returns a tuple of (tokens, start_indices, end_indices), while the ScriptImpl expects a list of (token, start_idx, end_idx) tuples so we need to unzip these

**prepare_input**(*row*)

Tokenize, look up in vocabulary, return tokenized_texts in raw text

Similarly to the above function, tokenization is done with the original and not the torchscriptified tokenizer.

**sort_key**(*row*)

**tensorize**(*batch*)

Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**class** pytext.data.token_tensorizer.**TokenTensorizerScriptImpl**(*add_bos_token: bool, add_eos_token: bool, use_eos_token_for_bos: bool, max_seq_len: int, vocab: pytext.data.utils.Vocabulary, tokenizer: Optional[pytext.data.tokenizers.tokenizer.Toke*

Bases: *pytext.data.tensorizers.TensorizerScriptImpl*

**forward**(*inputs: pytext.torchscript.utils.ScriptBatchInput*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_texts_by_index**(*texts: Optional[List[List[str]]], index: int*) → Optional[str]

**get_tokens_by_index**(*tokens: Optional[List[List[List[str]]]], index: int*) → Optional[List[str]]

**numberize**(*text_tokens: List[Tuple[str, int, int]]*) → Tuple[List[int], int, List[Tuple[int, int]]]

This functions will receive the outputs from function: tokenize() or will be called directly from PyTextTensorizer function: numberize().

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**tensorize**(*tokens_2d: List[List[int]], seq_lens_1d: List[int], positions_2d: List[List[Tuple[int, int]]]*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

This functions will receive a list(e.g a batch) of outputs from function numberize(), padding and convert to output tensors.

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

**tokenize**(*row_text: Optional[str], row_pre_tokenized: Optional[List[str]]*) → List[Tuple[str, int, int]]

This functions will receive the inputs from Clients, usually there are two possible inputs 1) a row of texts: List[str] 2) a row of pre-processed tokens: List[List[str]]

Override this function to be TorchScriptable, e.g you need to declare concrete input arguments with type hints.

### pytext.data.utils module

**class** pytext.data.utils.**VocabBuilder**(*delimiter=' '*)

Bases: object

Helper class for aggregating and building *Vocabulary* objects.

**add**(*value*) → None

Count a single value in the vocabulary.

**add_all**(*values*) → None

Count a value or nested container of values in the vocabulary.

**add_from_file**(*file_pointer*, *skip_header_line*, *lowercase_tokens*, *size*)

**has_added_tokens**()

**make_vocab**() → pytext.data.utils.Vocabulary

Build a Vocabulary object from the values seen by the builder.

**truncate_to_vocab_size**(*vocab_size=-1*, *min_counts=-1*) → None

**class** pytext.data.utils.**Vocabulary**(*vocab_list: List[str], counts: List[T] = None, replacements: Optional[Dict[str, str]] = None, unk_token: str = '__UNKNOWN__', pad_token: str = '__PAD__', bos_token: str = '__BEGIN_OF_SENTENCE__', eos_token: str = '__END_OF_SENTENCE__', mask_token: str = '__MASK__'*)

Bases: object

A mapping from indices to vocab elements.

**get_bos_index**(*value=None*)

**get_eos_index**(*value=None*)

**get_mask_index**(*value=None*)

**get_pad_index**(*value=None*)

**get_unk_index**(*value=None*)

**lookup_all**(*nested_values*)

**lookup_all_internal**(*nested_values*)

Look up a value or nested container of values in the vocab index. The return value will have the same shape as the input, with all values replaced with their respective indicies.

**replace_tokens**(*replacements*)

Replace tokens in vocab with given replacement. Used for replacing special strings for special tokens. e.g. '[UNK]' for UNK

pytext.data.utils.**align_target_label**(*targets: List[float], labels: List[str], label_vocab: Dict[str, int]*) → List[float]

Given *targets* that are ordered according to *labels*, align the targets to match the order of *label_vocab*.

pytext.data.utils.**align_target_labels**(*targets_list: List[List[float]], labels_list: List[List[str]], label_vocab: Dict[str, int]*) → List[List[float]]

Given *targets_list* that are ordered according to *labels_list*, align the targets to match the order of *label_vocab*.

pytext.data.utils.**pad**(*nested_lists*, *pad_token*, *pad_shape=None*)
Pad the input lists with the pad token. If pad_shape is provided, pad to that shape, otherwise infer the input shape and pad out to a square tensor shape.

pytext.data.utils.**pad_and_tensorize**(*batch*, *pad_token=0*, *pad_shape=None*, *dtype=torch.int64*)

pytext.data.utils.**shard**(*rows*, *rank*, *num_workers*)
Only return every num_workers example for distributed training.

pytext.data.utils.**should_iter**(*i*)
Whether or not an object looks like a python iterable (not including strings).

### pytext.data.xlm_constants module

### pytext.data.xlm_dictionary module

**class** pytext.data.xlm_dictionary.**Dictionary**(*id2word*, *word2id*, *counts*)
Bases: object

**check_valid**()
Check that the dictionary is valid.

**index**(*word*, *no_unk=False*)
Returns the index of the specified word.

**static index_data**(*path*, *bin_path*, *dico*)
Index sentences with a dictionary.

**max_vocab**(*max_vocab*)
Limit the vocabulary size.

**min_count**(*min_count*)
Threshold on the word frequency counts.

**static read_vocab**(*vocab_path*)
Create a dictionary from a vocabulary file.

### pytext.data.xlm_tensorizer module

**class** pytext.data.xlm_tensorizer.**XLMTensorizer**(*columns: List[str] = ['text'], vocab: pytext.data.utils.Vocabulary = None, tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None, max_seq_len: int = 256, language_column: str = 'language', lang2id: Dict[str, int] = {'ar': 0, 'bg': 1, 'de': 2, 'el': 3, 'en': 4, 'es': 5, 'fr': 6, 'hi': 7, 'ru': 8, 'sw': 9, 'th': 10, 'tr': 11, 'ur': 12, 'vi': 13, 'zh': 14}, use_language_embeddings: bool = True, has_language_in_data: bool = False*)
Bases: pytext.data.bert_tensorizer.BERTTensorizerBase

Tensorizer for Cross-lingual LM tasks. Works for single sentence as well as sentence pair.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.xlm_tensorizer.XLMTensorizer.Config*)

**get_lang_id**(*row: Dict[KT, VT], col: str*) → int

**numberize**(*row: Dict[KT, VT]*) → Tuple[Any, ...]
This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model.

**tensorizer_script_impl = None**

**class** pytext.data.xlm_tensorizer.**XLMTensorizerScriptImpl**(*tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer, vocab: pytext.data.utils.Vocabulary, max_seq_len: int, language_vocab: List[str], default_language: str*)

Bases: *pytext.data.bert_tensorizer.BERTTensorizerBaseScriptImpl*

**forward**(*inputs: pytext.torchscript.utils.ScriptBatchInput*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]
Wire up tokenize(), numberize() and tensorize() functions for data processing.

**numberize**(*per_sentence_tokens: List[List[Tuple[str, int, int]]], per_sentence_languages: List[int]*) → Tuple[List[int], List[int], int, List[int]]
This function contains logic for converting tokens into ids based on the specified vocab. It also outputs, for each instance, the vectors needed to run the actual model.

**Parameters**

- **per_sentence_tokens** – list of tokens per sentence level in one row,
- **token represented by token string, start and end indices.** (*each*) –

**Returns** List[int], a list of token ids, concatenate all sentences token ids. segment_labels: List[int], denotes each token belong to which sentence. seq_len: int, tokens length positions: List[int], token positions

**Return type** tokens

## Module contents

**class** pytext.data.**AlternatingRandomizedBatchSampler**(*unnormalized_iterator_probs: Dict[str, float], second_unnormalized_iterator_probs: Dict[str, float]*)

Bases: *pytext.data.batch_sampler.RandomizedBatchSampler*

This sampler takes in a dictionary of iterators and returns batches alternating between keys and probabilities specified by *unnormalized_iterator_probs* and 'second_unnormalized_iterator_probs', This is used for example in XLM pre-training where we alternate between MLM and TLM batches.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.data.batch_sampler.AlternatingRandomizedBatchSampler.Config*)

**class** pytext.data.**Batcher**(*train_batch_size=16*, *eval_batch_size=16*, *test_batch_size=16*)

Bases: *pytext.config.component.Component*

Batcher designed to batch rows of data, before padding.

**batchify**(*iterable: Iterable[pytext.data.sources.data_source.RawExample], sort_key=None, stage=<Stage.TRAIN: 'Training'>*)

Group rows by batch_size. Assume iterable of dicts, yield dict of lists. The last batch will be of length len(iterable) % batch_size.

**classmethod from_config**(*config: pytext.data.data.Batcher.Config*)

**class** pytext.data.**BaseBatchSampler**

Bases: *pytext.config.component.Component*

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.config.component.Component.Config*)

**class** pytext.data.**BatchIterator**(*batches*, *processor*, *include_input=True*, *include_target=True*, *include_context=True*, *is_train=True*, *num_batches=0*)

Bases: object

BatchIterator is a wrapper of TorchText. Iterator that provide flexibility to map batched data to a tuple of (input, target, context) and other additional steps such as dealing with distributed training.

**Parameters**

- **batches** (*Iterator[TorchText.Batch]*) – iterator of TorchText.Batch, which shuffles/batches the data in __iter__ and return a batch of data in __next__
- **processor** – function to run after getting batched data from TorchText.Iterator, the function should define a way to map to data into (input, target, context)
- **include_input** (*bool*) – if input data should be returned, default is true
- **include_target** (*bool*) – if target data should be returned, default is true
- **include_context** (*bool*) – if context data should be returned, default is true
- **is_train** (*bool*) – if the batch data is for training
- **num_batches** (*int*) – total batches to generate, this param if for distributed training due to a limitation in PyTorch's distributed training backend that enforces all the parallel workers to have the same number of batches we workaround it by adding dummy batches at the end

**class** pytext.data.**CommonMetadata**

Bases: object

**class** pytext.data.**Data**(*data_source: pytext.data.sources.data_source.DataSource, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], batcher: pytext.data.data.Batcher = None, sort_key: Optional[str] = None, in_memory: Optional[bool] = True, init_tensorizers: Optional[bool] = True, init_tensorizers_from_scratch: Optional[bool] = True*)

Bases: *pytext.config.component.Component*

Data is an abstraction that handles all of the following:

- Initialize model metadata parameters
- Create batches of tensors for model training or prediction

It can accomplish these in any way it needs to. The base implementation utilizes *pytext.data.sources.DataSource*, and sends batches to *pytext.data.tensorizers.Tensorizer* to create tensors.

The *tensorizers* dict passed to the initializer should be considered something like a signature for the model. Each batch should be a dictionary with the same keys as the *tensorizers* dict, and values should be tensors arranged in the way specified by that tensorizer. The tensorizers dict doubles as a simple baseline implementation of that same signature, but subclasses of Data can override the implementation using other methods. This value is how the model specifies what inputs it's looking for.

**add_row_indices**(*rows*)

**batches**(*stage: pytext.common.constants.Stage*, *data_source=None*, *load_early=False*)

Create batches of tensors to pass to model train_batch. This function yields dictionaries that mirror the *tensorizers* dict passed to *__init__*, ie. the keys will be the same, and the tensors will be the shape expected from the respective tensorizers.

*stage* is used to determine which data source is used to create batches. if data_source is provided, it is used instead of the configured data_sorce this is to allow setting a different data_source for testing a model.

Passing in *load_early* = True disables loading all data in memory and using PoolingBatcher, so that we get the first batch as quickly as possible.

**cache**(*numberized_rows*, *stage*)

**classmethod from_config**(*config: pytext.data.data.Data.Config, schema: Dict[str, Type[CT_co]], tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], rank=0, world_size=1, init_tensorizers=True, **kwargs*)

**numberize_rows**(*rows*)

**class** pytext.data.**DataHandler**(*raw_columns: List[str], labels: Dict[str, pytext.fields.field.Field], features: Dict[str, pytext.fields.field.Field], featurizer: pytext.data.featurizer.featurizer.Featurizer, extra_fields: Dict[str, pytext.fields.field.Field] = None, text_feature_name: str = 'word_feat', shuffle: bool = True, sort_within_batch: bool = True, train_path: str = 'train.tsv', eval_path: str = 'eval.tsv', test_path: str = 'test.tsv', train_batch_size: int = 128, eval_batch_size: int = 128, test_batch_size: int = 128, max_seq_len: int = -1, pass_index: bool = True, column_mapping: Dict[str, str] = None, **kwargs*)

Bases: `pytext.config.component.Component`

DataHandler is the central place to prepare data for model training/testing. The class is responsible of:

- Define pipeline to process data and generate batch of tensors to be consumed by model. Each batch is a (input, target, extra_data) tuple, in which input can be feed directly into model.
- Initialize global context, such as build vocab, load pretrained embeddings. Store the context as metadata, and provide function to serialize/deserialize the metadata

The data processing pipeline contains the following steps:

- Read data from file into a list of raw data examples
- Convert each row of row data to a TorchText Example. This logic happens in process_row function and will:
  - Invoke featurizer, which contains data processing steps to apply for both training and inference time, e.g: tokenization
  - Use the raw data and results from featurizer to do any preprocessing
- Generate a TorchText.Dataset that contains the list of Example, the Dataset also has a list of TorchText.Field, which defines how to do padding and numericalization while batching data.

- Return a BatchIterator which will give a tuple of (input, target, context) tensors for each iteration. By default the tensors have a 1:1 mapping to the TorchText.Field fields, but this behavior can be overwritten by _input_from_batch, _target_from_batch, _context_from_batch functions.

**raw_columns**
columns to read from data source. The order should match the data stored in that file.

**Type** List[str]

**featurizer**
perform data preprocessing that should be shared between training and inference

**Type** Featurizer

**features**
a dict of name -> field that used to process data as model input

**Type** Dict[str, Field]

**labels**
a dict of name -> field that used to process data as training target

**Type** Dict[str, Field]

**extra_fields**
fields that process any extra data used neither as model input nor target. This is None by default

**Type** Dict[str, Field]

**text_feature_name**
name of the text field, used to define the default sort key of data

**Type** str

**shuffle**
if the dataset should be shuffled, true by default

**Type** bool

**sort_within_batch**
if data within same batch should be sorted, true by default

**Type** bool

**train_path**
path of training data file

**Type** str

**eval_path**
path of evaluation data file

**Type** str

**test_path**
path of test data file

**Type** str

**train_batch_size**
training batch size, 128 by default

**Type** int

**eval_batch_size**
evaluation batch size, 128 by default

**Type** int

**test_batch_size**
test batch size, 128 by default

**Type** int

**max_seq_len**
maximum length of tokens to keep in sequence

**Type** int

**pass_index**
if the original index of data in the batch should be passed along to downstream steps, default is true

**Type** bool

**gen_dataset**(*data: Iterable[Dict[str, Any]], include_label_fields: bool = True, shard_range: Tuple[int, int] = None*) → torchtext.legacy.data.dataset.Dataset
Generate torchtext Dataset from raw in memory data. :returns: dataset (TorchText.Dataset)

**gen_dataset_from_path**(*path: str, rank: int = 0, world_size: int = 1, include_label_fields: bool = True, use_cache: bool = True*) → torchtext.legacy.data.dataset.Dataset
Generate a dataset from file :returns: dataset (TorchText.Dataset)

**get_eval_iter**()

**get_predict_iter**(*data: Iterable[Dict[str, Any]], batch_size: Optional[int] = None*)

**get_test_iter**()

**get_test_iter_from_path**(*test_path: str, batch_size: int*) → pytext.data.data_handler.BatchIterator

**get_test_iter_from_raw_data**(*test_data: List[Dict[str, Any]], batch_size: int*) → pytext.data.data_handler.BatchIterator

**get_train_iter**(*rank: int = 0, world_size: int = 1*)

**get_train_iter_from_path**(*train_path: str, batch_size: int, rank: int = 0, world_size: int = 1*) → pytext.data.data_handler.BatchIterator
Generate data batch iterator for training data. See *_get_train_iter()* for details

**Parameters**

- **train_path** (*str*) – file path of training data
- **batch_size** (*int*) – batch size
- **rank** (*int*) – used for distributed training, the rank of current Gpu, don't set it to anything but 0 for non-distributed training
- **world_size** (*int*) – used for distributed training, total number of Gpu

**get_train_iter_from_raw_data**(*train_data: List[Dict[str, Any]], batch_size: int, rank: int = 0, world_size: int = 1*) → pytext.data.data_handler.BatchIterator

**init_feature_metadata**(*train_data: torchtext.legacy.data.dataset.Dataset, eval_data: torchtext.legacy.data.dataset.Dataset, test_data: torchtext.legacy.data.dataset.Dataset*)

**init_metadata**()
Initialize metadata using data from configured path

**init_metadata_from_path**(*train_path, eval_path, test_path*)
Initialize metadata using data from file

**init_metadata_from_raw_data**(**data*)
Initialize metadata using in memory data

**init_target_metadata**(*train_data: torchtext.legacy.data.dataset.Dataset, eval_data: torchtext.legacy.data.dataset.Dataset, test_data: torchtext.legacy.data.dataset.Dataset*)

**load_metadata**(*metadata: pytext.data.data_handler.CommonMetadata*)
Load previously saved metadata

**load_vocab**(*vocab_file, vocab_size, lowercase_tokens: bool = False*)
Loads items into a set from a file containing one item per line. Items are added to the set from top of the file to bottom. So, the items in the file should be ordered by a preference (if any), e.g., it makes sense to order tokens in descending order of frequency in corpus.

**Parameters**

- **vocab_file** (*str*) – vocab file to load
- **vocab_size** (*int*) – maximum tokens to load, will only load the first n if the actual vocab size is larger than this parameter
- **lowercase_tokens** (*bool*) – if the tokens should be lowercased

**metadata_to_save**()
Save metadata, pretrained_embeds_weight should be excluded

**preprocess**(*data: Iterable[Dict[str, Any]]*)
preprocess the raw data to create TorchText.Example, this is the second step in whole processing pipeline :returns: data (Generator[Dict[str, Any]])

**preprocess_row**(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

**read_from_file**(*file_name: str, columns_to_use: Union[Dict[str, int], List[str]]*) → Generator[Dict[KT, VT], None, None]
Read data from csv file. Input file format is required to be tab-separated columns

**Parameters**

- **file_name** (*str*) – csv file name
- **columns_to_use** (*Union[Dict[str, int], List[str]]*) – either a list of column names or a dict of column name -> column index in the file

**sort_key**(*example: torchtext.legacy.data.example.Example*) → Any
How to sort data in every batch, default behavior is by the length of input text :param example: one torchtext example :type example: Example

**class** pytext.data.**DisjointMultitaskData**(*data_dict: Dict[str, pytext.data.data.Data], samplers: Dict[pytext.common.constants.Stage, pytext.data.batch_sampler.BaseBatchSampler], test_key: str = None, task_key: str = 'task_name'*)

Bases: *pytext.data.data.Data*

Wrapper for doing multitask training using multiple data objects. Takes a dictionary of data objects, does round robin over their iterators using BatchSampler.

**Parameters**

- **config** (*Config*) – Configuration object of type DisjointMultitaskData.Config.
- **data_dict** (*Dict[str, Data]*) – Data objects to do roundrobin over.
- ***args** (*type*) – Extra arguments to be passed down to sub data handlers.

- ****kwargs** (*type*) – Extra arguments to be passed down to sub data handlers.

**data_dict**
Data handlers to do roundrobin over.

**Type** type

**batches**(*stage: pytext.common.constants.Stage*, *data_source=None*, *load_early=False*)
Yield batches from each task, sampled according to a given sampler. This batcher additionally exposes a task name in the batch to allow the model to filter examples to the appropriate tasks.

**classmethod from_config**(*config: pytext.data.disjoint_multitask_data.DisjointMultitaskData.Config, data_dict: Dict[str, pytext.data.data.Data], task_key: str = 'task_name', rank=0, world_size=1, init_tensorizers=True*)

**class** pytext.data.**DisjointMultitaskDataHandler**(*config: pytext.data.disjoint_multitask_data_handler.DisjointMultitaskData.*, *data_handlers: Dict[str, pytext.data.data_handler.DataHandler], target_task_name: Optional[str] = None, *args, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

Wrapper for doing multitask training using multiple data handlers. Takes a dictionary of data handlers, does round robin over their iterators using RoundRobinBatchIterator.

**Parameters**

- **config** (*Config*) – Configuration object of type DisjointMultitaskDataHandler.Config.
- **data_handlers** (*Dict[str, DataHandler]*) – Data handlers to do roundrobin over.
- **target_task_name** (*Optional[str]*) – Used to select best epoch, and set batch_per_epoch.
- ***args** (*type*) – Extra arguments to be passed down to sub data handlers.
- ****kwargs** (*type*) – Extra arguments to be passed down to sub data handlers.

**data_handlers**
Data handlers to do roundrobin over.

**Type** type

**target_task_name**
Used to select best epoch, and set batch_per_epoch.

**Type** type

**upsample**
If upsample, keep cycling over each iterator in round-robin. Iterators with less batches will get more passes. If False, we do single pass over each iterator, the ones which run out will sit idle. This is used for evaluation. Default True.

**Type** bool

**get_eval_iter**() → pytext.data.data_handler.BatchIterator

**get_test_iter**() → pytext.data.data_handler.BatchIterator

**get_train_iter**(*rank: int = 0*, *world_size: int = 1*) → Tuple[pytext.data.data_handler.BatchIterator, ...]

**init_metadata**()
Initialize metadata using data from configured path

**load_metadata**(*metadata*)
Load previously saved metadata

**metadata_to_save**()
Save metadata, pretrained_embeds_weight should be excluded

**class** pytext.data.**DynamicPoolingBatcher**(*train_batch_size=16, eval_batch_size=16, test_batch_size=16, pool_num_batches=1000, num_shuffled_pools=1, scheduler_config=<pytext.data.dynamic_pooling_batcher.BatcherSchedulerConf object>*)

Bases: pytext.data.data.PoolingBatcher

Allows dynamic batch training, extends pooling batcher with a scheduler config, which specifies how batch size should increase

**batchify**(*iterable: Iterable[pytext.data.sources.data_source.RawExample], sort_key=None, stage=<Stage.TRAIN: 'Training'>*)
From an iterable of dicts, yield dicts of lists:

1. Load *num_shuffled_pools* pools of data, and shuffle them.
2. Load a pool (*batch_size * pool_num_batches* examples).
3. Sort rows, if necessary.
4. Shuffle the order in which the batches are returned, if necessary.

**compute_dynamic_batch_size**(*curr_epoch: int, scheduler_config: pytext.data.dynamic_pooling_batcher.BatcherSchedulerConfig, curr_steps: int*) → int

**finished_dynamic**() → bool

**classmethod from_config**(*config: pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher.Config*)

**get_batch_size**(*stage: pytext.common.constants.Stage*) → int

**step_epoch**()

**class** pytext.data.**EvalBatchSampler**

Bases: pytext.data.batch_sampler.BaseBatchSampler

This sampler takes in a dictionary of Iterators and returns batches associated with each key in the dictionary. It guarentees that we will see each batch associated with each key exactly once in the epoch.

### Example

Iterator 1: [A, B, C, D], Iterator 2: [a, b]

Output: [A, B, C, D, a, b]

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)
Loop through each key in the input dict and generate batches from the iterator associated with that key.

**Parameters** **iterators** – Dictionary of iterators

pytext.data.**generator_iterator**(*fn*)
Turn a generator into a GeneratorIterator-wrapped function. Effectively this allows iterating over a generator multiple times by recording the call arguments, and calling the generator with them anew each item __iter__ is called on the returned object.

**class** pytext.data.**PoolingBatcher**(*train_batch_size=16, eval_batch_size=16, test_batch_size=16, pool_num_batches=1000, num_shuffled_pools=1*)

Bases: `pytext.data.data.Batcher`

Batcher that shuffles and (if requested) sorts data.

**Rationale**

There is a trade-off between having batches of data that are truly randomly shuffled, and batches of data that are efficiently padded. If we wanted to maximise the efficiency of padding (i.e. minimise the amount of padding that is needed), we would have to enforce that all inputs of a similar length appear in the same batch. This however would lead to a dramatic decrease in the randomness of batches. On the other end of the spectrum, if we wanted to maximise randomness, we would often end up with inputs of wildly different lengths in the same batch, which would lead to a lot of padding.

**Operation**

This batcher uses a multi-staged approach.

1. It first loads a number of "pools" of data, and shuffles them (this is controlled by *num_shuffled_pools*).
2. It then splits up the shuffled data sequentially into individual pools, and the examples within each pool are sorted (if requested).
3. Finally, each pool is split up sequentially into batches, and yielded. If sorting was requested in step #2, the order in which the batches are yielded is randomised.

The size of a pool is expressed as a multiple of the batch size, and is controlled by *pool_num_batches*.

**Examples**

Assuming sorting is enabled, with the default settings of *pool_num_batches: 1000* and *num_shuffled_pools: 1*, a pool of *1k * batch_size* examples is loaded, sorted by length, and split up into 1k batches. These batches are then yielded in random order. Once they run out, a new pool is loaded, and the process is repeated. An advantage of this approach is that padding will be somewhat reduced. A disadvantage is that, for every epoch, the first 1k batches will be always the same (albeit in a different order).

On the other hand, specifying *pool_num_batches: 1000* and *num_shuffled_pools: 1000* would achieve the following: *1k * 1k * batch_size* examples are loaded, and shuffled. These are then split up into pools of size *1k * batch_size*, which are then sorted internally, split into individual batches, and yielded in random order. Compared to the previous example, we no longer have the problem that the first 1k batches are always the same in each epoch, but we've had to load in memory 1M examples.

**batchify**(*iterable: Iterable[pytext.data.sources.data_source.RawExample], sort_key=None, stage=<Stage.TRAIN: 'Training'>*)

From an iterable of dicts, yield dicts of lists:

1. Load *num_shuffled_pools* pools of data, and shuffle them.
2. Load a pool (*batch_size * pool_num_batches* examples).
3. Sort rows, if necessary.
4. Shuffle the order in which the batches are returned, if necessary.

**classmethod from_config**(*config: pytext.data.data.PoolingBatcher.Config*)

**get_batch_size**(*stage: pytext.common.constants.Stage*) → int

**class** pytext.data.**RandomizedBatchSampler**(*unnormalized_iterator_probs: Dict[str, float]*)

Bases: `pytext.data.batch_sampler.BaseBatchSampler`

This sampler takes in a dictionary of iterators and returns batches according to the specified probabilities by *unnormalized_iterator_probs*. We cycle through the iterators (restarting any that "run out") indefinitely. Set batches_per_epoch in Trainer.Config.

### Example

Iterator A: [A, B, C, D], Iterator B: [a, b]

batches_per_epoch = 3, unnormalized_iterator_probs = {"A": 0, "B": 1} Epoch 1 = [a, b, a] Epoch 2 = [b, a, b]

**Parameters** **unnormalized_iterator_probs** (*Dict[str, float]*) – Iterator sampling probabilities. The keys should be the same as the keys of the underlying iterators, and the values will be normalized to sum to 1.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.data.batch_sampler.RandomizedBatchSampler.Config*)

**class** pytext.data.**RoundRobinBatchSampler**(*iter_to_set_epoch: Optional[str] = None*)
Bases: *pytext.data.batch_sampler.BaseBatchSampler*

This sampler takes a dictionary of Iterators and returns batches in a round robin fashion till a the end of one of the iterators is reached. The end is specified by *iter_to_set_epoch*.

If *iter_to_set_epoch* is set, cycle batches from each iterator until one epoch of the target iterator is fulfilled. Iterators with fewer batches than the target iterator are repeated, so they never run out.

If *iter_to_set_epoch* is None, cycle over batches from each iterator until the shortest iterator completes one epoch.

### Example

Iterator 1: [A, B, C, D], Iterator 2: [a, b]

iter_to_set_epoch = "Iterator 1" Output: [A, a, B, b, C, a, D, b]

iter_to_set_epoch = None Output: [A, a, B, b]

**Parameters** **iter_to_set_epoch** (*Optional[str]*) – Name of iterator to define epoch size. If this is not set, epoch size defaults to the length of the shortest iterator.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)
Loop through each key in the input dict and generate batches from the iterator associated with that key until the target iterator reaches its end.

**Parameters** **iterators** – Dictionary of iterators

**classmethod from_config**(*config: pytext.data.batch_sampler.RoundRobinBatchSampler.Config*)

**class** pytext.data.**NaturalBatchSampler**(*dataset_counts: Dict[str, int]*)
Bases: *pytext.data.batch_sampler.RandomizedBatchSampler*

This sampler iterates over all the datasets, sampling according to the weighted number of samples in each dataset.

**batchify**(*iterators: Dict[str, collections.abc.Iterator]*)

**classmethod from_config**(*config: pytext.data.batch_sampler.NaturalBatchSampler.Config*)

**class** pytext.data.**Tensorizer**(*is_input: bool = True*)
Bases: *pytext.config.component.Component*

Tensorizers are a component that converts from batches of *pytext.data.type.DataType* instances to tensors. These tensors will eventually be inputs to the model, but the model is aware of the tensorizers and can arrange the tensors they create to conform to its model.

Tensorizers have an initialize function. This function allows the tensorizer to read through the training dataset to build up any data that it needs for creating the model. Commonly this is valuable for things like inferring a vocabulary from the training set, or learning the entire set of training labels, or slot labels, etc.

**column_schema**
Generic types don't pickle well pre-3.7, so we don't actually want to store the schema as an attribute. We're already storing all of the columns anyway, so until there's a better solution, schema is a property.

**classmethod from_config**(*config: pytext.data.tensorizers.Tensorizer.Config*)

**initialize**(*from_scratch=True*)
The initialize function is carefully designed to allow us to read through the training dataset only once, and not store it in memory. As such, it can't itself manually iterate over the data source. Instead, the initialize function is a coroutine, which is sent row data. This should look roughly like:

```
# set up variables here
...
try:
    # start reading through data source
    while True:
        # row has type Dict[str, types.DataType]
        row = yield
        # update any variables, vocabularies, etc.
        ...
except GeneratorExit:
    # finalize your initialization, set instance variables, etc.
    ...
```

See *WordTokenizer.initialize* for a more concrete example.

**numberize**(*row*)

**prepare_input**(*row*)
Return preprocessed input tensors/blob for caffe2 prediction net.

**sort_key**(*row*)

**stringify**(*token_indices*)

**tensorize**(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

**tensorizer_script_impl = None**

**torchscriptify**()

## pytext.exporters package

### Submodules

### pytext.exporters.custom_exporters module

**class** pytext.exporters.custom_exporters.**DenseFeatureExporter**(*config, input_names, dummy_model_input, vocab_map, output_names*)

Bases: *pytext.exporters.exporter.ModelExporter*

Exporter for models that have DenseFeatures as input to the decoder

**classmethod get_feature_metadata**(*feature_config: pytext.config.field_config.FeatureConfig, feature_meta: Dict[str, pytext.fields.field.FieldMeta]*)

**class** pytext.exporters.custom_exporters.**InitPredictNetExporter**(*config, input_names, dummy_model_input, vocab_map, output_names*)

Bases: *pytext.exporters.exporter.ModelExporter*

Exporter for converting models to their caffe2 init and predict nets. Does not rely on c2_prepared, but rather splits the ONNX model into the init and predict nets directly.

**export_to_caffe2**(*model, export_path: str, export_onnx_path: str = None*) → List[str]

export pytorch model to caffe2 by first using ONNX to convert logic in forward function to a caffe2 net, and then prepend/append additional operators to the caffe2 net according to the model

**Parameters**

- **model** (*Model*) – pytorch model to export
- **export_path** (*str*) – path to save the exported caffe2 model
- **export_onnx_path** (*str*) – path to save the exported onnx model

**Returns** list of caffe2 model output names

**Return type** final_output_names

**get_export_paths**(*path*)

**postprocess_output**(*init_net, predict_net, workspace, output_names: List[str], model*)

Postprocess the model output, generate additional blobs for human readable prediction. By default it use export function of output layer from pytorch model to append additional operators to caffe2 net

**Parameters**

- **init_net** (*caffe2.python.Net*) – caffe2 init net created by the current graph
- **predict_net** (*caffe2.python.Net*) – caffe2 net created by the current graph
- **workspace** (*caffe2.python.workspace*) – caffe2 current workspace
- **output_names** (*List[str]*) – current output names of the caffe2 net
- **py_model** (*Model*) – original pytorch model object

**Returns** list of blobs that will be added to the caffe2 model final_output_names: list of output names of the blobs to add

**Return type** result

**prepend_operators**(*init_net, predict_net, input_names: List[str]*)

Prepend operators to the converted caffe2 net, do nothing by default

**Parameters**

- **c2_prepared** (*Caffe2Rep*) – caffe2 net rep
- **input_names** (*List[str]*) – current input names to the caffe2 net

**Returns** caffe2 net with prepended operators input_names (List[str]): list of input names for the new net

**Return type** c2_prepared (Caffe2Rep)

`pytext.exporters.custom_exporters.`**`get_exporter`**(*name*)

`pytext.exporters.custom_exporters.`**`save_caffe2_pb_net`**(*path*, *model*)

## pytext.exporters.exporter module

**class** `pytext.exporters.exporter.`**`ModelExporter`**(*config*, *input_names*, *dummy_model_input*, *vocab_map*, *output_names*)

Bases: *pytext.config.component.Component*

Model exporter exports a PyTorch model to Caffe2 model using ONNX

**input_names**

names of the input variables to model forward function, in a flattened way. e.g: forward(tokens, dict) where tokens is List[Tensor] and dict is a tuple of value and length: (List[Tensor], List[Tensor]) the input names should looks like ['token', 'dict_value', 'dict_length']

**Type** List[Str]

**dummy_model_input**

dummy values to define the shape of input tensors, should exactly match the shape of the model forward function

**Type** Tuple[torch.Tensor]

**vocab_map**

dict of input feature names to corresponding index_to_string array, e.g:

```
{
    "text": ["<UNK>", "W1", "W2", "W3", "W4", "W5", "W6", "W7", "W8"],
    "dict": ["<UNK>", "D1", "D2", "D3", "D4", "D5", "D6", "D7", "D8"]
}
```

**Type** Dict[str, List[str]]

**output_names**

names of output variables

**Type** List[Str]

**export_to_caffe2**(*model*, *export_path: str*, *export_onnx_path: str = None*) → List[str]

export pytorch model to caffe2 by first using ONNX to convert logic in forward function to a caffe2 net, and then prepend/append additional operators to the caffe2 net according to the model

**Parameters**

- **model** (*Model*) – pytorch model to export
- **export_path** (*str*) – path to save the exported caffe2 model

- **export_onnx_path** (*str*) – path to save the exported onnx model

**Returns** list of caffe2 model output names

**Return type** final_output_names

**export_to_metrics** (*model*, *metric_channels*)

Exports the pytorch model to tensorboard as a graph.

**Parameters**

- **model** (*Model*) – pytorch model to export
- **metric_channels** (*List[Channel]*) – outputs of model's execution graph

**classmethod from_config** (*config, feature_config: pytext.config.field_config.FeatureConfig, target_config: Union[pytext.config.pytext_config.ConfigBase, List[pytext.config.pytext_config.ConfigBase]], meta: pytext.data.data_handler.CommonMetadata, *args, **kwargs*)

Gather all the necessary metadata from configs and global metadata to be used in exporter

**get_extra_params** () → List[str]

**Returns** list of blobs to be added as extra params to the caffe2 model

**classmethod get_feature_metadata** (*feature_config: pytext.config.field_config.FeatureConfig, feature_meta: Dict[str, pytext.fields.field.FieldMeta]*)

**postprocess_output** (*init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, output_names: List[str], py_model*)

Postprocess the model output, generate additional blobs for human readable prediction. By default it use export function of output layer from pytorch model to append additional operators to caffe2 net

**Parameters**

- **init_net** (*caffe2.python.Net*) – caffe2 init net created by the current graph
- **predict_net** (*caffe2.python.Net*) – caffe2 net created by the current graph
- **workspace** (*caffe2.python.workspace*) – caffe2 current workspace
- **output_names** (*List[str]*) – current output names of the caffe2 net
- **py_model** (*Model*) – original pytorch model object

**Returns** list of blobs that will be added to the caffe2 model final_output_names: list of output names of the blobs to add

**Return type** result

**prepend_operators** (*c2_prepared: caffe2.python.onnx.backend_rep.Caffe2Rep, input_names: List[str]*) → Tuple[caffe2.python.onnx.backend_rep.Caffe2Rep, List[str]]

Prepend operators to the converted caffe2 net, do nothing by default

**Parameters**

- **c2_prepared** (*Caffe2Rep*) – caffe2 net rep
- **input_names** (*List[str]*) – current input names to the caffe2 net

**Returns** caffe2 net with prepended operators input_names (List[str]): list of input names for the new net

**Return type** c2_prepared (Caffe2Rep)

### Module contents

**class** pytext.exporters.**ModelExporter**(*config*, *input_names*, *dummy_model_input*, *vocab_map*, *output_names*)

Bases: *pytext.config.component.Component*

Model exporter exports a PyTorch model to Caffe2 model using ONNX

**input_names**

names of the input variables to model forward function, in a flattened way. e.g: forward(tokens, dict) where tokens is List[Tensor] and dict is a tuple of value and length: (List[Tensor], List[Tensor]) the input names should looks like ['token', 'dict_value', 'dict_length']

**Type** List[Str]

**dummy_model_input**

dummy values to define the shape of input tensors, should exactly match the shape of the model forward function

**Type** Tuple[torch.Tensor]

**vocab_map**

dict of input feature names to corresponding index_to_string array, e.g:

```
{
    "text": ["<UNK>", "W1", "W2", "W3", "W4", "W5", "W6", "W7", "W8"],
    "dict": ["<UNK>", "D1", "D2", "D3", "D4", "D5", "D6", "D7", "D8"]
}
```

**Type** Dict[str, List[str]]

**output_names**

names of output variables

**Type** List[Str]

**export_to_caffe2**(*model*, *export_path: str*, *export_onnx_path: str = None*) → List[str]

export pytorch model to caffe2 by first using ONNX to convert logic in forward function to a caffe2 net, and then prepend/append additional operators to the caffe2 net according to the model

**Parameters**

- **model** (*Model*) – pytorch model to export
- **export_path** (*str*) – path to save the exported caffe2 model
- **export_onnx_path** (*str*) – path to save the exported onnx model

**Returns** list of caffe2 model output names

**Return type** final_output_names

**export_to_metrics**(*model*, *metric_channels*)

Exports the pytorch model to tensorboard as a graph.

**Parameters**

- **model** (*Model*) – pytorch model to export
- **metric_channels** (*List[Channel]*) – outputs of model's execution graph

**classmethod from_config**(*config, feature_config: pytext.config.field_config.FeatureConfig, target_config: Union[pytext.config.pytext_config.ConfigBase, List[pytext.config.pytext_config.ConfigBase]], meta: pytext.data.data_handler.CommonMetadata, *args, **kwargs*)

Gather all the necessary metadata from configs and global metadata to be used in exporter

**get_extra_params**() → List[str]

**Returns** list of blobs to be added as extra params to the caffe2 model

**classmethod get_feature_metadata**(*feature_config: pytext.config.field_config.FeatureConfig, feature_meta: Dict[str, pytext.fields.field.FieldMeta]*)

**postprocess_output**(*init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, output_names: List[str], py_model*)

Postprocess the model output, generate additional blobs for human readable prediction. By default it use export function of output layer from pytorch model to append additional operators to caffe2 net

**Parameters**

- **init_net** (*caffe2.python.Net*) – caffe2 init net created by the current graph
- **predict_net** (*caffe2.python.Net*) – caffe2 net created by the current graph
- **workspace** (*caffe2.python.workspace*) – caffe2 current workspace
- **output_names** (*List[str]*) – current output names of the caffe2 net
- **py_model** (*Model*) – original pytorch model object

**Returns** list of blobs that will be added to the caffe2 model final_output_names: list of output names of the blobs to add

**Return type** result

**prepend_operators**(*c2_prepared: caffe2.python.onnx.backend_rep.Caffe2Rep, input_names: List[str]*) → Tuple[caffe2.python.onnx.backend_rep.Caffe2Rep, List[str]]

Prepend operators to the converted caffe2 net, do nothing by default

**Parameters**

- **c2_prepared** (*Caffe2Rep*) – caffe2 net rep
- **input_names** (*List[str]*) – current input names to the caffe2 net

**Returns** caffe2 net with prepended operators input_names (List[str]): list of input names for the new net

**Return type** c2_prepared (Caffe2Rep)

**class** pytext.exporters.**DenseFeatureExporter**(*config, input_names, dummy_model_input, vocab_map, output_names*)

Bases: *pytext.exporters.exporter.ModelExporter*

Exporter for models that have DenseFeatures as input to the decoder

**classmethod get_feature_metadata**(*feature_config: pytext.config.field_config.FeatureConfig, feature_meta: Dict[str, pytext.fields.field.FieldMeta]*)

**class** pytext.exporters.**InitPredictNetExporter**(*config, input_names, dummy_model_input, vocab_map, output_names*)

Bases: *pytext.exporters.exporter.ModelExporter*

Exporter for converting models to their caffe2 init and predict nets. Does not rely on c2_prepared, but rather splits the ONNX model into the init and predict nets directly.

**export_to_caffe2** (*model*, *export_path: str*, *export_onnx_path: str = None*) → List[str]

export pytorch model to caffe2 by first using ONNX to convert logic in forward function to a caffe2 net, and then prepend/append additional operators to the caffe2 net according to the model

**Parameters**

- **model** (*Model*) – pytorch model to export
- **export_path** (*str*) – path to save the exported caffe2 model
- **export_onnx_path** (*str*) – path to save the exported onnx model

**Returns** list of caffe2 model output names

**Return type** final_output_names

**get_export_paths** (*path*)

**postprocess_output** (*init_net, predict_net, workspace, output_names: List[str], model*)

Postprocess the model output, generate additional blobs for human readable prediction. By default it use export function of output layer from pytorch model to append additional operators to caffe2 net

**Parameters**

- **init_net** (*caffe2.python.Net*) – caffe2 init net created by the current graph
- **predict_net** (*caffe2.python.Net*) – caffe2 net created by the current graph
- **workspace** (*caffe2.python.workspace*) – caffe2 current workspace
- **output_names** (*List[str]*) – current output names of the caffe2 net
- **py_model** (*Model*) – original pytorch model object

**Returns** list of blobs that will be added to the caffe2 model final_output_names: list of output names of the blobs to add

**Return type** result

**prepend_operators** (*init_net, predict_net, input_names: List[str]*)

Prepend operators to the converted caffe2 net, do nothing by default

**Parameters**

- **c2_prepared** (*Caffe2Rep*) – caffe2 net rep
- **input_names** (*List[str]*) – current input names to the caffe2 net

**Returns** caffe2 net with prepended operators input_names (List[str]): list of input names for the new net

**Return type** c2_prepared (Caffe2Rep)

## pytext.fields package

### Submodules

### pytext.fields.char_field module

**class** pytext.fields.char_field.**CharFeatureField**(*pad_token='<pad>', unk_token='<unk>', batch_first=True, max_word_length=20, min_freq=1, **kwargs*)

Bases: *pytext.fields.field.VocabUsingField*

**build_vocab**(**args, **kwargs*)

Construct the Vocab object for this field from one or more datasets.

**Parameters**

- **arguments** (*Positional*) – Dataset objects or other iterable data sources from which to construct the Vocab object that represents the set of possible values for this field. If a Dataset object is provided, all columns corresponding to this field are used; individual columns can also be provided directly.
- **keyword arguments** (*Remaining*) – Passed to the constructor of Vocab.

**dummy_model_input = tensor([[[1, 1, 1]], [[1, 1, 1]]])**

**numericalize**(*batch, device=None*)

Turn a batch of examples that use this field into a Variable.

If the field has include_lengths=True, a tensor of lengths will be included in the return value.

**Parameters**

- **arr** (*List[List[str]], or tuple of (List[List[str]], List[int])*) – List of tokenized and padded examples, or tuple of List of tokenized and padded examples and List of lengths of each example if self.include_lengths is True.
- **device** (*str or torch.device*) – A string or instance of *torch.device* specifying which device the Variables are going to be created on. If left as default, the tensors will be created on cpu. Default: None.

**pad**(*minibatch: List[List[List[str]]]*) → List[List[List[str]]]

Example of minibatch:

```
[[['p', 'l', 'a', 'y', '<PAD>', '<PAD>'],
  ['t', 'h', 'a', 't', '<PAD>', '<PAD>'],
  ['t', 'r', 'a', 'c', 'k', '<PAD>'],
  ['o', 'n', '<PAD>', '<PAD>', '<PAD>', '<PAD>'],
  ['r', 'e', 'p', 'e', 'a', 't']
 ], ...
]
```

### pytext.fields.contextual_token_embedding_field module

**class** pytext.fields.contextual_token_embedding_field.**ContextualTokenEmbeddingField**(***kwargs*)

Bases: *pytext.fields.field.Field*

**numericalize**(*batch, device=None*)

Turn a batch of examples that use this field into a Variable.

If the field has include_lengths=True, a tensor of lengths will be included in the return value.

**Parameters**

- **arr** (*List[List[str]], or tuple of (List[List[str]], List[int])*) – List of tokenized and padded examples, or tuple of List of tokenized and padded examples and List of lengths of each example if self.include_lengths is True.
- **device** (*str or torch.device*) – A string or instance of *torch.device* specifying which device the Variables are going to be created on. If left as default, the tensors will be created on cpu. Default: None.

**pad**(*minibatch: List[List[List[float]]]*) → List[List[List[float]]]

Example of padded minibatch:

```
[[[0.1, 0.2, 0.3, 0.4, 0.5],
  [1.1, 1.2, 1.3, 1.4, 1.5],
  [2.1, 2.2, 2.3, 2.4, 2.5],
  [3.1, 3.2, 3.3, 3.4, 3.5],
 ],
 [[0.1, 0.2, 0.3, 0.4, 0.5],
  [1.1, 1.2, 1.3, 1.4, 1.5],
  [2.1, 2.2, 2.3, 2.4, 2.5],
  [0.0, 0.0, 0.0, 0.0, 0.0],
 ],
 [[0.1, 0.2, 0.3, 0.4, 0.5],
  [1.1, 1.2, 1.3, 1.4, 1.5],
  [0.0, 0.0, 0.0, 0.0, 0.0],
  [0.0, 0.0, 0.0, 0.0, 0.0],
 ],
]
```

## pytext.fields.dict_field module

**class** pytext.fields.dict_field.**DictFeatureField**(*pad_token='<pad>', unk_token='<unk>', batch_first=True, left_pad=False, **kwargs*)

Bases: *pytext.fields.field.VocabUsingField*

**build_vocab**(**args, **kwargs*)

Construct the Vocab object for this field from one or more datasets.

**Parameters**

- **arguments** (*Positional*) – Dataset objects or other iterable data sources from which to construct the Vocab object that represents the set of possible values for this field. If a Dataset object is provided, all columns corresponding to this field are used; individual columns can also be provided directly.
- **keyword arguments** (*Remaining*) – Passed to the constructor of Vocab.

**dummy_model_input = (tensor([[1], [1]]), tensor([[1.5000], [2.5000]]), tensor([[1], [1**

**numericalize**(*arr, device=None*)

Turn a batch of examples that use this field into a Variable.

If the field has include_lengths=True, a tensor of lengths will be included in the return value.

**Parameters**

- **arr** (*List[List[str]], or tuple of (List[List[str]], List[int])*) – List of tokenized and padded examples, or tuple of List of tokenized and padded examples and List of lengths of each example if self.include_lengths is True.
- **device** (*str or torch.device*) – A string or instance of *torch.device* specifying which device the Variables are going to be created on. If left as default, the tensors will be created on cpu. Default: None.

**pad**(*minibatch: List[Tuple[List[int], List[float], List[int]]]*) → Tuple[List[List[int]], List[List[float]], List[int]]

Pad a batch of examples using this field.

Pads to self.fix_length if provided, otherwise pads to the length of the longest example in the batch. Prepends self.init_token and appends self.eos_token if those attributes are not None. Returns a tuple of the padded list and a list containing lengths of each example if *self.include_lengths* is *True* and *self.sequential* is *True*, else just returns the padded list. If *self.sequential* is *False*, no padding is applied.

### pytext.fields.field module

**class** pytext.fields.field.**ActionField**(***kwargs*)

Bases: *pytext.fields.field.VocabUsingField*

**class** pytext.fields.field.**DocLabelField**(***kwargs*)

Bases: *pytext.fields.field.Field*

**class** pytext.fields.field.**Field**(**args*, ***kwargs*)

Bases: torchtext.legacy.data.field.Field

**classmethod from_config**(*config*)

**get_meta**() → pytext.fields.field.FieldMeta

**load_meta**(*metadata: pytext.fields.field.FieldMeta*)

**pad_length**(*n*)

Override to make pad_length to be multiple of 8 to support fp16 training

**class** pytext.fields.field.**FieldMeta**

Bases: object

**class** pytext.fields.field.**FloatField**(***kwargs*)

Bases: *pytext.fields.field.Field*

**class** pytext.fields.field.**FloatVectorField**(*dim=0*, *dim_error_check=False*, ***kwargs*)

Bases: *pytext.fields.field.Field*

**class** pytext.fields.field.**NestedField**(**args*, ***kwargs*)

Bases: *pytext.fields.field.Field*, torchtext.legacy.data.field.NestedField

**get_meta**()

**load_meta**(*metadata: pytext.fields.field.FieldMeta*)

**class** pytext.fields.field.**RawField**(**args*, *is_target=False*, ***kwargs*)

Bases: torchtext.legacy.data.field.RawField

**get_meta**() → pytext.fields.field.FieldMeta

**class** pytext.fields.field.**SeqFeatureField**(*pretrained_embeddings_path=''*, *embed_dim=0*, *embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>*, *vocab_file=''*, *vocab_size=''*, *vocab_from_train_data=True*, *vocab_from_all_data=False*, *vocab_from_pretrained_embeddings=False*, *postprocessing=None*, *use_vocab=True*, *include_lengths=True*, *pad_token='<pad_seq>'*, *init_token=None*, *eos_token=None*, *tokenize=<function no_tokenize>*, *nesting_field=None*, ***kwargs*)

Bases: *pytext.fields.field.VocabUsingNestedField*

**dummy_model_input = tensor([[[1]], [[1]]])**

**class** pytext.fields.field.**TextFeatureField**(*pretrained_embeddings_path=''*, *embed_dim=0*, *embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>*, *vocab_file=''*, *vocab_size=''*, *vocab_from_train_data=True*, *vocab_from_all_data=False*, *vocab_from_pretrained_embeddings=False*, *postprocessing=None*, *use_vocab=True*, *include_lengths=True*, *batch_first=True*, *sequential=True*, *pad_token='<pad>'*, *unk_token='<unk>'*, *init_token=None*, *eos_token=None*, *lower=False*, *tokenize=<function no_tokenize>*, *fix_length=None*, *pad_first=None*, *min_freq=1*, ***kwargs*)

Bases: *pytext.fields.field.VocabUsingField*

**dummy_model_input = tensor([[1], [1]])**

**class** pytext.fields.field.**VocabUsingField**(*pretrained_embeddings_path=''*, *embed_dim=0*, *embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>*, *vocab_file=''*, *vocab_size=''*, *vocab_from_train_data=True*, *vocab_from_all_data=False*, *vocab_from_pretrained_embeddings=False*, *min_freq=1*, **args*, ***kwargs*)

Bases: *pytext.fields.field.Field*

Base class for all fields that need to build a vocabulary.

**class** pytext.fields.field.**VocabUsingNestedField**(*pretrained_embeddings_path=''*, *embed_dim=0*, *embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>*, *vocab_file=''*, *vocab_size=''*, *vocab_from_train_data=True*, *vocab_from_all_data=False*, *vocab_from_pretrained_embeddings=False*, *min_freq=1*, **args*, ***kwargs*)

Bases: *pytext.fields.field.VocabUsingField*, *pytext.fields.field.NestedField*

Base class for all nested fields that need to build a vocabulary.

**class** pytext.fields.field.**WordLabelField**(*use_bio_labels*, ***kwargs*)

Bases: *pytext.fields.field.Field*

**get_meta**()

pytext.fields.field.**create_fields**(*fields_config*, *field_cls_dict*)

pytext.fields.field.**create_label_fields**(*label_configs*, *label_cls_dict*)

### pytext.fields.text_field_with_special_unk module

**class** pytext.fields.text_field_with_special_unk.**TextFeatureFieldWithSpecialUnk**(**args*, *unkify_func=<func unkify>*, ***kwargs*)

Bases: *pytext.fields.field.TextFeatureField*

**build_vocab**(**args*, *min_freq=1*, ***kwargs*)

Code is exactly same as as torchtext.legacy.data.Field.build_vocab() before the UNKification logic. The reason super().build_vocab() cannot be called is because the Counter object computed in torchtext.legacy.data.Field.build_vocab() is required for UNKification and, that object cannot be recovered after super().build_vocab() call is made.

**numericalize**(*arr: Union[List[List[str]], Tuple[List[List[str]], List[int]]], device: Union[str, torch.device, None] = None*)

Code is exactly same as torchtext.legacy.data.Field.numericalize() except the call to self._get_idx(x) instead of self.vocab.stoi[x] for getting the index of an item from vocab. This is needed because torchtext doesn't allow custom UNKification. So, TextFeatureFieldWithSpecialUnk field's constructor accepts a function unkify_func() that can be used to UNKifying instead of assigning all UNKs a default value.

### Module contents

pytext.fields.**create_fields**(*fields_config*, *field_cls_dict*)

pytext.fields.**create_label_fields**(*label_configs*, *label_cls_dict*)

**class** pytext.fields.**ActionField**(***kwargs*)

Bases: *pytext.fields.field.VocabUsingField*

**class** pytext.fields.**CharFeatureField**(*pad_token='<pad>'*, *unk_token='<unk>'*, *batch_first=True*, *max_word_length=20*, *min_freq=1*, ***kwargs*)

Bases: *pytext.fields.field.VocabUsingField*

**build_vocab**(**args*, ***kwargs*)

Construct the Vocab object for this field from one or more datasets.

**Parameters**

- **arguments** (*Positional*) – Dataset objects or other iterable data sources from which to construct the Vocab object that represents the set of possible values for this field. If a Dataset object is provided, all columns corresponding to this field are used; individual columns can also be provided directly.
- **keyword arguments** (*Remaining*) – Passed to the constructor of Vocab.

**dummy_model_input = tensor([[[1, 1, 1]], [[1, 1, 1]]])**

**numericalize**(*batch*, *device=None*)

Turn a batch of examples that use this field into a Variable.

If the field has include_lengths=True, a tensor of lengths will be included in the return value.

**Parameters**

- **arr** (*List[List[str]], or tuple of (List[List[str]], List[int])*) – List of tokenized and padded examples, or tuple of List of tokenized and padded examples and List of lengths of each example if self.include_lengths is True.
- **device** (*str or torch.device*) – A string or instance of *torch.device* specifying which device the Variables are going to be created on. If left as default, the tensors will be created on cpu. Default: None.

**pad**(*minibatch: List[List[List[str]]]*) → List[List[List[str]]]

Example of minibatch:

```
[[['p', 'l', 'a', 'y', '<PAD>', '<PAD>'],
  ['t', 'h', 'a', 't', '<PAD>', '<PAD>'],
  ['t', 'r', 'a', 'c', 'k', '<PAD>'],
  ['o', 'n', '<PAD>', '<PAD>', '<PAD>', '<PAD>'],
  ['r', 'e', 'p', 'e', 'a', 't']
 ], ...
]
```

**class** pytext.fields.**ContextualTokenEmbeddingField**(***kwargs*)

Bases: *pytext.fields.field.Field*

**numericalize**(*batch*, *device=None*)

Turn a batch of examples that use this field into a Variable.

If the field has include_lengths=True, a tensor of lengths will be included in the return value.

**Parameters**

- **arr** (*List[List[str]], or tuple of (List[List[str]], List[int])*) – List of tokenized and padded examples, or tuple of List of tokenized and padded examples and List of lengths of each example if self.include_lengths is True.
- **device** (*str or torch.device*) – A string or instance of *torch.device* specifying which device the Variables are going to be created on. If left as default, the tensors will be created on cpu. Default: None.

**pad**(*minibatch: List[List[List[float]]]*) → List[List[List[float]]]

Example of padded minibatch:

```
[[[0.1, 0.2, 0.3, 0.4, 0.5],
  [1.1, 1.2, 1.3, 1.4, 1.5],
  [2.1, 2.2, 2.3, 2.4, 2.5],
  [3.1, 3.2, 3.3, 3.4, 3.5],
 ],
 [[0.1, 0.2, 0.3, 0.4, 0.5],
  [1.1, 1.2, 1.3, 1.4, 1.5],
  [2.1, 2.2, 2.3, 2.4, 2.5],
  [0.0, 0.0, 0.0, 0.0, 0.0],
 ],
```

(continues on next page)

(continued from previous page)

```
 [[0.1, 0.2, 0.3, 0.4, 0.5],
  [1.1, 1.2, 1.3, 1.4, 1.5],
  [0.0, 0.0, 0.0, 0.0, 0.0],
  [0.0, 0.0, 0.0, 0.0, 0.0],
 ],
]
```

**class** pytext.fields.**DictFeatureField**(*pad_token='<pad>', unk_token='<unk>', batch_first=True, left_pad=False, **kwargs*)

Bases: *pytext.fields.field.VocabUsingField*

**build_vocab**(**args, **kwargs*)

Construct the Vocab object for this field from one or more datasets.

**Parameters**

- **arguments** (*Positional*) – Dataset objects or other iterable data sources from which to construct the Vocab object that represents the set of possible values for this field. If a Dataset object is provided, all columns corresponding to this field are used; individual columns can also be provided directly.
- **keyword arguments** (*Remaining*) – Passed to the constructor of Vocab.

**dummy_model_input = (tensor([[1], [1]]), tensor([[1.5000], [2.5000]]), tensor([[1], [1**

**numericalize**(*arr, device=None*)

Turn a batch of examples that use this field into a Variable.

If the field has include_lengths=True, a tensor of lengths will be included in the return value.

**Parameters**

- **arr** (*List[List[str]], or tuple of (List[List[str]], List[int])*) – List of tokenized and padded examples, or tuple of List of tokenized and padded examples and List of lengths of each example if self.include_lengths is True.
- **device** (*str or torch.device*) – A string or instance of *torch.device* specifying which device the Variables are going to be created on. If left as default, the tensors will be created on cpu. Default: None.

**pad**(*minibatch: List[Tuple[List[int], List[float], List[int]]]*) → Tuple[List[List[int]], List[List[float]], List[int]]

Pad a batch of examples using this field.

Pads to self.fix_length if provided, otherwise pads to the length of the longest example in the batch. Prepends self.init_token and appends self.eos_token if those attributes are not None. Returns a tuple of the padded list and a list containing lengths of each example if *self.include_lengths* is *True* and *self.sequential* is *True*, else just returns the padded list. If *self.sequential* is *False*, no padding is applied.

**class** pytext.fields.**DocLabelField**(***kwargs*)

Bases: *pytext.fields.field.Field*

**class** pytext.fields.**Field**(**args, **kwargs*)

Bases: torchtext.legacy.data.field.Field

**classmethod from_config**(*config*)

**get_meta**() → pytext.fields.field.FieldMeta

**load_meta**(*metadata: pytext.fields.field.FieldMeta*)

**pad_length**(*n*)
Override to make pad_length to be multiple of 8 to support fp16 training

**class** pytext.fields.**FieldMeta**
Bases: object

**class** pytext.fields.**FloatField**(***kwargs*)
Bases: *pytext.fields.field.Field*

**class** pytext.fields.**FloatVectorField**(*dim=0, dim_error_check=False, **kwargs*)
Bases: *pytext.fields.field.Field*

**class** pytext.fields.**RawField**(**args, is_target=False, **kwargs*)
Bases: torchtext.legacy.data.field.RawField

**get_meta**() → pytext.fields.field.FieldMeta

**class** pytext.fields.**TextFeatureField**(*pretrained_embeddings_path=", embed_dim=0, embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>, vocab_file=", vocab_size=", vocab_from_train_data=True, vocab_from_all_data=False, vocab_from_pretrained_embeddings=False, postprocessing=None, use_vocab=True, include_lengths=True, batch_first=True, sequential=True, pad_token='<pad>', unk_token='<unk>', init_token=None, eos_token=None, lower=False, tokenize=<function no_tokenize>, fix_length=None, pad_first=None, min_freq=1, **kwargs*)
Bases: *pytext.fields.field.VocabUsingField*

**dummy_model_input = tensor([[1], [1]])**

**class** pytext.fields.**VocabUsingField**(*pretrained_embeddings_path=", embed_dim=0, embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>, vocab_file=", vocab_size=", vocab_from_train_data=True, vocab_from_all_data=False, vocab_from_pretrained_embeddings=False, min_freq=1, *args, **kwargs*)
Bases: *pytext.fields.field.Field*

Base class for all fields that need to build a vocabulary.

**class** pytext.fields.**WordLabelField**(*use_bio_labels, **kwargs*)
Bases: *pytext.fields.field.Field*

**get_meta**()

**class** pytext.fields.**NestedField**(**args, **kwargs*)
Bases: *pytext.fields.field.Field*, torchtext.legacy.data.field.NestedField

**get_meta**()

**load_meta**(*metadata: pytext.fields.field.FieldMeta*)

**class** pytext.fields.**VocabUsingNestedField**(*pretrained_embeddings_path=''*, *embed_dim=0*, *embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>*, *vocab_file=''*, *vocab_size=''*, *vocab_from_train_data=True*, *vocab_from_all_data=False*, *vocab_from_pretrained_embeddings=False*, *min_freq=1*, **args*, ***kwargs*)

Bases: *pytext.fields.field.VocabUsingField*, *pytext.fields.field.NestedField*

Base class for all nested fields that need to build a vocabulary.

**class** pytext.fields.**SeqFeatureField**(*pretrained_embeddings_path=''*, *embed_dim=0*, *embedding_init_strategy=<EmbedInitStrategy.RANDOM: 'random'>*, *vocab_file=''*, *vocab_size=''*, *vocab_from_train_data=True*, *vocab_from_all_data=False*, *vocab_from_pretrained_embeddings=False*, *postprocessing=None*, *use_vocab=True*, *include_lengths=True*, *pad_token='<pad_seq>'*, *init_token=None*, *eos_token=None*, *tokenize=<function no_tokenize>*, *nesting_field=None*, ***kwargs*)

Bases: *pytext.fields.field.VocabUsingNestedField*

**dummy_model_input = tensor([[[1]], [[1]]])**

**class** pytext.fields.**TextFeatureFieldWithSpecialUnk**(**args*, *unkify_func=<function unkify>*, ***kwargs*)

Bases: *pytext.fields.field.TextFeatureField*

**build_vocab**(**args*, *min_freq=1*, ***kwargs*)

Code is exactly same as as torchtext.legacy.data.Field.build_vocab() before the UNKification logic. The reason super().build_vocab() cannot be called is because the Counter object computed in torchtext.legacy.data.Field.build_vocab() is required for UNKification and, that object cannot be recovered after super().build_vocab() call is made.

**numericalize**(*arr: Union[List[List[str]], Tuple[List[List[str]], List[int]]], device: Union[str, torch.device, None] = None*)

Code is exactly same as torchtext.legacy.data.Field.numericalize() except the call to self._get_idx(x) instead of self.vocab.stoi[x] for getting the index of an item from vocab. This is needed because torchtext doesn't allow custom UNKification. So, TextFeatureFieldWithSpecialUnk field's constructor accepts a function unkify_func() that can be used to UNKifying instead of assigning all UNKs a default value.

### pytext.loss package

### Submodules

### pytext.loss.loss module

**class** pytext.loss.loss.**AUCPRHingeLoss**(*config*, *weights=None*, **args*, ***kwargs*)

Bases: torch.nn.modules.module.Module, *pytext.loss.loss.Loss*

area under the precision-recall curve loss, Reference: "Scalable Learning of Non-Decomposable Objectives", Section 5 TensorFlow Implementation: https://github.com/tensorflow/models/tree/master/research/global_objectives

**forward**(*logits*, *targets*, *reduce=True*, *size_average=True*, *weights=None*)

**Parameters**

- **logits** – Variable $(N, C)$ where *C = number of classes*
- **targets** – Variable $(N)$ where each value is *0 <= targets[i] <= C-1*
- **weights** – Coefficients for the loss. Must be a *Tensor* of shape [N] or [N, C], where *N = batch_size*, *C = number of classes*.
- **size_average** (*bool, optional*) – By default, the losses are averaged over observations for each minibatch. However, if the field sizeAverage is set to False, the losses are instead summed for each minibatch. Default: `True`
- **reduce** (*bool, optional*) – By default, the losses are averaged or summed over observations for each minibatch depending on size_average. When reduce is False, returns a loss per input/target element instead and ignores size_average. Default: True

**class** pytext.loss.loss.**BinaryCrossEntropyLoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**BinaryCrossEntropyWithLogitsLoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**CosineEmbeddingLoss**(*config*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**CrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**HingeLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**KLDivergenceBCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**KLDivergenceCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**LabelSmoothedCrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**Loss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.config.component.Component*

Base class for loss functions

**class** pytext.loss.loss.**MAELoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Mean absolute error or L1 loss, for regression tasks.

**class** pytext.loss.loss.**MSELoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Mean squared error or L2 loss, for regression tasks.

**class** pytext.loss.loss.**MultiLabelSoftMarginLoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**NLLLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)
Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.loss.**PairwiseRankingLoss**(*config=None*, **args*, ***kwargs*)
Bases: *pytext.loss.loss.Loss*

Given embeddings for a query, positive response and negative response computes pairwise ranking hinge loss

**static get_similarities**(*embeddings*)

**class** pytext.loss.loss.**SourceType**
Bases: enum.Enum

An enumeration.

**LOGITS = 'logits'**

**LOG_PROBS = 'log_probs'**

**PROBS = 'probs'**

pytext.loss.loss.**maybe_log_normalize**(*logits*, *logits_type*, *dim=-1*)
Optionally log normalizes logits on the given dimension.

### pytext.loss.regularized_loss module

**class** pytext.loss.regularized_loss.**LabelSmoothingLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.loss.Loss*

Label loss with an optional regularizer for smoothing.

**class** pytext.loss.regularized_loss.**NARSamplewiseSequenceLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.regularized_loss.NARSequenceLoss*

Non-autoregressive sequence loss with sample-wise logging.

**class** pytext.loss.regularized_loss.**NARSequenceLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.loss.Loss*

Joint loss over labels and length of sequences for non-autoregressive modeling.

**class** pytext.loss.regularized_loss.**SamplewiseLabelSmoothingLoss**(*config*, *ignore_index=-1*)
Bases: *pytext.loss.regularized_loss.LabelSmoothingLoss*

Label smoothing loss with sample-wise logging.

### pytext.loss.regularizer module

**class** pytext.loss.regularizer.**AdaptiveRegularizer**(*config*, *ignore_index=1*)
Bases: *pytext.loss.regularizer.Regularizer*

Adaptive variant of *UniformRegularizer* which learns the mix-in noise distribution.

Learning Better Structured Representations using Low-Rank Adaptive Label Smoothing (Ghoshal+ 2021; https://openreview.net/pdf?id=5NsEIflpbSv)

**compute_adaptive_loss**(*logits*, *targets*, *label_embedding*)
Using Equation 3 and 4, computes several terms of the adaptive penalty. Specifically, we implement adaptive smoothing (*smooth_term*) and an entropy constraint (*eta_term*).

**class** pytext.loss.regularizer.**EntropyRegularizer**(*config*, *ignore_index=1*)
Bases: *pytext.loss.regularizer.Regularizer*

**Entropy of the predicted distribution. Defined as:** H[P(Y|X)] = - sum_i P(Y_i|X) * log P(Y_i|X)

**class** pytext.loss.regularizer.**Regularizer**(*config*, *ignore_index=1*)
Bases: *pytext.loss.loss.Loss*

Generic regularization function to be added to a surrogate loss (e.g., cross-entropy).

**class** pytext.loss.regularizer.**UniformRegularizer**(*config*, *ignore_index=1*)
Bases: *pytext.loss.regularizer.Regularizer*

**Negative KL between the uniform and predicted distribution.**

**Defined as:**

- **KL(U || P(Y|X)) = - sum_i U_i * log (P(Y_i | X) / U_i)** = - sum_i U_i * log P(Y_i|X) + H[U] = - (1/n) * sum_i log P(Y_i | X) + H[U]

H[U] does not depend on X, thus it is omitted during optimization.

### pytext.loss.structured_loss module

**class** pytext.loss.structured_loss.**CostFunctionType**
Bases: enum.Enum

An enumeration.

**HAMMING = 'hamming'**

**class** pytext.loss.structured_loss.**StructuredLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.loss.Loss*

Generic loss function applied to structured outputs.

**class** pytext.loss.structured_loss.**StructuredMarginLoss**(*config*, *ignore_index=1*, **args*, ***kwargs*)
Bases: *pytext.loss.structured_loss.StructuredLoss*

Margin-based loss which requires a gold structure Y to score at least *cost(Y, Y')* above a hypothesis structure *Y'*. The cost function used is variable, but should reflect the underlying semantics of the task (e.g., BLEU in machine translation).

pytext.loss.structured_loss.**get_cost_fn**(*cost_fn_type: pytext.loss.structured_loss.CostFunctionType*)
Retrieves a cost function corresponding to *cost_fn_type*.

pytext.loss.structured_loss.**hamming_distance**(*logits*, *targets*, *cost_scale=1.0*)
Computes Hamming distance (https://en.wikipedia.org/wiki/Hamming_distance), which is defined as the number of positions where two sequences of equal length differ. We apply Hamming distance locally, incrementing non-gold token scores by *cost_scale*.

` Example: Given targets = [0, 1] and cost_scale = 1.0, we have the following: logits (before) = [[-1.0, 1.0, 2.0], [-2.0, -1.0, 1.0]] logits (after) = [[-1.0, 2.0, 3.0], [-1.0, -1.0, 2.0]] `

### Module contents

**class** pytext.loss.**AUCPRHingeLoss**(*config*, *weights=None*, **args*, ***kwargs*)
Bases: torch.nn.modules.module.Module, *pytext.loss.loss.Loss*

area under the precision-recall curve loss, Reference: "Scalable Learning of Non-Decomposable Objectives", Section 5 TensorFlow Implementation: https://github.com/tensorflow/models/tree/master/research/global_objectives

**forward**(*logits*, *targets*, *reduce=True*, *size_average=True*, *weights=None*)

**Parameters**

- **logits** – Variable $(N, C)$ where *C = number of classes*
- **targets** – Variable $(N)$ where each value is *0 <= targets[i] <= C-1*
- **weights** – Coefficients for the loss. Must be a *Tensor* of shape [N] or [N, C], where *N = batch_size*, *C = number of classes*.
- **size_average** (*bool, optional*) – By default, the losses are averaged over observations for each minibatch. However, if the field sizeAverage is set to False, the losses are instead summed for each minibatch. Default: `True`
- **reduce** (*bool, optional*) – By default, the losses are averaged or summed over observations for each minibatch depending on size_average. When reduce is False, returns a loss per input/target element instead and ignores size_average. Default: True

**class** pytext.loss.**Loss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.config.component.Component*

Base class for loss functions

**class** pytext.loss.**CrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**CosineEmbeddingLoss**(*config*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**BinaryCrossEntropyLoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**BinaryCrossEntropyWithLogitsLoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**HingeLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**MultiLabelSoftMarginLoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**KLDivergenceBCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**KLDivergenceCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**MAELoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Mean absolute error or L1 loss, for regression tasks.

**class** pytext.loss.**MSELoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Mean squared error or L2 loss, for regression tasks.

**class** pytext.loss.**NLLLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)
Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**PairwiseRankingLoss**(*config=None*, **args*, ***kwargs*)
Bases: *pytext.loss.loss.Loss*

Given embeddings for a query, positive response and negative response computes pairwise ranking hinge loss

**static get_similarities**(*embeddings*)

**class** pytext.loss.**LabelSmoothedCrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)
Bases: *pytext.loss.loss.Loss*

**class** pytext.loss.**SourceType**
Bases: enum.Enum

An enumeration.

**LOGITS = 'logits'**

**LOG_PROBS = 'log_probs'**

**PROBS = 'probs'**

**class** pytext.loss.**CostFunctionType**
Bases: enum.Enum

An enumeration.

**HAMMING = 'hamming'**

**class** pytext.loss.**StructuredLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.loss.Loss*

Generic loss function applied to structured outputs.

**class** pytext.loss.**StructuredMarginLoss**(*config*, *ignore_index=1*, **args*, ***kwargs*)
Bases: *pytext.loss.structured_loss.StructuredLoss*

Margin-based loss which requires a gold structure Y to score at least *cost(Y, Y')* above a hypothesis structure *Y'*. The cost function used is variable, but should reflect the underlying semantics of the task (e.g., BLEU in machine translation).

**class** pytext.loss.**LabelSmoothingLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.loss.Loss*

Label loss with an optional regularizer for smoothing.

**class** pytext.loss.**SamplewiseLabelSmoothingLoss**(*config*, *ignore_index=-1*)
Bases: *pytext.loss.regularized_loss.LabelSmoothingLoss*

Label smoothing loss with sample-wise logging.

**class** pytext.loss.**NARSequenceLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.loss.Loss*

Joint loss over labels and length of sequences for non-autoregressive modeling.

**class** pytext.loss.**NARSamplewiseSequenceLoss**(*config*, *ignore_index=1*)
Bases: *pytext.loss.regularized_loss.NARSequenceLoss*

Non-autoregressive sequence loss with sample-wise logging.

**class** pytext.loss.**UniformRegularizer**(*config*, *ignore_index=1*)
Bases: *pytext.loss.regularizer.Regularizer*

**Negative KL between the uniform and predicted distribution.**

**Defined as:**

- **KL(U || P(Y|X)) = - sum_i U_i * log (P(Y_i | X) / U_i)** = - sum_i U_i * log P(Y_i|X) + H[U] = - (1/n) * sum_i log P(Y_i | X) + H[U]

H[U] does not depend on X, thus it is omitted during optimization.

**class** pytext.loss.**EntropyRegularizer**(*config*, *ignore_index=1*)

Bases: *pytext.loss.regularizer.Regularizer*

**Entropy of the predicted distribution. Defined as:** H[P(Y|X)] = - sum_i P(Y_i|X) * log P(Y_i|X)

**class** pytext.loss.**AdaptiveRegularizer**(*config*, *ignore_index=1*)

Bases: *pytext.loss.regularizer.Regularizer*

Adaptive variant of *UniformRegularizer* which learns the mix-in noise distribution.

Learning Better Structured Representations using Low-Rank Adaptive Label Smoothing (Ghoshal+ 2021; https://openreview.net/pdf?id=5NsEIflpbSv)

**compute_adaptive_loss**(*logits*, *targets*, *label_embedding*)

Using Equation 3 and 4, computes several terms of the adaptive penalty. Specifically, we implement adaptive smoothing (*smooth_term*) and an entropy constraint (*eta_term*).

## pytext.metric_reporters package

### Submodules

### pytext.metric_reporters.calibration_metric_reporter module

**class** pytext.metric_reporters.calibration_metric_reporter.**CalibrationMetricReporter**(*channels: List[pytex pad_inde int = - 1*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**aggregate_preds**(*batch_preds: torch.Tensor, batch_context=typing.Dict[str, typing.Any]*)

**aggregate_scores**(*batch_scores: torch.Tensor*)

**aggregate_targets**(*batch_targets: torch.Tensor, batch_context=typing.Dict[str, typing.Any]*)

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config: pytext.config.pytext_config.PyTextConfig, pad_index: int = -1*)

### pytext.metric_reporters.channel module

**class** pytext.metric_reporters.channel.**Channel**(*stages: Tuple[pytext.common.constants.Stage, ...] = (<Stage.TRAIN: 'Training'>, <Stage.EVAL: 'Evaluation'>, <Stage.TEST: 'Test'>, <Stage.OTHERS: 'Others'>)*)

Bases: `object`

Channel defines how to format and report the result of a PyText job to an output stream.

**stages**
in which stages the report will be triggered, default is all stages, which includes train, eval, test

**close**()

**export**(*model*, *input_to_model=None*, ***kwargs*)

**report**(*stage*, *epoch*, *metrics*, *model_select_metric*, *loss*, *preds*, *targets*, *scores*, *context*, **args*)
Defines how to format and report data to the output channel.

**Parameters**

- **stage** (*Stage*) – train, eval or test
- **epoch** (*int*) – current epoch
- **metrics** (*Any*) – all metrics
- **model_select_metric** (*double*) – a single numeric metric to pick best model
- **loss** (*double*) – average loss
- **preds** (*List[Any]*) – list of predictions
- **targets** (*List[Any]*) – list of targets
- **scores** (*List[Any]*) – list of scores
- **context** (*Dict[str, List[Any]]*) – dict of any additional context data, each context is a list of data that maps to each example

**class** pytext.metric_reporters.channel.**ConsoleChannel**(*stages: Tuple[pytext.common.constants.Stage, ...] = (<Stage.TRAIN: 'Training'>, <Stage.EVAL: 'Evaluation'>, <Stage.TEST: 'Test'>, <Stage.OTHERS: 'Others'>)*)

Bases: *pytext.metric_reporters.channel.Channel*

Simple Channel that prints results to console.

**report**(*stage*, *epoch*, *metrics*, *model_select_metric*, *loss*, *preds*, *targets*, *scores*, *context*, **args*)
Defines how to format and report data to the output channel.

**Parameters**

- **stage** (*Stage*) – train, eval or test
- **epoch** (*int*) – current epoch
- **metrics** (*Any*) – all metrics
- **model_select_metric** (*double*) – a single numeric metric to pick best model

- **loss** (*double*) – average loss
- **preds** (*List[Any]*) – list of predictions
- **targets** (*List[Any]*) – list of targets
- **scores** (*List[Any]*) – list of scores
- **context** (*Dict[str, List[Any]]*) – dict of any additional context data, each context is a list of data that maps to each example

**class** pytext.metric_reporters.channel.**FileChannel**(*stages*, *file_path*)

Bases: *pytext.metric_reporters.channel.Channel*

Simple Channel that writes results to a TSV file.

**gen_content**(*metrics*, *loss*, *preds*, *targets*, *scores*, *context*)

**get_title**(*context_keys=()*)

**report**(*stage*, *epoch*, *metrics*, *model_select_metric*, *loss*, *preds*, *targets*, *scores*, *context*, **args*)

Defines how to format and report data to the output channel.

**Parameters**

- **stage** (*Stage*) – train, eval or test
- **epoch** (*int*) – current epoch
- **metrics** (*Any*) – all metrics
- **model_select_metric** (*double*) – a single numeric metric to pick best model
- **loss** (*double*) – average loss
- **preds** (*List[Any]*) – list of predictions
- **targets** (*List[Any]*) – list of targets
- **scores** (*List[Any]*) – list of scores
- **context** (*Dict[str, List[Any]]*) – dict of any additional context data, each context is a list of data that maps to each example

**class** pytext.metric_reporters.channel.**TensorBoardChannel**(*summary_writer=None*, *metric_name='accuracy'*)

Bases: *pytext.metric_reporters.channel.Channel*

TensorBoardChannel defines how to format and report the result of a PyText job to TensorBoard.

**summary_writer**

An instance of the TensorBoard SummaryWriter class, or an object that implements the same interface. https://pytorch.org/docs/stable/tensorboard.html

**metric_name**

The name of the default metric to display on the TensorBoard dashboard, defaults to "accuracy"

**train_step**

The training step count

**add_scalars**(*prefix*, *metrics*, *epoch*)

Recursively flattens the metrics object and adds each field name and value as a scalar for the corresponding epoch using the summary writer.

**Parameters**

- **prefix** (*str*) – The tag prefix for the metric. Each field name in the metrics object will be prepended with the prefix.
- **metrics** (*Any*) – The metrics object.

**add_texts** (*tag*, *metrics*)

Recursively flattens the metrics object and adds each field name and value as a text using the summary writer. For example, if tag = "test", and metrics = { accuracy: 0.7, scores: { precision: 0.8, recall: 0.6 } }, then under "tag=test" we will display "accuracy=0.7", and under "tag=test/scores" we will display "precision=0.8" and "recall=0.6" in TensorBoard.

**Parameters**

- **tag** (*str*) – The tag name for the metric. If a field needs to be flattened further, it will be prepended as a prefix to the field name.
- **metrics** (*Any*) – The metrics object/dict.

**close** ()

Closes the summary writer.

**export** (*model*, *input_to_model=None*, ***kwargs*)

Draws the neural network representation graph in TensorBoard.

**Parameters**

- **model** (*Any*) – the model object.
- **input_to_model** (*Any*) – the input to the model (required for PyTorch models, since its execution graph is defined by run).

**log_loss** (*prefix*, *loss*, *epoch*)

**log_vector** (*key*, *val*, *epoch*)

**report** (*stage*, *epoch*, *metrics*, *model_select_metric*, *loss*, *preds*, *targets*, *scores*, *context*, *meta*, *model*, *optimizer*, *log_gradient*, *gradients*, **args*)

Defines how to format and report data to TensorBoard using the summary writer. In the current implementation, during the train/eval phase we recursively report each metric field as scalars, and during the test phase we report the final metrics to be displayed as texts.

Also visualizes the internal model states (weights, biases) as histograms in TensorBoard.

**Parameters**

- **stage** (*Stage*) – train, eval or test
- **epoch** (*int*) – current epoch
- **metrics** (*Any*) – all metrics
- **model_select_metric** (*double*) – a single numeric metric to pick best model
- **loss** (*double*) – average loss
- **preds** (*List[Any]*) – list of predictions
- **targets** (*List[Any]*) – list of targets
- **scores** (*List[Any]*) – list of scores
- **context** (*Dict[str, List[Any]]*) – dict of any additional context data, each context is a list of data that maps to each example
- **meta** (*Dict[str, Any]*) – global metadata, such as target names
- **model** (*nn.Module*) – the PyTorch neural network model

### pytext.metric_reporters.classification_metric_reporter module

**class** pytext.metric_reporters.classification_metric_reporter.**ClassificationMetricReporter**(

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *meta: pytext.data.data_handler.CommonMetadata = None*, *tensorizers=None*)

**classmethod from_config_and_label_names**(*config, label_names: List[str]*)

**get_meta**()
Get global meta data that is not specific to any batch, the data will be pass along to channels

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**predictions_to_report**()
Generate human readable predictions

**targets_to_report**()
Generate human readable targets

**class** pytext.metric_reporters.classification_metric_reporter.**ComparableClassificationMetric**
Bases: enum.Enum

An enumeration.

**ACCURACY = 'accuracy'**

**LABEL_AVG_PRECISION = 'label_avg_precision'**

**LABEL_F1 = 'label_f1'**

**LABEL_ROC_AUC = 'label_roc_auc'**

**MACRO_F1 = 'macro_f1'**

**MCC = 'mcc'**

**NEGATIVE_LOSS = 'negative_loss'**

**ROC_AUC = 'roc_auc'**

**class** pytext.metric_reporters.classification_metric_reporter.**MultiLabelClassificationMetric**

Bases: *pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter*

**calculate_metric**()

Calculate metrics, each sub class should implement it

**predictions_to_report**()
Generate human readable predictions

**targets_to_report**()
Generate human readable targets

### pytext.metric_reporters.compositional_metric_reporter module

**class** pytext.metric_reporters.compositional_metric_reporter.**CompositionalMetricReporter**(*acti cha nels List text str = 'to- k- eniz to- k- eniz py- text = Non*

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**create_frame_prediction_pairs**()

**classmethod from_config**(*config*, *metadata: pytext.data.data_handler.CommonMetadata = None*, *tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer] = None*)

**gen_extra_context**(**args*)
Generate any extra intermediate context data for metric calculation

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**static node_to_metrics_node**(*node: Union[pytext.data.data_structures.annotation.Intent, pytext.data.data_structures.annotation.Slot], start: int = 0*) → pytext.metrics.intent_slot_metrics.Node
The input start is the absolute start position in utterance

**predictions_to_report**()
Generate human readable predictions

**targets_to_report**()
Generate human readable targets

**static tree_from_tokens_and_indx_actions**(*token_str_list: List[str], actions_vocab: List[str], actions_indices: List[int], validate_tree: bool = True*)

**static tree_to_metric_node**(*tree: pytext.data.data_structures.annotation.Tree*) → pytext.metrics.intent_slot_metrics.Node

Creates a Node from tree assuming the utterance is a concatenation of the tokens by whitespaces. The function does not necessarily reproduce the original utterance as extra whitespaces can be introduced.

### pytext.metric_reporters.compositional_utils module

pytext.metric_reporters.compositional_utils.**extract_beam_subtrees**(*beam: List[List[str]]*) → List[List[str]]

pytext.metric_reporters.compositional_utils.**extract_subtree**(*beam: List[str]*) → Optional[List[str]]

pytext.metric_reporters.compositional_utils.**filter_invalid_beams**(*beam: List[List[str]]*) → List[List[str]]

pytext.metric_reporters.compositional_utils.**is_valid_tree**(*beam: List[str]*) → bool

### pytext.metric_reporters.dense_retrieval_metric_reporter module

**class** pytext.metric_reporters.dense_retrieval_metric_reporter.**DenseRetrievalMetricNames**

Bases: enum.Enum

An enumeration.

**ACCURACY = 'accuracy'**

**AVG_RANK = 'avg_rank'**

**MEAN_RECIPROCAL_RANK = 'mean_reciprocal_rank'**

**NEGATIVE_LOSS = 'negative_loss'**

**class** pytext.metric_reporters.dense_retrieval_metric_reporter.**DenseRetrievalMetricReporter**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**aggregate_preds**(*preds, context*)

**batch_context**(*raw_batch, batch*) → Dict[str, Any]

**calculate_metric**() → pytext.metrics.dense_retrieval_metrics.DenseRetrievalMetrics
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, **args*, *tensorizers=None*, ***kwargs*)

**get_model_select_metric**(*metrics: pytext.metrics.dense_retrieval_metrics.DenseRetrievalMetrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

### pytext.metric_reporters.disjoint_multitask_metric_reporter module

**class** pytext.metric_reporters.disjoint_multitask_metric_reporter.**DisjointMultitaskMetricRep**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**add_channel**(*channel*)

**batch_context**(*raw_batch*, *batch*)

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**lower_is_better = False**

**report_metric**(*model*, *stage*, *epoch*, *reset=True*, *print_to_channels=True*, *optimizer=None*)
Calculate metrics and average loss, report all statistic data to channels

**Parameters**

- **model** (*nn.Module*) – the PyTorch neural network model.

- **stage** (*Stage*) – training, evaluation or test
- **epoch** (*int*) – current epoch
- **reset** (*bool*) – if all data should be reset after report, default is True
- **print_to_channels** (*bool*) – if report data to channels, default is True

**report_realtime_metric**(*stage*)

### pytext.metric_reporters.intent_slot_detection_metric_reporter module

**class** pytext.metric_reporters.intent_slot_detection_metric_reporter.**IntentSlotMetricReporte**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**aggregate_preds**(*batch_preds*, *batch_context*)

**aggregate_scores**(*batch_scores*)

**aggregate_targets**(*batch_targets*, *batch_context*)

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *tensorizers: Optional[Dict[KT, VT]] = None*)

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**get_raw_slot_str**(*raw_data_row*)

**predictions_to_report**()
Generate human readable predictions

**targets_to_report**()
Generate human readable targets

pytext.metric_reporters.intent_slot_detection_metric_reporter.**create_frame**(*text*, *intent_label*, *slot_names_str*, *byte_len*)

pytext.metric_reporters.intent_slot_detection_metric_reporter.**frame_to_str**(*frame: pytext.metrics.intent_slot*

### pytext.metric_reporters.language_model_metric_reporter module

**class** pytext.metric_reporters.language_model_metric_reporter.**LanguageModelChannel**(*stages*, *file_path*)

Bases: *pytext.metric_reporters.channel.FileChannel*

**gen_content**(*metrics*, *loss*, *preds*, *targets*, *scores*, *contexts*)

**get_title**(*context_keys=()*)

**class** pytext.metric_reporters.language_model_metric_reporter.**LanguageModelMetricReporter**(*ch m da te so er ag gr ga pe pl ity pe lo*

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**LABELS_COLUMN = 'labels'**

**RAW_TEXT_COLUMN = 'text'**

**TOKENS_COLUMN = 'tokens'**

**UTTERANCE_COLUMN = 'utterance'**

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch

- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**aggregate_context**(*context*)

**aggregate_scores**(*scores*)

**batch_context**(*raw_batch*, *batch*)

**calculate_loss**() → float
Calculate the average loss for all aggregated batch

**calculate_metric**() → pytext.metrics.language_model_metrics.LanguageModelMetric
Calculate metrics, each sub class should implement it

**compute_scores**(*logits*, *targets*)

**classmethod from_config**(*config: pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricRepo meta: pytext.data.data_handler.CommonMetadata = None*, *tensorizers=None*)

**get_model_select_metric**(*metrics*) → float
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**lower_is_better = True**

**class** pytext.metric_reporters.language_model_metric_reporter.**MaskedLMMetricReporter**(*channels*, *metadata*, *tensorizers*, *aggregate*, *aggregate_met*, *perplexity_type*, *pep_form*, *log_grad*

Bases: *pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter*

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch

- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**calculate_loss**() → float
Calculate the average loss for all aggregated batch

**classmethod from_config**(*config*, *meta: pytext.data.data_handler.CommonMetadata = None*, *tensorizers=None*)

**report_realtime_metric**(*stage*)

pytext.metric_reporters.language_model_metric_reporter.**get_perplexity_func**(*perplexity_type*)

### pytext.metric_reporters.mask_compositional module

### pytext.metric_reporters.metric_reporter module

**class** pytext.metric_reporters.metric_reporter.**MetricReporter**(*channels*, *log_gradient=False*, *pep_format=False*)

Bases: *pytext.config.component.Component*

MetricReporter is responsible of three things:

1. Aggregate output from trainer, which includes model inputs, predictions, targets, scores, and loss.
2. Calculate metrics using the aggregated output, and define how the metric is used to find best model
3. Optionally report the metrics and aggregated output to various channels

**lower_is_better**
Whether a lower metric indicates better performance. Set to True for e.g. perplexity, and False for e.g. accuracy. Default is False

**Type** bool

**channels**
A list of Channel that will receive metrics and the aggregated trainer output then format and report them in any customized way.

**Type** List[Channel]

MetricReporter is tightly-coupled with metric aggregation and computation which makes inheritance hard to reuse the parent functionalities and attributes. Next step is to decouple the metric aggregation and computation vs metric reporting.

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch

- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**add_channel**(*channel*)

**add_gradients**(*model*)

**classmethod aggregate_data**(*all_data*, *new_batch*)
Aggregate a batch of data, basically just convert tensors to list of native python data

**aggregate_preds**(*batch_preds*, *batch_context=None*)

**aggregate_scores**(*batch_scores*)

**aggregate_targets**(*batch_targets*, *batch_context=None*)

**batch_context**(*raw_batch*, *batch*)

**calculate_loss**()
Calculate the average loss for all aggregated batch

**calculate_metric**()
Calculate metrics, each sub class should implement it

**compare_metric**(*new_metric*, *old_metric*)
Check if new metric indicates better model performance

**Returns** bool, true if model with new_metric performs better

**gen_extra_context**(**args*)
Generate any extra intermediate context data for metric calculation

**get_gradients**()

**get_meta**()
Get global meta data that is not specific to any batch, the data will be pass along to channels

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**log_gradient = False**

**lower_is_better = False**

**predictions_to_report**()
Generate human readable predictions

**report_metric**(*model*, *stage*, *epoch*, *reset=True*, *print_to_channels=True*, *optimizer=None*)
Calculate metrics and average loss, report all statistic data to channels

**Parameters**

- **model** (*nn.Module*) – the PyTorch neural network model.
- **stage** (*Stage*) – training, evaluation or test
- **epoch** (*int*) – current epoch
- **reset** (*bool*) – if all data should be reset after report, default is True
- **print_to_channels** (*bool*) – if report data to channels, default is True

**report_realtime_metric**(*stage*)

**targets_to_report**()
Generate human readable targets

**class** pytext.metric_reporters.metric_reporter.**PureLossMetricReporter**(*channels*, *log_gradient=False*, *pep_format=False*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, **args*, ***kwargs*)

**lower_is_better = True**

### pytext.metric_reporters.pairwise_ranking_metric_reporter module

**class** pytext.metric_reporters.pairwise_ranking_metric_reporter.**PairwiseRankingMetricReporte**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *meta: pytext.data.data_handler.CommonMetadata = None*, *tensorizers=None*)

**static get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

### pytext.metric_reporters.regression_metric_reporter module

**class** pytext.metric_reporters.regression_metric_reporter.**RegressionMetricReporter**(*channels*, *log_gradient*
*pep_format=*

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *tensorizers=None*)

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**lower_is_better = False**

### pytext.metric_reporters.seq2seq_compositional module

**class** pytext.metric_reporters.seq2seq_compositional.**CompositionalSeq2SeqFileChannel**(*stages, file_path, tensorizers, accept_flat_*

Bases: *pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqFileChannel*

**gen_content**(*metrics, loss, preds, targets, scores, context*)

**get_title**(*context_keys=()*)

**validated_annotation**(*predicted_output_sequence*)

**class** pytext.metric_reporters.seq2seq_compositional.**Seq2SeqCompositionalMetricReporter**(*chan log_g tensoriz ers, accept_*

Bases: *pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter*

**aggregate_preds**(*new_batch, context=None*)

**aggregate_targets**(*new_batch, context=None*)

**batch_context**(*raw_batch, batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**create_frame_prediction_pairs**()

**classmethod from_config**(*config: pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositionalMetricReporte tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**get_annotation_from_string**(*stringified_tree_str: str*) → pytext.data.data_structures.annotation.Annotation

**stringify_annotation_tree**(*tree_tokens, tree_vocab*)

### pytext.metric_reporters.seq2seq_metric_reporter module

**class** pytext.metric_reporters.seq2seq_metric_reporter.**Seq2SeqFileChannel**(*stages, file_path, tensorizers*)

Bases: *pytext.metric_reporters.channel.FileChannel*

**gen_content** (*metrics*, *loss*, *preds*, *targets*, *scores*, *context*)

**get_title** (*context_keys=()*)

**class** pytext.metric_reporters.seq2seq_metric_reporter.**Seq2SeqMetricReporter** (*channels*, *log_gradient*, *tensorizers*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**add_batch_stats** (*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)

Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**aggregate_preds** (*new_batch*, *context=None*)

**aggregate_src_tokens** (*new_batch*)

**aggregate_targets** (*new_batch*, *context=None*)

**batch_context** (*raw_batch*, *batch*)

**calculate_metric** ()

Calculate metrics, each sub class should implement it

**classmethod from_config** (*config: pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**gen_extra_context** (**args*)

Generate any extra intermediate context data for metric calculation

**get_model_select_metric** (*metrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**lower_is_better = True**

### pytext.metric_reporters.seq2seq_utils module

pytext.metric_reporters.seq2seq_utils.**stringify** (*token_indices*, *vocab*)

### pytext.metric_reporters.squad_metric_reporter module

**class** pytext.metric_reporters.squad_metric_reporter.**SquadFileChannel** (*stages*, *file_path*)

Bases: *pytext.metric_reporters.channel.FileChannel*

**gen_content**(*metrics*, *loss*, *preds*, *targets*, *scores*, *contexts*, **args*)

**get_title**(*context_keys=()*)

**class** pytext.metric_reporters.squad_metric_reporter.**SquadMetricReporter**(*channels: List[pytext.metric_reporters n_best_size: int, max_answer_length: int, ignore_impossible: bool, has_answer_labels: List[str], tensorizer=None, false_label='False'*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**ANSWERS_COLUMN = 'answers'**

**DOC_COLUMN = 'doc'**

**QUES_COLUMN = 'question'**

**ROW_INDEX = 'id'**

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***contexts*)

Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**aggregate_preds**(*new_batch*, *context=None*)

**aggregate_scores**(*new_batch*)

**aggregate_targets**(*new_batch*, *context=None*)

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, **args*, *tensorizers=None*, ***kwargs*)

**get_model_select_metric**(*metric: pytext.metrics.squad_metrics.SquadMetrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

### pytext.metric_reporters.word_tagging_metric_reporter module

**class** pytext.metric_reporters.word_tagging_metric_reporter.**MultiLabelSequenceTaggingMetricR**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**aggregate_preds**(*batch_preds*, *batch_context=None*)

**aggregate_scores**(*batch_scores*)

**aggregate_targets**(*batch_targets*, *batch_context=None*)

**aggregate_tuple_data**(*all_data*, *new_batch*)

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *tensorizers*)

**static get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.word_tagging_metric_reporter.**NERMetricReporter**(*label_names: List[str], pad_idx: int, channels: List[pytext.metric_r use_bio_labels: bool = True*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**() → pytext.metrics.PRF1Metrics
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *tensorizer*)

**static get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.word_tagging_metric_reporter.**SequenceTaggingMetricReporter**(*la pa ch ne*

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *tensorizer*)

**static get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.word_tagging_metric_reporter.**Span**(*label*, *start*, *end*)

Bases: tuple

**end**
Alias for field number 2

**label**
Alias for field number 0

**start**
Alias for field number 1

**class** pytext.metric_reporters.word_tagging_metric_reporter.**WordTaggingMetricReporter**(*label_na List[str] use_bio bool, chan- nels: List[pyt*

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**calculate_loss**()
Calculate the average loss for all aggregated batch

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *meta: pytext.data.data_handler.CommonMetadata*)

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**process_pred**(*pred: List[int]*) → List[str]
pred is a list of token label index

pytext.metric_reporters.word_tagging_metric_reporter.**convert_bio_to_spans**(*bio_sequence: List[str]*) → List[pytext.metric_repor

Process the output and convert to spans for evaluation.

pytext.metric_reporters.word_tagging_metric_reporter.**get_slots**(*word_names*)

### Module contents

**class** pytext.metric_reporters.**Channel**(*stages: Tuple[pytext.common.constants.Stage, ...] = (<Stage.TRAIN: 'Training'>, <Stage.EVAL: 'Evaluation'>, <Stage.TEST: 'Test'>, <Stage.OTHERS: 'Others'>)*)

Bases: object

Channel defines how to format and report the result of a PyText job to an output stream.

**stages**
in which stages the report will be triggered, default is all stages, which includes train, eval, test

**close**()

**export**(*model*, *input_to_model=None*, ***kwargs*)

**report**(*stage*, *epoch*, *metrics*, *model_select_metric*, *loss*, *preds*, *targets*, *scores*, *context*, **args*)
Defines how to format and report data to the output channel.

**Parameters**

- **stage** (*Stage*) – train, eval or test
- **epoch** (*int*) – current epoch
- **metrics** (*Any*) – all metrics
- **model_select_metric** (*double*) – a single numeric metric to pick best model
- **loss** (*double*) – average loss
- **preds** (*List[Any]*) – list of predictions
- **targets** (*List[Any]*) – list of targets
- **scores** (*List[Any]*) – list of scores
- **context** (*Dict[str, List[Any]]*) – dict of any additional context data, each context is a list of data that maps to each example

**class** pytext.metric_reporters.**MetricReporter**(*channels*, *log_gradient=False*, *pep_format=False*)

Bases: *pytext.config.component.Component*

MetricReporter is responsible of three things:

1. Aggregate output from trainer, which includes model inputs, predictions, targets, scores, and loss.
2. Calculate metrics using the aggregated output, and define how the metric is used to find best model
3. Optionally report the metrics and aggregated output to various channels

**lower_is_better**
Whether a lower metric indicates better performance. Set to True for e.g. perplexity, and False for e.g. accuracy. Default is False

**Type** bool

**channels**
A list of Channel that will receive metrics and the aggregated trainer output then format and report them in any customized way.

**Type** List[Channel]

MetricReporter is tightly-coupled with metric aggregation and computation which makes inheritance hard to reuse the parent functionalities and attributes. Next step is to decouple the metric aggregation and computation vs metric reporting.

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch

- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**add_channel**(*channel*)

**add_gradients**(*model*)

**classmethod aggregate_data**(*all_data*, *new_batch*)
Aggregate a batch of data, basically just convert tensors to list of native python data

**aggregate_preds**(*batch_preds*, *batch_context=None*)

**aggregate_scores**(*batch_scores*)

**aggregate_targets**(*batch_targets*, *batch_context=None*)

**batch_context**(*raw_batch*, *batch*)

**calculate_loss**()
Calculate the average loss for all aggregated batch

**calculate_metric**()
Calculate metrics, each sub class should implement it

**compare_metric**(*new_metric*, *old_metric*)
Check if new metric indicates better model performance

**Returns** bool, true if model with new_metric performs better

**gen_extra_context**(**args*)
Generate any extra intermediate context data for metric calculation

**get_gradients**()

**get_meta**()
Get global meta data that is not specific to any batch, the data will be pass along to channels

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**log_gradient = False**

**lower_is_better = False**

**predictions_to_report**()
Generate human readable predictions

**report_metric**(*model*, *stage*, *epoch*, *reset=True*, *print_to_channels=True*, *optimizer=None*)
Calculate metrics and average loss, report all statistic data to channels

**Parameters**

- **model** (*nn.Module*) – the PyTorch neural network model.
- **stage** (*Stage*) – training, evaluation or test
- **epoch** (*int*) – current epoch

- **reset** (*bool*) – if all data should be reset after report, default is True
- **print_to_channels** (*bool*) – if report data to channels, default is True

**report_realtime_metric**(*stage*)

**targets_to_report**()
Generate human readable targets

**class** pytext.metric_reporters.**CalibrationMetricReporter**(*channels: List[pytext.metric_reporters.channel.Channel], pad_index: int = -1*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**aggregate_preds**(*batch_preds: torch.Tensor, batch_context=typing.Dict[str, typing.Any]*)

**aggregate_scores**(*batch_scores: torch.Tensor*)

**aggregate_targets**(*batch_targets: torch.Tensor, batch_context=typing.Dict[str, typing.Any]*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config: pytext.config.pytext_config.PyTextConfig, pad_index: int = -1*)

**class** pytext.metric_reporters.**ClassificationMetricReporter**(*label_names: List[str], channels: List[pytext.metric_reporters.channel.Channel], model_select_metric: pytext.metric_reporters.classification_metric_rep = <ComparableClassificationMetric.ACCURACY: 'accuracy'>, target_label: Optional[str] = None, text_column_names: List[str] = ['text'], additional_column_names: List[str] = [], recall_at_precision_thresholds: List[float] = [0.2, 0.4, 0.6, 0.8, 0.9], is_memory_efficient: bool = False*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**add_batch_stats**(*n_batches, preds, targets, scores, loss, m_input, **context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch

- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *meta: pytext.data.data_handler.CommonMetadata = None*, *tensorizers=None*)

**classmethod from_config_and_label_names**(*config, label_names: List[str]*)

**get_meta**()
Get global meta data that is not specific to any batch, the data will be pass along to channels

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**predictions_to_report**()
Generate human readable predictions

**targets_to_report**()
Generate human readable targets

**class** pytext.metric_reporters.**MultiLabelClassificationMetricReporter**(*label_names: List[str], channels: List[pytext.metric_reporters.cha model_select_metric: pytext.metric_reporters.classificati = <ComparableClassificationMetric.ACCURACY: 'accuracy'>, target_label: Optional[str] = None, text_column_names: List[str] = ['text'], additional_column_names: List[str] = [], recall_at_precision_thresholds: List[float] = [0.2, 0.4, 0.6, 0.8, 0.9], is_memory_efficient: bool = False*)

Bases: *pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter*

**calculate_metric**()

Calculate metrics, each sub class should implement it

**predictions_to_report**()

Generate human readable predictions

**targets_to_report**()

Generate human readable targets

**class** pytext.metric_reporters.**MultiLabelSequenceTaggingMetricReporter**(*label_names, pad_idx, channels, label_vocabs=None*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**aggregate_preds**(*batch_preds, batch_context=None*)

**aggregate_scores**(*batch_scores*)

**aggregate_targets**(*batch_targets, batch_context=None*)

**aggregate_tuple_data**(*all_data, new_batch*)

**batch_context**(*raw_batch, batch*)

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config, tensorizers*)

**static get_model_select_metric**(*metrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.**RegressionMetricReporter**(*channels, log_gradient=False, pep_format=False*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config, tensorizers=None*)

**get_model_select_metric**(*metrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**lower_is_better = False**

**class** pytext.metric_reporters.**IntentSlotMetricReporter**(*doc_label_names: List[str], word_label_names: List[str], use_bio_labels: bool, channels: List[pytext.metric_reporters.channel.Channel], slot_column_name: str = 'slots', text_column_name: str = 'text', token_tensorizer_name: str = 'tokens'*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**aggregate_preds**(*batch_preds, batch_context*)

**aggregate_scores**(*batch_scores*)

**aggregate_targets**(*batch_targets, batch_context*)

**batch_context**(*raw_batch, batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *tensorizers: Optional[Dict[KT, VT]] = None*)

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**get_raw_slot_str**(*raw_data_row*)

**predictions_to_report**()
Generate human readable predictions

**targets_to_report**()
Generate human readable targets

**class** pytext.metric_reporters.**LanguageModelMetricReporter**(*channels*, *metadata*, *tensorizers*, *aggregate_metrics*, *perplexity_type*, *pep_format*, *log_gradient=False*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**LABELS_COLUMN = 'labels'**

**RAW_TEXT_COLUMN = 'text'**

**TOKENS_COLUMN = 'tokens'**

**UTTERANCE_COLUMN = 'utterance'**

**add_batch_stats**(*n_batches*, *preds*, *targets*, *scores*, *loss*, *m_input*, ***context*)
Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**aggregate_context**(*context*)

**aggregate_scores**(*scores*)

**batch_context**(*raw_batch*, *batch*)

**calculate_loss**() → float
Calculate the average loss for all aggregated batch

**calculate_metric**() → pytext.metrics.language_model_metrics.LanguageModelMetric
Calculate metrics, each sub class should implement it

**compute_scores**(*logits*, *targets*)

**classmethod from_config**(*config: pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricRepo meta: pytext.data.data_handler.CommonMetadata = None, tensorizers=None*)

**get_model_select_metric**(*metrics*) → float

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**lower_is_better = True**

**class** pytext.metric_reporters.**SquadMetricReporter**(*channels: List[pytext.metric_reporters.channel.Channel], n_best_size: int, max_answer_length: int, ignore_impossible: bool, has_answer_labels: List[str], tensorizer=None, false_label='False'*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**ANSWERS_COLUMN = 'answers'**

**DOC_COLUMN = 'doc'**

**QUES_COLUMN = 'question'**

**ROW_INDEX = 'id'**

**add_batch_stats**(*n_batches, preds, targets, scores, loss, m_input, **contexts*)

Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**aggregate_preds**(*new_batch, context=None*)

**aggregate_scores**(*new_batch*)

**aggregate_targets**(*new_batch, context=None*)

**batch_context**(*raw_batch, batch*)

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config, *args, tensorizers=None, **kwargs*)

**get_model_select_metric**(*metric: pytext.metrics.squad_metrics.SquadMetrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.**WordTaggingMetricReporter**(*label_names: List[str], use_bio_labels: bool, channels: List[pytext.metric_reporters.channel.Channel]*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**calculate_loss**()
Calculate the average loss for all aggregated batch

**calculate_metric**()
Calculate metrics, each sub class should implement it

**classmethod from_config**(*config, meta: pytext.data.data_handler.CommonMetadata*)

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**process_pred**(*pred: List[int]*) → List[str]
pred is a list of token label index

**class** pytext.metric_reporters.**CompositionalMetricReporter**(*actions_vocab, channels: List[pytext.metric_reporters.channel.Channel], text_column_name: str = 'tokenized_text', tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**batch_context**(*raw_batch, batch*)

**calculate_metric**()
Calculate metrics, each sub class should implement it

**create_frame_prediction_pairs**()

**classmethod from_config**(*config, metadata: pytext.data.data_handler.CommonMetadata = None, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer] = None*)

**gen_extra_context**(**args*)
Generate any extra intermediate context data for metric calculation

**get_model_select_metric**(*metrics*)
Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**static node_to_metrics_node**(*node: Union[pytext.data.data_structures.annotation.Intent, pytext.data.data_structures.annotation.Slot], start: int = 0*) → pytext.metrics.intent_slot_metrics.Node
The input start is the absolute start position in utterance

**predictions_to_report**()
Generate human readable predictions

**targets_to_report**()
Generate human readable targets

**static tree_from_tokens_and_indx_actions**(*token_str_list: List[str], actions_vocab: List[str], actions_indices: List[int], validate_tree: bool = True*)

**static tree_to_metric_node**(*tree: pytext.data.data_structures.annotation.Tree*) → pytext.metrics.intent_slot_metrics.Node

Creates a Node from tree assuming the utterance is a concatenation of the tokens by whitespaces. The function does not necessarily reproduce the original utterance as extra whitespaces can be introduced.

**class** pytext.metric_reporters.**PairwiseRankingMetricReporter**(*channels, log_gradient=False, pep_format=False*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**add_batch_stats**(*n_batches, preds, targets, scores, loss, m_input, **context*)

Aggregates a batch of output data (predictions, scores, targets/true labels and loss).

**Parameters**

- **n_batches** (*int*) – number of current batch
- **preds** (*torch.Tensor*) – predictions of current batch
- **targets** (*torch.Tensor*) – targets of current batch
- **scores** (*torch.Tensor*) – scores of current batch
- **loss** (*double*) – average loss of current batch
- **m_input** (*Tuple[torch.Tensor, ..]*) – model inputs of current batch
- **context** (*Dict[str, Any]*) – any additional context data, it could be either a list of data which maps to each example, or a single value for the batch

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config, meta: pytext.data.data_handler.CommonMetadata = None, tensorizers=None*)

**static get_model_select_metric**(*metrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.**SequenceTaggingMetricReporter**(*label_names, pad_idx, channels*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**batch_context**(*raw_batch, batch*)

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config, tensorizer*)

**static get_model_select_metric**(*metrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.**PureLossMetricReporter**(*channels, log_gradient=False, pep_format=False*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

**calculate_metric**()

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config, *args, **kwargs*)

**lower_is_better = True**

**class** pytext.metric_reporters.**NERMetricReporter**(*label_names: List[str], pad_idx: int, channels: List[pytext.metric_reporters.channel.Channel], use_bio_labels: bool = True*)

Bases: pytext.metric_reporters.metric_reporter.MetricReporter

**batch_context**(*raw_batch*, *batch*)

**calculate_metric**() → pytext.metrics.PRF1Metrics

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, *tensorizer*)

**static get_model_select_metric**(*metrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

**class** pytext.metric_reporters.**DenseRetrievalMetricReporter**(*channels: List[pytext.metric_reporters.channel.Channel], text_column_names: List[str], model_select_metric: pytext.metric_reporters.dense_retrieval_metric_r task_batch_size: int, num_negative_ctxs: int = 0*)

Bases: pytext.metric_reporters.metric_reporter.MetricReporter

**aggregate_preds**(*preds*, *context*)

**batch_context**(*raw_batch*, *batch*) → Dict[str, Any]

**calculate_metric**() → pytext.metrics.dense_retrieval_metrics.DenseRetrievalMetrics

Calculate metrics, each sub class should implement it

**classmethod from_config**(*config*, **args*, *tensorizers=None*, ***kwargs*)

**get_model_select_metric**(*metrics: pytext.metrics.dense_retrieval_metrics.DenseRetrievalMetrics*)

Return a single numeric metric value that is used for model selection, returns the metric itself by default, but usually metrics will be more complicated data structures

## pytext.metrics package

### Submodules

### pytext.metrics.calibration_metrics module

**class** pytext.metrics.calibration_metrics.**AllCalibrationMetrics**(*calibration_metrics*)

Bases: tuple

**calibration_metrics**

Alias for field number 0

**print_metrics**(*report_pep=False*) → None

**class** pytext.metrics.calibration_metrics.**CalibrationMetrics**(*expected_error, max_error, total_error*)

Bases: tuple

**expected_error**
Alias for field number 0

**max_error**
Alias for field number 1

**print_metrics**(*report_pep=False*) → None

**total_error**
Alias for field number 2

pytext.metrics.calibration_metrics.**calculate_error**(*n_samples: int, bucket_values: List[List[float]], bucket_confidence: List[List[float]], bucket_accuracy: List[List[float]]*) → Tuple[float, float, float]

Computes several metrics used to measure calibration error, including expected calibration error (ECE), maximum calibration error (MCE), and total calibration error (TCE).

pytext.metrics.calibration_metrics.**compute_calibration**(*label_predictions: List[pytext.metrics.LabelPrediction]*) → Tuple[float, float, float]

pytext.metrics.calibration_metrics.**get_bucket_accuracy**(*bucket_values: List[List[float]], y_true: List[float], y_pred: List[float]*) → List[float]

Computes accuracy for each bucket. If a bucket does not have any predictions, uses -1 as a placeholder.

pytext.metrics.calibration_metrics.**get_bucket_confidence**(*bucket_values: List[List[float]]*) → List[float]

Computes average confidence for each bucket. If a bucket does not have any predictions, uses -1 as a placeholder.

pytext.metrics.calibration_metrics.**get_bucket_scores**(*y_score: List[float], buckets: int = 10*) → Tuple[List[List[float]], List[int]]

Organizes real-valued posterior probabilities into buckets. For example, if we have 10 buckets, the probabilities 0.0, 0.1, 0.2 are placed into buckets 0 (0.0 <= p < 0.1), 1 (0.1 <= p < 0.2), and 2 (0.2 <= p < 0.3), respectively.

### pytext.metrics.dense_retrieval_metrics module

**class** pytext.metrics.dense_retrieval_metrics.**DenseRetrievalMetrics**

Bases: `tuple`

Metric class for dense passage retrieval.

**num_examples**
number of samples

**Type** int

**accuracy**
how many times did we get the +ve doc from list of docs

**Type** float

**average_rank**
average rank of positive passage

**Type** float

**mean_reciprocal_rank**
average 1/rank of positive passage

**Type** float

**accuracy**
Alias for field number 1

**average_rank**
Alias for field number 2

**mean_reciprocal_rank**
Alias for field number 3

**num_examples**
Alias for field number 0

**print_metrics**() → None

### pytext.metrics.intent_slot_metrics module

**class** pytext.metrics.intent_slot_metrics.**AllMetrics**
Bases: tuple

Aggregated class for intent-slot related metrics.

**top_intent_accuracy**
Accuracy of the top-level intent.

**frame_accuracy**
Frame accuracy.

**frame_accuracies_by_depth**
Frame accuracies bucketized by depth of the gold tree.

**bracket_metrics**
Bracket metrics for intents and slots. For details, see the function *compute_intent_slot_metrics()*.

**tree_metrics**
Tree metrics for intents and slots. For details, see the function *compute_intent_slot_metrics()*.

**loss**
Cross entropy loss.

**bracket_metrics**
Alias for field number 4

**frame_accuracies_by_depth**
Alias for field number 3

**frame_accuracy**
Alias for field number 1

**frame_accuracy_top_k**
Alias for field number 2

**loss**
Alias for field number 6

**print_metrics**() → None

**top_intent_accuracy**
Alias for field number 0

**tree_metrics**
Alias for field number 5

pytext.metrics.intent_slot_metrics.**FrameAccuraciesByDepth = typing.Dict[int, pytext.metrics**
Frame accuracies bucketized by depth of the gold tree.

**class** pytext.metrics.intent_slot_metrics.**FrameAccuracy**
Bases: tuple

Frame accuracy for a collection of intent frame predictions.

Frame accuracy means the entire tree structure of the predicted frame matches that of the gold frame.

**frame_accuracy**
Alias for field number 1

**num_samples**
Alias for field number 0

**class** pytext.metrics.intent_slot_metrics.**FramePredictionPair**
Bases: tuple

Pair of predicted and gold intent frames.

**expected_frame**
Alias for field number 1

**predicted_frame**
Alias for field number 0

**class** pytext.metrics.intent_slot_metrics.**IntentSlotConfusions**
Bases: tuple

Aggregated class for intent and slot confusions.

**intent_confusions**
Confusion counts for intents.

**slot_confusions**
Confusion counts for slots.

**intent_confusions**
Alias for field number 0

**slot_confusions**
Alias for field number 1

**class** pytext.metrics.intent_slot_metrics.**IntentSlotMetrics**
Bases: tuple

Precision/recall/F1 metrics for intents and slots.

**intent_metrics**
Precision/recall/F1 metrics for intents.

**slot_metrics**
Precision/recall/F1 metrics for slots.

**overall_metrics**
Combined precision/recall/F1 metrics for all nodes (merging intents and slots).

**intent_metrics**
Alias for field number 0

**overall_metrics**
Alias for field number 2

**print_metrics**() → None

**slot_metrics**
Alias for field number 1

**class** pytext.metrics.intent_slot_metrics.**IntentsAndSlots**
Bases: tuple

Collection of intents and slots in an intent frame.

**intents**
Alias for field number 0

**slots**
Alias for field number 1

**class** pytext.metrics.intent_slot_metrics.**Node**(*label: str, span: pytext.data.data_structures.node.Span, children: Optional[AbstractSet[Node]] = None, text: str = None*)
Bases: *pytext.data.data_structures.node.Node*

Subclass of the base Node class, used for metric purposes. It is immutable so that hashing can be done on the class.

**label**
Label of the node.

**Type** str

**span**
Span of the node.

**Type** Span

**children**
frozenset of the node's children, left empty when computing bracketing metrics.

**Type** frozenset of *Node*

**text**
Text the node covers (=utterance[span.start:span.end])

**Type** str

**class** pytext.metrics.intent_slot_metrics.**NodesPredictionPair**
Bases: tuple

Pair of predicted and expected sets of nodes.

**expected_nodes**
Alias for field number 1

**predicted_nodes**
Alias for field number 0

pytext.metrics.intent_slot_metrics.**compare_frames**(*predicted_frame: pytext.metrics.intent_slot_metrics.Node, expected_frame: pytext.metrics.intent_slot_metrics.Node, tree_based: bool, intent_per_label_confusions: Optional[pytext.metrics.PerLabelConfusions] = None, slot_per_label_confusions: Optional[pytext.metrics.PerLabelConfusions] = None*) → pytext.metrics.intent_slot_metrics.IntentSlotConfusions

Compares two intent frames and returns TP, FP, FN counts for intents and slots. Optionally collects the per label TP, FP, FN counts.

**Parameters**

- **predicted_frame** – Predicted intent frame.
- **expected_frame** – Gold intent frame.
- **tree_based** – Whether to get the tree-based confusions (if True) or bracket-based confusions (if False). For details, see the function *compute_intent_slot_metrics()*.
- **intent_per_label_confusions** – If provided, update the per label confusions for intents as well. Defaults to None.
- **slot_per_label_confusions** – If provided, update the per label confusions for slots as well. Defaults to None.

**Returns** IntentSlotConfusions, containing confusion counts for intents and slots.

pytext.metrics.intent_slot_metrics.**compute_all_metrics**(*frame_pairs: Sequence[pytext.metrics.intent_slot_metrics.FramePrec top_intent_accuracy: bool = True, frame_accuracy: bool = True, frame_accuracies_by_depth: bool = True, bracket_metrics: bool = True, tree_metrics: bool = True, overall_metrics: bool = False, all_predicted_frames: List[List[pytext.metrics.intent_slot_metrics.Node]] = None, calculated_loss: float = None, length_metrics: Dict[KT, VT] = None*) → pytext.metrics.intent_slot_metrics.AllMetrics

Given a list of predicted and gold intent frames, computes intent-slot related metrics.

**Parameters**

- **frame_pairs** – List of predicted and gold intent frames.
- **top_intent_accuracy** – Whether to compute top intent accuracy or not. Defaults to True.
- **frame_accuracy** – Whether to compute frame accuracy or not. Defaults to True.

- **frame_accuracies_by_depth** – Whether to compute frame accuracies by depth or not. Defaults to True.
- **bracket_metrics** – Whether to compute bracket metrics or not. Defaults to True.
- **tree_metrics** – Whether to compute tree metrics or not. Defaults to True.
- **overall_metrics** – If *bracket_metrics* or *tree_metrics* is true, decides whether to compute overall (merging intents and slots) metrics for them. Defaults to False.

**Returns** AllMetrics which contains intent-slot related metrics.

pytext.metrics.intent_slot_metrics.**compute_frame_accuracies_by_depth**(*frame_pairs: Sequence[pytext.metrics.intent_slo* → Dict[int, pytext.metrics.intent_slot_metrics.

Given a list of predicted and gold intent frames, splits the predictions into buckets according to the depth of the gold trees, and computes frame accuracy for each bucket.

**Parameters** **frame_pairs** – List of predicted and gold intent frames.

**Returns** FrameAccuraciesByDepth, a map from depths to their corresponding frame accuracies.

pytext.metrics.intent_slot_metrics.**compute_frame_accuracy**(*frame_pairs: Sequence[pytext.metrics.intent_slot_metrics.Frame* → float

Computes frame accuracy given a list of predicted and gold intent frames.

**Parameters** **frame_pairs** – List of predicted and gold intent frames.

**Returns** Frame accuracy. For a prediction, frame accuracy is achieved if the entire tree structure of the predicted frame matches that of the gold frame.

pytext.metrics.intent_slot_metrics.**compute_frame_accuracy_top_k**(*frame_pairs: List[pytext.metrics.intent_slot_metrics.I all_frames: List[List[pytext.metrics.intent_slot_met* → float

pytext.metrics.intent_slot_metrics.**compute_intent_slot_metrics**(*frame_pairs: Sequence[pytext.metrics.intent_slot_metric tree_based: bool, overall_metrics: bool = True*) → pytext.metrics.intent_slot_metrics.IntentSlo

Given a list of predicted and gold intent frames, computes precision, recall and F1 metrics for intents and slots, either in tree-based or bracket-based manner.

The following assumptions are taken on intent frames: 1. The root node is an intent, 2. Children of intents are always slots, and children of slots are always intents.

For tree-based metrics, a node (an intent or slot) in the predicted frame is considered a true positive only if the subtree rooted at this node has an exact copy in the gold frame, otherwise it is considered a false positive. A false negative is a node in the gold frame that does not have an exact subtree match in the predicted frame.

For bracket-based metrics, a node in the predicted frame is considered a true positive if there is a node in the gold frame having the same label and span (but not necessarily the same children). The definitions of false positives and false negatives are similar to the above.

**Parameters**

- **frame_pairs** – List of predicted and gold intent frames.
- **tree_based** – Whether to compute tree-based metrics (if True) or bracket-based metrics (if False).
- **overall_metrics** – Whether to compute overall (merging intents and slots) metrics or not. Defaults to True.

**Returns** IntentSlotMetrics, containing precision/recall/F1 metrics for intents and slots.

pytext.metrics.intent_slot_metrics.**compute_metric_at_k**(*references: List[pytext.metrics.intent_slot_metrics.Node], hypothesis: List[List[pytext.metrics.intent_slot_metrics.Node]], metric_fn: Callable[[pytext.metrics.intent_slot_metrics.Node, pytext.metrics.intent_slot_metrics.Node], bool] = <function <lambda>>*) → List[float]

Computes a boolean metric at each position in the ranked list of hypothesis, and returns an average for each position over all examples. By default metric_fn is comparing if frames are equal.

pytext.metrics.intent_slot_metrics.**compute_prf1_metrics**(*nodes_pairs: Sequence[pytext.metrics.intent_slot_metrics.NodesPre*) → Tuple[pytext.metrics.AllConfusions, pytext.metrics.PRF1Metrics]

Computes precision/recall/F1 metrics given a list of predicted and expected sets of nodes.

**Parameters** **nodes_pairs** – List of predicted and expected node sets.

**Returns** A tuple, of which the first member contains the confusion information, and the second member contains the computed precision/recall/F1 metrics.

pytext.metrics.intent_slot_metrics.**compute_top_intent_accuracy**(*frame_pairs: Sequence[pytext.metrics.intent_slot_metric*) → float

Computes accuracy of the top-level intent.

**Parameters** **frame_pairs** – List of predicted and gold intent frames.

**Returns** Prediction accuracy of the top-level intent.

### pytext.metrics.language_model_metrics module

**class** pytext.metrics.language_model_metrics.**LanguageModelMetric**

Bases: tuple

Class for language model metrics.

**perplexity_per_word**

Average perplexity per word of the dataset.

**perplexity_per_word**
Alias for field number 0

**print_metrics**()

pytext.metrics.language_model_metrics.**compute_language_model_metric**(*loss_per_word: float*) → pytext.metrics.language_model_met

### pytext.metrics.mask_metrics module

### pytext.metrics.seq2seq_metrics module

**class** pytext.metrics.seq2seq_metrics.**Seq2SeqMetrics**(*loss*, *exact_match*, *f1*, *bleu*)
Bases: tuple

**bleu**
Alias for field number 3

**exact_match**
Alias for field number 1

**f1**
Alias for field number 2

**loss**
Alias for field number 0

**print_metrics**() → None

**class** pytext.metrics.seq2seq_metrics.**Seq2SeqTopKMetrics**
Bases: *pytext.metrics.seq2seq_metrics.Seq2SeqMetrics*

**print_metrics**() → None

pytext.metrics.seq2seq_metrics.**compute_f1**(*hypothesis_list*, *reference_list*, *eps=1e-08*)
Computes token F1 given a hypothesis and reference. This is defined as F1 = 2 * ((P * R) / (P + R + eps)) where P = precision, R = recall, and eps = epsilon for smoothing zero denominators. By default, eps = 1e-8.

### pytext.metrics.squad_metrics module

**class** pytext.metrics.squad_metrics.**SquadMetrics**(*classification_metrics*, *num_examples*, *exact_matches*, *f1_score*)
Bases: tuple

**classification_metrics**
Alias for field number 0

**exact_matches**
Alias for field number 2

**f1_score**
Alias for field number 3

**num_examples**
Alias for field number 1

**print_metrics**() → None

### Module contents

**class** pytext.metrics.**AllConfusions**

Bases: object

Aggregated class for per label confusions.

**per_label_confusions**

Per label confusion information.

**confusions**

Overall TP, FP and FN counts across the labels in *per_label_confusions*.

**compute_metrics**() → pytext.metrics.PRF1Metrics

**confusions**

**per_label_confusions**

**class** pytext.metrics.**ClassificationMetrics**

Bases: tuple

Metric class for various classification metrics.

**accuracy**

Overall accuracy of predictions.

**macro_prf1_metrics**

Macro precision/recall/F1 scores.

**per_label_soft_scores**

Per label soft metrics.

**mcc**

Matthews correlation coefficient.

**roc_auc**

Area under the Receiver Operating Characteristic curve.

**loss**

Training loss (only used for selecting best model, no need to print).

**accuracy**

Alias for field number 0

**loss**

Alias for field number 5

**macro_prf1_metrics**

Alias for field number 1

**mcc**

Alias for field number 3

**per_label_soft_scores**

Alias for field number 2

**print_metrics**(*report_pep=False*) → None

**print_pep**()

**roc_auc**

Alias for field number 4

**class** pytext.metrics.**Confusions**(*TP: int = 0, FP: int = 0, FN: int = 0*)

Bases: object

Confusion information for a collection of predictions.

**TP**
Number of true positives.

**FP**
Number of false positives.

**FN**
Number of false negatives.

**FN**

**FP**

**TP**

**compute_metrics**() → pytext.metrics.PRF1Scores

**class** pytext.metrics.**LabelListPrediction**

Bases: tuple

Label list predictions of an example.

**label_scores**
Confidence scores that each label receives.

**predicted_label**
List of indices of the predicted label.

**expected_label**
List of indices of the true label.

**expected_label**
Alias for field number 2

**label_scores**
Alias for field number 0

**predicted_label**
Alias for field number 1

**class** pytext.metrics.**LabelPrediction**

Bases: tuple

Label predictions of an example.

**label_scores**
Confidence scores that each label receives.

**predicted_label**
Index of the predicted label. This is usually the label with the highest confidence score in label_scores.

**expected_label**
Index of the true label.

**expected_label**
Alias for field number 2

**label_scores**
Alias for field number 0

**predicted_label**
Alias for field number 1

**class** pytext.metrics.**MacroPRF1Metrics**
Bases: tuple

Aggregated metric class for macro precision/recall/F1 scores.

**per_label_scores**
Mapping from label string to the corresponding precision/recall/F1 scores.

**macro_scores**
Macro precision/recall/F1 scores across the labels in *per_label_scores*.

**macro_scores**
Alias for field number 1

**per_label_scores**
Alias for field number 0

**print_metrics**(*indentation=''*) → None

**class** pytext.metrics.**MacroPRF1Scores**
Bases: tuple

Macro precision/recall/F1 scores (averages across each label).

**num_label**
Number of distinct labels.

**precision**
Equally weighted average of precisions for each label.

**recall**
Equally weighted average of recalls for each label.

**f1**
Equally weighted average of F1 scores for each label.

**f1**
Alias for field number 3

**num_labels**
Alias for field number 0

**precision**
Alias for field number 1

**recall**
Alias for field number 2

**class** pytext.metrics.**MultiLabelSoftClassificationMetrics**
Bases: tuple

Classification scores that are independent of thresholds.

**average_label_precision**
Alias for field number 0

**average_label_recall**
Alias for field number 2

**average_overall_accuracy**
Alias for field number 11

**average_overall_auc**
Alias for field number 9

**average_overall_precision**
Alias for field number 1

**average_overall_recall**
Alias for field number 3

**decision_thresh_at_precision**
Alias for field number 5

**decision_thresh_at_recall**
Alias for field number 7

**label_accuracy**
Alias for field number 10

**precision_at_recall**
Alias for field number 6

**recall_at_precision**
Alias for field number 4

**roc_auc**
Alias for field number 8

pytext.metrics.**PRECISION_AT_RECALL_THRESHOLDS = [0.2, 0.4, 0.6, 0.8, 0.9]**
Basic metric classes and functions for single-label prediction problems. Extending to multi-label support

**class** pytext.metrics.**PRF1Metrics**
Bases: tuple

Metric class for all types of precision/recall/F1 scores.

**per_label_scores**
Map from label string to the corresponding precision/recall/F1 scores.

**macro_scores**
Macro precision/recall/F1 scores across the labels in *per_label_scores*.

**micro_scores**
Micro (regular) precision/recall/F1 scores for the same collection of predictions.

**macro_scores**
Alias for field number 1

**micro_scores**
Alias for field number 2

**per_label_scores**
Alias for field number 0

**print_metrics**() → None

**class** pytext.metrics.**PRF1Scores**
Bases: tuple

Precision/recall/F1 scores for a collection of predictions.

**true_positives**
Number of true positives.

**false_positives**
Number of false positives.

**false_negatives**
Number of false negatives.

**precision**
TP / (TP + FP).

**recall**
TP / (TP + FN).

**f1**
2 * TP / (2 * TP + FP + FN).

**f1**
Alias for field number 5

**false_negatives**
Alias for field number 2

**false_positives**
Alias for field number 1

**precision**
Alias for field number 3

**recall**
Alias for field number 4

**true_positives**
Alias for field number 0

**class** pytext.metrics.**PairwiseRankingMetrics**
Bases: `tuple`

Metric class for pairwise ranking

**num_examples**
number of samples

**Type** int

**accuracy**
how many times did we rank in the correct order

**Type** float

**average_score_difference**
average score(higherRank) - score(lowerRank)

**Type** float

**accuracy**
Alias for field number 1

**average_score_difference**
Alias for field number 2

**num_examples**
Alias for field number 0

**print_metrics**() → None

**class** pytext.metrics.**PerLabelConfusions**
Bases: `object`

Per label confusion information.

**label_confusions_map**
Map from label string to the corresponding confusion counts.

**compute_metrics**() → pytext.metrics.MacroPRF1Metrics

**label_confusions_map**

**update**(*label: str*, *item: str*, *count: int*) → None
Increase one of TP, FP or FN count for a label by certain amount.

**Parameters**

- **label** – Label to be modified.
- **item** – Type of count to be modified, should be one of "TP", "FP" or "FN".
- **count** – Amount to be added to the count.

**Returns** None

**class** pytext.metrics.**RealtimeMetrics**
Bases: tuple

Realtime Metrics for tracking training progress and performance.

**samples**
number of samples

**Type** int

**tps**
tokens per second

**Type** float

**ups**
updates per second

**Type** float

**samples**
Alias for field number 0

**tps**
Alias for field number 1

**ups**
Alias for field number 2

**class** pytext.metrics.**RegressionMetrics**
Bases: tuple

Metrics for regression tasks.

**num_examples**
number of examples

**Type** int

**pearson_correlation**
correlation between predictions and labels

**Type** float

**mse**
mean-squared error between predictions and labels

**Type** float

**mse**
Alias for field number 2

**num_examples**
Alias for field number 0

**pearson_correlation**
Alias for field number 1

**print_metrics**()

**class** pytext.metrics.**SoftClassificationMetrics**
Bases: tuple

Classification scores that are independent of thresholds.

**average_precision**
Alias for field number 0

**decision_thresh_at_precision**
Alias for field number 2

**decision_thresh_at_recall**
Alias for field number 4

**precision_at_recall**
Alias for field number 3

**recall_at_precision**
Alias for field number 1

**roc_auc**
Alias for field number 5

pytext.metrics.**average_precision_score**(*y_true_sorted: numpy.ndarray, y_score_sorted: numpy.ndarray*) → float

Computes average precision, which summarizes the precision-recall curve as the precisions achieved at each threshold weighted by the increase in recall since the previous threshold.

**Parameters**

- **y_true_sorted** – Numpy array sorted according to decreasing confidence scores indicating whether each prediction is correct.
- **Numpy array of confidence scores for the predictions in** (*y_score_sorted*) – decreasing order.

**Returns** Average precision score.

TODO: This is too slow, improve the performance

pytext.metrics.**compute_average_recall**(*predictions: Sequence[pytext.metrics.LabelPrediction], label_names: Sequence[str], average_precisions: Dict[str, float]*) → float

pytext.metrics.**compute_classification_metrics**(*predictions: Sequence[pytext.metrics.LabelPrediction], label_names: Sequence[str], loss: float, average_precisions: bool = True, recall_at_precision_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9], precision_at_recall_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9]*) → pytext.metrics.ClassificationMetrics

A general function that computes classification metrics given a list of label predictions.

**Parameters**

- **predictions** – Label predictions, including the confidence score for each label.
- **label_names** – Indexed label names.
- **average_precisions** – Whether to compute average precisions for labels or not. Defaults to True.
- **recall_at_precision_thresholds** – precision thresholds at which to calculate recall
- **precision_at_recall_thresholds** – recall thresholds at which to calculate precision

**Returns** ClassificationMetrics which contains various classification metrics.

pytext.metrics.**compute_macro_avg**(*soft_metrics: Dict[str, pytext.metrics.SoftClassificationMetrics], metric: str*)

pytext.metrics.**compute_matthews_correlation_coefficients**(*TP: int, FP: int, FN: int, TN: int*) → float

Computes Matthews correlation coefficient, a way to summarize all four counts (TP, FP, FN, TN) in the confusion matrix of binary classification.

**Parameters**

- **TP** – Number of true positives.
- **FP** – Number of false positives.
- **FN** – Number of false negatives.
- **TN** – Number of true negatives.

**Returns** Matthews correlation coefficient, which is *sqrt((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))*.

pytext.metrics.**compute_multi_label_classification_metrics**(*predictions: Sequence[pytext.metrics.LabelListPrediction], label_names: Sequence[str], loss: float, average_precisions: bool = True, recall_at_precision_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9], precision_at_recall_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9]*) → pytext.metrics.ClassificationMetrics

A general function that computes classification metrics given a list of multi-label predictions.

**Parameters**

- **predictions** – multi-label predictions, including the confidence score for each label.
- **label_names** – Indexed label names.
- **average_precisions** – Whether to compute average precisions for labels or not. Defaults to True.
- **recall_at_precision_thresholds** – precision thresholds at which to calculate recall
- **precision_at_recall_thresholds** – recall thresholds at which to calculate precision

**Returns** ClassificationMetrics which contains various classification metrics.

pytext.metrics.**compute_multi_label_full_vector_classification_metrics**(*predictions: Sequence[pytext.metrics.LabelLi la-bel_names: Se-quence[str], loss: float, aver-age_precisions: bool = True, re-call_at_precision_thresholds: Se-quence[float] = [0.2, 0.4, 0.6, 0.8, 0.9], preci-sion_at_recall_thresholds: Se-quence[float] = [0.2, 0.4, 0.6, 0.8, 0.9]*) → py-text.metrics.ClassificationMetr

A general function that computes classification metrics given a list of multi-label predictions.

**Parameters**

- **predictions** – multi-label predictions, including the confidence score for each label.
- **label_names** – Indexed label names.
- **average_precisions** – Whether to compute average precisions for labels or not. Defaults to True.
- **recall_at_precision_thresholds** – precision thresholds at which to calculate recall
- **precision_at_recall_thresholds** – recall thresholds at which to calculate precision

**Returns** ClassificationMetrics which contains various classification metrics.

pytext.metrics.**compute_multi_label_multi_class_soft_metrics**(*predictions: Sequence[Sequence[pytext.metrics.LabelPredic label_names: Sequence[str], label_vocabs: Sequence[Sequence[str]], recall_at_precision_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9], precision_at_recall_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9]*) → pytext.metrics.MultiLabelSoftClassificationMet

Computes multi-label soft classification metrics with multi-class accommodation

**Parameters**

- **predictions** – multi-label predictions, including the confidence score for each label.
- **label_names** – Indexed label names.
- **recall_at_precision_thresholds** – precision thresholds at which to calculate recall
- **precision_at_recall_thresholds** – recall thresholds at which to calculate precision

**Returns** Dict from label strings to their corresponding soft metrics.

pytext.metrics.**compute_multi_label_soft_full_vector_metrics**(*predictions: Sequence[pytext.metrics.LabelListPrediction], label_names: Sequence[str], recall_at_precision_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9], precision_at_recall_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9]*) → Dict[str, pytext.metrics.SoftClassificationMetrics]

Computes multi-label soft classification metrics

**Parameters**

- **predictions** – multi-label predictions, including the confidence score for each label.
- **label_names** – Indexed label names. May contain duplicate label names.
- **recall_at_precision_thresholds** – precision thresholds at which to calculate recall
- **precision_at_recall_thresholds** – recall thresholds at which to calculate precision

**Returns** Dict from label strings to their corresponding soft metrics.

pytext.metrics.**compute_multi_label_soft_metrics**(*predictions: Sequence[pytext.metrics.LabelListPrediction], label_names: Sequence[str], recall_at_precision_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9], precision_at_recall_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9]*) → Dict[str, pytext.metrics.SoftClassificationMetrics]

Computes multi-label soft classification metrics

**Parameters**

- **predictions** – multi-label predictions, including the confidence score for each label.
- **label_names** – Indexed label names.
- **recall_at_precision_thresholds** – precision thresholds at which to calculate recall
- **precision_at_recall_thresholds** – recall thresholds at which to calculate precision

**Returns** Dict from label strings to their corresponding soft metrics.

pytext.metrics.**compute_pairwise_ranking_metrics**(*predictions: Sequence[int], scores: Sequence[float]*) → pytext.metrics.PairwiseRankingMetrics

Computes metrics for pairwise ranking given sequences of predictions and scores

**Parameters**

- **predictions** – 1 if ranking was correct, 0 if ranking was incorrect
- **scores** – score(higher-ranked-sample) - score(lower-ranked-sample)

**Returns** PairwiseRankingMetrics object

pytext.metrics.**compute_prf1**(*tp: int, fp: int, fn: int*) → Tuple[float, float, float]

pytext.metrics.**compute_regression_metrics**(*predictions: Sequence[float], targets: Sequence[float]*) → pytext.metrics.RegressionMetrics

Computes metrics for regression tasks.abs

**Parameters**

- **predictions** – 1-D sequence of float predictions
- **targets** – 1-D sequence of float labels

**Returns** RegressionMetrics object

pytext.metrics.**compute_roc_auc**(*predictions: Sequence[pytext.metrics.LabelPrediction], target_class: int = 0*) → Optional[float]

Computes area under the Receiver Operating Characteristic curve, for binary classification. Implementation based off of (and explained at) https://www.ibm.com/developerworks/community/blogs/jfp/entry/Fast_Computation_of_AUC_ROC_score?lang=en.

pytext.metrics.**compute_roc_auc_given_sorted_positives**(*y_true_sorted: numpy.ndarray*) → Optional[float]

pytext.metrics.**compute_soft_metrics**(*predictions: Sequence[pytext.metrics.LabelPrediction], label_names: Sequence[str], recall_at_precision_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9], precision_at_recall_thresholds: Sequence[float] = [0.2, 0.4, 0.6, 0.8, 0.9]*) → Dict[str, pytext.metrics.SoftClassificationMetrics]

Computes soft classification metrics given a list of label predictions.

**Parameters**

- **predictions** – Label predictions, including the confidence score for each label.
- **label_names** – Indexed label names.
- **recall_at_precision_thresholds** – precision thresholds at which to calculate recall
- **precision_at_recall_thresholds** – recall thresholds at which to calculate precision

**Returns** Dict from label strings to their corresponding soft metrics.

pytext.metrics.**precision_at_recall**(*y_true_sorted: numpy.ndarray, y_score_sorted: numpy.ndarray, thresholds: Sequence[float]*) → Tuple[Dict[float, float], Dict[float, float]]

Computes precision at various recall levels

**Parameters**

- **y_true_sorted** – Numpy array sorted according to decreasing confidence scores indicating whether each prediction is correct.
- **y_score_sorted** – Numpy array of confidence scores for the predictions in decreasing order.
- **thresholds** – Sequence of floats indicating the requested recall thresholds

**Returns** Dictionary of maximum precision at requested recall thresholds. Dictionary of decision thresholds resulting in max precision at requested recall thresholds.

pytext.metrics.**recall_at_precision**(*y_true_sorted: numpy.ndarray, y_score_sorted: numpy.ndarray, thresholds: Sequence[float]*) → Dict[float, float]

Computes recall at various precision levels

**Parameters**

- **y_true_sorted** – Numpy array sorted according to decreasing confidence scores indicating whether each prediction is correct.
- **y_score_sorted** – Numpy array of confidence scores for the predictions in decreasing order.
- **thresholds** – Sequence of floats indicating the requested precision thresholds

**Returns** Dictionary of maximum recall at requested precision thresholds.

`pytext.metrics.safe_division`(*n: Union[int, float], d: int*) → float

`pytext.metrics.sort_by_score`(*y_true_list: Sequence[bool], y_score_list: Sequence[float]*)

### pytext.models package

### Subpackages

### pytext.models.decoders package

### Submodules

### pytext.models.decoders.decoder_base module

**class** `pytext.models.decoders.decoder_base.DecoderBase`(*config: pytext.config.pytext_config.ConfigBase*)

Bases: `pytext.models.module.Module`

Base class for all decoder modules.

**Parameters** **config** (*ConfigBase*) – Configuration object.

**in_dim**

Dimension of input Tensor passed to the decoder.

**Type** int

**out_dim**

Dimension of output Tensor produced by the decoder.

**Type** int

**forward**(**input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_decoder**()

Returns the decoder module.

**get_in_dim**() → int

Returns the dimension of the input Tensor that the decoder accepts.

**get_out_dim**() → int

Returns the dimension of the input Tensor that the decoder emits.

### pytext.models.decoders.intent_slot_model_decoder module

**class** pytext.models.decoders.intent_slot_model_decoder.**IntentSlotModelDecoder**(*config: pytext.models.decode in_dim_doc: int, in_dim_word: int, out_dim_doc: int, out_dim_word: int*)

Bases: *pytext.models.decoders.decoder_base.DecoderBase*

*IntentSlotModelDecoder* implements the decoder layer for intent-slot models. Intent-slot models jointly predict intent and slots from an utterance. At the core these models learn to jointly perform document classification and word tagging tasks.

***IntentSlotModelDecoder* accepts arguments for decoding both document** classification and word tagging tasks, namely, *in_dim_doc* and *in_dim_word*.

**Parameters**

- **config** (*type*) – Configuration object of type IntentSlotModelDecoder.Config.
- **in_dim_doc** (*type*) – Dimension of input Tensor for projecting document
- **representation.** –
- **in_dim_word** (*type*) – Dimension of input Tensor for projecting word
- **representation.** –
- **out_dim_doc** (*type*) – Dimension of projected output Tensor for document
- **classification.** –
- **out_dim_word** (*type*) – Dimension of projected output Tensor for word tagging.

**use_doc_probs_in_word**

Whether to use intent probabilities for

**Type** bool

**predicting slots.**

**doc_decoder**

Document/intent decoder module.

**Type** type

**word_decoder**

Word/slot decoder module.

**Type** type

**forward**(*x_d: torch.Tensor, x_w: torch.Tensor, dense: Optional[torch.Tensor] = None*) → Tuple[torch.Tensor, torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**get_decoder**() → List[torch.nn.modules.module.Module]
Returns the document and word decoder modules.

### pytext.models.decoders.mlp_decoder module

**class** pytext.models.decoders.mlp_decoder.**MLPDecoder**(*config: pytext.models.decoders.mlp_decoder.MLPDecoder.Config*, *in_dim: int*, *out_dim: int = 0*)

Bases: `pytext.models.decoders.decoder_base.DecoderBase`

*MLPDecoder* implements a fully connected network and uses ReLU as the activation function. The module projects an input tensor to *out_dim*.

**Parameters**

- **config** (*Config*) – Configuration object of type MLPDecoder.Config.
- **in_dim** (*int*) – Dimension of input Tensor passed to MLP.
- **out_dim** (*int*) – Dimension of output Tensor produced by MLP. Defaults to 0.

**mlp**
Module that implements the MLP.

**Type** type

**out_dim**
Dimension of the output of this module.

**Type** type

**hidden_dims**
Dimensions of the outputs of hidden layers.

**Type** List[int]

**forward**(**input*) → torch.Tensor
Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**get_decoder**() → List[torch.nn.modules.module.Module]
Returns the MLP module that is used as a decoder.

### pytext.models.decoders.mlp_decoder_query_response module

**class** pytext.models.decoders.mlp_decoder_query_response.**MLPDecoderQueryResponse**(*config: pytext.models.dec from_dim: int, to_dim: int*)

Bases: *pytext.models.decoders.decoder_base.DecoderBase*

Implements a 'two-tower' MLP: one for query and one for response Used in search pairwise ranking: both pos_response and neg_response use the response-MLP

**forward**(**x*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_decoder**() → List[torch.nn.modules.module.Module]

Returns the decoder module.

**static get_mlp**(*from_dim: int, to_dim: int, hidden_dims: List[int]*)

### pytext.models.decoders.mlp_decoder_two_tower module

**class** pytext.models.decoders.mlp_decoder_two_tower.**ExportType**

Bases: `enum.Enum`

An enumeration.

**LEFT = 'LEFT'**

**NONE = 'NONE'**

**RIGHT = 'RIGHT'**

**class** pytext.models.decoders.mlp_decoder_two_tower.**MLPDecoderTwoTower**(*config: pytext.models.decoders.mlp_dec right_dim: int, left_dim: int, to_dim: int, export_type=<ExportType.NON 'NONE'>*)

Bases: *pytext.models.decoders.decoder_base.DecoderBase*

Implements a 'two-tower' MLPDecoder: one for left and one for right

**forward**(**x*) → torch.Tensor
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_decoder**() → List[torch.nn.modules.module.Module]
Returns the decoder module.

**static get_mlp**(*from_dim: int, to_dim: int, hidden_dims: List[int], layer_norm: bool, dropout: float, export_embedding: bool = False*)

### pytext.models.decoders.multilabel_decoder module

**class** pytext.models.decoders.multilabel_decoder.**MultiLabelDecoder**(*config: pytext.models.decoders.multilabel_dec in_dim: int, output_dim: Dict[str, int], label_names: List[str]*)

Bases: *pytext.models.decoders.decoder_base.DecoderBase*

Implements a 'n-tower' MLP: one for each of the multi labels Used in USM/EA: the user satisfaction modeling, pTSR prediction and Error Attribution are all 3 label sets that need predicting.

**forward**(**input*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_decoder**() → List[torch.nn.modules.module.Module]
Returns the decoder module.

**static get_mlp**(*in_dim: int, out_dim: int, hidden_dims: List[int]*)

### Module contents

**class** pytext.models.decoders.**DecoderBase**(*config: pytext.config.pytext_config.ConfigBase*)
Bases: *pytext.models.module.Module*

Base class for all decoder modules.

**Parameters** **config** (*ConfigBase*) – Configuration object.

**in_dim**
Dimension of input Tensor passed to the decoder.

**Type** int

**out_dim**
Dimension of output Tensor produced by the decoder.

**Type** int

**forward**(**input*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_decoder**()
Returns the decoder module.

**get_in_dim**() → int
Returns the dimension of the input Tensor that the decoder accepts.

**get_out_dim**() → int
Returns the dimension of the input Tensor that the decoder emits.

**class** pytext.models.decoders.**MLPDecoder**(*config: pytext.models.decoders.mlp_decoder.MLPDecoder.Config*, *in_dim: int*, *out_dim: int = 0*)
Bases: `pytext.models.decoders.decoder_base.DecoderBase`

*MLPDecoder* implements a fully connected network and uses ReLU as the activation function. The module projects an input tensor to *out_dim*.

**Parameters**

- **config** (*Config*) – Configuration object of type MLPDecoder.Config.
- **in_dim** (*int*) – Dimension of input Tensor passed to MLP.
- **out_dim** (*int*) – Dimension of output Tensor produced by MLP. Defaults to 0.

**mlp**
Module that implements the MLP.

**Type** type

**out_dim**
Dimension of the output of this module.

**Type** type

**hidden_dims**
Dimensions of the outputs of hidden layers.

**Type** List[int]

**forward**(**input*) → torch.Tensor
Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**get_decoder**() → List[torch.nn.modules.module.Module]
Returns the MLP module that is used as a decoder.

**class** pytext.models.decoders.**IntentSlotModelDecoder**(*config: pytext.models.decoders.intent_slot_model_decoder.IntentSlo in_dim_doc: int, in_dim_word: int, out_dim_doc: int, out_dim_word: int*)

Bases: *pytext.models.decoders.decoder_base.DecoderBase*

*IntentSlotModelDecoder* implements the decoder layer for intent-slot models. Intent-slot models jointly predict intent and slots from an utterance. At the core these models learn to jointly perform document classification and word tagging tasks.

***IntentSlotModelDecoder* accepts arguments for decoding both document** classification and word tagging tasks, namely, *in_dim_doc* and *in_dim_word*.

**Parameters**

- **config** (*type*) – Configuration object of type IntentSlotModelDecoder.Config.
- **in_dim_doc** (*type*) – Dimension of input Tensor for projecting document
- **representation.** –
- **in_dim_word** (*type*) – Dimension of input Tensor for projecting word
- **representation.** –
- **out_dim_doc** (*type*) – Dimension of projected output Tensor for document
- **classification.** –
- **out_dim_word** (*type*) – Dimension of projected output Tensor for word tagging.

**use_doc_probs_in_word**
Whether to use intent probabilities for

**Type** bool

**predicting slots.**

**doc_decoder**
Document/intent decoder module.

**Type** type

**word_decoder**
Word/slot decoder module.

**Type** type

**forward**(*x_d: torch.Tensor, x_w: torch.Tensor, dense: Optional[torch.Tensor] = None*) → Tuple[torch.Tensor, torch.Tensor]
Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**get_decoder**() → List[torch.nn.modules.module.Module]
Returns the document and word decoder modules.

## pytext.models.embeddings package

### Submodules

### pytext.models.embeddings.char_embedding module

**class** pytext.models.embeddings.char_embedding.**CharacterEmbedding**(*num_embeddings: int, embed_dim: int, out_channels: int, kernel_sizes: List[int], highway_layers: int, projection_dim: Optional[int], *args, **kwargs*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for character aware CNN embeddings for tokens. It uses convolution followed by max-pooling over character embeddings to obtain an embedding vector for each token.

Implementation is loosely based on https://arxiv.org/abs/1508.06615.

**Parameters**

- **num_embeddings** (*int*) – Total number of characters (vocabulary size).
- **embed_dim** (*int*) – Size of character embeddings to be passed to convolutions.
- **out_channels** (*int*) – Number of output channels.
- **kernel_sizes** (*List[int]*) – Dimension of input Tensor passed to MLP.
- **highway_layers** (*int*) – Number of highway layers applied to pooled output.
- **projection_dim** (*int*) – If specified, size of output embedding for token, via a linear projection from convolution output.

**char_embed**
Character embedding table.

**Type** nn.Embedding

**convs**
Convolution layers that operate on character

**Type** nn.ModuleList

**embeddings.**

**highway_layers**
Highway layers on top of convolution output.

**Type** nn.Module

**projection**
Final linear layer to token embedding.

**Type** nn.Module

**embedding_dim**
Dimension of the final token embedding produced.

**Type** int

**forward**(*chars: torch.Tensor*) → torch.Tensor
Given a batch of sentences such that tokens are broken into character ids, produce token embedding vectors for each sentence in the batch.

**Parameters**

- **chars** (*torch.Tensor*) – Batch of sentences where each token is broken
- **characters.** (*into*) –
- **Dimension** – batch size X maximum sentence length X maximum word length

**Returns** Embedded batch of sentences. Dimension: batch size X maximum sentence length, token embedding size. Token embedding size = *out_channels * len(self.convs))*

**Return type** torch.Tensor

**classmethod from_config**(*config: pytext.config.field_config.CharFeatConfig*, *metadata: Optional[pytext.fields.field.FieldMeta] = None*, *vocab_size: Optional[int] = None*)
Factory method to construct an instance of CharacterEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*CharFeatConfig*) – Configuration object specifying all the parameters of CharacterEmbedding.
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of CharacterEmbedding.

**Return type** type

**class** pytext.models.embeddings.char_embedding.**Highway**(*input_dim: int*, *num_layers: int = 1*)

Bases: torch.nn.modules.module.Module

A *Highway layer <https://arxiv.org/abs/1505.00387>*. Adopted from the AllenNLP implementation.

**forward**(*x: torch.Tensor*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**reset_parameters**()

### pytext.models.embeddings.contextual_token_embedding module

**class** pytext.models.embeddings.contextual_token_embedding.**ContextualTokenEmbedding**(*embed_dim: int, downsample_dim: Optional[int] = None*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for providing token embeddings from a pretrained model.

**forward**(*embedding: torch.Tensor*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.config.field_config.ContextualTokenEmbeddingConfig, *args, **kwargs*)

### pytext.models.embeddings.dict_embedding module

**class** pytext.models.embeddings.dict_embedding.**DictEmbedding**(*num_embeddings: int, embed_dim: int, pooling_type: pytext.config.module_config.PoolingType, pad_index: int = 1, unk_index: int = 0, mobile: bool = False*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for dictionary feature embeddings for tokens. Dictionary features are also known as gazetteer features. These are per token discrete features that the module learns embeddings for. Example: For the utterance *Order coffee from Starbucks*, the dictionary features could be

```
[
    {"tokenIdx": 1, "features": {"drink/beverage": 0.8, "music/song": 0.2}},
```

(continues on next page)

(continued from previous page)

```
    {"tokenIdx": 3, "features": {"store/coffee_shop": 1.0}}
]
```

:: Thus, for a given token there can be more than one dictionary features each of which has a confidence score. The final embedding for a token is the weighted average of the dictionary embeddings followed by a pooling operation such that the module produces an embedding vector per token.

**Parameters**

- **num_embeddings** (*int*) – Total number of dictionary features (vocabulary size).
- **embed_dim** (*int*) – Size of embedding vector.
- **pooling_type** (*PoolingType*) – Type of pooling for combining the dictionary feature embeddings.

**pooling_type**

Type of pooling for combining the dictionary feature embeddings.

**Type** PoolingType

**find_and_replace**(*tensor: torch.Tensor*, *find_val: int*, *replace_val: int*) → torch.Tensor

*torch.where* is not supported for mobile ONNX, this hack allows a mobile exported version of *torch.where* which is computationally more expensive

**forward**(*feats: torch.Tensor*, *weights: torch.Tensor*, *lengths: torch.Tensor*) → torch.Tensor

Given a batch of sentences such containing dictionary feature ids per token, produce token embedding vectors for each sentence in the batch.

**Parameters**

- **feats** (*torch.Tensor*) – Batch of sentences with dictionary feature ids. shape: [bsz, seq_len * max_feat_per_token]
- **weights** (*torch.Tensor*) – Batch of sentences with dictionary feature weights for the dictionary features. shape: [bsz, seq_len * max_feat_per_token]
- **lengths** (*torch.Tensor*) – Batch of sentences with the number of dictionary features per token. shape: [bsz, seq_len]

**Returns** Embedded batch of sentences. Dimension: batch size X maximum sentence length, token embedding size. Token embedding size = *embed_dim* passed to the constructor.

**Return type** torch.Tensor

**classmethod from_config**(*config: pytext.config.field_config.DictFeatConfig*, *metadata: Optional[pytext.fields.field.FieldMeta] = None*, *labels: Optional[pytext.data.utils.Vocabulary] = None*, *tensorizer: Optional[pytext.data.tensorizers.Tensorizer] = None*)

Factory method to construct an instance of DictEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*DictFeatConfig*) – Configuration object specifying all the
- **of DictEmbedding.** (*parameters*) –
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of DictEmbedding.

**Return type** type

### pytext.models.embeddings.embedding_base module

**class** pytext.models.embeddings.embedding_base.**EmbeddingBase**(*embedding_dim: int*)

Bases: *pytext.models.module.Module*

Base class for token level embedding modules.

**Parameters** **embedding_dim** (*int*) – Size of embedding vector.

**num_emb_modules**
Number of ways to embed a token.

**Type** int

**embedding_dim**
Size of embedding vector.

**Type** int

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)
Overridden in sub classes to implement Tensorboard visualization of embedding space

### pytext.models.embeddings.embedding_list module

**class** pytext.models.embeddings.embedding_list.**EmbeddingList**(*embeddings: Iterable[pytext.models.embeddings.embedding_b concat: bool*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*, torch.nn.modules.container.ModuleList

There are more than one way to embed a token and this module provides a way to generate a list of sub-embeddings, concat embedding tensors into a single Tensor or return a tuple of Tensors that can be used by downstream modules.

**Parameters**

- **embeddings** (*Iterable[EmbeddingBase]*) – A sequence of embedding modules to
- **a token.** (*embed*) –
- **concat** (*bool*) – Whether to concatenate the embedding vectors emitted from
- **modules.** (*embeddings*) –

**num_emb_modules**
Number of flattened embeddings in *embeddings*, e.g: ((e1, e2), e3) has 3 in total

**Type** int

**input_start_indices**
List of indices of the sub-embeddings in the embedding list.

**Type** List[int]

**concat**
Whether to concatenate the embedding vectors emitted from *embeddings* modules.

**Type** bool

**embedding_dim**
Total embedding size, can be a single int or tuple of int depending on concat setting

**forward**(**emb_input*) → Union[torch.Tensor, Tuple[torch.Tensor]]

Get embeddings from all sub-embeddings and either concatenate them into one Tensor or return them in a tuple.

**Parameters** ***emb_input** (*type*) – Sequence of token level embeddings to combine. The inputs should match the size of configured embeddings. Each of them is either a Tensor or a tuple of Tensors.

**Returns**

**If *concat* is True then** a Tensor is returned by concatenating all embeddings. Otherwise all embeddings are returned in a tuple.

**Return type** Union[torch.Tensor, Tuple[torch.Tensor]]

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

### pytext.models.embeddings.mlp_embedding module

**class** pytext.models.embeddings.mlp_embedding.**MLPEmbedding**(*embedding_dim: int = 300, embeddings_weight: Optional[torch.Tensor] = None, init_range: Optional[List[int]] = None, init_std: Optional[float] = None, mlp_layer_dims: List[int] = ()*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

An MLP embedding wrapper module around *torch.nn.Embedding* to add transformations for float tensors.

**Parameters**

- **num_embeddings** (*int*) – Total number of words/tokens (vocabulary size).
- **embedding_dim** (*int*) – Size of embedding vector.
- **embeddings_weight** (*torch.Tensor*) – Pretrained weights to initialize the embedding table with.
- **init_range** (*List[int]*) – Range of uniform distribution to initialize the weights with if *embeddings_weight* is None.
- **mlp_layer_dims** (*List[int]*) – List of layer dimensions (if any) to add on top of the embedding lookup.

**forward**(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.config.field_config.MLPFeatConfig, metadata: Optional[pytext.fields.field.FieldMeta] = None, tensorizer: Optional[pytext.data.tensorizers.Tensorizer] = None, init_from_saved_state: Optional[bool] = False*)

Factory method to construct an instance of MLPEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*MLPFeatConfig*) – Configuration object specifying all the
- **of MLPEmbedding.** (*parameters*) –
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of MLPEmbedding.

**Return type** type

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

### pytext.models.embeddings.scriptable_embedding_list module

**class** pytext.models.embeddings.scriptable_embedding_list.**ScriptableEmbeddingList**(*embeddings: Iterable[pytext.mo*

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

This class is a Torchscript-friendly version of pytext.models.embeddings.EmbeddingList. The main differences are that it requires input arguments to be passed in as a list of Tensors, since Torchscript does not allow variable arguments, and that it only supports concat mode, since Torchscript does not support return value variance.

**class Wrapper1**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase*)

Bases: torch.nn.modules.module.Module

**forward**(*xs: List[torch.Tensor]*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class Wrapper3**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase*)

Bases: torch.nn.modules.module.Module

**forward**(*xs: List[torch.Tensor]*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**forward**(*emb_input: List[List[torch.Tensor]]*) → torch.Tensor

Get embeddings from all sub-embeddings and either concatenate them into one Tensor or return them in a tuple.

**Parameters** **emb_input** (*type*) – Sequence of token level embeddings to combine. The inputs should match the size of configured embeddings. Each of them is a List of Tensors.

**Returns** a Tensor is returned by concatenating all embeddings.

**Return type** torch.Tensor

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

### pytext.models.embeddings.word_embedding module

**class** pytext.models.embeddings.word_embedding.**WordEmbedding**(*num_embeddings: int, embedding_dim: int = 300, embeddings_weight: Optional[torch.Tensor] = None, init_range: Optional[List[int]] = None, init_std: Optional[float] = None, unk_token_idx: int = 0, mlp_layer_dims: List[int] = (), padding_idx: Optional[int] = None, vocab: Optional[List[str]] = None*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

A word embedding wrapper module around *torch.nn.Embedding* with options to initialize the word embedding weights and add MLP layers acting on each word.

Note: Embedding weights for UNK token are always initialized to zeros.

**Parameters**

- **num_embeddings** (*int*) – Total number of words/tokens (vocabulary size).
- **embedding_dim** (*int*) – Size of embedding vector.
- **embeddings_weight** (*torch.Tensor*) – Pretrained weights to initialize the embedding table with.
- **init_range** (*List[int]*) – Range of uniform distribution to initialize the weights with if *embeddings_weight* is None.
- **unk_token_idx** (*int*) – Index of UNK token in the word vocabulary.
- **mlp_layer_dims** (*List[int]*) – List of layer dimensions (if any) to add on top of the embedding lookup.

**forward**(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**freeze**()

**classmethod from_config**(*config: pytext.config.field_config.WordFeatConfig, metadata: Optional[pytext.fields.field.FieldMeta] = None, tensorizer: Optional[pytext.data.tensorizers.Tensorizer] = None, init_from_saved_state: Optional[bool] = False*)

Factory method to construct an instance of WordEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*WordFeatConfig*) – Configuration object specifying all the
- **of WordEmbedding.** (*parameters*) –
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of WordEmbedding.

**Return type** type

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

### pytext.models.embeddings.word_seq_embedding module

**class** pytext.models.embeddings.word_seq_embedding.**WordSeqEmbedding**(*lstm_config: pytext.models.representations.bilstm.l num_embeddings: int, word_embed_dim: int = 300, embeddings_weight: Optional[torch.Tensor] = None, init_range: Optional[List[int]] = None, init_std: Optional[float] = None, unk_token_idx: int = 0, padding_idx: Optional[int] = None, vocab: Optional[List[str]] = None*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

An embedding module represents a sequence of sentences

**Parameters**

- **lstm_config** (*BiLSTM.Config*) – config of the lstm layer
- **num_embeddings** (*int*) – Total number of words/tokens (vocabulary size).
- **embedding_dim** (*int*) – Size of embedding vector.
- **embeddings_weight** (*torch.Tensor*) – Pretrained weights to initialize the embedding table with.
- **init_range** (*List[int]*) – Range of uniform distribution to initialize the weights with if *embeddings_weight* is None.
- **unk_token_idx** (*int*) – Index of UNK token in the word vocabulary.

**forward**(*seq_token_idx, seq_token_count*)

**Parameters**

- **seq_token_idx** – shape [batch_size * max_seq_len * max_token_count]
- **seq_token_count** – shape [batch_size * max_seq_len]

**Returns** shape (batch_size * max_seq_len * output_dim)

**Return type** embedding

**freeze**()

**classmethod from_config**(*config: pytext.models.embeddings.word_seq_embedding.WordSeqEmbedding.Config, tensorizer: pytext.data.tensorizers.Tensorizer = None, init_from_saved_state: Optional[bool] = False*)

Factory method to construct an instance of WordEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*WordSeqEmbedding.Config*) – Configuration object specifying all the
- **of WordEmbedding.** (*parameters*) –

**Returns** An instance of WordSeqEmbedding.

**Return type** type

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

### Module contents

**class** pytext.models.embeddings.**EmbeddingBase**(*embedding_dim: int*)

Bases: *pytext.models.module.Module*

Base class for token level embedding modules.

**Parameters** **embedding_dim** (*int*) – Size of embedding vector.

**num_emb_modules**

Number of ways to embed a token.

**Type** int

**embedding_dim**

Size of embedding vector.

**Type** int

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

**class** pytext.models.embeddings.**EmbeddingList**(*embeddings: Iterable[pytext.models.embeddings.embedding_base.EmbeddingBase], concat: bool*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*, torch.nn.modules.container.ModuleList

There are more than one way to embed a token and this module provides a way to generate a list of sub-embeddings, concat embedding tensors into a single Tensor or return a tuple of Tensors that can be used by downstream modules.

**Parameters**

- **embeddings** (*Iterable[EmbeddingBase]*) – A sequence of embedding modules to
- **a token.** (*embed*) –
- **concat** (*bool*) – Whether to concatenate the embedding vectors emitted from

- **modules.** (*embeddings*) –

**num_emb_modules**
Number of flattened embeddings in *embeddings*, e.g: ((e1, e2), e3) has 3 in total

**Type** int

**input_start_indices**
List of indices of the sub-embeddings in the embedding list.

**Type** List[int]

**concat**
Whether to concatenate the embedding vectors emitted from *embeddings* modules.

**Type** bool

**embedding_dim**
Total embedding size, can be a single int or tuple of int depending on concat setting

**forward**(**emb_input*) → Union[torch.Tensor, Tuple[torch.Tensor]]
Get embeddings from all sub-embeddings and either concatenate them into one Tensor or return them in a tuple.

**Parameters** ***emb_input** (*type*) – Sequence of token level embeddings to combine. The inputs should match the size of configured embeddings. Each of them is either a Tensor or a tuple of Tensors.

**Returns**

**If *concat* is True then** a Tensor is returned by concatenating all embeddings. Otherwise all embeddings are returned in a tuple.

**Return type** Union[torch.Tensor, Tuple[torch.Tensor]]

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)
Overridden in sub classes to implement Tensorboard visualization of embedding space

**class** pytext.models.embeddings.**WordEmbedding**(*num_embeddings: int, embedding_dim: int = 300, embeddings_weight: Optional[torch.Tensor] = None, init_range: Optional[List[int]] = None, init_std: Optional[float] = None, unk_token_idx: int = 0, mlp_layer_dims: List[int] = (), padding_idx: Optional[int] = None, vocab: Optional[List[str]] = None*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

A word embedding wrapper module around *torch.nn.Embedding* with options to initialize the word embedding weights and add MLP layers acting on each word.

Note: Embedding weights for UNK token are always initialized to zeros.

**Parameters**

- **num_embeddings** (*int*) – Total number of words/tokens (vocabulary size).
- **embedding_dim** (*int*) – Size of embedding vector.
- **embeddings_weight** (*torch.Tensor*) – Pretrained weights to initialize the embedding table with.
- **init_range** (*List[int]*) – Range of uniform distribution to initialize the weights with if *embeddings_weight* is None.

- **unk_token_idx** (*int*) – Index of UNK token in the word vocabulary.
- **mlp_layer_dims** (*List[int]*) – List of layer dimensions (if any) to add on top of the embedding lookup.

**forward**(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**freeze**()

**classmethod from_config**(*config: pytext.config.field_config.WordFeatConfig, metadata: Optional[pytext.fields.field.FieldMeta] = None, tensorizer: Optional[pytext.data.tensorizers.Tensorizer] = None, init_from_saved_state: Optional[bool] = False*)

Factory method to construct an instance of WordEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*WordFeatConfig*) – Configuration object specifying all the
- **of WordEmbedding.** (*parameters*) –
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of WordEmbedding.

**Return type** type

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

**class** pytext.models.embeddings.**DictEmbedding**(*num_embeddings: int, embed_dim: int, pooling_type: pytext.config.module_config.PoolingType, pad_index: int = 1, unk_index: int = 0, mobile: bool = False*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for dictionary feature embeddings for tokens. Dictionary features are also known as gazetteer features. These are per token discrete features that the module learns embeddings for. Example: For the utterance *Order coffee from Starbucks*, the dictionary features could be

```
[
    {"tokenIdx": 1, "features": {"drink/beverage": 0.8, "music/song": 0.2}},
    {"tokenIdx": 3, "features": {"store/coffee_shop": 1.0}}
]
```

:: Thus, for a given token there can be more than one dictionary features each of which has a confidence score. The final embedding for a token is the weighted average of the dictionary embeddings followed by a pooling operation such that the module produces an embedding vector per token.

**Parameters**

- **num_embeddings** (*int*) – Total number of dictionary features (vocabulary size).

- **embed_dim** (*int*) – Size of embedding vector.
- **pooling_type** (*PoolingType*) – Type of pooling for combining the dictionary feature embeddings.

**pooling_type**
Type of pooling for combining the dictionary feature embeddings.

**Type** PoolingType

**find_and_replace**(*tensor: torch.Tensor*, *find_val: int*, *replace_val: int*) → torch.Tensor
*torch.where* is not supported for mobile ONNX, this hack allows a mobile exported version of *torch.where* which is computationally more expensive

**forward**(*feats: torch.Tensor*, *weights: torch.Tensor*, *lengths: torch.Tensor*) → torch.Tensor
Given a batch of sentences such containing dictionary feature ids per token, produce token embedding vectors for each sentence in the batch.

**Parameters**

- **feats** (*torch.Tensor*) – Batch of sentences with dictionary feature ids. shape: [bsz, seq_len * max_feat_per_token]
- **weights** (*torch.Tensor*) – Batch of sentences with dictionary feature weights for the dictionary features. shape: [bsz, seq_len * max_feat_per_token]
- **lengths** (*torch.Tensor*) – Batch of sentences with the number of dictionary features per token. shape: [bsz, seq_len]

**Returns** Embedded batch of sentences. Dimension: batch size X maximum sentence length, token embedding size. Token embedding size = *embed_dim* passed to the constructor.

**Return type** torch.Tensor

**classmethod from_config**(*config: pytext.config.field_config.DictFeatConfig*, *metadata: Optional[pytext.fields.field.FieldMeta] = None*, *labels: Optional[pytext.data.utils.Vocabulary] = None*, *tensorizer: Optional[pytext.data.tensorizers.Tensorizer] = None*)
Factory method to construct an instance of DictEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*DictFeatConfig*) – Configuration object specifying all the
- **of DictEmbedding.** (*parameters*) –
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of DictEmbedding.

**Return type** type

**class** pytext.models.embeddings.**CharacterEmbedding**(*num_embeddings: int, embed_dim: int, out_channels: int, kernel_sizes: List[int], highway_layers: int, projection_dim: Optional[int], *args, **kwargs*)
Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for character aware CNN embeddings for tokens. It uses convolution followed by max-pooling over character embeddings to obtain an embedding vector for each token.

Implementation is loosely based on https://arxiv.org/abs/1508.06615.

**Parameters**

- **num_embeddings** (*int*) – Total number of characters (vocabulary size).
- **embed_dim** (*int*) – Size of character embeddings to be passed to convolutions.
- **out_channels** (*int*) – Number of output channels.
- **kernel_sizes** (*List[int]*) – Dimension of input Tensor passed to MLP.
- **highway_layers** (*int*) – Number of highway layers applied to pooled output.
- **projection_dim** (*int*) – If specified, size of output embedding for token, via a linear projection from convolution output.

**char_embed**

Character embedding table.

**Type** nn.Embedding

**convs**

Convolution layers that operate on character

**Type** nn.ModuleList

**embeddings.**

**highway_layers**

Highway layers on top of convolution output.

**Type** nn.Module

**projection**

Final linear layer to token embedding.

**Type** nn.Module

**embedding_dim**

Dimension of the final token embedding produced.

**Type** int

**forward**(*chars: torch.Tensor*) → torch.Tensor

Given a batch of sentences such that tokens are broken into character ids, produce token embedding vectors for each sentence in the batch.

**Parameters**

- **chars** (*torch.Tensor*) – Batch of sentences where each token is broken
- **characters.** (*into*) –
- **Dimension** – batch size X maximum sentence length X maximum word length

**Returns** Embedded batch of sentences. Dimension: batch size X maximum sentence length, token embedding size. Token embedding size = *out_channels * len(self.convs))*

**Return type** torch.Tensor

**classmethod from_config**(*config: pytext.config.field_config.CharFeatConfig, metadata: Optional[pytext.fields.field.FieldMeta] = None, vocab_size: Optional[int] = None*)

Factory method to construct an instance of CharacterEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*CharFeatConfig*) – Configuration object specifying all the parameters of CharacterEmbedding.
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of CharacterEmbedding.

**Return type** type

**class** pytext.models.embeddings.**ContextualTokenEmbedding**(*embed_dim: int, downsample_dim: Optional[int] = None*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for providing token embeddings from a pretrained model.

**forward**(*embedding: torch.Tensor*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.config.field_config.ContextualTokenEmbeddingConfig, *args, **kwargs*)

**class** pytext.models.embeddings.**WordSeqEmbedding**(*lstm_config: pytext.models.representations.bilstm.BiLSTM.Config, num_embeddings: int, word_embed_dim: int = 300, embeddings_weight: Optional[torch.Tensor] = None, init_range: Optional[List[int]] = None, init_std: Optional[float] = None, unk_token_idx: int = 0, padding_idx: Optional[int] = None, vocab: Optional[List[str]] = None*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

An embedding module represents a sequence of sentences

**Parameters**

- **lstm_config** (*BiLSTM.Config*) – config of the lstm layer
- **num_embeddings** (*int*) – Total number of words/tokens (vocabulary size).
- **embedding_dim** (*int*) – Size of embedding vector.
- **embeddings_weight** (*torch.Tensor*) – Pretrained weights to initialize the embedding table with.
- **init_range** (*List[int]*) – Range of uniform distribution to initialize the weights with if *embeddings_weight* is None.
- **unk_token_idx** (*int*) – Index of UNK token in the word vocabulary.

**forward**(*seq_token_idx, seq_token_count*)

**Parameters**

- **seq_token_idx** – shape [batch_size * max_seq_len * max_token_count]
- **seq_token_count** – shape [batch_size * max_seq_len]

**Returns** shape (batch_size * max_seq_len * output_dim)

**Return type** embedding

**freeze**()

**classmethod from_config**(*config: pytext.models.embeddings.word_seq_embedding.WordSeqEmbedding.Config, tensorizer: pytext.data.tensorizers.Tensorizer = None, init_from_saved_state: Optional[bool] = False*)

Factory method to construct an instance of WordEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*WordSeqEmbedding.Config*) – Configuration object specifying all the
- **of WordEmbedding.** (*parameters*) –

**Returns** An instance of WordSeqEmbedding.

**Return type** type

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

**class** pytext.models.embeddings.**MLPEmbedding**(*embedding_dim: int = 300, embeddings_weight: Optional[torch.Tensor] = None, init_range: Optional[List[int]] = None, init_std: Optional[float] = None, mlp_layer_dims: List[int] = ()*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

An MLP embedding wrapper module around *torch.nn.Embedding* to add transformations for float tensors.

**Parameters**

- **num_embeddings** (*int*) – Total number of words/tokens (vocabulary size).
- **embedding_dim** (*int*) – Size of embedding vector.
- **embeddings_weight** (*torch.Tensor*) – Pretrained weights to initialize the embedding table with.
- **init_range** (*List[int]*) – Range of uniform distribution to initialize the weights with if *embeddings_weight* is None.
- **mlp_layer_dims** (*List[int]*) – List of layer dimensions (if any) to add on top of the embedding lookup.

**forward**(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.config.field_config.MLPFeatConfig, metadata: Optional[pytext.fields.field.FieldMeta] = None, tensorizer: Optional[pytext.data.tensorizers.Tensorizer] = None, init_from_saved_state: Optional[bool] = False*)

Factory method to construct an instance of MLPEmbedding from the module's config object and the field's metadata object.

**Parameters**

- **config** (*MLPFeatConfig*) – Configuration object specifying all the
- **of MLPEmbedding.** (*parameters*) –
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

**Returns** An instance of MLPEmbedding.

**Return type** type

**visualize**(*summary_writer: <Mock name='mock.SummaryWriter' id='140050279154320'>*)

Overridden in sub classes to implement Tensorboard visualization of embedding space

## pytext.models.ensembles package

### Submodules

### pytext.models.ensembles.bagging_doc_ensemble module

**class** pytext.models.ensembles.bagging_doc_ensemble.**BaggingDocEnsembleModel**(*config: pytext.models.ensembles.mod- els: List[pytext.models.mod *args, **kwargs*)

Bases: *pytext.models.ensembles.ensemble.EnsembleModel*

Ensemble class that uses bagging for ensembling document classification models.

**forward**(**args, **kwargs*) → torch.Tensor

Call *forward()* method of each document classification sub-model by passing all arguments and named arguments to the sub-models, collect the logits from them and average their values.

**Returns** Logits from the ensemble.

**Return type** torch.Tensor

### pytext.models.ensembles.bagging_intent_slot_ensemble module

**class** pytext.models.ensembles.bagging_intent_slot_ensemble.**BaggingIntentSlotEnsembleModel**(*c*

Bases: *pytext.models.ensembles.ensemble.EnsembleModel*

Ensemble class that uses bagging for ensembling intent-slot models.

**Parameters**

- **config** (*Config*) – Configuration object specifying all the parameters of BaggingIntentSlotEnsemble.
- **models** (*List[Model]*) – List of intent-slot model objects.

**use_crf**

Whether to use CRF for word tagging task.

**Type** bool

**output_layer**

Output layer of intent-slot model responsible for computing loss and predictions.

**Type** IntentSlotOutputLayer

**forward**(**args*, ***kwargs*) → Tuple[torch.Tensor, torch.Tensor]

Call *forward()* method of each intent-slot sub-model by passing all arguments and named arguments to the sub-models, collect the logits from them and average their values.

**Returns** Logits from the ensemble.

**Return type** torch.Tensor

**load_state_dict**(*state_dict: Dict[str, torch.Tensor], strict: bool = True*)

Copies parameters and buffers from `state_dict` into this module and its descendants. If `strict` is `True`, then the keys of `state_dict` must exactly match the keys returned by this module's `state_dict()` function.

**Parameters**

- **state_dict** (*dict*) – a dict containing parameters and persistent buffers.
- **strict** (*bool, optional*) – whether to strictly enforce that the keys in `state_dict` match the keys returned by this module's `state_dict()` function. Default: `True`

**Returns**

- **missing_keys** is a list of str containing the missing keys
- **unexpected_keys** is a list of str containing the unexpected keys

**Return type** `NamedTuple` with `missing_keys` and `unexpected_keys` fields

**merge_sub_models**() → None

Merges all sub-models' transition matrices when using CRF. Otherwise does nothing.

**torchscriptify**(*tensorizers*, *traced_model*)

### pytext.models.ensembles.ensemble module

**class** pytext.models.ensembles.ensemble.**EnsembleModel**(*config: pytext.models.ensembles.ensemble.EnsembleModel.Config models: List[pytext.models.model.Model], *args, **kwargs*)

Bases: *pytext.models.model.Model*

Base class for ensemble models.

**Parameters**

- **config** (*Config*) – Configuration object specifying all the parameters of Ensemble.
- **models** (*List[Model]*) – List of sub-model objects.

**output_layer**

Responsible for computing loss and predictions.

**Type** OutputLayerBase

**models**

ModuleList container for sub-model objects.

**Type** nn.ModuleList]

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**forward**(**args*, ***kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.ensembles.ensemble.EnsembleModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], *args, **kwargs*)

Factory method to construct an instance of Ensemble or one its derived classes from the module's config object and tensorizers It creates sub-models in the ensemble using the sub-model's configuration.

**Parameters**

- **config** (*Config*) – Configuration object specifying all the parameters of Ensemble.
- **tensorizers** (*Dict[str, Tensorizer]*) – Tensorizer specifying all the parameters of the input features to the model.

**Returns** An instance of Ensemble.

**Return type** type

**get_export_input_names**(*tensorizers*)

**get_export_output_names**(*tensorizers*)

**merge_sub_models**()

**save_modules**(*base_path: str = ", suffix: str = "*) → None
Saves the modules of all sub_models in the *Ensemble*.

**Parameters**

- **base_path** (*str*) – Path of base directory. Defaults to "".
- **suffix** (*str*) – Suffix to add to the file name to save. Defaults to "".

**torchscriptify**(*tensorizers*, *traced_model*)

**vocab_to_export**(*tensorizers*)

### Module contents

**class** pytext.models.ensembles.**BaggingDocEnsembleModel**(*config: pytext.models.ensembles.ensemble.EnsembleModel.Confi models: List[pytext.models.model.Model], *args, **kwargs*)

Bases: *pytext.models.ensembles.ensemble.EnsembleModel*

Ensemble class that uses bagging for ensembling document classification models.

**forward**(**args, **kwargs*) → torch.Tensor
Call *forward()* method of each document classification sub-model by passing all arguments and named arguments to the sub-models, collect the logits from them and average their values.

**Returns** Logits from the ensemble.

**Return type** torch.Tensor

**class** pytext.models.ensembles.**BaggingIntentSlotEnsembleModel**(*config: pytext.models.ensembles.bagging_intent_slot_ models: List[pytext.models.model.Model], *args, **kwargs*)

Bases: *pytext.models.ensembles.ensemble.EnsembleModel*

Ensemble class that uses bagging for ensembling intent-slot models.

**Parameters**

- **config** (*Config*) – Configuration object specifying all the parameters of BaggingIntentSlotEnsemble.
- **models** (*List[Model]*) – List of intent-slot model objects.

**use_crf**
Whether to use CRF for word tagging task.

**Type** bool

**output_layer**
Output layer of intent-slot model responsible for computing loss and predictions.

**Type** IntentSlotOutputLayer

**forward**(**args, **kwargs*) → Tuple[torch.Tensor, torch.Tensor]
Call *forward()* method of each intent-slot sub-model by passing all arguments and named arguments to the sub-models, collect the logits from them and average their values.

**Returns** Logits from the ensemble.

**Return type** torch.Tensor

**load_state_dict**(*state_dict: Dict[str, torch.Tensor], strict: bool = True*)

Copies parameters and buffers from `state_dict` into this module and its descendants. If `strict` is `True`, then the keys of `state_dict` must exactly match the keys returned by this module's `state_dict()` function.

**Parameters**

- **state_dict** (*dict*) – a dict containing parameters and persistent buffers.
- **strict** (*bool, optional*) – whether to strictly enforce that the keys in `state_dict` match the keys returned by this module's `state_dict()` function. Default: `True`

**Returns**

- **missing_keys** is a list of str containing the missing keys
- **unexpected_keys** is a list of str containing the unexpected keys

**Return type** `NamedTuple` with `missing_keys` and `unexpected_keys` fields

**merge_sub_models**() → None

Merges all sub-models' transition matrices when using CRF. Otherwise does nothing.

**torchscriptify**(*tensorizers, traced_model*)

**class** pytext.models.ensembles.**EnsembleModel**(*config: pytext.models.ensembles.ensemble.EnsembleModel.Config, models: List[pytext.models.model.Model], *args, **kwargs*)

Bases: *pytext.models.model.Model*

Base class for ensemble models.

**Parameters**

- **config** (*Config*) – Configuration object specifying all the parameters of Ensemble.
- **models** (*List[Model]*) – List of sub-model objects.

**output_layer**

Responsible for computing loss and predictions.

**Type** OutputLayerBase

**models**

ModuleList container for sub-model objects.

**Type** nn.ModuleList]

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**forward**(**args, **kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.models.ensembles.ensemble.EnsembleModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], *args, **kwargs*)

Factory method to construct an instance of Ensemble or one its derived classes from the module's config object and tensorizers It creates sub-models in the ensemble using the sub-model's configuration.

**Parameters**

- **config** (*Config*) – Configuration object specifying all the parameters of Ensemble.
- **tensorizers** (*Dict[str, Tensorizer]*) – Tensorizer specifying all the parameters of the input features to the model.

**Returns** An instance of Ensemble.

**Return type** type

**get_export_input_names**(*tensorizers*)

**get_export_output_names**(*tensorizers*)

**merge_sub_models**()

**save_modules**(*base_path: str = ", suffix: str = "*) → None

Saves the modules of all sub_models in the *Ensemble*.

**Parameters**

- **base_path** (*str*) – Path of base directory. Defaults to "".
- **suffix** (*str*) – Suffix to add to the file name to save. Defaults to "".

**torchscriptify**(*tensorizers, traced_model*)

**vocab_to_export**(*tensorizers*)

## pytext.models.language_models package

### Submodules

### pytext.models.language_models.lmlstm module

**class** pytext.models.language_models.lmlstm.**LMLSTM**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase = <pytext.config.field_config.WordFeatConfig object>, representation: pytext.models.representations.representation_base.Representat = <pytext.models.representations.bilstm.BiLSTM.Config object>, decoder: pytext.models.decoders.decoder_base.DecoderBase = <pytext.models.decoders.mlp_decoder.MLPDecoder.Config object>, output_layer: pytext.models.output_layers.output_layer_base.OutputLayerBa = <pytext.models.output_layers.lm_output_layer.LMOutputLayer.C object>, stateful: bool = False, exporter: object = <class 'pytext.exporters.exporter.ModelExporter'>*)

Bases: `pytext.models.model.BaseModel`

*LMLSTM* implements a word-level language model that uses LSTMs to represent the document.

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**caffe2_export**(*tensorizers*, *tensor_dict*, *path*, *export_onnx_path=None*)

**classmethod checkTokenConfig**(*tokens: Optional[pytext.data.tensorizers.TokenTensorizer.Config]*)

**cpu**()

Moves all model parameters and buffers to the CPU.

**Returns** self

**Return type** Module

**forward**(*tokens: torch.Tensor*, *seq_len: torch.Tensor*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.language_models.lmlstm.LMLSTM.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**get_export_input_names**(*tensorizers*)

**get_export_output_names**(*tensorizers*)

**get_num_examples_from_batch**(*batch*)

**init_hidden**(*bsz: int*) → Tuple[torch.Tensor, torch.Tensor]

Initialize the hidden states of the LSTM if the language model is stateful.

**Parameters** **bsz** (*int*) – Batch size.

**Returns** Initialized hidden state and cell state of the LSTM.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**vocab_to_export** (*tensorizers*)

pytext.models.language_models.lmlstm.**repackage_hidden**(*hidden: Union[torch.Tensor, Tuple[torch.Tensor, ...]]*) → Union[torch.Tensor, Tuple[torch.Tensor, ...]]

Wraps hidden states in new Tensors, to detach them from their history.

**Parameters** **hidden** (*Union[torch.Tensor, Tuple[torch.Tensor, ..]]*) – Tensor or a tuple of tensors to repackage.

**Returns** Repackaged output

**Return type** Union[torch.Tensor, Tuple[torch.Tensor, ..]]

### Module contents

### pytext.models.output_layers package

### Submodules

### pytext.models.output_layers.distance_output_layer module

**class** pytext.models.output_layers.distance_output_layer.**DenseRetrievalOutputLayer**(*target_name Optional[List[s = None, loss_fn: Union[pytex pytext.loss.loss pytext.loss.loss pytext.loss.loss pytext.loss.loss = None, score_thresh bool = 0.9, score_type: pytext.models.c = <OutputScore.nor 2>*)

Bases: *pytext.models.output_layers.distance_output_layer.PairwiseCosineDistanceOutputLayer*

**get_loss**(*logits: Tuple[torch.Tensor, torch.Tensor], targets: torch.Tensor, context: Optional[Dict[str, Any]] = None, reduce: bool = True*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logits: torch.Tensor, targets: torch.Tensor, *args, **kwargs*)

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.

- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by `LMOutputLayer`. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.distance_output_layer.**OutputScore**

Bases: `enum.IntEnum`

An enumeration.

**norm_cosine = 2**

**raw_cosine = 1**

**sigmoid_cosine = 3**

**class** pytext.models.output_layers.distance_output_layer.**PairwiseCosineDistanceOutputLayer**(*t*

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

**classmethod from_config**(*config*, *metadata: Optional[pytext.fields.field.FieldMeta] = None*, *labels: Optional[pytext.data.utils.Vocabulary] = None*)

**get_loss**(*logits: Tuple[torch.Tensor, torch.Tensor]*, *targets: torch.Tensor*, *context: Optional[Dict[str, Any]] = None*, *reduce: bool = True*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.

- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logits: torch.Tensor*, *targets: torch.Tensor*, **args*, ***kwargs*)

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by LMOutputLayer. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

pytext.models.output_layers.distance_output_layer.**get_norm_cosine_scores**(*cosine_sim_scores*)

pytext.models.output_layers.distance_output_layer.**get_sigmoid_scores**(*cosine_sim_scores*)

### pytext.models.output_layers.doc_classification_output_layer module

**class** pytext.models.output_layers.doc_classification_output_layer.**BinaryClassificationOutpu**

Bases: *pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer*

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>*, *init_net: caffe2.python.core.Net*, *predict_net: caffe2.python.core.Net*, *model_out: torch.Tensor*, *output_name: str*) → List[caffe2.python.core.BlobReference]

See *OutputLayerBase.export_to_caffe2()*.

**get_pred**(*logit*, **args*, ***kwargs*)

See *OutputLayerBase.get_pred()*.

**torchscript_predictions**()

**class** pytext.models.output_layers.doc_classification_output_layer.**ClassificationOutputLayer**

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for document classification models. It supports *CrossEntropyLoss* and *BinaryCrossEntropyLoss* per document.

**Parameters** **loss_fn** (*Union[CrossEntropyLoss, BinaryCrossEntropyLoss]*) – The loss function to use for computing loss. Defaults to None.

**loss_fn**
The loss function to use for computing loss.

**classmethod from_config**(*config: pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLay metadata: Optional[pytext.fields.field.FieldMeta] = None, labels: Optional[pytext.data.utils.Vocabulary] = None*)

**get_pred**(*logit, *args, **kwargs*)
Compute and return prediction and scores from the model.

Prediction is computed using argmax over the document label/target space.

Scores are sigmoid or softmax scores over the model logits depending on the loss component being used.

**Parameters** **logit** (*torch.Tensor*) – Logits returned *DocModel*.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.doc_classification_output_layer.**ClassificationScores**(*class score score*

Bases: torch.jit._script.ScriptModule

**class** pytext.models.output_layers.doc_classification_output_layer.**MultiLabelOutputLayer**(*targ Op- tion = Nor loss Op- tion = Nor *ar **k*

Bases: *pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer*

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

See *OutputLayerBase.export_to_caffe2()*.

**get_pred**(*logit, *args, **kwargs*)

See *OutputLayerBase.get_pred()*.

**torchscript_predictions**()

**class** pytext.models.output_layers.doc_classification_output_layer.**MulticlassOutputLayer**(*targ Op- tion = Nor loss Op- tion = Nor *ar **k*

Bases: *pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer*

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

See *OutputLayerBase.export_to_caffe2()*.

**get_pred**(*logit, *args, **kwargs*)

See *OutputLayerBase.get_pred()*.

**torchscript_predictions**()

## pytext.models.output_layers.doc_regression_output_layer module

**class** pytext.models.output_layers.doc_regression_output_layer.**PairwiseCosineRegressionOutpu**

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for pair (two-tower) regression models. Accepts two embedding tensors, and computes cosine similarity. The loss is between regression label and cosine similarity.

**classmethod from_config**(*config: pytext.models.output_layers.doc_regression_output_layer.PairwiseCosineRegressionO **kwargs*)

**get_loss**(*logits: Tuple[torch.Tensor, torch.Tensor], target: torch.Tensor, context: Optional[Dict[str, Any]] = None, reduce: bool = True*) → torch.Tensor

**Parameters**

- **logits** (*Tuple[torch.Tensor, torch.Tensor]*) – Logits returned from pairwise model

- **target** (*torch.Tensor*) – Float target to compute loss against
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary from data handler
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logits*, **args*, ***kwargs*)

**Parameters** **logits** (*Tuple[torch.Tensor, torch.Tensor]*) – Logits returned from pairwise model

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.doc_regression_output_layer.**RegressionOutputLayer**(*loss_fn: pytext.loss.l squash_to bool = False*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for doc regression models. Currently only supports Mean Squared Error loss.

**Parameters**

- **loss** (*MSELoss*) – config for MSE loss
- **squash_to_unit_range** (*bool*) – whether to clamp the output to the range [0, 1], via a sigmoid.

**classmethod from_config**(*config: pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer.Con*

**get_loss**(*logit: torch.Tensor*, *target: torch.Tensor*, *context: Optional[Dict[str, Any]] = None*, *reduce: bool = True*) → torch.Tensor

Compute regression loss from logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logit*, **args*, ***kwargs*)

Compute predictions and scores from the model (unlike in classification, where prediction = "most likely class" and scores = "log probs", here these are the same values). If *squash_to_unit_range* is True, fit prediction to [0, 1] via a sigmoid.

**Parameters** **logit** (*torch.Tensor*) – Logits returned from the model.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.doc_regression_output_layer.**RegressionScores**(*squash_to_unit_r bool*)

Bases: torch.jit._script.ScriptModule

## pytext.models.output_layers.intent_slot_output_layer module

**class** pytext.models.output_layers.intent_slot_output_layer.**IntentSlotOutputLayer**(*doc_output: pytext.models.ou word_output: pytext.models.ou*

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for joint intent classification and slot-filling models. Intent classification is a document classification problem and slot filling is a word tagging problem. Thus terms these can be used interchangeably in the documentation.

**Parameters**

- **doc_output** (*ClassificationOutputLayer*) – Output layer for intent classification task. See *ClassificationOutputLayer* for details.
- **word_output** (*WordTaggingOutputLayer*) – Output layer for slot filling task. See *WordTaggingOutputLayer* for details.

**doc_output**

Output layer for intent classification task.

**Type** type

**word_output**

Output layer for slot filling task.

**Type** type

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: List[torch.Tensor], doc_out_name: str, word_out_name: str*) → List[caffe2.python.core.BlobReference]

Exports the intent slot output layer to Caffe2. See *OutputLayerBase.export_to_caffe2()* for details.

**classmethod from_config**(*config: pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer.Config, doc_labels: pytext.data.utils.Vocabulary, word_labels: pytext.data.utils.Vocabulary*)

**get_loss**(*logits: Tuple[torch.Tensor, torch.Tensor], targets: Tuple[torch.Tensor, torch.Tensor], context: Dict[str, Any] = None, *args, **kwargs*) → torch.Tensor

Compute and return the averaged intent and slot-filling loss.

**Parameters**

- **logit** (*Tuple[torch.Tensor, torch.Tensor]*) – Logits returned by JointModel. It is a tuple containing logits for intent classification and slot filling.

- **targets** (*Tuple[torch.Tensor, torch.Tensor]*) – Tuple of target Tensors containing true document label/target and true word labels/targets.
- **context** (*Dict[str, Any]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.

**Returns** Averaged intent and slot loss.

**Return type** torch.Tensor

**get_pred**(*logits: Tuple[torch.Tensor, torch.Tensor], targets: Optional[torch.Tensor] = None, context: Optional[Dict[str, Any]] = None*) → Tuple[torch.Tensor, torch.Tensor]

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*Tuple[torch.Tensor, torch.Tensor]*) – Logits returned by `JointModel`. It's tuple containing logits for intent classification and slot filling.
- **targets** (*Optional[torch.Tensor]*) – Not applicable. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.intent_slot_output_layer.**IntentSlotScores**(*doc_scores: torch.jit._script.Script word_scores: torch.jit._script.Script*

Bases: `torch.nn.modules.module.Module`

**forward**(*logits: Tuple[torch.Tensor, torch.Tensor], context: Dict[str, torch.Tensor]*) → Tuple[List[Dict[str, float]], List[List[Dict[str, float]]]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.output_layers.lm_output_layer module

**class** pytext.models.output_layers.lm_output_layer.**LMOutputLayer**(*target_names: List[str], loss_fn: pytext.loss.loss.Loss = None, config=None, pad_token_idx=-100*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for language models. It supports *CrossEntropyLoss* per word.

**Parameters** **loss_fn** (*CrossEntropyLoss*) – Cross-entropy loss component. Defaults to None.

**loss_fn**
Cross-entropy loss component for computing loss.

**static calculate_perplexity**(*sequence_loss: torch.Tensor*) → torch.Tensor

**classmethod from_config**(*config: pytext.models.output_layers.lm_output_layer.LMOutputLayer.Config*, *metadata: Optional[pytext.fields.field.FieldMeta] = None*, *labels: Optional[pytext.data.utils.Vocabulary] = None*)

**get_loss**(*logit: torch.Tensor, target: torch.Tensor, context: Dict[str, Any], reduce=True*) → torch.Tensor
Compute word prediction loss by comparing prediction of each word in the sentence with the true word.

**Parameters**

- **logit** (*torch.Tensor*) – Logit returned by *LMLSTM*.
- **targets** (*torch.Tensor*) – Not applicable for language models.
- **context** (*Dict[str, Any]*) – Not applicable. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Word prediction loss.

**Return type** torch.Tensor

**get_pred**(*logits: torch.Tensor*, **args*, ***kwargs*) → Tuple[torch.Tensor, torch.Tensor]
Compute and return prediction and scores from the model. Prediction is computed using argmax over the word label/target space. Scores are softmax scores over the model logits.

**Parameters**

- **logits** (*torch.Tensor*) – Logits returned *LMLSTM*.
- **targets** (*torch.Tensor*) – True words.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

### pytext.models.output_layers.multi_label_classification_layer module

**class** pytext.models.output_layers.multi_label_classification_layer.**MultiLabelClassificatio**

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for multilabel sequence classification models.

**Parameters**

- **outputs** (*Dict[str, ClassificationOutputLayer]*) – Output for multilabels

- **label_weights** (*optional*) – Dict of label_names along with the
- **for label** (*weight*) –

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: List[torch.Tensor], out_name: str*) → List[caffe2.python.core.BlobReference]

Exports the multilabel output layer to Caffe2. See *OutputLayerBase.export_to_caffe2()* for details.

**classmethod from_config**(*config: pytext.models.output_layers.multi_label_classification_layer.MultiLabelClassification label_tensorizers: [typing.Dict[str, pytext.data.tensorizers.Tensorizer]]*)

**get_loss**(*logits, targets: List[torch.Tensor], context: Optional[Dict[str, Any]] = None, *args, **kwargs*) → torch.Tensor

Compute and return the averaged intent and slot-filling loss.

**Parameters**

- **logits** (*List[torch.Tensor]*) – Logits returned by `MultiLabelDecoder`. It's list containing logits for all label tasks here pTSR, autoUSM and EA.
- **targets** (*List[torch.Tensor]*) – Targets as computed by the true labels
- **context** (*Optional[torch.Tensor]*) – Not applicable. Defaults to None.

**Returns** Averaged Loss across all label losses.

**Return type** torch.Tensor

**get_pred**(*logits: List[torch.Tensor], targets: List[torch.Tensor], context: Optional[Dict[str, Any]] = None, *args, **kwargs*) → Tuple[torch.Tensor, torch.Tensor]

Compute and return prediction and scores from the model.

Prediction is computed using argmax over the document label/target space.

Scores are sigmoid or softmax scores over the model logits depending on the loss component being used.

**Parameters**

- **logits** (*List[torch.Tensor]*) – Logits returned by `MultiLabelDecoder`. It's list containing logits for all label tasks here pTSR, autoUSM and EA.
- **targets** (*List[torch.Tensor]*) – Targets as computed by the true labels
- **of logits corresponding the respective label.** (*ordering*) –
- **context** (*Optional[torch.Tensor]*) – Not applicable. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.multi_label_classification_layer.**MultiLabelClassificatio**

Bases: `torch.nn.modules.module.Module`

**forward**(*logits: List[torch.Tensor]*) → List[List[Dict[str, float]]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.output_layers.output_layer_base module

**class** pytext.models.output_layers.output_layer_base.**OutputLayerBase**(*target_names: Optional[List[str]] = None, loss_fn: Optional[pytext.loss.loss.Loss] = None, *args, **kwargs*)

Bases: *pytext.models.module.Module*

Base class for all output layers in PyText. The responsibilities of this layer are

1. Implement how loss is computed from logits and targets.
2. Implement how to get predictions from logits.
3. **Implement the Caffe2 operator for performing the above tasks. This is** used when PyText exports PyTorch model to Caffe2.

**Parameters** **loss_fn** (*type*) – The loss function object to use for computing loss. Defaults to None.

**loss_fn**

The loss function object to use for computing loss.

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

Exports the output layer to Caffe2 by manually adding the necessary operators to the init_net and predict_net and, returns the list of external output blobs to be added to the model. By default this does nothing, so any sub-class must override this method (if necessary).

To learn about Caffe2 computation graphs and why we need two networks, *init_net* and *predict_net/exec_net* read https://caffe2.ai/docs/intro-tutorial#null__nets-and-operators.

**Parameters**

- **workspace** (*core.workspace*) – Caffe2 *workspace* to use for adding the operator. See https://caffe2.ai/docs/workspace.html to learn about Caffe2 workspace.
- **init_net** (*core.Net*) – Caffe2 *init_net* to add the operator to.
- **predict_net** (*core.Net*) – Caffe2 *predict_net* to add the operator to.
- **model_out** (*torch.Tensor*) – Output logit Tensor from the model to .
- **output_name** (*str*) – Name of *model_out* to use in Caffe2 net.

- **label_names** (*List[str]*) – List of names of the targets/labels to expose from the Caffe2 net.

**Returns**

**List of output blobs that the *output_layer*** generates.

**Return type** List[core.BlobReference]

**get_loss**(*logit: torch.Tensor*, *target: torch.Tensor*, *context: Optional[Dict[str, Any]] = None*, *reduce: bool = True*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logit: torch.Tensor*, *targets: Optional[torch.Tensor] = None*, *context: Optional[Dict[str, Any]] = None*) → Tuple[torch.Tensor, torch.Tensor]

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by `LMOutputLayer`. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

### pytext.models.output_layers.pairwise_ranking_output_layer module

**class** pytext.models.output_layers.pairwise_ranking_output_layer.**PairwiseRankingOutputLayer**

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

**classmethod from_config**(*config*)

**get_pred**(*logit*, *targets*, *context*)

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by LMOutputLayer. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

### pytext.models.output_layers.squad_output_layer module

**class** pytext.models.output_layers.squad_output_layer.**SquadOutputLayer**(*loss_fn: pytext.loss.loss.Loss, ignore_impossible: bool = True, pos_loss_weight: float = 0.5, has_answer_loss_weight: float = 0.5, has_answer_labels: Iterable[str] = ('False', 'True'), false_label: str = 'False', max_answer_len: int = 30, hard_weight: float = 0.0, is_kd: bool = False*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

**classmethod from_config**(*config*, *metadata: Optional[pytext.fields.field.FieldMeta] = None*, *labels: Optional[Iterable[str]] = None*, *is_kd: bool = False*)

**get_loss**(*logits: Tuple[torch.Tensor, ...], targets: Tuple[torch.Tensor, ...], contexts: Optional[Dict[str, Any]] = None, *args, **kwargs*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.=

**Returns** Model loss.

**Return type** torch.Tensor

**get_position_preds**(*start_pos_logits: torch.Tensor, end_pos_logits: torch.Tensor, max_span_length: int*)

**get_pred**(*logits: torch.Tensor, targets: torch.Tensor, contexts: Dict[str, List[Any]]*) → Tuple[Tuple[torch.Tensor, torch.Tensor], Tuple[torch.Tensor, torch.Tensor]]

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by `LMOutputLayer`. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

### pytext.models.output_layers.utils module

**class** pytext.models.output_layers.utils.**OutputLayerUtils**

Bases: `object`

**static gen_additional_blobs**(*predict_net: caffe2.python.core.Net, probability_out, model_out: torch.Tensor, output_name: str, label_names: List[str]*) → List[caffe2.python.core.BlobReference]

Utility method to generate additional blobs for human readable result for models that use explicit labels.

pytext.models.output_layers.utils.**query_word_reprs**(*encoder_repr: torch.Tensor, token_indices: torch.Tensor*) → torch.Tensor

Given an encoder_repr (B x T_1 x H) and token_indices (B x T_2) where T_2 <= T_1, collect embeddings from encoder_repr pertaining to indices in token_indices. In the context of fine-tuning pre-trained encoders on sequence labeling, our goal is to build token-level representations as opposed to subword-level represenatations for alignment with other token-level cues, such as dictionary features. Currently, a token representation is built by taking its first subword representation.

### pytext.models.output_layers.word_tagging_output_layer module

**class** pytext.models.output_layers.word_tagging_output_layer.**CRFOutputLayer**(*num_tags, labels: pytext.data.utils.Vocabula *args*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for word tagging models that use Conditional Random Field.

**Parameters** **num_tags** (*int*) – Total number of possible word tags.

**num_tags**

Total number of possible word tags.

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

Exports the CRF output layer to Caffe2. See *OutputLayerBase.export_to_caffe2()* for details.

**classmethod from_config**(*config: pytext.config.component.ComponentMeta.__new__.<locals>.Config, labels: pytext.data.utils.Vocabulary*)

**get_loss**(*logit: torch.Tensor, target: torch.Tensor, context: Dict[str, Any], reduce=True*)

Compute word tagging loss by using CRF.

**Parameters**

- **logit** (*torch.Tensor*) – Logit returned by `WordTaggingModel`.
- **targets** (*torch.Tensor*) – True document label/target.
- **context** (*Dict[str, Any]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**get_pred**(*logit: torch.Tensor, target: Optional[torch.Tensor] = None, context: Optional[Dict[str, Any]] = None*)

Compute and return prediction and scores from the model.

Prediction is computed using CRF decoding.

Scores are softmax scores over the model logits where the logits are computed by rearranging the word logits such that decoded word tag has the highest valued logits. This is done because with CRF, the highest valued word tag for a given may not be part of the overall set of word tags. In order for argmax to work, we rearrange the logit values.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned `WordTaggingModel`.
- **target** (*torch.Tensor*) – Not applicable. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.word_tagging_output_layer.**CRFWordTaggingScores**(*classes: List[str], crf*)

Bases: *pytext.models.output_layers.word_tagging_output_layer.WordTaggingScores*

**forward**(*logits: torch.Tensor, context: Dict[str, torch.Tensor]*) → List[List[Dict[str, float]]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.output_layers.word_tagging_output_layer.**WordTaggingOutputLayer**(*target_nam Optional[List[ = None, loss_fn: Optional[pyte = None, *args, **kwargs*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for word tagging models. It supports *CrossEntropyLoss* per word.

**Parameters** **loss_fn** (*CrossEntropyLoss*) – Cross-entropy loss component. Defaults to None.

**loss_fn**

Cross-entropy loss component.

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

Exports the word tagging output layer to Caffe2.

**classmethod from_config**(*config: pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer.Co labels: pytext.data.utils.Vocabulary*)

**get_loss**(*logit: torch.Tensor, target: Union[torch.Tensor, Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], context: Dict[str, Any], reduce: bool = True*) → torch.Tensor

Compute word tagging loss by comparing prediction of each word in the sentence with its true label/target.

**Parameters**

- **logit** (*torch.Tensor*) – Logit returned by *WordTaggingModel*.

- **targets** (*torch.Tensor*) – True document label/target.
- **context** (*Dict[str, Any]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Word tagging loss for all words in the sentence.

**Return type** torch.Tensor

**get_pred**(*logit: torch.Tensor*, **args*, ***kwargs*) → Tuple[torch.Tensor, torch.Tensor]

Compute and return prediction and scores from the model. Prediction is computed using argmax over the word label/target space. Scores are softmax scores over the model logits.

**Parameters** **logit** (*torch.Tensor*) – Logits returned *WordTaggingModel*.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.word_tagging_output_layer.**WordTaggingScores**(*classes*)

Bases: torch.nn.modules.module.Module

**forward**(*logits: torch.Tensor*, *context: Optional[Dict[str, torch.Tensor]] = None*) → List[List[Dict[str, float]]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

### Module contents

**class** pytext.models.output_layers.**OutputLayerBase**(*target_names: Optional[List[str]] = None*, *loss_fn: Optional[pytext.loss.loss.Loss] = None*, **args*, ***kwargs*)

Bases: *pytext.models.module.Module*

Base class for all output layers in PyText. The responsibilities of this layer are

1. Implement how loss is computed from logits and targets.
2. Implement how to get predictions from logits.
3. **Implement the Caffe2 operator for performing the above tasks. This is** used when PyText exports PyTorch model to Caffe2.

**Parameters** **loss_fn** (*type*) – The loss function object to use for computing loss. Defaults to None.

**loss_fn**

The loss function object to use for computing loss.

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

Exports the output layer to Caffe2 by manually adding the necessary operators to the init_net and predict_net and, returns the list of external output blobs to be added to the model. By default this does nothing, so any sub-class must override this method (if necessary).

To learn about Caffe2 computation graphs and why we need two networks, *init_net* and *predict_net/exec_net* read https://caffe2.ai/docs/intro-tutorial#null__nets-and-operators.

**Parameters**

- **workspace** (`core.workspace`) – Caffe2 *workspace* to use for adding the operator. See https://caffe2.ai/docs/workspace.html to learn about Caffe2 workspace.
- **init_net** (`core.Net`) – Caffe2 *init_net* to add the operator to.
- **predict_net** (`core.Net`) – Caffe2 *predict_net* to add the operator to.
- **model_out** (`torch.Tensor`) – Output logit Tensor from the model to .
- **output_name** (`str`) – Name of *model_out* to use in Caffe2 net.
- **label_names** (`List[str]`) – List of names of the targets/labels to expose from the Caffe2 net.

**Returns**

**List of output blobs that the *output_layer*** generates.

**Return type** List[core.BlobReference]

**get_loss**(*logit: torch.Tensor, target: torch.Tensor, context: Optional[Dict[str, Any]] = None, reduce: bool = True*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (`torch.Tensor`) – Logits returned `Model`.
- **target** (`torch.Tensor`) – True label/target to compute loss against.
- **context** (`Optional[Dict[str, Any]]`) – Context is a dictionary of items that's passed as additional metadata by the `DataHandler`. Defaults to None.
- **reduce** (`bool`) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logit: torch.Tensor, targets: Optional[torch.Tensor] = None, context: Optional[Dict[str, Any]] = None*) → Tuple[torch.Tensor, torch.Tensor]

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (`torch.Tensor`) – Logits returned `Model`.
- **targets** (`Optional[torch.Tensor]`) – True label/target. Only used by `LMOutputLayer`. Defaults to None.
- **context** (`Optional[Dict[str, Any]]`) – Context is a dictionary of items that's passed as additional metadata by the `DataHandler`. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.**CRFOutputLayer**(*num_tags, labels: pytext.data.utils.Vocabulary, *args*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for word tagging models that use Conditional Random Field.

**Parameters** **num_tags** (*int*) – Total number of possible word tags.

**num_tags**

Total number of possible word tags.

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

Exports the CRF output layer to Caffe2. See *OutputLayerBase.export_to_caffe2()* for details.

**classmethod from_config**(*config: pytext.config.component.ComponentMeta.__new__.<locals>.Config, labels: pytext.data.utils.Vocabulary*)

**get_loss**(*logit: torch.Tensor, target: torch.Tensor, context: Dict[str, Any], reduce=True*)

Compute word tagging loss by using CRF.

**Parameters**

- **logit** (*torch.Tensor*) – Logit returned by `WordTaggingModel`.
- **targets** (*torch.Tensor*) – True document label/target.
- **context** (*Dict[str, Any]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**get_pred**(*logit: torch.Tensor, target: Optional[torch.Tensor] = None, context: Optional[Dict[str, Any]] = None*)

Compute and return prediction and scores from the model.

Prediction is computed using CRF decoding.

Scores are softmax scores over the model logits where the logits are computed by rearranging the word logits such that decoded word tag has the highest valued logits. This is done because with CRF, the highest valued word tag for a given may not be part of the overall set of word tags. In order for argmax to work, we rearrange the logit values.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned `WordTaggingModel`.
- **target** (*torch.Tensor*) – Not applicable. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.**ClassificationOutputLayer**(*target_names: Optional[List[str]] = None, loss_fn: Optional[pytext.loss.loss.Loss] = None, *args, **kwargs*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for document classification models. It supports *CrossEntropyLoss* and *BinaryCrossEntropyLoss* per document.

**Parameters** **loss_fn** (*Union[CrossEntropyLoss, BinaryCrossEntropyLoss]*) – The loss function to use for computing loss. Defaults to None.

**loss_fn**

The loss function to use for computing loss.

**classmethod from_config**(*config: pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLay metadata: Optional[pytext.fields.field.FieldMeta] = None, labels: Optional[pytext.data.utils.Vocabulary] = None*)

**get_pred**(*logit, *args, **kwargs*)

Compute and return prediction and scores from the model.

Prediction is computed using argmax over the document label/target space.

Scores are sigmoid or softmax scores over the model logits depending on the loss component being used.

**Parameters** **logit** (*torch.Tensor*) – Logits returned *DocModel*.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.**RegressionOutputLayer**(*loss_fn: pytext.loss.loss.MSELoss, squash_to_unit_range: bool = False*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for doc regression models. Currently only supports Mean Squared Error loss.

**Parameters**

- **loss** (*MSELoss*) – config for MSE loss
- **squash_to_unit_range** (*bool*) – whether to clamp the output to the range [0, 1], via a sigmoid.

**classmethod from_config**(*config: pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer.Con*

**get_loss**(*logit: torch.Tensor, target: torch.Tensor, context: Optional[Dict[str, Any]] = None, reduce: bool = True*) → torch.Tensor

Compute regression loss from logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logit*, **args*, ***kwargs*)

Compute predictions and scores from the model (unlike in classification, where prediction = "most likely class" and scores = "log probs", here these are the same values). If *squash_to_unit_range* is True, fit prediction to [0, 1] via a sigmoid.

**Parameters** **logit** (*torch.Tensor*) – Logits returned from the model.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.**WordTaggingOutputLayer**(*target_names: Optional[List[str]] = None, loss_fn: Optional[pytext.loss.loss.Loss] = None, *args, **kwargs*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Output layer for word tagging models. It supports *CrossEntropyLoss* per word.

**Parameters** **loss_fn** (*CrossEntropyLoss*) – Cross-entropy loss component. Defaults to None.

**loss_fn**

Cross-entropy loss component.

**export_to_caffe2**(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext/envs/master/lib/python3.7/site-packages/caffe2/python/workspace.py'>, init_net: caffe2.python.core.Net, predict_net: caffe2.python.core.Net, model_out: torch.Tensor, output_name: str*) → List[caffe2.python.core.BlobReference]

Exports the word tagging output layer to Caffe2.

**classmethod from_config**(*config: pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer.Co labels: pytext.data.utils.Vocabulary*)

**get_loss**(*logit: torch.Tensor, target: Union[torch.Tensor, Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], context: Dict[str, Any], reduce: bool = True*) → torch.Tensor

Compute word tagging loss by comparing prediction of each word in the sentence with its true label/target.

**Parameters**

- **logit** (*torch.Tensor*) – Logit returned by *WordTaggingModel*.
- **targets** (*torch.Tensor*) – True document label/target.
- **context** (*Dict[str, Any]*) – Context is a dictionary of items that's passed as additional metadata. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Word tagging loss for all words in the sentence.

**Return type** torch.Tensor

**get_pred**(*logit: torch.Tensor*, **args*, ***kwargs*) → Tuple[torch.Tensor, torch.Tensor]

Compute and return prediction and scores from the model. Prediction is computed using argmax over the word label/target space. Scores are softmax scores over the model logits.

**Parameters** **logit** (*torch.Tensor*) – Logits returned *WordTaggingModel*.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**torchscript_predictions**()

**class** pytext.models.output_layers.**PairwiseRankingOutputLayer**(*target_names: Optional[List[str]] = None*, *loss_fn: Optional[pytext.loss.loss.Loss] = None*, **args*, ***kwargs*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

**classmethod from_config**(*config*)

**get_pred**(*logit*, *targets*, *context*)

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by LMOutputLayer. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.**PairwiseCosineDistanceOutputLayer**(*target_names: Optional[List[str]] = None, loss_fn: Union[pytext.loss.loss.BinaryCro pytext.loss.loss.CosineEmbeddingLo pytext.loss.loss.MAELoss, pytext.loss.loss.MSELoss, pytext.loss.loss.NLLLoss] = None, score_threshold: bool = 0.9, score_type: pytext.models.output_layers.distanc = <OutputScore.norm_cosine: 2>*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

**classmethod from_config**(*config, metadata: Optional[pytext.fields.field.FieldMeta] = None, labels: Optional[pytext.data.utils.Vocabulary] = None*)

**get_loss**(*logits: Tuple[torch.Tensor, torch.Tensor], targets: torch.Tensor, context: Optional[Dict[str, Any]] = None, reduce: bool = True*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logits: torch.Tensor, targets: torch.Tensor, *args, **kwargs*)

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by `LMOutputLayer`. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.**PairwiseCosineRegressionOutputLayer**(*loss_fn: Union[pytext.loss.loss.MSELo pytext.loss.loss.MAELoss]*)

Bases: `pytext.models.output_layers.output_layer_base.OutputLayerBase`

Output layer for pair (two-tower) regression models. Accepts two embedding tensors, and computes cosine similarity. The loss is between regression label and cosine similarity.

**classmethod from_config**(*config: pytext.models.output_layers.doc_regression_output_layer.PairwiseCosineRegressionC **kwargs*)

**get_loss**(*logits: Tuple[torch.Tensor, torch.Tensor], target: torch.Tensor, context: Optional[Dict[str, Any]] = None, reduce: bool = True*) → torch.Tensor

**Parameters**

- **logits** (`Tuple[torch.Tensor, torch.Tensor]`) – Logits returned from pairwise model
- **target** (`torch.Tensor`) – Float target to compute loss against
- **context** (`Optional[Dict[str, Any]]`) – Context is a dictionary from data handler
- **reduce** (`bool`) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logits, *args, **kwargs*)

**Parameters** **logits** (`Tuple[torch.Tensor, torch.Tensor]`) – Logits returned from pairwise model

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.**DenseRetrievalOutputLayer**(*target_names: Optional[List[str]] = None, loss_fn: Union[pytext.loss.loss.BinaryCrossEntropyLo pytext.loss.loss.CosineEmbeddingLoss, pytext.loss.loss.MAELoss, pytext.loss.loss.MSELoss, pytext.loss.loss.NLLLoss] = None, score_threshold: bool = 0.9, score_type: pytext.models.output_layers.distance_output_la = <OutputScore.norm_cosine: 2>*)

Bases: *pytext.models.output_layers.distance_output_layer.PairwiseCosineDistanceOutputLayer*

**get_loss**(*logits: Tuple[torch.Tensor, torch.Tensor], targets: torch.Tensor, context: Optional[Dict[str, Any]] = None, reduce: bool = True*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

**get_pred**(*logits: torch.Tensor, targets: torch.Tensor, *args, **kwargs*)

Compute and return prediction and scores from the model.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned *Model*.
- **targets** (*Optional[torch.Tensor]*) – True label/target. Only used by `LMOutputLayer`. Defaults to None.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the *DataHandler*. Defaults to None.

**Returns** Model prediction and scores.

**Return type** Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.output_layers.**OutputLayerUtils**

Bases: `object`

**static gen_additional_blobs**(*predict_net: caffe2.python.core.Net, probability_out, model_out: torch.Tensor, output_name: str, label_names: List[str]*) → List[caffe2.python.core.BlobReference]

Utility method to generate additional blobs for human readable result for models that use explicit labels.

## pytext.models.qna package

### Submodules

### pytext.models.qna.bert_squad_qa module

**class** pytext.models.qna.bert_squad_qa.**BertSquadQAModel**(*encoder: torch.nn.modules.module.Module, decoder: torch.nn.modules.module.Module, has_ans_decoder: torch.nn.modules.module.Module, output_layer: torch.nn.modules.module.Module, stage: pytext.common.constants.Stage = <Stage.TRAIN: 'Training'>, is_kd: bool = False*)

Bases: pytext.models.bert_classification_models.NewBertModel

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**forward**(**inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.qna.bert_squad_qa.BertSquadQAModel.Config, tensorizers*)

### pytext.models.qna.dr_qa module

**class** pytext.models.qna.dr_qa.**DrQAModel**(*dropout: torch.nn.modules.module.Module, embedding: torch.nn.modules.module.Module, ques_rnn: torch.nn.modules.module.Module, doc_rnn: torch.nn.modules.module.Module, ques_self_attn: torch.nn.modules.module.Module, ques_aligned_doc_attn: torch.nn.modules.module.Module, start_attn: torch.nn.modules.module.Module, end_attn: torch.nn.modules.module.Module, doc_rep_pool: torch.nn.modules.module.Module, has_ans_decoder: torch.nn.modules.module.Module, output_layer: torch.nn.modules.module.Module, is_kd: bool = False*)

Bases: pytext.models.model.BaseModel

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**classmethod create_embedding**(*model_config: pytext.models.qna.dr_qa.DrQAModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**forward**(*doc_tokens: torch.Tensor*, *doc_seq_len: torch.Tensor*, *doc_mask: torch.Tensor*, *ques_tokens: torch.Tensor*, *ques_seq_len: torch.Tensor*, *ques_mask: torch.Tensor*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.qna.dr_qa.DrQAModel.Config*, *tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

### Module contents

### pytext.models.representations package

### Subpackages

### pytext.models.representations.transformer package

### Submodules

### pytext.models.representations.transformer.multihead_attention module

**class** pytext.models.representations.transformer.multihead_attention.**MultiheadSelfAttention**

Bases: `torch.nn.modules.module.Module`

This is a TorchScriptable implementation of MultiheadAttention from fairseq for the purposes of creating a productionized RoBERTa model. It distills just the elements which are required to implement the RoBERTa use cases of MultiheadAttention, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

The default constructor values match those required to import the public RoBERTa weights. Unless you are pretraining your own model, there's no need to change them.

**forward**(*query*, *key_padding_mask*)

Input shape: Time x Batch x Channel Timesteps can be masked by supplying a T x T mask in the *attn_mask* argument. Padding elements can be excluded from the key by passing a binary ByteTensor (*key_padding_mask*) with shape: batch x source_length, where padding elements are indicated by 1s.

**prune_multi_heads**(*heads: List[int]*)

### pytext.models.representations.transformer.multihead_linear_attention module

**class** pytext.models.representations.transformer.multihead_linear_attention.**MultiheadLinearA**

Bases: torch.nn.modules.module.Module

This is a TorchScriptable implementation of MultiheadLinearAttention: https://arxiv.org/pdf/2006.04768.pdf. from fairseq for the purposes of creating a productionized Linformer model. It distills just the elements which are required to implement the RoBERTa use cases of MultiheadLinearAttention, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

The default constructor values match those required to import the public RoBERTa weights. Unless you are pretraining your own model, there's no need to change them.

**forward**(*query*, *key_padding_mask*)

Input shape: Time x Batch x Channel Timesteps can be masked by supplying a T x T mask in the *attn_mask* argument. Padding elements can be excluded from the key by passing a binary ByteTensor (*key_padding_mask*) with shape: batch x source_length, where padding elements are indicated by 1s.

**get_compressed_projection**(*k_input: torch.Tensor*, *v_input: torch.Tensor*, *target_length: int*) → Tuple[torch.Tensor, torch.Tensor]

**prune_multi_linear_heads**(*heads: List[int]*)

**class** pytext.models.representations.transformer.multihead_linear_attention.**QuantizedMultihe**

Bases: *pytext.models.representations.transformer.multihead_linear_attention.MultiheadLinearAttention*

**get_compressed_projection**(*k_input: torch.Tensor*, *v_input: torch.Tensor*, *target_length: int*) → Tuple[torch.Tensor, torch.Tensor]

### pytext.models.representations.transformer.positional_embedding module

**class** pytext.models.representations.transformer.positional_embedding.**PositionalEmbedding**(*nu in em be di in pa Op tio = Na*

Bases: torch.nn.modules.module.Module

This module learns positional embeddings up to a fixed maximum size. Padding ids are ignored by either offsetting based on pad_index or by setting pad_index to None and ensuring that the appropriate position ids are passed to the forward function.

This is a TorchScriptable implementation of PositionalEmbedding from fairseq for the purposes of creating a productionized RoBERTa model. It distills just the elements which are required to implement the RoBERTa use cases of MultiheadAttention, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

**forward**(*input*)

Input is expected to be of size [batch_size x sequence_length].

**max_positions**()

Maximum number of supported positions.

`pytext.models.representations.transformer.positional_embedding.`**`make_positions`**(*tensor*, *pad_index: int*)

Replace non-padding symbols with their position numbers. Position numbers begin at pad_index+1. Padding symbols are ignored.

### pytext.models.representations.transformer.representation module

**class** `pytext.models.representations.transformer.representation.`**`TransformerRepresentation`**(*co py te en be in*

Bases: *`pytext.models.module.Module`*

Representation consisting of stacked multi-head self-attention and position-wise feed-forward layers. Unlike *Transformer*, we assume inputs are already embedded, thus this representation can be used as a drop-in replacement for other temporal representations over text inputs (e.g., *BiLSTM* and *DeepCNNDeepCNNRepresentation*).

**forward**(*embedded_tokens: torch.Tensor*, *padding_mask: torch.Tensor*) → torch.Tensor

Forward inputs through the transformer layers.

**Parameters**

- **embedded_tokens** (*B x T x H*) – Tokens previously encoded with token,
- **and segment embeddings.** (*positional,*) –
- **padding_mask** (*B x T*) – Boolean mask specifying token positions that
- **should not operate on.** (*self-attention*) –

**Returns** Final transformer layer state.

**Return type** last_state (B x T x H)

### pytext.models.representations.transformer.residual_mlp module

**class** `pytext.models.representations.transformer.residual_mlp.`**`GeLU`**

Bases: `torch.nn.modules.module.Module`

Component class to wrap F.gelu.

**forward**(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.representations.transformer.residual_mlp.**ResidualMLP**(*input_dim: int, hidden_dims: List[int], dropout: float = 0.1, activation=<class 'pytext.models.representation*

Bases: torch.nn.modules.module.Module

A square MLP component which can learn a bias on an input vector. This MLP in particular defaults to using GeLU as its activation function (this can be changed by passing a different activation function), and retains a residual connection to its original input to help with gradient propogation.

Unlike pytext's MLPDecoder it doesn't currently allow adding a LayerNorm in between hidden layers.

**forward**(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.transformer.sentence_encoder module

**class** pytext.models.representations.transformer.sentence_encoder.**PostEncoder**(*transformer: Optional[pytext.models = None*)

Bases: *pytext.models.representations.transformer.sentence_encoder.SentenceEncoder*

**extract_features**(*tokens: torch.Tensor, dense: List[torch.Tensor]*)

**forward**(*tokens: torch.Tensor, dense: List[torch.Tensor]*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.transformer.sentence_encoder.**SentenceEncoder**(*transformer: Optional[pytext.r = None*)

Bases: torch.nn.modules.module.Module

This is a TorchScriptable implementation of RoBERTa from fairseq for the purposes of creating a productionized RoBERTa model. It distills just the elements which are required to implement the RoBERTa model, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

This SentenceEncoder can load in the public RoBERTa weights directly with *load_roberta_state_dict*, which will translate the keys as they exist in the publicly released RoBERTa to the correct structure for this implementation. The default constructor value will have the same size and shape as that model.

To use RoBERTa with this, download the RoBERTa public weights as *roberta.weights*

```
>>> encoder = SentenceEncoder()
>>> weights = torch.load("roberta.weights")
>>> encoder.load_roberta_state_dict(weights)
```

Within this you will still need to preprocess inputs using fairseq and the publicly released vocabs, and finally place this encoder in a model alongside say an MLP output layer to do classification.

**extract_features**(*tokens*)

**forward**(*tokens*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**load_roberta_state_dict**(*state_dict*)

pytext.models.representations.transformer.sentence_encoder.**check_state_keys**(*state, keys_regex*)

check if keys exists in state using full python paths

pytext.models.representations.transformer.sentence_encoder.**merge_input_projection**(*state*)

New checkpoints of fairseq multihead attention split in_projections into k,v,q projections. This function merge them back to to make it compatible.

pytext.models.representations.transformer.sentence_encoder.**remove_state_keys**(*state, keys_regex*)

Remove keys from state that match a regex

pytext.models.representations.transformer.sentence_encoder.**rename_component_from_root**(*state, old_na new_n*

Rename keys from state using full python paths

pytext.models.representations.transformer.sentence_encoder.**rename_state_keys**(*state, keys_regex, replacement*)

Rename keys from state that match a regex; replacement can use capture groups

pytext.models.representations.transformer.sentence_encoder.**translate_roberta_state_dict**(*stat*

Translate the public RoBERTa weights to ones which match SentenceEncoder.

### pytext.models.representations.transformer.transformer module

**class** pytext.models.representations.transformer.transformer.**SELFIETransformer**(*vocab_size: int = 50265, embedding_dim: int = 768, padding_idx: int = 1, max_seq_len: int = 514, layers: List[pytext.models. = (), dropout: float = 0.1*)

Bases: *pytext.models.representations.transformer.transformer.Transformer*

**forward**(*tokens: torch.Tensor, dense: List[torch.Tensor]*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.representations.transformer.transformer.**Transformer**(*vocab_size: int = 50265, embedding_dim: int = 768, padding_idx: int = 1, max_seq_len: int = 514, layers: List[pytext.models.represen = (), dropout: float = 0.1*)

Bases: torch.nn.modules.module.Module

**forward**(*tokens: torch.Tensor*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.representations.transformer.transformer.**TransformerLayer**(*embedding_dim: int = 768, attention: Optional[pytext.models = None, residual_mlp: Optional[pytext.models = None, dropout: float = 0.1*)

Bases: torch.nn.modules.module.Module

**forward**(*input*, *key_padding_mask*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### Module contents

This directory contains modules for implementing a productionized RoBERTa model. These modules implement the same Transformer components that are implemented in the fairseq library, however they're distilled down to just the elements which are used in the final RoBERTa model, and within that are restructured and rewritten to be able to be compiled by TorchScript for production use cases.

The SentenceEncoder specifically can be used to load model weights directly from the publicly release RoBERTa weights, and it will translate these weights to the corresponding values in this implementation.

**class** pytext.models.representations.transformer.**MultiheadLinearAttention**(*embed_dim: int, num_heads: int, scaling: float = 0.125, dropout: float = 0.1, compress_layer=None, bias: bool = True*)

Bases: torch.nn.modules.module.Module

This is a TorchScriptable implementation of MultiheadLinearAttention: https://arxiv.org/pdf/2006.04768.pdf. from fairseq for the purposes of creating a productionized Linformer model. It distills just the elements which are required to implement the RoBERTa use cases of MultiheadLinearAttention, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

The default constructor values match those required to import the public RoBERTa weights. Unless you are pretraining your own model, there's no need to change them.

**forward**(*query, key_padding_mask*)

Input shape: Time x Batch x Channel Timesteps can be masked by supplying a T x T mask in the *attn_mask* argument. Padding elements can be excluded from the key by passing a binary ByteTensor (*key_padding_mask*) with shape: batch x source_length, where padding elements are indicated by 1s.

**get_compressed_projection**(*k_input: torch.Tensor, v_input: torch.Tensor, target_length: int*) → Tuple[torch.Tensor, torch.Tensor]

**prune_multi_linear_heads**(*heads: List[int]*)

**class** pytext.models.representations.transformer.**QuantizedMultiheadLinearAttention**(*embed_dim: int, num_heads: int, scaling: float = 0.125, dropout: float = 0.1, compress_layer= bias: bool = True*)

Bases: *pytext.models.representations.transformer.multihead_linear_attention.MultiheadLinearAttention*

**get_compressed_projection**(*k_input: torch.Tensor*, *v_input: torch.Tensor*, *target_length: int*) → Tuple[torch.Tensor, torch.Tensor]

**class** pytext.models.representations.transformer.**MultiheadSelfAttention**(*embed_dim: int*, *num_heads: int*, *scaling: float = 0.125*, *dropout: float = 0.1*)

Bases: torch.nn.modules.module.Module

This is a TorchScriptable implementation of MultiheadAttention from fairseq for the purposes of creating a productionized RoBERTa model. It distills just the elements which are required to implement the RoBERTa use cases of MultiheadAttention, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

The default constructor values match those required to import the public RoBERTa weights. Unless you are pretraining your own model, there's no need to change them.

**forward**(*query*, *key_padding_mask*)

Input shape: Time x Batch x Channel Timesteps can be masked by supplying a T x T mask in the *attn_mask* argument. Padding elements can be excluded from the key by passing a binary ByteTensor (*key_padding_mask*) with shape: batch x source_length, where padding elements are indicated by 1s.

**prune_multi_heads**(*heads: List[int]*)

**class** pytext.models.representations.transformer.**PositionalEmbedding**(*num_embeddings: int*, *embedding_dim: int*, *pad_index: Optional[int] = None*)

Bases: torch.nn.modules.module.Module

This module learns positional embeddings up to a fixed maximum size. Padding ids are ignored by either offsetting based on pad_index or by setting pad_index to None and ensuring that the appropriate position ids are passed to the forward function.

This is a TorchScriptable implementation of PositionalEmbedding from fairseq for the purposes of creating a productionized RoBERTa model. It distills just the elements which are required to implement the RoBERTa use cases of MultiheadAttention, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

**forward**(*input*)

Input is expected to be of size [batch_size x sequence_length].

**max_positions**()
Maximum number of supported positions.

**class** pytext.models.representations.transformer.**ResidualMLP**(*input_dim: int, hidden_dims: List[int], dropout: float = 0.1, activation=<class 'pytext.models.representations.transformer.residu*

Bases: torch.nn.modules.module.Module

A square MLP component which can learn a bias on an input vector. This MLP in particular defaults to using GeLU as its activation function (this can be changed by passing a different activation function), and retains a residual connection to its original input to help with gradient propogation.

Unlike pytext's MLPDecoder it doesn't currently allow adding a LayerNorm in between hidden layers.

**forward**(*input*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.transformer.**SentenceEncoder**(*transformer: Optional[pytext.models.representations.tra = None*)

Bases: torch.nn.modules.module.Module

This is a TorchScriptable implementation of RoBERTa from fairseq for the purposes of creating a productionized RoBERTa model. It distills just the elements which are required to implement the RoBERTa model, and within that is restructured and rewritten to be able to be compiled by TorchScript for production use cases.

This SentenceEncoder can load in the public RoBERTa weights directly with *load_roberta_state_dict*, which will translate the keys as they exist in the publicly released RoBERTa to the correct structure for this implementation. The default constructor value will have the same size and shape as that model.

To use RoBERTa with this, download the RoBERTa public weights as *roberta.weights*

```
>>> encoder = SentenceEncoder()
>>> weights = torch.load("roberta.weights")
>>> encoder.load_roberta_state_dict(weights)
```

Within this you will still need to preprocess inputs using fairseq and the publicly released vocabs, and finally place this encoder in a model alongside say an MLP output layer to do classification.

**extract_features**(*tokens*)

**forward**(*tokens*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

**load_roberta_state_dict**(*state_dict*)

**class** pytext.models.representations.transformer.**PostEncoder**(*transformer: Optional[pytext.models.representations.transfor = None*)

Bases: pytext.models.representations.transformer.sentence_encoder.SentenceEncoder

**extract_features**(*tokens: torch.Tensor, dense: List[torch.Tensor]*)

**forward**(*tokens: torch.Tensor, dense: List[torch.Tensor]*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.transformer.**SELFIETransformer**(*vocab_size: int = 50265, embedding_dim: int = 768, padding_idx: int = 1, max_seq_len: int = 514, layers: List[pytext.models.representations.tr = (), dropout: float = 0.1*)

Bases: pytext.models.representations.transformer.transformer.Transformer

**forward**(*tokens: torch.Tensor, dense: List[torch.Tensor]*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.transformer.**Transformer**(*vocab_size: int = 50265, embedding_dim: int = 768, padding_idx: int = 1, max_seq_len: int = 514, layers: List[pytext.models.representations.transforme = (), dropout: float = 0.1*)

Bases: torch.nn.modules.module.Module

**forward**(*tokens: torch.Tensor*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.transformer.**TransformerLayer**(*embedding_dim: int = 768, attention: Optional[pytext.models.representations.t = None, residual_mlp: Optional[pytext.models.representations.t = None, dropout: float = 0.1*)

Bases: torch.nn.modules.module.Module

**forward**(*input, key_padding_mask*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.transformer.**TransformerRepresentation**(*config: pytext.models.representatio embed_dim: int*)

Bases: *pytext.models.module.Module*

Representation consisting of stacked multi-head self-attention and position-wise feed-forward layers. Unlike *Transformer*, we assume inputs are already embedded, thus this representation can be used as a drop-in replacement for other temporal representations over text inputs (e.g., *BiLSTM* and *DeepCNNDeepCNNRepresentation*).

**forward**(*embedded_tokens: torch.Tensor*, *padding_mask: torch.Tensor*) → torch.Tensor
Forward inputs through the transformer layers.

**Parameters**

- **embedded_tokens** (*B x T x H*) – Tokens previously encoded with token,
- **and segment embeddings.** (*positional,*) –
- **padding_mask** (*B x T*) – Boolean mask specifying token positions that
- **should not operate on.** (*self-attention*) –

**Returns** Final transformer layer state.

**Return type** last_state (B x T x H)

## Submodules

### pytext.models.representations.attention module

**class** pytext.models.representations.attention.**DotProductSelfAttention**(*input_dim*)
Bases: *pytext.models.module.Module*

Given vector w and token vectors = {t1, t2, . . . , t_n}, compute self attention weights to weighs the tokens * a_j = softmax(w . t_j)

**forward**(*tokens*, *tokens_mask*)

**Input:** x: batch_size * seq_len * input_dim x_mask: batch_size * seq_len (1 for padding, 0 for true)

**Output:** alpha: batch_size * seq_len

**classmethod from_config**(*config: pytext.models.representations.attention.DotProductSelfAttention.Config*)

**class** pytext.models.representations.attention.**MultiplicativeAttention**(*p_hidden_dim*, *q_hidden_dim*, *normalize*)
Bases: *pytext.models.module.Module*

Given sequence P and vector q, computes attention weights for each element in P by matching q with each element in P using multiplicative attention. * a_i = softmax(p_i . W . q)

**forward**(*p_seq: torch.Tensor*, *q: torch.Tensor*, *p_mask: torch.Tensor*)

**Input:** p_seq: batch_size * p_seq_len * p_hidden_dim q: batch_size * q_hidden_dim p_mask: batch_size * p_seq_len (1 for padding, 0 for true)

**Output:** attn_scores: batch_size * p_seq_len

**classmethod from_config**(*config: pytext.models.representations.attention.MultiplicativeAttention.Config*)

**class** pytext.models.representations.attention.**SequenceAlignedAttention**(*proj_dim*)
Bases: *pytext.models.module.Module*

Given sequences P and Q, computes attention weights for each element in P by matching Q with each element in P. * a_i_j = softmax(p_i . q_j) where softmax is computed by summing over q_j

**forward**(*p: torch.Tensor*, *q: torch.Tensor*, *q_mask: torch.Tensor*)

**Input:** p: batch_size * p_seq_len * dim q: batch_size * q_seq_len * dim q_mask: batch_size * q_seq_len (1 for padding, 0 for true)

**Output:** matched_seq: batch_size * doc_seq_len * dim

**classmethod from_config**(*config: pytext.models.representations.attention.SequenceAlignedAttention.Config*)

### pytext.models.representations.augmented_lstm module

**class** pytext.models.representations.augmented_lstm.**AugmentedLSTM**(*config: pytext.models.representations.augmented_lstm.AugmentedLSTM.Config, embed_dim: int, padding_value: float = 0.0*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

*AugmentedLSTM* implements a generic AugmentedLSTM representation layer. AugmentedLSTM is an LSTM which optionally appends an optional highway network to the output layer. Furthermore the dropout controlls the level of variational dropout done.

**Parameters**

- **config** (*Config*) – Configuration object of type BiLSTM.Config.
- **embed_dim** (*int*) – The number of expected features in the input.
- **padding_value** (*float*) – Value for the padded elements. Defaults to 0.0.

**padding_value**

Value for the padded elements.

**Type** float

**forward_layers**

A module list of unidirectional AugmentedLSTM layers moving forward in time.

**Type** nn.ModuleList

**backward_layers**

A module list of unidirectional AugmentedLSTM layers moving backward in time.

**Type** nn.ModuleList

**representation_dim**

The calculated dimension of the output features of AugmentedLSTM.

**Type** int

**forward**(*embedded_tokens: torch.Tensor, seq_lengths: torch.Tensor, states: Optional[Tuple[torch.Tensor, torch.Tensor]] = None*) → Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

Given an input batch of sequential data such as word embeddings, produces a AugmentedLSTM representation of the sequential input and new state tensors.

**Parameters**

- **embedded_tokens** (*torch.Tensor*) – Input tensor of shape (bsize x seq_len x input_dim).
- **seq_lengths** (*torch.Tensor*) – List of sequences lengths of each batch element.
- **states** (*Tuple[torch.Tensor, torch.Tensor]*) – Tuple of tensors containing the initial hidden state and the cell state of each element in the batch. Each of these tensors have a dimension of (bsize x num_layers x num_directions * nhid). Defaults to *None*.

**Returns** AgumentedLSTM representation of input and the state of the LSTM *t = seq_len*. Shape of representation is (bsize x seq_len x representation_dim). Shape of each state is (bsize x num_layers * num_directions x nhid).

**Return type** Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

**class** pytext.models.representations.augmented_lstm.**AugmentedLSTMCell**(*embed_dim: int, lstm_dim: int, use_highway: bool, use_bias: bool = True*)

Bases: torch.nn.modules.module.Module

*AugmentedLSTMCell* implements a AugmentedLSTM cell. :param embed_dim: The number of expected features in the input. :type embed_dim: int :param lstm_dim: Number of features in the hidden state of the LSTM. :type lstm_dim: int :param Defaults to 32.: :param use_highway: If *True* we append a highway network to the :type use_highway: bool :param outputs of the LSTM.: :param use_bias: If *True* we use a bias in our LSTM calculations, otherwise :type use_bias: bool :param we don't.:

**input_linearity**

Fused weight matrix which computes a linear function over the input.

**Type** nn.Module

**state_linearity**

Fused weight matrix which computes a linear function over the states.

**Type** nn.Module

**forward**(*x: torch.Tensor, states=typing.Tuple[torch.Tensor, torch.Tensor], variational_dropout_mask: Optional[torch.Tensor] = None*) → Tuple[torch.Tensor, torch.Tensor]

Warning: DO NOT USE THIS LAYER DIRECTLY, INSTEAD USE the AugmentedLSTM class

**Parameters**

- **x** (*torch.Tensor*) – Input tensor of shape (bsize x input_dim).
- **states** (*Tuple[torch.Tensor, torch.Tensor]*) – Tuple of tensors containing the hidden state and the cell state of each element in the batch. Each of these tensors have a dimension of (bsize x nhid). Defaults to *None*.

**Returns** Returned states. Shape of each state is (bsize x nhid).

**Return type** Tuple[torch.Tensor, torch.Tensor]

**reset_parameters**()

**class** pytext.models.representations.augmented_lstm.**AugmentedLSTMUnidirectional**(*embed_dim: int, lstm_dim: int, go_forward: bool = True, re-cur-rent_dropout_pro float = 0.0, use_highway: bool = True, use_input_projec bool = True*)

Bases: torch.nn.modules.module.Module

*AugmentedLSTMUnidirectional* implements a one-layer single directional AugmentedLSTM layer. AugmentedLSTM is an LSTM which optionally appends an optional highway network to the output layer. Furthermore the dropout controlls the level of variational dropout done.

**Parameters**

- **embed_dim** (*int*) – The number of expected features in the input.
- **lstm_dim** (*int*) – Number of features in the hidden state of the LSTM. Defaults to 32.
- **go_forward** (*bool*) – Whether to compute features left to right (forward) or right to left (backward).
- **recurrent_dropout_probability** (*float*) – Variational dropout probability to use. Defaults to 0.0.
- **use_highway** (*bool*) – If *True* we append a highway network to the outputs of the LSTM.
- **use_input_projection_bias** (*bool*) – If *True* we use a bias in our LSTM calculations, otherwise we don't.

**cell**

AugmentedLSTMCell that is applied at every timestep.

**Type** AugmentedLSTMCell

**forward**(*inputs: torch.nn.utils.rnn.PackedSequence, states: Optional[Tuple[torch.Tensor, torch.Tensor]] = None*) → Tuple[torch.nn.utils.rnn.PackedSequence, Tuple[torch.Tensor, torch.Tensor]]

Warning: DO NOT USE THIS LAYER DIRECTLY, INSTEAD USE the AugmentedLSTM class

Given an input batch of sequential data such as word embeddings, produces a single layer unidirectional AugmentedLSTM representation of the sequential input and new state tensors.

**Parameters**

- **inputs** (*PackedSequence*) – Input tensor of shape (bsize x seq_len x input_dim).
- **states** (*Tuple[torch.Tensor, torch.Tensor]*) – Tuple of tensors containing the initial hidden state and the cell state of each element in the batch. Each of these tensors have a dimension of (1 x bsize x num_directions * nhid). Defaults to *None*.

**Returns** AgumentedLSTM representation of input and the state of the LSTM *t = seq_len*. Shape of representation is (bsize x seq_len x representation_dim). Shape of each state is (1 x bsize x nhid).

**Return type** Tuple[PackedSequence, Tuple[torch.Tensor, torch.Tensor]]

**get_dropout_mask**(*dropout_probability: float*, *tensor_for_masking: torch.Tensor*) → torch.Tensor

### pytext.models.representations.bilstm module

**class** pytext.models.representations.bilstm.**BiLSTM**(*config: pytext.models.representations.bilstm.BiLSTM.Config*, *embed_dim: int*, *padding_value: float = 0.0*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

*BiLSTM* implements a multi-layer bidirectional LSTM representation layer preceded by a dropout layer.

**Parameters**

- **config** (*Config*) – Configuration object of type BiLSTM.Config.
- **embed_dim** (*int*) – The number of expected features in the input.
- **padding_value** (*float*) – Value for the padded elements. Defaults to 0.0.

**padding_value**

Value for the padded elements.

**Type** float

**dropout**

Dropout layer preceding the LSTM.

**Type** nn.Dropout

**lstm**

LSTM layer that operates on the inputs.

**Type** nn.LSTM

**representation_dim**

The calculated dimension of the output features of BiLSTM.

**Type** int

**forward**(*embedded_tokens: torch.Tensor*, *seq_lengths: torch.Tensor*, *states: Optional[Tuple[torch.Tensor, torch.Tensor]] = None*) → Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

Given an input batch of sequential data such as word embeddings, produces a bidirectional LSTM representation of the sequential input and new state tensors.

**Parameters**

- **embedded_tokens** (*torch.Tensor*) – Input tensor of shape (bsize x seq_len x input_dim).
- **seq_lengths** (*torch.Tensor*) – List of sequences lengths of each batch element.

- **states** (*Tuple[torch.Tensor, torch.Tensor]*) – Tuple of tensors containing the initial hidden state and the cell state of each element in the batch. Each of these tensors have a dimension of (bsize x num_layers * num_directions x nhid). Defaults to *None*.

**Returns**

**Bidirectional** LSTM representation of input and the state of the LSTM *t* = *seq_len*. Shape of representation is (bsize x seq_len x representation_dim). Shape of each state is (bsize x num_layers * num_directions x nhid).

**Return type** Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

### pytext.models.representations.bilstm_doc_attention module

**class** pytext.models.representations.bilstm_doc_attention.**BiLSTMDocAttention**(*config: pytext.models.representa*
*embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

*BiLSTMDocAttention* implements a multi-layer bidirectional LSTM based representation for documents with or without pooling. The pooling can be max pooling, mean pooling or self attention.

**Parameters**

- **config** (*Config*) – Configuration object of type BiLSTMDocAttention.Config.
- **embed_dim** (*int*) – The number of expected features in the input.

**dropout**

Dropout layer preceding the LSTM.

**Type** nn.Dropout

**lstm**

Module that implements the LSTM.

**Type** nn.Module

**attention**

Module that implements the attention or pooling.

**Type** nn.Module

**dense**

Module that implements the non-linear projection over attended representation.

**Type** nn.Module

**representation_dim**

The calculated dimension of the output features of the *BiLSTMDocAttention* representation.

**Type** int

**forward**(*embedded_tokens: torch.Tensor*, *seq_lengths: torch.Tensor*, **args*, *states: Tuple[torch.Tensor, torch.Tensor] = None*) → Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

Given an input batch of sequential data such as word embeddings, produces a bidirectional LSTM representation with or without pooling of the sequential input and new state tensors.

**Parameters**

- **embedded_tokens** (*torch.Tensor*) – Input tensor of shape (bsize x seq_len x input_dim).
- **seq_lengths** (*torch.Tensor*) – List of sequences lengths of each batch element.
- **states** (*Tuple[torch.Tensor, torch.Tensor]*) – Tuple of tensors containing the initial hidden state and the cell state of each element in the batch. Each of these tensors have a dimension of (bsize x num_layers * num_directions x nhid). Defaults to *None*.

**Returns**

**Bidirectional** LSTM representation of input and the state of the LSTM at *t* = *seq_len*.

**Return type** Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

### pytext.models.representations.bilstm_doc_slot_attention module

**class** pytext.models.representations.bilstm_doc_slot_attention.**BiLSTMDocSlotAttention**(*config: pytext.mod em- bed_dim int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

*BiLSTMDocSlotAttention* implements a multi-layer bidirectional LSTM based representation with support for various attention mechanisms.

In default mode, when attention configuration is not provided, it behaves like a multi-layer LSTM encoder and returns the output features from the last layer of the LSTM, for each t. When document_attention configuration is provided, it produces a fixed-sized document representation. When slot_attention configuration is provide, it attends on output of each cell of LSTM module to produce a fixed sized word representation.

**Parameters**

- **config** (*Config*) – Configuration object of type BiLSTMDocSlotAttention.Config.
- **embed_dim** (*int*) – The number of expected features in the input.

**dropout**

Dropout layer preceding the LSTM.

**Type** nn.Dropout

**relu**

An instance of the ReLU layer.

**Type** nn.ReLU

**lstm**

Module that implements the LSTM.

**Type** nn.Module

**use_doc_attention**

If *True*, indicates using document attention.

**Type** bool

**doc_attention**

Module that implements document attention.

**Type** nn.Module

`self.`**`projection_d`**
A sequence of dense layers for projection over document representation.

**Type** nn.Sequential

**`use_word_attention`**
If *True*, indicates using word attention.

**Type** bool

**`word_attention`**
Module that implements word attention.

**Type** nn.Module

`self.`**`projection_w`**
A sequence of dense layers for projection over word representation.

**Type** nn.Sequential

**`representation_dim`**
The calculated dimension of the output features of the *BiLSTMDocAttention* representation.

**Type** int

**`forward`**(*embedded_tokens: torch.Tensor*, *seq_lengths: torch.Tensor*, **args*, *states: torch.Tensor = None*) → Tuple[torch.Tensor, torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]
Given an input batch of sequential data such as word embeddings, produces a bidirectional LSTM representation the appropriate attention.

**Parameters**

- **`embedded_tokens`** (*`torch.Tensor`*) – Input tensor of shape (bsize x seq_len x input_dim).
- **`seq_lengths`** (*`torch.Tensor`*) – List of sequences lengths of each batch element.
- **`states`** (*`Tuple[torch.Tensor, torch.Tensor]`*) – Tuple of tensors containing the initial hidden state and the cell state of each element in the batch. Each of these tensors have a dimension of (bsize x num_layers * num_directions x nhid). Defaults to *None*.

**Returns** Tensors containing the document and the word representation of the input.

**Return type** Tuple[torch.Tensor, torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

### pytext.models.representations.bilstm_slot_attn module

**class** `pytext.models.representations.bilstm_slot_attn.`**`BiLSTMSlotAttention`**(*config: pytext.models.representation em- bed_dim: int*)

Bases: *`pytext.models.representations.representation_base.RepresentationBase`*

*BiLSTMSlotAttention* implements a multi-layer bidirectional LSTM based representation with attention over slots.

**Parameters**

- **config** (*Config*) – Configuration object of type BiLSTMSlotAttention.Config.
- **embed_dim** (*int*) – The number of expected features in the input.

**dropout**
Dropout layer preceding the LSTM.

**Type** nn.Dropout

**lstm**
Module that implements the LSTM.

**Type** nn.Module

**attention**
Module that implements the attention.

**Type** nn.Module

**dense**
Module that implements the non-linear projection over attended representation.

**Type** nn.Module

**representation_dim**
The calculated dimension of the output features of the *SlotAttention* representation.

**Type** int

**forward**(*embedded_tokens: torch.Tensor*, *seq_lengths: torch.Tensor*, **args*, *states: torch.Tensor = None*, ***kwargs*) → torch.Tensor
Given an input batch of sequential data such as word embeddings, produces a bidirectional LSTM representation with or without Slot attention.

**Parameters**

- **embedded_tokens** (*torch.Tensor*) – Input tensor of shape (bsize x seq_len x input_dim).
- **seq_lengths** (*torch.Tensor*) – List of sequences lengths of each batch element.
- **states** (*Tuple[torch.Tensor, torch.Tensor]*) – Tuple of tensors containing the initial hidden state and the cell state of each element in the batch. Each of these tensors have a dimension of (bsize x num_layers * num_directions x nhid). Defaults to *None*.

**Returns**

**Bidirectional LSTM representation of input with or** without slot attention.

**Return type** torch.Tensor

### pytext.models.representations.biseqcnn module

**class** pytext.models.representations.biseqcnn.**BSeqCNNRepresentation**(*config: pytext.models.representations.biseqcn*
*embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

This class is an implementation of the paper https://arxiv.org/pdf/1606.07783. It is a bidirectional CNN model that captures context like RNNs do.

The module expects that input mini-batch is already padded.

TODO: Current implementation has a single layer conv-maxpool operation.

**forward**(*inputs: torch.Tensor*, **args*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.biseqcnn.**ContextualWordConvolution**(*in_channels: int, out_channels: int, kernel_sizes: List[int]*)

Bases: `torch.nn.modules.module.Module`

**forward**(*words: torch.Tensor*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

## pytext.models.representations.contextual_intent_slot_rep module

**class** pytext.models.representations.contextual_intent_slot_rep.**ContextualIntentSlotRepresen**

Bases: *pytext.models.representations.representation_base.RepresentationBase*

Representation for a contextual intent slot model

The inputs are two embeddings: word level embedding containing dictionary features, sequence (contexts) level embedding. See following diagram for the representation implementation that combines the two embeddings. Seq_representation is concatenated with word_embeddings.

```
+-----------+
| word_embed|--------------------------->+   +--------------------+
+-----------+                            |   | doc_representation |
```

(continues on next page)

(continued from previous page)

```
+-----------+   +-------------------+    |-->+--------------------+
| seq_embed |-->| seq_representation|--->+   | word_representation|
+-----------+   +-------------------+        +--------------------+
                                              joint_representation
```

**forward**(*word_seq_embed: Tuple[torch.Tensor, torch.Tensor], word_lengths: torch.Tensor, seq_lengths: torch.Tensor, *args*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.deepcnn module

**class** pytext.models.representations.deepcnn.**DeepCNNRepresentation**(*config: pytext.models.representations.deepcnn.*, *embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

*DeepCNNRepresentation* implements CNN representation layer preceded by a dropout layer. CNN representation layer is based on the encoder in the architecture proposed by Gehring et. al. in Convolutional Sequence to Sequence Learning.

**Parameters**

- **config** (*Config*) – Configuration object of type DeepCNNRepresentation.Config.
- **embed_dim** (*int*) – The number of expected features in the input.

**forward**(*inputs: torch.Tensor, *args*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.deepcnn.**SeparableConv1d**(*input_channels: int, output_channels: int, kernel_size: int, padding: int, dilation: int, bottleneck: int*)

Bases: `torch.nn.modules.module.Module`

Implements a 1d depthwise separable convolutional layer. In regular convolutional layers, the input channels are mixed with each other to produce each output channel. Depthwise separable convolutions decompose this process into two smaller convolutions – a depthwise and pointwise convolution.

The depthwise convolution spatially convolves each input channel separately, then the pointwise convolution projects this result into a new channel space. This process reduces the number of FLOPS used to compute a convolution and also exhibits a regularization effect. The general behavior – including the input parameters – is equivalent to *nn.Conv1d*.

*bottleneck* controls the behavior of the pointwise convolution. Instead of upsampling directly, we split the pointwise convolution into two pieces: the first convolution downsamples into a (sufficiently small) low dimension and the second convolution upsamples into the target (higher) dimension. Creating this bottleneck significantly cuts the number of parameters with minimal loss in performance.

**forward**(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.representations.deepcnn.**Trim1d**(*trim*)

Bases: `torch.nn.modules.module.Module`

Trims a 1d convolutional output. Used to implement history-padding by removing excess padding from the right.

**forward**(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

pytext.models.representations.deepcnn.**create_conv_package**(*index: int*, *activation: pytext.config.module_config.Activation*, *in_channels: int*, *out_channels: int*, *kernel_size: int*, *causal: bool*, *dilated: bool*, *separable: bool*, *bottleneck: int*, *weight_norm: bool*)

Creates a convolutional layer with the specified arguments.

**Parameters**

- **index** (*int*) – Index of a convolutional layer in the stack.
- **activation** (*Activation*) – Activation function.
- **in_channels** (*int*) – Number of input channels.
- **out_channels** (*int*) – Number of output channels.
- **kernel_size** (*int*) – Size of 1d convolutional filter.
- **causal** (*bool*) – Whether the convolution is causal or not. If set, it

- **for the temporal ordering of the inputs.** (*accounts*) –
- **dilated** (*bool*) – Whether the convolution is dilated or not. If set,
- **receptive field of the convolutional stack grows exponentially.** (*the*) –
- **separable** (*bool*) – Whether to use depthwise separable convolutions
- **not -- see SeparableConv1d.** (*or*) –
- **bottleneck** (*int*) – Bottleneck channel dimension for depthwise separable
- **See SeparableConv1d for an in-depth explanation.** (*convolutions.*) –
- **weight_norm** (*bool*) – Whether to add weight normalization to the
- **convolutions or not.** (*regular*) –

pytext.models.representations.deepcnn.**pool**(*pooling_type*, *words*)

### pytext.models.representations.docnn module

**class** pytext.models.representations.docnn.**DocNNRepresentation**(*config: pytext.models.representations.docnn.DocNNR embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

CNN based representation of a document.

**conv_and_pool**(*x*, *conv*)

**forward**(*embedded_tokens: torch.Tensor*, **args*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.huggingface_bert_sentence_encoder module

**class** pytext.models.representations.huggingface_bert_sentence_encoder.**HuggingFaceBertSente**

Bases: *pytext.models.representations.transformer_sentence_encoder_base.TransformerSentenceEncoderBase*

Generate sentence representation using the open source HuggingFace BERT model. This class implements loading the model weights from a pre-trained model file.

### pytext.models.representations.huggingface_electra_sentence_encoder module

**class** pytext.models.representations.huggingface_electra_sentence_encoder.**HuggingFaceElectra**

Bases: *pytext.models.representations.transformer_sentence_encoder_base.TransformerSentenceEncoderBase*

Generate sentence representation using the open source HuggingFace Electra model. This class implements loading the model weights from a pre-trained model file.

### pytext.models.representations.jointcnn_rep module

**class** pytext.models.representations.jointcnn_rep.**JointCNNRepresentation**(*config: pytext.models.representations.*
*embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

**forward**(*embedded_tokens: torch.Tensor*, **args*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.jointcnn_rep.**SharedCNNRepresentation**(*config: pytext.models.representation*
*embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

**forward**(*embedded_tokens: torch.Tensor*, **args*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.ordered_neuron_lstm module

**class** pytext.models.representations.ordered_neuron_lstm.**OrderedNeuronLSTM**(*config: pytext.models.representatio embed_dim: int, padding_value: Optional[float] = 0.0*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

**forward**(*rep: torch.Tensor, seq_lengths: torch.Tensor, states: Optional[Tuple[torch.Tensor, torch.Tensor]] = None*) → Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.representations.ordered_neuron_lstm.**OrderedNeuronLSTMLayer**(*embed_dim: int, lstm_dim: int, padding_value: float, dropout: float*)

Bases: *pytext.models.module.Module*

**forward**(*embedded_tokens: torch.Tensor, states: Tuple[torch.Tensor, torch.Tensor], seq_lengths: List[int]*) → Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

### pytext.models.representations.pair_rep module

**class** pytext.models.representations.pair_rep.**PairRepresentation**(*config: pytext.models.representations.pair_rep.Pa embed_dim: Tuple[int, ...]*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

Wrapper representation for a pair of inputs.

Takes a tuple of inputs: the left sentence, and the right sentence(s). Returns a representation of the pair of sentences, either as a concatenation of the two sentence embeddings or as a "siamese" representation which also includes their difference and elementwise product (arXiv:1705.02364). If more than two inputs are provided, the extra inputs are assumed to be extra "right" sentences, and the output will be the stacked pair representations of the left sentence together with all right sentences. This is more efficient than separately computing all these pair representations, because the left sentence will not need to be re-embedded multiple times.

**forward**(*embeddings: Tuple[torch.Tensor, ...], *lengths*) → torch.Tensor

Computes the pair representations.

**Parameters**

- **embeddings** – token embeddings of the left sentence, followed by the token embeddings of the right sentence(s).
- **lengths** – the corresponding sequence lengths.

**Returns** A tensor of shape *(num_right_inputs, batch_size, rep_size)*, with the first dimension squeezed if one.

### pytext.models.representations.pass_through module

**class** pytext.models.representations.pass_through.**PassThroughRepresentation**(*config: pytext.config.component.C embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

**forward**(*embedded_tokens: torch.Tensor*, **args*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.pooling module

**class** pytext.models.representations.pooling.**BoundaryPool**(*config: pytext.models.representations.pooling.BoundaryPoo n_input: int*)

Bases: *pytext.models.module.Module*

**forward**(*inputs: torch.Tensor*, *seq_lengths: torch.Tensor = None*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.pooling.**LastTimestepPool**(*config: pytext.config.module_config.ModuleConfig*, *n_input: int*)

Bases: *pytext.models.module.Module*

**forward**(*inputs: torch.Tensor*, *seq_lengths: torch.Tensor*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.pooling.**MaxPool**(*config: pytext.config.module_config.ModuleConfig*, *n_input: int*)

Bases: *pytext.models.module.Module*

**forward**(*inputs: torch.Tensor*, *seq_lengths: torch.Tensor = None*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.pooling.**MeanPool**(*config: pytext.config.module_config.ModuleConfig*, *n_input: int*)

Bases: *pytext.models.module.Module*

**forward**(*inputs: torch.Tensor*, *seq_lengths: torch.Tensor*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.pooling.**NoPool**(*config: pytext.config.module_config.ModuleConfig*, *n_input: int*)

Bases: *pytext.models.module.Module*

**forward**(*inputs: torch.Tensor*, *seq_lengths: torch.Tensor = None*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

**class** pytext.models.representations.pooling.**SelfAttention**(*config: pytext.models.representations.pooling.SelfAttention.Config, n_input: int*)

Bases: *pytext.models.module.Module*

**forward**(*inputs: torch.Tensor, seq_lengths: torch.Tensor = None*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**init_weights**(*init_range: float = 0.1*) → None

### pytext.models.representations.pure_doc_attention module

**class** pytext.models.representations.pure_doc_attention.**PureDocAttention**(*config: pytext.models.representations.pure_doc_attention.PureDocAttention.Config, embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

pooling (e.g. max pooling or self attention) followed by optional MLP

**forward**(*embedded_tokens: torch.Tensor, seq_lengths: torch.Tensor = None, *args*) → Any

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.representation_base module

**class** pytext.models.representations.representation_base.**RepresentationBase**(*config*)

Bases: *pytext.models.module.Module*

**forward**(**inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**get_representation_dim**()

### pytext.models.representations.seq_rep module

**class** pytext.models.representations.seq_rep.**SeqRepresentation**(*config: pytext.models.representations.seq_rep.SeqRe embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

Representation for a sequence of sentences Each sentence will be embedded with a DocNN model, then all the sentences are embedded with another DocNN/BiLSTM model

**forward**(*embedded_seqs: torch.Tensor*, *seq_lengths: torch.Tensor*, **args*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.slot_attention module

**class** pytext.models.representations.slot_attention.**SlotAttention**(*config: pytext.models.representations.slot_atten n_input: int, batch_first: bool = True*)

Bases: *pytext.models.module.Module*

**forward**(*inputs: torch.Tensor*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.sparse_transformer_sentence_encoder module

**class** pytext.models.representations.sparse_transformer_sentence_encoder.**SparseTransformerSe**

Bases: *pytext.models.representations.transformer_sentence_encoder.TransformerSentenceEncoder*

Implementation of the Transformer Sentence Encoder. This directly makes use of the TransformerSentenceEncoder module in Fairseq.

**A few interesting config options:**

- encoder_normalize_before detemines whether the layer norm is applied before or after self_attention. This is similar to original implementation from Google.
- activation_fn can be set to 'gelu' instead of the default of 'relu'.
- project_representation adds a linear projection + tanh to the pooled output in the style of BERT.

### pytext.models.representations.stacked_bidirectional_rnn module

**class** pytext.models.representations.stacked_bidirectional_rnn.**RnnType**

Bases: enum.Enum

An enumeration.

**GRU = 'gru'**

**LSTM = 'lstm'**

**RNN = 'rnn'**

**class** pytext.models.representations.stacked_bidirectional_rnn.**StackedBidirectionalRNN**(*config.*
*py-*
*text.mo*
*in-*
*put_siz*
*int*,
*paddin*
*float*
*=*
*0.0*)

Bases: *pytext.models.module.Module*

*StackedBidirectionalRNN* implements a multi-layer bidirectional RNN with an option to return outputs from all the layers of RNN.

**Parameters**

- **config** (*Config*) – Configuration object of type BiLSTM.Config.
- **embed_dim** (*int*) – The number of expected features in the input.
- **padding_value** (*float*) – Value for the padded elements. Defaults to 0.0.

**padding_value**
Value for the padded elements.

**Type** float

**dropout**
Dropout layer preceding the LSTM.

**Type** nn.Dropout

**lstm**
LSTM layer that operates on the inputs.

**Type** nn.LSTM

**representation_dim**
The calculated dimension of the output features of BiLSTM.

**Type** int

**forward** (*tokens*, *tokens_mask*)

**Parameters**

- **tokens** – batch, max_seq_len, hidden_size
- **tokens_mask** – batch, max_seq_len (1 for padding, 0 for true)

**Output:**

**tokens_encoded: batch, max_seq_len, hidden_size * num_layers if** concat_layers = True else batch, max_seq_len, hidden_size

### pytext.models.representations.traced_transformer_encoder module

**class** pytext.models.representations.traced_transformer_encoder.**TraceableTransformerWrapper**

Bases: torch.nn.modules.module.Module

**forward** (*tokens: torch.Tensor*, *segment_labels: torch.Tensor = None*, *positions: torch.Tensor = None*, *token_embeddings: torch.Tensor = None*, *attn_mask: torch.Tensor = None*) → Tuple[torch.Tensor, torch.Tensor]
Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.representations.traced_transformer_encoder.**TracedTransformerEncoder**(*eag fair to- ken torc seg- mer torc = Nor po- si- tion torc = Nor to- ken torc = Nor attn torc = Nor*

Bases: torch.nn.modules.module.Module

**forward**(*tokens: torch.Tensor*, *segment_labels: torch.Tensor = None*, *positions: torch.Tensor = None*, *token_embeddings: torch.Tensor = None*, *attn_mask: torch.Tensor = None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.models.representations.transformer_sentence_encoder module

**class** pytext.models.representations.transformer_sentence_encoder.**TransformerSentenceEncoder**

Bases: *pytext.models.representations.transformer_sentence_encoder_base.*

*TransformerSentenceEncoderBase*

Implementation of the Transformer Sentence Encoder. This directly makes use of the TransformerSentenceEncoder module in Fairseq.

**A few interesting config options:**

- encoder_normalize_before detemines whether the layer norm is applied before or after self_attention. This is similar to original implementation from Google.
- activation_fn can be set to 'gelu' instead of the default of 'relu'.

**load_state_dict**(*state_dict*)

Copies parameters and buffers from `state_dict` into this module and its descendants. If `strict` is `True`, then the keys of `state_dict` must exactly match the keys returned by this module's `state_dict()` function.

**Parameters**

- **state_dict** (*dict*) – a dict containing parameters and persistent buffers.
- **strict** (*bool, optional*) – whether to strictly enforce that the keys in `state_dict` match the keys returned by this module's `state_dict()` function. Default: `True`

**Returns**

- **missing_keys** is a list of str containing the missing keys
- **unexpected_keys** is a list of str containing the unexpected keys

**Return type** `NamedTuple` with `missing_keys` and `unexpected_keys` fields

**upgrade_state_dict_named**(*state_dict*)

### pytext.models.representations.transformer_sentence_encoder_base module

**class** pytext.models.representations.transformer_sentence_encoder_base.**PoolingMethod**

Bases: `enum.Enum`

Pooling Methods are chosen from the "Feature-based Approachs" section in https://arxiv.org/pdf/1810.04805.pdf

**AVG_CONCAT_LAST_4_LAYERS = 'avg_concat_last_4_layers'**

**AVG_LAST_LAYER = 'avg_last_layer'**

**AVG_SECOND_TO_LAST_LAYER = 'avg_second_to_last_layer'**

**AVG_SUM_LAST_4_LAYERS = 'avg_sum_last_4_layers'**

**CLS_TOKEN = 'cls_token'**

**NO_POOL = 'no_pool'**

**class** pytext.models.representations.transformer_sentence_encoder_base.**TransformerSentenceEn**

Bases: *pytext.models.representations.representation_base.RepresentationBase*

Base class for all Bi-directional Transformer based Sentence Encoders. All children of this class should implement an _encoder function which takes as input: tokens, [optional] segment labels and a pad mask and outputs both the sentence representation (output of _pool_encoded_layers) and the output states of all the intermediate Transformer layers as a list of tensors.

Input tuple consists of the following elements: 1) tokens: torch tensor of size B x T which contains tokens ids 2) pad_mask: torch tensor of size B x T generated with the condition tokens != self.vocab.get_pad_index() 3) segment_labels: torch tensor of size B x T which contains the segment id of each token

Output tuple consists of the following elements: 1) encoded_layers: List of torch tensors where each tensor has shape B x T x C and there are num_transformer_layers + 1 of these. Each tensor represents the output of the intermediate transformer layers with the 0th element being the input to the first transformer layer (token + segment + position emebdding). 2) [Optional] pooled_output: Output of the pooling operation associated with config.pooling_method to the encoded_layers. Size B x C (or B x 4C if pooling = AVG_CONCAT_LAST_4_LAYERS)

**forward**(*input_tuple: Tuple[torch.Tensor, ...], *args*) → Tuple[torch.Tensor, ...]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.representations.transformer_sentence_encoder_base.TransformerSente output_encoded_layers=False, *args, **kwargs*)

### Module contents

### pytext.models.semantic_parsers package

### Subpackages

### pytext.models.semantic_parsers.rnng package

### Submodules

### pytext.models.semantic_parsers.rnng.rnng_constant module

### pytext.models.semantic_parsers.rnng.rnng_data_structures module

**class** pytext.models.semantic_parsers.rnng.rnng_data_structures.**CompositionalNN**(*lstm_dim: int*)

Bases: `torch.jit._script.ScriptModule`

Combines a list / sequence of embeddings into one using a biLSTM

**class** pytext.models.semantic_parsers.rnng.rnng_data_structures.**CompositionalSummationNN**(*lstm int*)

Bases: `torch.jit._script.ScriptModule`

Simpler version of CompositionalNN

**class** pytext.models.semantic_parsers.rnng.rnng_data_structures.**Element**(*node: Any*)

Bases: object

Generic element representing a token / non-terminal / sub-tree on a stack. Used to compute valid actions in the RNNG parser.

**class** pytext.models.semantic_parsers.rnng.rnng_data_structures.**ParserState**(*parser=None*)

Bases: object

Maintains state of the Parser. Useful for beam search

**copy**()

**finished**()

**class** pytext.models.semantic_parsers.rnng.rnng_data_structures.**StackLSTM**(*lstm: torch.nn.modules.rnn.LST*

Bases: collections.abc.Sized, typing.Generic

The Stack LSTM from Dyer et al: https://arxiv.org/abs/1505.08075

**copy**()

**element_from_top**(*index: int*) → pytext.models.semantic_parsers.rnng.rnng_data_structures.Element

**embedding**() → torch.Tensor

**Shapes:** return value: (1, lstm_hidden_dim)

**pop**() → Tuple[torch.Tensor, pytext.models.semantic_parsers.rnng.rnng_data_structures.Element]

Pops and returns tuple of output embedding (1, lstm_hidden_dim) and element

**push**(*expression: torch.Tensor*, *element: pytext.models.semantic_parsers.rnng.rnng_data_structures.Element*) → None

**Shapes:** expression: (1, lstm_input_dim)

### pytext.models.semantic_parsers.rnng.rnng_parser module

**class** pytext.models.semantic_parsers.rnng.rnng_parser.**RNNGParser**(*ablation: pytext.models.semantic_parsers.rnng.rn constraints: pytext.models.semantic_parsers.rnng.rn lstm_num_layers: int, lstm_dim: int, max_open_NT: int, dropout: float, actions_vocab, shift_idx: int, reduce_idx: int, ignore_subNTs_roots: List[int], valid_NT_idxs: List[int], valid_IN_idxs: List[int], valid_SL_idxs: List[int], embedding: pytext.models.embeddings.embedding_li p_compositional*)

Bases: *pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserBase*

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**get_export_input_names**(*tensorizers*)

**get_export_output_names**(*tensorizers*)

**vocab_to_export**(*tensorizers*)

**class** pytext.models.semantic_parsers.rnng.rnng_parser.**RNNGParserBase**(*ablation: pytext.models.semantic_parsers.rn con- straints: pytext.models.semantic_parsers.rn lstm_num_layers: int, lstm_dim: int, max_open_NT: int, dropout: float, ac- tions_vocab, shift_idx: int, re- duce_idx: int, ig- nore_subNTs_roots: List[int], valid_NT_idxs: List[int], valid_IN_idxs: List[int], valid_SL_idxs: List[int], embed- ding: pytext.models.embeddings.embedd p_compositional*)

Bases: `pytext.models.model.BaseModel`

The Recurrent Neural Network Grammar (RNNG) parser from Dyer et al.: https://arxiv.org/abs/1602.07776 and Gupta et al.: https://arxiv.org/abs/1810.07942d. RNNG is a neural constituency parsing algorithm that explicitly models compositional structure of a sentence. It is able to learn about hierarchical relationship among the words and phrases in a given sentence thereby learning the underlying tree structure. The paper proposes generative as well as discriminative approaches. In PyText we have implemented the discriminative approach for modeling intent slot models. It is a top-down shift-reduce parser than can output trees with non-terminals (intent and slot labels) and terminals (tokens)

**contextualize**(*context*)

Add additional context into model. *context* can be anything that helps maintaining/updating state. For example, it is used by `DisjointMultitaskModel` for changing the task that should be trained with a given iterator.

**forward**(*tokens: torch.Tensor, seq_lens: torch.Tensor, dict_feat: Optional[Tuple[torch.Tensor, ...]] = None, actions: Optional[List[List[int]]] = None, contextual_token_embeddings: Optional[torch.Tensor] = None, beam_size=1, top_k=1*) → List[Tuple[torch.Tensor, torch.Tensor]]

RNNG forward function.

**Parameters**

- **tokens** (*torch.Tensor*) – list of tokens
- **seq_lens** (*torch.Tensor*) – list of sequence lengths
- **dict_feat** (*Optional[Tuple[torch.Tensor, ..]]*) – dictionary or gazetteer features for each token
- **actions** (*Optional[List[List[int]]]*) – Used only during training. Oracle actions for the instances.

**Returns** list of top k tuple of predicted actions tensor and corresponding scores tensor. Tensor shape: (batch_size, action_length) (batch_size, action_length, number_of_actions)

**classmethod from_config**(*model_config, feature_config=None, metadata: pytext.data.data_handler.CommonMetadata = None, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer] = None*)

**get_loss**(*logits: List[Tuple[torch.Tensor, torch.Tensor]], target_actions: torch.Tensor, context: torch.Tensor*)

**Shapes:** logits[1]: action scores: (1, action_length, number_of_actions) target_actions: (1, action_length)

**get_pred**(*logits: List[Tuple[torch.Tensor, torch.Tensor]], context=None, *args*)

**Return Shapes:** preds: batch (1) * topk * action_len scores: batch (1) * topk * (action_len * number_of_actions)

**get_single_pred**(*logits: Tuple[torch.Tensor, torch.Tensor], *args*)

**push_action**(*state: pytext.models.semantic_parsers.rnng.rnng_data_structures.ParserState, target_action_idx: int*) → None

Used for updating the state with a target next action

**Parameters**

- **state** (*ParserState*) – The state of the stack, buffer and action
- **target_action_idx** (*int*) – Index of the action to process

**save_modules**(**args, **kwargs*)

Save each sub-module in separate files for reusing later.

**valid_actions**(*state: pytext.models.semantic_parsers.rnng.rnng_data_structures.ParserState*) → List[int]

Used for restricting the set of possible action predictions

**Parameters** **state** (*ParserState*) – The state of the stack, buffer and action

**Returns** indices of the valid actions

**Return type** List[int]

### Module contents

### Module contents

### pytext.models.seq_models package

### Submodules

### pytext.models.seq_models.attention module

**class** pytext.models.seq_models.attention.**DecoupledMultiheadAttention**(*embed_dim: int, context_dim: int, num_heads: int, dropout: float, unseen_mask=False, src_length_mask=True*)

Bases: torch.nn.modules.module.Module

Multiheaded Scaled Dot Product Attention. This function has the same exact signature as the one used in pytorch_translate with the added benefit of supporting torchscript

**forward**(*decoder_state: torch.Tensor, source_hids: torch.Tensor, src_len_mask: Optional[torch.Tensor], squeeze: bool = True*) → Tuple[torch.Tensor, torch.Tensor]

Computes MultiheadAttention with respect to either a vector or a tensor

**Inputs:**

**decoder_state: (bsz x decoder_hidden_state_dim) or** (bsz x T x decoder_hidden_state_dim)

source_hids: srclen x bsz x context_dim src_lengths: bsz x 1, actual sequence lengths squeeze: Whether or not to squeeze on the time dimension.

Even if decoder_state.dim() is 2 dimensional an explicit time step dimension will be unsqueezed.

**Outputs:**

**[batch_size, max_src_len] if decoder_state.dim() == 2 & squeeze** or

**[batch_size, 1, max_src_len] if decoder_state.dim() == 2 & !squeeze** or

**[batch_size, T, max_src_len] if decoder_state.dim() == 3 & !squeeze** or

**[batch_size, T, max_src_len] if decoder_state.dim() == 3 & squeeze & T != 1** or

[batch_size, max_src_len] if decoder_state.dim() == 3 & squeeze & T == 1

**class** pytext.models.seq_models.attention.**DotAttention**(*decoder_hidden_state_dim, context_dim, force_projection=False, src_length_masking=True*)

Bases: torch.nn.modules.module.Module

**forward**(*decoder_state, source_hids, src_lengths*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.seq_models.attention.**MultiheadAttention**(*embed_dim*, *num_heads*, *dropout*, *kdim=None*, *vdim=None*, *bias=True*)

Bases: *pytext.models.seq_models.base.PyTextIncrementalDecoderComponent*

Refer Attention is All You Need for more details.

This is a simplified implementation of multihead attention optimized for exporting using torchscript. Usage of nn.Linear() instead of F.Linear() helps to quantize the linear layers.

Query represents the output from last decoder step. Key and Values are obtained from encoder. Attention weights are obtained from the dot product of query and key. Attention weights multiplied by the value gives output.

**forward**(*query: torch.Tensor, key: torch.Tensor, value: torch.Tensor, key_padding_mask: Optional[torch.Tensor], need_weights: bool, incremental_state: Optional[Dict[str, torch.Tensor]] = None*) → Tuple[torch.Tensor, Optional[torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config*, *embed_dim*, *num_heads*)

**reorder_incremental_state**(*incremental_state: Dict[str, torch.Tensor], new_order: torch.Tensor*)

Reorder buffered internal state (for incremental generation).

pytext.models.seq_models.attention.**create_src_lengths_mask**(*batch_size: int, src_lengths*)

Generate boolean mask to prevent attention beyond the end of source

**Inputs:** batch_size : int src_lengths : [batch_size] of sentence lengths

**Outputs:** [batch_size, max_src_len]

pytext.models.seq_models.attention.**masked_softmax**(*scores, src_lengths, src_length_masking: bool = True*)

Apply source length masking then softmax. Input and output have shape bsz x src_len

### pytext.models.seq_models.base module

**class** pytext.models.seq_models.base.**PlaceholderAttentionIdentity**

Bases: `torch.nn.modules.module.Module`

**forward**(*query, key, value, need_weights: bool = None, key_padding_mask: Optional[torch.Tensor] = None, incremental_state: Optional[Dict[str, torch.Tensor]] = None*) → Tuple[torch.Tensor, Optional[torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**reorder_incremental_state**(*incremental_state: Dict[str, torch.Tensor], new_order: torch.Tensor*)

**class** pytext.models.seq_models.base.**PlaceholderIdentity**

Bases: `torch.nn.modules.module.Module`

**class Config**(***kwargs*)

Bases: `pytext.config.module_config.Module.Config`

**forward**(*x, incremental_state: Optional[Dict[str, torch.Tensor]] = None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.models.seq_models.base.**PyTextIncrementalDecoderComponent**

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*

**get_incremental_state**(*incremental_state: Dict[str, torch.Tensor], key: str*) → Optional[torch.Tensor]

Helper for getting incremental state for an nn.Module.

**reorder_incremental_state**(*incremental_state: Dict[str, torch.Tensor], new_order: torch.Tensor*)

**set_incremental_state**(*incremental_state: Dict[str, torch.Tensor], key: str, value*)

Helper for setting incremental state for an nn.Module.

**class** pytext.models.seq_models.base.**PyTextSeq2SeqModule**

Bases: *pytext.models.module.Module*

**assign_id**()

**instance_id = None**

### pytext.models.seq_models.contextual_intent_slot module

**class** pytext.models.seq_models.contextual_intent_slot.**ContextualIntentSlotModel**(*default_doc_los de- fault_word_loss *args, **kwargs*)

Bases: *pytext.models.joint_model.IntentSlotModel*

Joint Model for Intent classification and slot tagging with inputs of contextual information (sequence of utterances) and dictionary feature of the last utterance.

Training data should include: doc_label (string): intent classification label of either the sequence of utterances or just the last sentence word_label (string): slot tagging label of the last utterance in the format of

start_idx:end_idx:slot_label, multiple slots are separated by a comma text (list of string): sequence of utterances for training dict_feat (dict): a dict of features that contains the feature of each word in the last utterance

Following is an example of raw columns from training data:

| | |
|---|---|
| doc_label | reply-where |
| word_label | 10:20:restaurant_name |
| text | ["dinner at 6?", "wanna try Tomi Sushi?"] |
| dict_feat | **{"tokenFeatList": [{"tokenIdx": 2, "features": {"poi:eatery": 0.66}** {"tokenIdx": 3, "features": {"poi:eatery": 0.66}}]} |

**arrange_model_inputs**(*tensor_dict*)

**classmethod create_embedding**(*config*, *tensorizers*)

**get_export_input_names**(*tensorizers*)

**vocab_to_export**(*tensorizers*)

### pytext.models.seq_models.conv_decoder module

**class** pytext.models.seq_models.conv_decoder.**ConvDecoderConfig**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

**combine_pos_embed = 'concat'**

**decoder_embed_dim = 128**

**decoder_input_dim = 128**

**decoder_learned_pos = False**

**decoder_normalize_before = False**

**decoder_output_dim = 128**

**dropout = 0.1**

**max_target_positions = 128**

**no_token_positional_embeddings = False**

**positional_embedding_type = 'learned'**

**class** pytext.models.seq_models.conv_decoder.**LightConvDecoder**(*target_dict*, *embed_tokens*, *layers*, *decoder_config*)

Bases: *pytext.models.seq_models.conv_decoder.LightConvDecoderBase*

**forward**(*prev_output_tokens: torch.Tensor, encoder_out: Dict[str, torch.Tensor], incremental_state: Optional[Dict[str, torch.Tensor]] = None, timestep: Optional[int] = None*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**get_probs**(*decoder_out: Tuple[torch.Tensor, Dict[str, torch.Tensor]]*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

**class** pytext.models.seq_models.conv_decoder.**LightConvDecoderBase**(*target_dict, embed_tokens, layers, decoder_config*)

Bases: *pytext.models.seq_models.base.PyTextIncrementalDecoderComponent*

**forward_unprojected**(*prev_output_tokens: torch.Tensor, encoder_out: Dict[str, torch.Tensor], incremental_state: Optional[Dict[str, torch.Tensor]] = None, timestep: Optional[int] = None*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

**classmethod from_config**(*config, tgt_dict, tgt_embedding*)

**get_probs**(*decoder_out: Tuple[torch.Tensor, Dict[str, torch.Tensor]]*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

**max_positions**()

Maximum output length supported by the decoder.

**pos_embed**(*x, src_tokens*)

**reorder_incremental_state**(*incremental_state: Dict[str, torch.Tensor], new_order: torch.Tensor*)

**class** pytext.models.seq_models.conv_decoder.**LightConvDecoderLayer**(*attention_dropout*, *decoder_attention_heads*, *self_attention_heads*, *decoder_conv_dim*, *decoder_conv_type*, *attention_type*, *self_attention_type*, *decoder_embed_dim*, *decoder_ffn_embed_dim*, *decoder_glu*, *decoder_normalize_before*, *dropout*, *input_dropout*, *relu_dropout*, *need_attention*, *convolution_type*, *conv=None*, *self_attention=None*, *attention=None*)

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*

**extra_repr**()

Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

**forward**(*x, encoder_out: torch.Tensor, encoder_padding_mask: Optional[torch.Tensor], decoder_padding_mask: Optional[torch.Tensor], incremental_state: Optional[Dict[str, torch.Tensor]]*)

**Parameters**

- **x** (*Tensor*) – input to the layer of shape *(seq_len, batch, embed_dim)*
- **encoder_padding_mask** (*ByteTensor*) – binary ByteTensor of shape *(batch, src_len)* where padding elements are indicated by `1`.

**Returns** encoded output of shape *(batch, src_len, embed_dim)*

**classmethod from_config**(*config*, *kernel_size*)

**maybe_layer_norm**(*before: bool = False*, *after: bool = False*)

This a utility function which helps to control the layer norm behavior *before* and *after* specific components using one variable in config. If self.normalize_before is set to True, output is true only when *before* is True

**reorder_incremental_state**(*incremental_state: Dict[str, torch.Tensor], new_order: torch.Tensor*)

**class** pytext.models.seq_models.conv_decoder.**LightConvDecoupledDecoder**(*target_dict, embed_tokens, layers, decoder_config, ontology_generation_only, decoupled_attention_heads, model_output_logprob*)

Bases: *pytext.models.seq_models.conv_decoder.LightConvDecoderBase*

**forward**(*prev_output_tokens: torch.Tensor, encoder_out: Dict[str, torch.Tensor], incremental_state: Optional[Dict[str, torch.Tensor]] = None, timestep: Optional[int] = None*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config, tgt_dict, tgt_embedding*)

### pytext.models.seq_models.conv_encoder module

**class** pytext.models.seq_models.conv_encoder.**ConvEncoderConfig**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

**combine_pos_embed = 'concat'**

**dropout = 0.1**

**embedding_dim = 128**

**encoder_embed_dim = 128**

**encoder_learned_pos = False**

**encoder_normalize_before = False**

**max_source_positions = 1024**

**max_target_positions = 100**

**no_token_positional_embeddings = False**

**positional_embedding_type = 'learned'**

**class** pytext.models.seq_models.conv_encoder.**LightConvEncoder**(*dictionary, embed_tokens, layers, encoder_config*)

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*, *pytext.models.seq_models.nar_modules.NAREncoderUtility*

**extra_repr**()
Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

**forward**(*src_tokens: torch.Tensor*, *src_embeddings: torch.Tensor*, *src_lengths: torch.Tensor*) → Dict[str, torch.Tensor]
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config*, *src_dict*, *src_embedding*)

**max_positions**()
Maximum input length supported by the encoder.

**pos_embed**(*x*, *src_tokens*)

**reorder_encoder_out**(*encoder_out: Dict[str, torch.Tensor], new_order: torch.Tensor*)

**tile_encoder_out**(*tile_size: int, encoder_out: Dict[str, torch.Tensor]*) → Dict[str, torch.Tensor]

**class** pytext.models.seq_models.conv_encoder.**LightConvEncoderLayer**(*dropout*, *encoder_conv_dim*, *encoder_conv_type*, *self_attention_type*, *encoder_embed_dim*, *encoder_ffn_embed_dim*, *self_attention_heads*, *encoder_glu*, *encoder_normalize_before*, *input_dropout*, *relu_dropout*, *convolution_type*, *conv=None*, *self_attention=None*)

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*

**extra_repr**()
Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

**forward**(*x*, *encoder_padding_mask: Optional[torch.Tensor] = None*)

**Parameters**

- **x** (*Tensor*) – input to the layer of shape *(seq_len, batch, embed_dim)*
- **encoder_padding_mask** (*ByteTensor*) – binary ByteTensor of shape *(batch, src_len)* where padding elements are indicated by 1.

**Returns** encoded output of shape *(batch, src_len, embed_dim)*

**classmethod from_config**(*config*, *kernel_size*)

**maybe_layer_norm**(*before: bool = False*, *after: bool = False*)

### pytext.models.seq_models.conv_model module

**class** pytext.models.seq_models.conv_model.**CNNModel**(*encoder*, *decoder*, *source_embedding*)

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*

**forward**(*src_tokens: torch.Tensor, additional_features: List[List[torch.Tensor]], src_lengths, prev_output_tokens, src_subword_begin_indices: Optional[torch.Tensor] = None*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.seq_models.conv_model.CNNModel.Config, src_dict, source_embedding, tgt_dict, target_embedding, dict_embedding=None*)

**get_embedding_module**()

**get_normalized_probs**(*net_output*, *log_probs*, *sample=None*)

**max_decoder_positions**()

**classmethod validate_config**(*config*)

**class** pytext.models.seq_models.conv_model.**DecoupledCNNModel**(*encoder*, *decoder*, *source_embedding*)

Bases: *pytext.models.seq_models.conv_model.CNNModel*

### pytext.models.seq_models.light_conv module

**class** pytext.models.seq_models.light_conv.**LightweightConv**(*input_size*, *kernel_size*, *convolution_type: str*, *num_heads*, *weight_softmax*, *bias*)

Bases: *pytext.models.seq_models.base.PyTextIncrementalDecoderComponent*

**extra_repr**()

Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

**forward**(*x*, *incremental_state: Optional[Dict[str, torch.Tensor]] = None*)

Assuming the input, x, of the shape T x B x C and producing an output in the shape T x B x C :param x: Input of shape T x B x C, i.e. (timesteps, batch_size, input_size) :param incremental_state: A dict to keep the state

**classmethod from_config**(*config*, *input_size*, *kernel_size*, *convolution_type*)

**reorder_incremental_state**(*incremental_state: Dict[str, torch.Tensor], new_order: torch.Tensor*)

**reset_parameters**()

### pytext.models.seq_models.mask_generator module

**class** pytext.models.seq_models.mask_generator.**BeamRankingAlgorithm**

Bases: enum.Enum

An enumeration.

**AVERAGE_TOKEN_LPROB = 'AVERAGE_TOKEN_LPROB'**

**LENGTH_CONDITIONED_AVERAGE_TOKEN_LPROB = 'LENGTH_CONDITIONED_AVERAGE_TOKEN_LPROB'**

**LENGTH_CONDITIONED_AVERAGE_TOKEN_LPROB_MULTIPLIED = 'LENGTH_CONDITIONED_AVERAGE_TOKEN_**

**LENGTH_CONDITIONED_RANK = 'LENGTH_CONDITIONED_RANK'**

**LENGTH_CONDITIONED_RANK_MUL = 'LENGTH_CONDITIONED_RANK_MUL'**

**LEN_ONLY = 'LEN_ONLY'**

**TOKEN_LPROB = 'TOKEN_LPROB'**

**class** pytext.models.seq_models.mask_generator.**EmbedQuantizeType**

Bases: enum.Enum

An enumeration.

**BIT_4 = '4bit'**

**BIT_8 = '8bit'**

**NONE = 'None'**

**class** pytext.models.seq_models.mask_generator.**MaskedSequenceGenerator**(*config*, *model*, *length_prediction_model*, *trg_vocab*, *beam_size*, *use_gold_length*, *beam_ranking_algorithm*, *quantize*, *embed_quantize*)

Bases: *pytext.models.module.Module*

**forward**(*src_tokens: torch.Tensor, dict_feats: Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], contextual_embed: Optional[torch.Tensor], char_feats: Optional[torch.Tensor], src_lengths: torch.Tensor, src_subword_begin_indices: Optional[torch.Tensor] = None, target_lengths: Optional[torch.Tensor] = None, beam_size: Optional[int] = None, src_index_tokens: Optional[torch.Tensor] = None*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config*, *model*, *length_prediction*, *trg_vocab*, *quantize=False*, *embed_quantize=False*)

**generate_hypo**(*tensors: Dict[str, torch.Tensor]*) → Tuple[Tuple[torch.Tensor, torch.Tensor], torch.Tensor]

Generates hypotheses using beam search, also returning their scores

**Inputs:**

- tensors: dictionary containing needed tensors for generation

**Outputs:**

- **(hypos, lens): tuple of Tensors**
  - hypos: Tensor of shape (batch_size, beam_size, MAX) containing the generated tokens. MAX refers to the longest sequence in batch.
  - lens: Tensor of shape (batch_size, beam_size) containing generated sequence lengths
- _hypo_scores: Tensor of shape (batch_size, beam_size) containing the scores for each generated sequence

**generate_non_autoregressive**(*encoder_out: Dict[str, torch.Tensor], tgt_tokens*)

**get_clip_length**(*src_lengths: torch.Tensor*)

**get_encoder_out**(*src_tokens: torch.Tensor, dict_feats: Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], contextual_embed: Optional[torch.Tensor], char_feats: Optional[torch.Tensor], src_subword_begin_indices: Optional[torch.Tensor], src_lengths: torch.Tensor, src_index_tokens: Optional[torch.Tensor] = None*) → Dict[str, torch.Tensor]

pytext.models.seq_models.mask_generator.**avg_token_lprob**(*token_lprob: torch.Tensor, length_lprob: torch.Tensor, target_lengths: torch.Tensor*) → torch.Tensor

pytext.models.seq_models.mask_generator.**get_beam_ranking_function**(*ranking_algorithm: pytext.models.seq_models.mask_genera*

pytext.models.seq_models.mask_generator.**length**(*token_lprob: torch.Tensor, length_lprob: torch.Tensor, target_lengths: torch.Tensor*) → torch.Tensor

pytext.models.seq_models.mask_generator.**length_conditioned_avg_lprob_rank**(*token_lprob: torch.Tensor, length_lprob: torch.Tensor, target_lengths: torch.Tensor*) → torch.Tensor

pytext.models.seq_models.mask_generator.**length_conditioned_avg_lprob_rank_mul**(*token_lprob: torch.Tensor, length_lprob: torch.Tensor, target_lengths: torch.Tensor*) → torch.Tensor

pytext.models.seq_models.mask_generator.**length_conditioned_rank**(*token_lprob: torch.Tensor, length_lprob: torch.Tensor, target_lengths: torch.Tensor*) → torch.Tensor

pytext.models.seq_models.mask_generator.**length_conditioned_rank_mul**(*token_lprob: torch.Tensor, length_lprob: torch.Tensor, target_lengths: torch.Tensor*) → torch.Tensor

pytext.models.seq_models.mask_generator.**prepare_masked_target_for_lengths**(*beam: torch.Tensor, mask_idx: int, pad_idx: int, length_beam_size: int = 1*) → Tuple[torch.Tensor, torch.Tensor]

pytext.models.seq_models.mask_generator.**token_prob**(*token_lprob: torch.Tensor, length_lprob: torch.Tensor, target_lengths: torch.Tensor*) → torch.Tensor

### pytext.models.seq_models.nar_length module

**class** pytext.models.seq_models.nar_length.**ConvLengthPredictionModule**(*embed_dim: int, conv_dim: int, max_target_positions: int, length_dropout: float, glu: bool, activation, pooling_type, conv_layers*)

Bases: *pytext.models.module.Module*

**forward**(*x: torch.Tensor, encoder_padding_mask: Optional[torch.Tensor] = None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.seq_models.nar_length.ConvLengthPredictionModule.Config, embed_dim: int*)

**class** pytext.models.seq_models.nar_length.**MaskedLengthPredictionModule**(*embed_dim: int, length_hidden_dim: int, max_target_positions: int, length_dropout: float*)

Bases: *pytext.models.module.Module*

**forward**(*x: torch.Tensor, encoder_padding_mask: Optional[torch.Tensor] = None*) → Tuple[torch.Tensor, torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.seq_models.nar_length.MaskedLengthPredictionModule.Config, embed_dim: int*)

pytext.models.seq_models.nar_length.**mean**(*rep: torch.Tensor, padding_mask: Optional[torch.Tensor]*)

pytext.models.seq_models.nar_length.**pool**(*pooling_type: str, words: torch.Tensor, encoder_padding_mask: Optional[torch.Tensor]*)

### pytext.models.seq_models.nar_modules module

**class** pytext.models.seq_models.nar_modules.**NAREncoderUtility**

Bases: object

**prepare_for_nar_inference**(*length_beam_size: int, encoder_out: Dict[str, torch.Tensor]*) → Dict[str, torch.Tensor]

During masked NAR inference, multiple lengths are predicted for each item in the batch. Hence tiling has to be done in such a way that all new rows related to each item should be placed together. This is the assumption that we are following in the rest of the nar generation code. Eg: [row1, row2, row3] should be tiled as [row1, row1, row1, row2, row2, row2, row3, row3, row3] NOT [row1, row2, row3, row1, row2, row3, row1, row2, row3]

### pytext.models.seq_models.nar_output_layer module

**class** pytext.models.seq_models.nar_output_layer.**NARSeq2SeqOutputLayer**(*target_names: Optional[List[str]] = None, loss_fn: Optional[pytext.loss.loss.Loss] = None, *args, **kwargs*)

Bases: *pytext.models.output_layers.output_layer_base.OutputLayerBase*

Non-autoregressive seq2seq output layer.

**classmethod from_config**(*config: pytext.models.seq_models.nar_output_layer.NARSeq2SeqOutputLayer.Config, vocab: pytext.data.utils.Vocabulary*)

**get_loss**(*model_outputs: Tuple[torch.Tensor, Dict[str, torch.Tensor]], targets: Tuple[Tuple[torch.Tensor, torch.Tensor], torch.Tensor], context: Dict[str, Any] = None, reduce=True*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

label_logits: B x T x V_1 label_targets: B x T length_logits: B x V_2 length_targets: B

### pytext.models.seq_models.positional module

**class** pytext.models.seq_models.positional.**LearnedPositionalEmbedding**(*num_embeddings: int, embedding_dim: int, padding_idx: int*)

Bases: torch.nn.modules.sparse.Embedding

This module learns positional embeddings up to a fixed maximum size. Padding ids are ignored by either offsetting based on padding_idx or by setting padding_idx to None and ensuring that the appropriate position ids are passed to the forward function.

**forward**(*input: torch.Tensor, incremental_state: Optional[Dict[str, Dict[str, Optional[torch.Tensor]]]] = None, positions: Optional[torch.Tensor] = None*)

Input is expected to be of size [bsz x seqlen].

**class** pytext.models.seq_models.positional.**PostionalEmbedCombine**

Bases: enum.Enum

An enumeration.

**CONCAT = 'concat'**

**SUM = 'sum'**

**class** pytext.models.seq_models.positional.**PostionalEmbedType**

Bases: enum.Enum

An enumeration.

**HYBRID = 'hybrid'**

**LEARNED = 'learned'**

**SINUSOIDAL = 'sinusoidal'**

**class** pytext.models.seq_models.positional.**SinusoidalPositionalEmbedding**(*embedding_dim, padding_idx, init_size=124, learned_embed=False*)

Bases: torch.nn.modules.module.Module

This module produces sinusoidal positional embeddings of any length.

Padding symbols are ignored.

**forward**(*input, incremental_state: Optional[Dict[str, torch.Tensor]] = None, timestep: Optional[int] = None*)

Input is expected to be of size [bsz x seqlen].

**max_positions**()

Maximum number of supported positions.

pytext.models.seq_models.positional.**build_positional_embedding**(*positional_embedding_type: pytext.models.seq_models.positional.Postio combine_pos_embed: pytext.models.seq_models.positional.Postio max_target_positions: int, input_embed_dim: int, embed_dim: int, padding_idx: int, no_token_positional_embeddings: bool*)

pytext.models.seq_models.positional.**get_sinusoidal_embedding**(*num_embeddings: int, embedding_dim: int, padding_idx: int*)

Build sinusoidal embeddings.

This matches the implementation in tensor2tensor, but differs slightly from the description in Section 3.5 of "Attention Is All You Need".

### pytext.models.seq_models.projection_layers module

**class** pytext.models.seq_models.projection_layers.**DecoderWithLinearOutputProjection**(*src_dict, dst_dict, out_embed, *args, **kwargs*)

Bases: torch.nn.modules.module.Module

Simple linear projection from the hidden vector to vocab.

**forward**(*encoder_out: Dict[str, torch.Tensor], decoder_out: Tuple[torch.Tensor, Dict[str, torch.Tensor]], incremental_state: Optional[Dict[str, torch.Tensor]] = None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_probs**(*decoder_out: Tuple[torch.Tensor, Dict[str, torch.Tensor]]*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

**reset_parameters**()

**class** pytext.models.seq_models.projection_layers.**DecoupledDecoderHead**(*src_dict, dst_dict, out_embed_dim=512, encoder_hidden_dim=None, pointer_attention_heads=1, fixed_generation_vocab=None, attention_dropout=0.2, model_output_logprob=True*)

Bases: torch.nn.modules.module.Module

**fixed_generation_vocab_expanded = typing_extensions.Final[torch.Tensor]**

**forward**(*encoder_out: Dict[str, torch.Tensor], decoder_out: Tuple[torch.Tensor, Dict[str, torch.Tensor]], incremental_state: Optional[Dict[str, torch.Tensor]] = None*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

B: Batch T_src: Length of source sequence T_trg: Length of target seuqence C: hidden dimension V_ont: Size of ontology vocabulary V_trg: Size of full target vocabulary

**get_pointer_src_tokens**(*encoder_out: Dict[str, torch.Tensor]*) → torch.Tensor

**get_probs**(*decoder_out: Tuple[torch.Tensor, Dict[str, torch.Tensor]]*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

**verify_encoder_out**(*encoder_out: Dict[str, torch.Tensor]*)

### pytext.models.seq_models.rnn_decoder module

**class** pytext.models.seq_models.rnn_decoder.**DecoderWithLinearOutputProjection**(*out_vocab_size, out_embed_dim=51*

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*

Common super class for decoder networks with output projection layers.

**forward**(*input_tokens, encoder_out: Dict[str, torch.Tensor], incremental_state: Optional[Dict[str, torch.Tensor]] = None, timestep: int = 0*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**forward_unprojected**(*input_tokens, encoder_out, incremental_state=None*)

Forward pass through the decoder without output projection.

**reset_parameters**()

**class** pytext.models.seq_models.rnn_decoder.**RNNDecoder**(*out_vocab_size, embed_tokens, encoder_hidden_dim, embed_dim, hidden_dim, out_embed_dim, cell_type, num_layers, dropout_in, dropout_out, attention_type, attention_heads, first_layer_attention, averaging_encoder*)

Bases: *pytext.models.seq_models.rnn_decoder.RNNDecoderBase*, *pytext.models.seq_models.rnn_decoder.DecoderWithLinearOutputProjection*

**class** pytext.models.seq_models.rnn_decoder.**RNNDecoderBase**(*embed_tokens, encoder_hidden_dim, embed_dim, hidden_dim, out_embed_dim, cell_type, num_layers, dropout_in, dropout_out, attention_type, attention_heads, first_layer_attention, averaging_encoder*)

Bases: *pytext.models.seq_models.base.PyTextIncrementalDecoderComponent*

RNN decoder with multihead attention. Attention is calculated using encoder output and output of decoder's first RNN layerself. Attention is applied after first RNN layer and concatenated to input of subsequent layers.

**forward_unprojected**(*input_tokens, encoder_out: Dict[str, torch.Tensor], incremental_state: Optional[Dict[str, torch.Tensor]] = None*) → Tuple[torch.Tensor, Dict[str, torch.Tensor]]

**classmethod from_config**(*config*, *out_vocab_size*, *target_embedding*)

**get_normalized_probs**(*net_output*, *log_probs*, *sample*)
Get normalized probabilities (or log probs) from a net's output.

**max_positions**()
Maximum output length supported by the decoder.

**reorder_incremental_state**(*incremental_state: Dict[str, torch.Tensor], new_order*)
Reorder buffered internal state (for incremental generation).

### pytext.models.seq_models.rnn_encoder module

**class** pytext.models.seq_models.rnn_encoder.**BiLSTM**(*num_layers*, *bidirectional*, *embed_dim*, *hidden_dim*, *dropout*)

Bases: `torch.nn.modules.module.Module`

Wrapper for nn.LSTM

Differences include: * weight initialization * the bidirectional option makes the first layer bidirectional only (and in that case the hidden dim is divided by 2)

**static LSTM**(*input_size*, *hidden_size*, ***kwargs*)

**forward**(*embeddings: torch.Tensor*, *lengths: torch.Tensor*, *enforce_sorted: bool = True*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.seq_models.rnn_encoder.**LSTMSequenceEncoder**(*embed_dim*, *hidden_dim*, *num_layers*, *dropout_in*, *dropout_out*, *bidirectional*)

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*

RNN encoder using nn.LSTM for cuDNN support / ONNX exportability.

**forward**(*src_tokens: torch.Tensor*, *embeddings: torch.Tensor*, *src_lengths*) → Dict[str, torch.Tensor]
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config*)

**max_positions**()
Maximum output length supported by the decoder.

**tile_encoder_out**(*beam_size: int, encoder_out: Dict[str, torch.Tensor]*) → Dict[str, torch.Tensor]

### pytext.models.seq_models.rnn_encoder_decoder module

**class** pytext.models.seq_models.rnn_encoder_decoder.**RNNModel**(*encoder, decoder, source_embeddings*)

Bases: *pytext.models.seq_models.base.PyTextSeq2SeqModule*

**forward**(*src_tokens: torch.Tensor, additional_features: List[List[torch.Tensor]], src_lengths, prev_output_tokens, incremental_state: Optional[Dict[str, torch.Tensor]] = None*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.seq_models.rnn_encoder_decoder.RNNModel.Config, source_vocab, source_embedding, target_vocab, target_embedding*)

**get_normalized_probs**(*net_output, log_probs, sample=None*)

**max_decoder_positions**()

### pytext.models.seq_models.seq2seq_model module

**class** pytext.models.seq_models.seq2seq_model.**Seq2SeqModel**(*model: pytext.models.seq_models.rnn_encoder_decoder.R output_layer: pytext.models.seq_models.seq2seq_output_layer.Se src_vocab: pytext.data.utils.Vocabulary, trg_vocab: pytext.data.utils.Vocabulary, dictfeat_vocab: pytext.data.utils.Vocabulary, generator_config=None*)

Bases: *pytext.models.model.Model*

Sequence to sequence model using an encoder-decoder architecture.

**arrange_model_inputs**(*tensor_dict*) → Tuple[torch.Tensor, Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], torch.Tensor, torch.Tensor]

**arrange_targets**(*tensor_dict*)

**forward**(*src_tokens: torch.Tensor, dict_feats: Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], contextual_token_embedding: Optional[torch.Tensor], src_lengths: torch.Tensor, trg_tokens: torch.Tensor*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.models.seq_models.seq2seq_model.Seq2SeqModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**get_pred**(*model_outputs*, *context=None*)

**max_decoder_positions**()

**torchscriptify**()

### pytext.models.seq_models.seq2seq_output_layer module

**class** pytext.models.seq_models.seq2seq_output_layer.**Seq2SeqOutputLayer**(*target_names: Optional[List[str]] = None, loss_fn: Optional[pytext.loss.loss.Loss] = None, *args, **kwargs*)

Bases: `pytext.models.output_layers.output_layer_base.OutputLayerBase`

**classmethod from_config**(*config: pytext.models.seq_models.seq2seq_output_layer.Seq2SeqOutputLayer.Config, vocab: pytext.data.utils.Vocabulary*)

**get_loss**(*model_outputs: Tuple[torch.Tensor, Dict[str, torch.Tensor]], targets: Tuple[torch.Tensor, torch.Tensor], context: Dict[str, Any] = None, reduce=True*) → torch.Tensor

Compute and return the loss given logits and targets.

**Parameters**

- **logit** (*torch.Tensor*) – Logits returned `Model`.
- **target** (*torch.Tensor*) – True label/target to compute loss against.
- **context** (*Optional[Dict[str, Any]]*) – Context is a dictionary of items that's passed as additional metadata by the `DataHandler`. Defaults to None.
- **reduce** (*bool*) – Whether to reduce loss over the batch. Defaults to True.

**Returns** Model loss.

**Return type** torch.Tensor

### pytext.models.seq_models.seqnn module

**class** pytext.models.seq_models.seqnn.**SeqNNModel**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase, representation: pytext.models.representations.representation_base.Representation decoder: pytext.models.decoders.decoder_base.DecoderBase, output_layer: pytext.models.output_layers.output_layer_base.OutputLayerBase*)

Bases: *pytext.models.doc_model.DocModel*

Classification model with sequence of utterances as input. It uses a docnn model (CNN or LSTM) to generate vector representation for each sequence, and then use an LSTM or BLSTM to capture the dynamics and produce labels for each sequence.

**arrange_model_inputs**(*tensor_dict*)

**class** pytext.models.seq_models.seqnn.**SeqNNModel_Deprecated**(*embedding: pytext.models.embeddings.embedding_base.Embe representation: pytext.models.representations.representation_bas decoder: pytext.models.decoders.decoder_base.DecoderBa output_layer: pytext.models.output_layers.output_layer_base.O*

Bases: *pytext.models.model.Model*

Classification model with sequence of utterances as input. It uses a docnn model (CNN or LSTM) to generate vector representation for each sequence, and then use an LSTM or BLSTM to capture the dynamics and produce labels for each sequence.

DEPRECATED: Use SeqNNModel

### pytext.models.seq_models.utils module

pytext.models.seq_models.utils.**Linear**(*in_features, out_features, bias=True*)

pytext.models.seq_models.utils.**extract_ontology_vocab**(*target_dictionary*)

pytext.models.seq_models.utils.**make_positions**(*input, padding_idx: int*)

Replace non-padding symbols with their position numbers.

Position numbers begin at padding_idx+1. Padding symbols are ignored.

pytext.models.seq_models.utils.**prepare_full_key**(*instance_id: str, key: str, secondary_key: Optional[str] = None*)

pytext.models.seq_models.utils.**unfold1d**(*x, kernel_size: int, padding_l: int, pad_value: float = 0*)

unfold T x B x C to T x B x C x K

pytext.models.seq_models.utils.**verify_encoder_out**(*encoder_out: Dict[str, torch.Tensor], keys: List[str]*)

### Module contents

## Submodules

### pytext.models.bert_classification_models module

**class** pytext.models.bert_classification_models.**BertPairwiseModel**(*encoder1*, *encoder2*, *decoder*, *output_layer*, *encode_relations*, *shared_encoder*)

Bases: pytext.models.bert_classification_models._EncoderPairwiseModel

Bert Pairwise classification model

The model takes two sets of tokens (left and right) and calculates their representations separately using shared BERT encoder. The final prediction can be the cosine similarity of the embeddings, or if encoder_relations is specified the concatenation of the embeddings, their absolute difference, and elementwise product.

**class** pytext.models.bert_classification_models.**NewBertModel**(*encoder*, *decoder*, *output_layer*, *stage=<Stage.TRAIN: 'Training'>*)

Bases: pytext.models.bert_classification_models._EncoderBaseModel

BERT single sentence classification.

### pytext.models.bert_regression_model module

**class** pytext.models.bert_regression_model.**BertPairwiseRegressionModel**(*encoder1*, *encoder2*, *decoder*, *output_layer*, *encode_relations*, *shared_encoder*)

Bases: *pytext.models.bert_classification_models.BertPairwiseModel*

Two-tower model for regression. Encode two texts separately and use the cosine similarity between sentence embeddings to predict regression label.

**class** pytext.models.bert_regression_model.**NewBertRegressionModel**(*encoder*, *decoder*, *output_layer*)

Bases: *pytext.models.bert_classification_models.NewBertModel*

BERT single sentence (or concatenated sentences) regression.

**classmethod from_config**(*config: pytext.models.bert_regression_model.NewBertRegressionModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

### pytext.models.crf module

**class** pytext.models.crf.**CRF**(*num_tags: int*, *ignore_index: int*, *default_label_pad_index: int*)

Bases: torch.nn.modules.module.Module

Compute the log-likelihood of the input assuming a conditional random field model.

**Parameters** **num_tags** – The number of tags

**decode**(*emissions: torch.Tensor*, *seq_lens: torch.Tensor*) → torch.Tensor

Given a set of emission probabilities, return the predicted tags.

**Parameters**

- **emissions** – Emission probabilities with expected shape of batch_size * seq_len * num_labels
- **seq_lens** – Length of each input.

**export_to_caffe2**(*workspace*, *init_net*, *predict_net*, *logits_output_name*)

Exports the crf layer to caffe2 by manually adding the necessary operators to the init_net and predict net.

**Parameters**

- **init_net** – caffe2 init net created by the current graph
- **predict_net** – caffe2 net created by the current graph
- **workspace** – caffe2 current workspace
- **output_names** – current output names of the caffe2 net
- **py_model** – original pytorch model object

**Returns** The updated predictions blob name

**Return type** string

**forward**(*emissions: torch.Tensor*, *tags: torch.Tensor*, *reduce: bool = True*) → torch.Tensor

Compute log-likelihood of input.

**Parameters**

- **emissions** – Emission values for different tags for each input. The expected shape is batch_size * seq_len * num_labels. Padding is should be on the right side of the input.
- **tags** – Actual tags for each token in the input. Expected shape is batch_size * seq_len

**get_transitions**()

**reset_parameters**() → None

**set_transitions**(*transitions: torch.Tensor = None*)

### pytext.models.disjoint_multitask_model module

**class** pytext.models.disjoint_multitask_model.**DisjointMultitaskModel**(*models*, *loss_weights*)

Bases: *pytext.models.model.Model*

Wrapper model to train multiple PyText models that share parameters. Designed to be used for multi-tasking when the tasks have disjoint datasets.

Modules which have the same shared_module_key and type share parameters. Only need to configure the first such module in full in each case.

**Parameters** **models** (*type*) – Dictionary of models of sub-tasks.

**current_model**
Current model to route the input batch to.

**Type** type

**contextualize**(*context*)
Add additional context into model. *context* can be anything that helps maintaining/updating state. For example, it is used by `DisjointMultitaskModel` for changing the task that should be trained with a given iterator.

**current_model**

**forward**(**inputs*) → List[torch.Tensor]
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_loss**(*logits*, *targets*, *context*)

**get_pred**(*logits*, *targets=None*, *context=None*, **args*)

**save_modules**(*base_path*, *suffix=''*)
Save each sub-module in separate files for reusing later.

**class** pytext.models.disjoint_multitask_model.**NewDisjointMultitaskModel**(*models*, *loss_weights*)
Bases: `pytext.models.disjoint_multitask_model.DisjointMultitaskModel`

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

### pytext.models.distributed_model module

**class** pytext.models.distributed_model.**DistributedModel**(**args*, ***kwargs*)
Bases: `torch.nn.parallel.distributed.DistributedDataParallel`

Wrapper model class to train models in distributed data parallel manner. The way to use this class to train your module in distributed manner is:

```
distributed_model = DistributedModel(
    module=model,
    device_ids=[device_id0, device_id1],
    output_device=device_id0,
    broadcast_buffers=False,
)
```

where, *model* is the object of the actual model class you want to train in distributed manner.

**cpu**()
Moves all model parameters and buffers to the CPU.

**Returns** self

**Return type** Module

**eval**(*stage=<Stage.TEST: 'Test'>*)
Override to set stage

**load_state_dict**(**args*, ***kwargs*)
Copies parameters and buffers from `state_dict` into this module and its descendants. If `strict` is `True`, then the keys of `state_dict` must exactly match the keys returned by this module's `state_dict()` function.

**Parameters**

- **state_dict** (*dict*) – a dict containing parameters and persistent buffers.
- **strict** (*bool, optional*) – whether to strictly enforce that the keys in `state_dict` match the keys returned by this module's `state_dict()` function. Default: `True`

**Returns**

- **missing_keys** is a list of str containing the missing keys
- **unexpected_keys** is a list of str containing the unexpected keys

**Return type** `NamedTuple` with `missing_keys` and `unexpected_keys` fields

**state_dict**(**args*, ***kwargs*)
Returns a dictionary containing a whole state of the module.

Both parameters and persistent buffers (e.g. running averages) are included. Keys are corresponding parameter and buffer names.

**Returns** a dictionary containing a whole state of the module

**Return type** dict

Example:

```
>>> module.state_dict().keys()
['bias', 'weight']
```

**train**(*mode=True*)
Override to set stage

## pytext.models.doc_model module

**class** pytext.models.doc_model.**ByteTokensDocumentModel**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingB representation: pytext.models.representations.representation_base.Repre decoder: pytext.models.decoders.decoder_base.DecoderBase, output_layer: pytext.models.output_layers.output_layer_base.OutputLa*)

Bases: `pytext.models.doc_model.DocModel`

DocModel that receives both word IDs and byte IDs as inputs (concatenating word and byte-token embeddings to represent input tokens).

**arrange_model_inputs**(*tensor_dict*)

**classmethod create_embedding**(*config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**get_export_input_names**(*tensorizers*)

**torchscriptify**(*tensorizers, traced_model*)

**class** pytext.models.doc_model.**DocModel**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase, representation: pytext.models.representations.representation_base.RepresentationBase, decoder: pytext.models.decoders.decoder_base.DecoderBase, output_layer: pytext.models.output_layers.output_layer_base.OutputLayerBase*)

Bases: `pytext.models.model.Model`

DocModel that's compatible with the new Model abstraction, which is responsible for describing which inputs it expects and arranging its input tensors.

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**caffe2_export**(*tensorizers, tensor_dict, path, export_onnx_path=None*)

**classmethod create_decoder**(*config: pytext.models.doc_model.DocModel.Config, representation_dim: int, num_labels: int*)

**classmethod create_embedding**(*config: pytext.models.doc_model.DocModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**classmethod create_output_layer**(*config: pytext.models.doc_model.DocModel.Config, labels: pytext.data.tensorizers.VocabConfig*)

**classmethod from_config**(*config: pytext.models.doc_model.DocModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**get_export_input_names**(*tensorizers*)

**get_export_output_names**(*tensorizers*)

**get_num_examples_from_batch**(*tensor_dict*)

**torchscriptify**(*tensorizers, traced_model*)

**vocab_to_export**(*tensorizers*)

**class** pytext.models.doc_model.**DocRegressionModel**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase, representation: pytext.models.representations.representation_base.Representatio decoder: pytext.models.decoders.decoder_base.DecoderBase, output_layer: pytext.models.output_layers.output_layer_base.OutputLayerBas*

Bases: `pytext.models.doc_model.DocModel`

Model that's compatible with the new Model abstraction, and is configured for regression tasks (specifically for labels, predictions, and loss).

**classmethod from_config**(*config: pytext.models.doc_model.DocRegressionModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**class** pytext.models.doc_model.**PersonalizedDocModel**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase, representation: pytext.models.representations.representation_base.Represent decoder: pytext.models.decoders.decoder_base.DecoderBase, output_layer: pytext.models.output_layers.output_layer_base.OutputLayerB user_embedding: Optional[pytext.models.embeddings.embedding_base.Embedd = None*)

Bases: *pytext.models.doc_model.DocModel*

DocModel that includes a user embedding which learns user features to produce personalized prediction. In this class, user-embedding is fed directly to the decoder (i.e., does not go through the encoders).

**arrange_model_inputs**(*tensor_dict*)

**forward**(**inputs*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**get_export_input_names**(*tensorizers*)

**torchscriptify**(*tensorizers, traced_model*)

**vocab_to_export**(*tensorizers*)

### pytext.models.joint_model module

**class** pytext.models.joint_model.**IntentSlotModel**(*default_doc_loss_weight, default_word_loss_weight, *args, **kwargs*)

Bases: *pytext.models.model.Model*

A joint intent-slot model. This is framed as a model to do document classification model and word tagging tasks where the embedding and text representation layers are shared for both tasks.

The supported representation layers are based on bidirectional LSTM or CNN.

It can be instantiated just like any other `Model`.

This is in the new data handling design involving tensorizers; that is the difference between this and JointModel

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**caffe2_export**(*tensorizers, tensor_dict, path, export_onnx_path=None*)

**classmethod create_embedding**(*config*, *tensorizers*)

**classmethod from_config**(*config*, *tensorizers*)

**get_export_input_names**(*tensorizers*)

**get_export_output_names**(*tensorizers*)

**get_weights_context**(*tensor_dict*)

**vocab_to_export**(*tensorizers*)

### pytext.models.masked_lm module

**class** pytext.models.masked_lm.**MaskedLanguageModel**(*encoder: pytext.models.representations.transformer_sentence_encoder_*, *decoder: pytext.models.decoders.mlp_decoder.MLPDecoder*, *output_layer: pytext.models.output_layers.lm_output_layer.LMOutputLayer*, *token_tensorizer: pytext.data.bert_tensorizer.BERTTensorizerBase*, *vocab: pytext.data.utils.Vocabulary*, *mask_prob: float = 0.15*, *mask_bos: float = False*, *masking_strategy: pytext.models.masking_utils.MaskingStrategy = <MaskingStrategy.RANDOM: 'random'>*, *stage: pytext.common.constants.Stage = <Stage.TRAIN: 'Training'>*)

Bases: *pytext.models.model.BaseModel*

Masked language model for BERT style pre-training.

**SUPPORT_FP16_OPTIMIZER = True**

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**forward**(**inputs*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.models.masked_lm.MaskedLanguageModel.Config*, *tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

### pytext.models.masking_utils module

**class** pytext.models.masking_utils.**MaskingStrategy**

Bases: `enum.Enum`

An enumeration.

**FREQUENCY = 'frequency_based'**

**RANDOM = 'random'**

pytext.models.masking_utils.**frequency_based_masking**(*tokens: None._VariableFunctionsClass.tensor, token_sampling_weights: numpy.ndarray, mask_prob: float*) → torch.Tensor

Function to mask tokens based on frequency.

**Inputs:**

1) tokens: Tensor with token ids of shape (batch_size x seq_len)
2) **token_sampling_weights: numpy array with shape (batch_size x seq_len)** and each element representing the sampling weight assicated with the corresponding token in tokens
3) mask_prob: Probability of masking a particular token

**Outputs:**

**mask: Tensor with same shape as input tokens (batch_size x seq_len)** with masked tokens represented by a 1 and everything else as 0.

pytext.models.masking_utils.**random_masking**(*tokens: None._VariableFunctionsClass.tensor, mask_prob: float*) → torch.Tensor

Function to mask tokens randomly.

**Inputs:**

1) tokens: Tensor with token ids of shape (batch_size x seq_len)
2) mask_prob: Probability of masking a particular token

**Outputs:**

**mask: Tensor with same shape as input tokens (batch_size x seq_len)** with masked tokens represented by a 1 and everything else as 0.

### pytext.models.model module

**class** pytext.models.model.**BaseModel**(*stage: pytext.common.constants.Stage = <Stage.TRAIN: 'Training'>*)

Bases: torch.nn.modules.module.Module, *pytext.config.component.Component*

Base model class which inherits from nn.Module. Also has a stage flag to indicate it's in *train*, *eval*, or *test* stage. This is because the built-in train/eval flag in PyTorch can't distinguish eval and test, which is required to support some use cases.

**SUPPORT_FP16_OPTIMIZER = False**

**arrange_caffe2_model_inputs**(*tensor_dict*)

Generate inputs for exported caffe2 model, default behavior is flatten the input tuples

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**caffe2_export**(*tensorizers, tensor_dict, path, export_onnx_path=None*)

**contextualize**(*context*)
Add additional context into model. *context* can be anything that helps maintaining/updating state. For example, it is used by `DisjointMultitaskModel` for changing the task that should be trained with a given iterator.

**eval**(*stage=<Stage.TEST: 'Test'>*)
Override to explicitly maintain the stage (train, eval, test).

**get_loss**(*logit*, *target*, *context*)

**get_num_examples_from_batch**(*batch*)

**get_pred**(*logit*, *target=None*, *context=None*, **args*)

**onnx_trace_input**(*tensor_dict*)

**prepare_for_onnx_export_**(***kwargs*)
Make model exportable via ONNX trace.

**quantize**()
Quantize the model during export.

**save_modules**(*base_path: str = ''*, *suffix: str = ''*)
Save each sub-module in separate files for reusing later.

**trace**(*inputs*)

**train**(*mode=True*)
Override to explicitly maintain the stage (train, eval, test).

**classmethod train_batch**(*model*, *batch*, *state=None*)

**class** `pytext.models.model.`**Model**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase*, *representation: pytext.models.representations.representation_base.RepresentationBase*, *decoder: pytext.models.decoders.decoder_base.DecoderBase*, *output_layer: pytext.models.output_layers.output_layer_base.OutputLayerBase*)

Bases: `pytext.models.model.BaseModel`

Generic single-task model class that expects four components:

1. *Embedding*
2. *Representation*
3. *Decoder*
4. *Output Layer*

Forward pass: *embedding -> representation -> decoder -> output_layer*

These four components have specific responsibilities as described below.

*Embedding* layer should implement the way to represent each token in the input text. It can be as simple as just token/word embedding or can be composed of multiple ways to represent a token, e.g., word embedding, character embedding, etc.

*Representation* layer should implement the way to encode the entire input text such that the output vector(s) can be used by decoder to produce logits. There is no restriction on the number of inputs it should encode. There is also not restriction on the number of ways to encode input.

*Decoder* layer should implement the way to consume the output of model's representation and produce logits that can be used by the output layer to compute loss or generate predictions (and prediction scores/confidence)

*Output layer* should implement the way loss computation is done as well as the logic to generate predictions from the logits.

Let us discuss the joint intent-slot model as a case to go over these layers. The model predicts intent of input utterance and the slots in the utterance. (Refer to *Train Intent-Slot model on ATIS Dataset* for details about intent-slot model.)

1. `EmbeddingList` layer is tasked with representing tokens. To do so we can use learnable word embedding table in conjunction with learnable character embedding table that are distilled to token level representation using CNN and pooling. Note: This class is meant to be reused by all models. It acts as a container of all the different ways of representing a token/word.
2. `BiLSTMDocSlotAttention` is tasked with encoding the embedded input string for intent classification and slot filling. In order to do that it has a shared bidirectional LSTM layer followed by separate attention layers for document level attention and word level attention. Finally it produces two vectors per utterance.
3. `IntentSlotModelDecoder` accepts the two input vectors from *BiLSTMDocSlotAttention* and produces logits for intent classification and slot filling. Conditioned on a flag it can also use the probabilities from intent classification for slot filling.
4. `IntentSlotOutputLayer` implements the logic behind computing loss and prediction, as well as, how to export this layer to export to Caffe2. This is used by model exporter as a post-processing Caffe2 operator.

**Parameters**

- **embedding** (*EmbeddingBase*) – Description of parameter *embedding*.
- **representation** (*RepresentationBase*) – Description of parameter *representation*.
- **decoder** (*DecoderBase*) – Description of parameter *decoder*.
- **output_layer** (*OutputLayerBase*) – Description of parameter *output_layer*.

**embedding**

**representation**

**decoder**

**output_layer**

**classmethod compose_embedding**(*sub_emb_module_dict: Dict[str, pytext.models.embeddings.embedding_base.EmbeddingBase], metadata*) → pytext.models.embeddings.embedding_list.EmbeddingList

Default implementation is to compose an instance of `EmbeddingList` with all the sub-embedding modules. You should override this class method if you want to implement a specific way to embed tokens/words.

**Parameters** **sub_emb_module_dict** (*Dict[str, EmbeddingBase]*) – Named dictionary of embedding modules each of which implement a way to embed/encode a token.

**Returns** An instance of `EmbeddingList`.

**Return type** EmbeddingList

**classmethod create_embedding**(*feat_config: pytext.config.field_config.FeatureConfig, metadata: pytext.data.data_handler.CommonMetadata*)

**classmethod create_sub_embs**(*emb_config: pytext.config.field_config.FeatureConfig, metadata: pytext.data.data_handler.CommonMetadata*) → Dict[str, pytext.models.embeddings.embedding_base.EmbeddingBase]

Creates the embedding modules defined in the *emb_config*.

**Parameters**

- **emb_config** (*FeatureConfig*) – Object containing all the sub-embedding configurations.
- **metadata** (*CommonMetadata*) – Object containing features and label metadata.

**Returns** Named dictionary of embedding modules.

**Return type** Dict[str, EmbeddingBase]

**forward**(**inputs*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.model.Model.Config, feat_config: pytext.config.field_config.FeatureConfig, metadata: pytext.data.data_handler.CommonMetadata*)

**class** pytext.models.model.**ModelInputBase**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

Base class for model inputs.

**class** pytext.models.model.**ModelInputMeta**

Bases: *pytext.config.pytext_config.ConfigBaseMeta*

### pytext.models.module module

**class** pytext.models.module.**Module**(*config=None*)

Bases: `torch.nn.modules.module.Module`, *pytext.config.component.Component*

Generic module class that serves as base class for all PyText modules.

**Parameters** **config** (*type*) – Module's *config* object. Specific contents of this object depends on the module. Defaults to None.

**freeze**() → None

pytext.models.module.**create_module**(*module_config, *args, create_fn=<function _create_module_from_registry>, **kwargs*)

Create module object given the module's config object. It depends on the global shared module registry. Hence, your module must be available for the registry. This entails that your module must be imported somewhere in the code path during module creation (ideally in your model class) for the module to be visible for registry.

**Parameters**

- **module_config** (*type*) – Module config object.
- **create_fn** (*type*) – The function to use for creating the module. Use this parameter if your module creation requires custom code and pass your function here. Defaults to *_create_module_from_registry()*.

**Returns** Description of returned object.

**Return type** type

### pytext.models.pair_classification_model module

**class** pytext.models.pair_classification_model.**BasePairwiseModel**(*decoder: pytext.models.decoders.decoder_base.Dec*
*output_layer: pytext.models.output_layers.output_layer_*
*encode_relations: bool*)

Bases: *pytext.models.model.BaseModel*

A base classification model that scores a pair of texts.

Subclasses need to implement the from_config, forward and save_modules.

**forward**(*input1: Tuple[torch.Tensor, ...], input2: Tuple[torch.Tensor, ...]*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.pair_classification_model.BasePairwiseModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**save_modules**(*base_path: str = ", suffix: str = "*)

Save each sub-module in separate files for reusing later.

**class** pytext.models.pair_classification_model.**PairwiseModel**(*embeddings: torch.nn.modules.container.ModuleList, representations: torch.nn.modules.container.ModuleList, decoder: pytext.models.decoders.mlp_decoder.MLPDeco*
*output_layer: pytext.models.output_layers.doc_classification_*
*encode_relations: bool, shared_representations: bool*)

Bases: *pytext.models.pair_classification_model.BasePairwiseModel*

A classification model that scores a pair of texts, for example, a model for natural language inference.

The model shares embedding space (so it doesn't support pairs of texts where left and right are in different languages). It uses bidirectional LSTM or CNN to represent the two documents, and concatenates them along with their absolute difference and elementwise product. This concatenated pair representation is passed to a multi-layer perceptron to decode to label/target space.

See https://arxiv.org/pdf/1705.02364.pdf for more details.

It can be instantiated just like any other `Model`.

**EMBEDDINGS = ['embedding']**

**INPUTS_PAIR = [['tokens1'], ['tokens2']]**

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**forward**(*input1: Tuple[torch.Tensor, ...], input2: Tuple[torch.Tensor, ...]*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.pair_classification_model.PairwiseModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**save_modules**(*base_path: str = ", suffix: str = "*)

Save each sub-module in separate files for reusing later.

## pytext.models.query_document_pairwise_ranking_model module

**class** pytext.models.query_document_pairwise_ranking_model.**QueryDocPairwiseRankingModel**(*embe torch rep- re- sen- ta- tions torch de- code py- text.n out- put_l py- text.n en- code bool, share bool*)

Bases: *pytext.models.pair_classification_model.PairwiseModel*

Pairwise ranking model This model takes in a query, and two responses (pos_response and neg_response) It passes representations of the query and the two responses to a decoder pos_response should be ranked higher than neg_response - this is ensured by training with a ranking hinge loss function

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**forward**(*pos_response: Tuple[torch.Tensor, torch.Tensor], neg_response: Tuple[torch.Tensor, torch.Tensor], query: Tuple[torch.Tensor, torch.Tensor]*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.models.query_document_pairwise_ranking_model.QueryDocPairwiseRanking*
*tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**get_num_examples_from_batch**(*tensor_dict*)

### pytext.models.r3f_models module

**class** pytext.models.r3f_models.**R3FConfigOptions**(***kwargs*)

Bases: *pytext.config.pytext_config.ConfigBase*

Configuration options for models using R3F

**eps = 1e-05**

**noise_type = 'uniform'**

**r3f_default_lambda = 0.5**

**r3f_lambda_by_loss = {}**

**class** pytext.models.r3f_models.**R3FNoiseContextManager**(*context*)

Bases: `contextlib.AbstractContextManager`

Context manager that adds a forward hook to the embedding module, to insert noise into the model and detatch embedding when doing this pass

**class** pytext.models.r3f_models.**R3FNoiseType**

Bases: `enum.Enum`

An enumeration.

**NORMAL = 'normal'**

**UNIFORM = 'uniform'**

**class** pytext.models.r3f_models.**R3FPyTextMixin**(*config: pytext.models.r3f_models.R3FConfigOptions*)

Bases: `object`

Mixin class for applying the R3F method, to apply R3F with any model inherit the class and implement the abstract functions.

For more details: https://arxiv.org/abs/2008.03156

**forward**(**args*, *use_r3f: bool = False*, ***kwargs*)

**forward_with_noise**(**args*, ***kwargs*)

**get_embedding_module**(**args*, ***kwargs*)

Given the core model outputs, this returns the embedding module that is used for the R3F loss, in particular noise will be injected to this module.

**get_r3f_loss_terms**(*model_outputs*, *noise_model_outputs*, *sample_size: int*) → torch.Tensor

Computes the auxillary loss for R3F, in particular computes a symmetric KL divergence between the result from the input embedding and the noise input embedding.

**get_r3f_model_output**(*model_output*)

Extracts the output from the model.forward() call that is used for the r3f loss term

**get_sample_size**(*model_inputs*, *targets*)
Gets the sample size of the model that is used as a regularization factor to the model itself

**original_forward**(**args*, ***kwargs*)
Runs the traditional forward of this model

**classmethod train_batch**(*model*, *batch*, *state=None*)
Runs training over a batch with the R3F method, training will use R3F while eval and test do not.

pytext.models.r3f_models.**build_noise_sampler**(*noise_type: pytext.models.r3f_models.R3FNoiseType*, *eps: float*)
Given a *noise_type* (*R3FNoiseType*): builds a *torch.distribution* capable of generating noise within the passed in *eps* (*float*).

pytext.models.r3f_models.**compute_symmetric_kl**(*noised_logits*, *input_logits*)
Computes symmetric KL loss by taking the KL for both the input logits and the noised logits and comparing the two

### pytext.models.roberta module

**class** pytext.models.roberta.**RoBERTa**(*encoder*, *decoder*, *output_layer*, *stage=<Stage.TRAIN: 'Training'>*)
Bases: *pytext.models.bert_classification_models.NewBertModel*

**graph_mode_quantize**(*inputs*, *data_loader*, *calibration_num_batches=64*, *qconfig_dict=None*, *force_quantize=False*)
Quantize the model during export with graph mode quantization.

**torchscriptify**(*tensorizers*, *traced_model*)
Using the traced model, create a ScriptModule which has a nicer API that includes generating tensors from simple data types, and returns classified values according to the output layer (eg. as a dict mapping class name to score)

**trace**(*inputs*)

**class** pytext.models.roberta.**RoBERTaEncoder**(*config: pytext.models.roberta.RoBERTaEncoder.Config*, *output_encoded_layers: bool*, ***kwarg*)
Bases: *pytext.models.roberta.RoBERTaEncoderBase*

A PyTorch RoBERTa implementation

**forward**(*input_tuple: Tuple[torch.Tensor, ...], *args*) → Tuple[torch.Tensor, ...]
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** pytext.models.roberta.**RoBERTaEncoderBase**(*config: pytext.models.representations.transformer_sentence_encoder_base.*
*output_encoded_layers=False*, **args*, ***kwargs*)
Bases: *pytext.models.representations.transformer_sentence_encoder_base.TransformerSentenceEncoderBase*

**class** pytext.models.roberta.**RoBERTaEncoderJit**(*config: pytext.models.roberta.RoBERTaEncoderJit.Config, output_encoded_layers: bool, **kwarg*)

Bases: pytext.models.roberta.RoBERTaEncoderBase

A TorchScript RoBERTa implementation

**class** pytext.models.roberta.**RoBERTaR3F**(*encoder, decoder, output_layer, r3f_options, stage=<Stage.TRAIN: 'Training'>*)

Bases: pytext.models.roberta.RoBERTa, pytext.models.r3f_models.R3FPyTextMixin

**forward**(**args, use_r3f: bool = False, **kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**get_embedding_module**(**args, **kwargs*)

Given the core model outputs, this returns the embedding module that is used for the R3F loss, in particular noise will be injected to this module.

**get_sample_size**(*model_inputs, targets*)

Gets the sample size of the model that is used as a regularization factor to the model itself

**original_forward**(**args, **kwargs*)

Runs the traditional forward of this model

**classmethod train_batch**(*model, batch, state=None*)

Runs training over a batch with the R3F method, training will use R3F while eval and test do not.

**class** pytext.models.roberta.**RoBERTaRegression**(*encoder, decoder, output_layer*)

Bases: pytext.models.bert_regression_model.NewBertRegressionModel

**torchscriptify**(*tensorizers, traced_model*)

Using the traced model, create a ScriptModule which has a nicer API that includes generating tensors from simple data types, and returns classified values according to the output layer (eg. as a dict mapping class name to score)

**class** pytext.models.roberta.**RoBERTaWordTaggingModel**(*encoder, decoder, output_layer, stage=<Stage.TRAIN: 'Training'>*)

Bases: pytext.models.model.BaseModel

Single Sentence Token-level Classification Model using XLM.

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**forward**(*encoder_inputs: Tuple[torch.Tensor, ...], *args*) → torch.Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

**classmethod from_config**(*config: pytext.models.roberta.RoBERTaWordTaggingModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**class** pytext.models.roberta.**SELFIE**(*encoder, decoder, output_layer, stage=<Stage.TRAIN: 'Training'>*)

Bases: *pytext.models.roberta.RoBERTa*

**forward**(*encoder_inputs: Tuple[torch.Tensor, ...], *args*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

pytext.models.roberta.**init_params**(*module*)

Initialize the RoBERTa weights for pre-training from scratch.

### pytext.models.two_tower_classification_model module

**class** pytext.models.two_tower_classification_model.**TwoTowerClassificationModel**(*right_encoder, left_encoder, decoder, output_layer, stage=<Stage.TR 'Training'>*)

Bases: *pytext.models.model.BaseModel*

**SUPPORT_FP16_OPTIMIZER = True**

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**caffe2_export**(*tensorizers, tensor_dict, path, export_onnx_path=None*)

**forward**(*right_encoder_inputs: Tuple[torch.Tensor, ...], left_encoder_inputs: Tuple[torch.Tensor, ...], *args*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**classmethod from_config**(*config: pytext.models.two_tower_classification_model.TwoTowerClassificationModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**graph_mode_quantize**(*inputs*, *data_loader*, *calibration_num_batches=64*)
Quantize the model during export with graph mode quantization for linformer encoder.

**torchscriptify**(*tensorizers*, *traced_model*)
Using the traced model, create a ScriptModule which has a nicer API that includes generating tensors from simple data types, and returns classified values according to the output layer (eg. as a dict mapping class name to score)

**trace**(*inputs*)

### pytext.models.utils module

pytext.models.utils.**normalize_embeddings**(*embeddings: torch.Tensor*)

### pytext.models.word_model module

**class** pytext.models.word_model.**WordTaggingLiteModel**(**args*, ***kwargs*)
Bases: *pytext.models.word_model.WordTaggingModel*

Also a word tagging model, but uses bytes as inputs to the model. Using bytes instead of words, the model does not need to store a word embedding table mapping words in the vocab to their embedding vector representations, but instead compute them on the fly using CharacterEmbedding. This produces an exported/serialized model that requires much less storage space as well as less memory during run/inference time.

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**classmethod create_embedding**(*config*, *tensorizers*)

**get_export_input_names**(*tensorizers*)

**torchscriptify**(*tensorizers*, *traced_model*)

**vocab_to_export**(*tensorizers*)

**class** pytext.models.word_model.**WordTaggingModel**(**args*, ***kwargs*)
Bases: *pytext.models.model.Model*

Word tagging model. It can be used for any task that requires predicting the tag for a word/token. For example, the following tasks can be modeled as word tagging tasks. This is not an exhaustive list. 1. Part of speech tagging. 2. Named entity recognition. 3. Slot filling for task oriented dialog.

It can be instantiated just like any other `Model`.

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**classmethod create_embedding**(*config*, *tensorizers*)

**classmethod from_config**(*config*, *tensorizers*)

**get_export_input_names**(*tensorizers*)

**get_export_output_names**(*tensorizers*)

**torchscriptify**(*tensorizers*, *traced_model*)

**vocab_to_export**(*tensorizers*)

### Module contents

**class** pytext.models.**Model**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase*, *representation: pytext.models.representations.representation_base.RepresentationBase*, *decoder: pytext.models.decoders.decoder_base.DecoderBase*, *output_layer: pytext.models.output_layers.output_layer_base.OutputLayerBase*)

Bases: `pytext.models.model.BaseModel`

Generic single-task model class that expects four components:

1. *Embedding*
2. *Representation*
3. *Decoder*
4. *Output Layer*

Forward pass: *embedding -> representation -> decoder -> output_layer*

These four components have specific responsibilities as described below.

*Embedding* layer should implement the way to represent each token in the input text. It can be as simple as just token/word embedding or can be composed of multiple ways to represent a token, e.g., word embedding, character embedding, etc.

*Representation* layer should implement the way to encode the entire input text such that the output vector(s) can be used by decoder to produce logits. There is no restriction on the number of inputs it should encode. There is also not restriction on the number of ways to encode input.

*Decoder* layer should implement the way to consume the output of model's representation and produce logits that can be used by the output layer to compute loss or generate predictions (and prediction scores/confidence)

*Output layer* should implement the way loss computation is done as well as the logic to generate predictions from the logits.

Let us discuss the joint intent-slot model as a case to go over these layers. The model predicts intent of input utterance and the slots in the utterance. (Refer to *Train Intent-Slot model on ATIS Dataset* for details about intent-slot model.)

1. `EmbeddingList` layer is tasked with representing tokens. To do so we can use learnable word embedding table in conjunction with learnable character embedding table that are distilled to token level representation using CNN and pooling. Note: This class is meant to be reused by all models. It acts as a container of all the different ways of representing a token/word.
2. `BiLSTMDocSlotAttention` is tasked with encoding the embedded input string for intent classification and slot filling. In order to do that it has a shared bidirectional LSTM layer followed by separate attention layers for document level attention and word level attention. Finally it produces two vectors per utterance.
3. `IntentSlotModelDecoder` accepts the two input vectors from *BiLSTMDocSlotAttention* and produces logits for intent classification and slot filling. Conditioned on a flag it can also use the probabilities from intent classification for slot filling.
4. `IntentSlotOutputLayer` implements the logic behind computing loss and prediction, as well as, how to export this layer to export to Caffe2. This is used by model exporter as a post-processing Caffe2 operator.

**Parameters**

- **embedding** (`EmbeddingBase`) – Description of parameter *embedding*.

- **representation** (*RepresentationBase*) – Description of parameter *representation*.
- **decoder** (*DecoderBase*) – Description of parameter *decoder*.
- **output_layer** (*OutputLayerBase*) – Description of parameter *output_layer*.

**embedding**

**representation**

**decoder**

**output_layer**

**classmethod compose_embedding**(*sub_emb_module_dict: Dict[str, pytext.models.embeddings.embedding_base.EmbeddingBase], metadata*) → pytext.models.embeddings.embedding_list.EmbeddingList

Default implementation is to compose an instance of `EmbeddingList` with all the sub-embedding modules. You should override this class method if you want to implement a specific way to embed tokens/words.

**Parameters** **sub_emb_module_dict** (*Dict[str, EmbeddingBase]*) – Named dictionary of embedding modules each of which implement a way to embed/encode a token.

**Returns** An instance of `EmbeddingList`.

**Return type** EmbeddingList

**classmethod create_embedding**(*feat_config: pytext.config.field_config.FeatureConfig, metadata: pytext.data.data_handler.CommonMetadata*)

**classmethod create_sub_embs**(*emb_config: pytext.config.field_config.FeatureConfig, metadata: pytext.data.data_handler.CommonMetadata*) → Dict[str, pytext.models.embeddings.embedding_base.EmbeddingBase]

Creates the embedding modules defined in the *emb_config*.

**Parameters**

- **emb_config** (*FeatureConfig*) – Object containing all the sub-embedding configurations.
- **metadata** (*CommonMetadata*) – Object containing features and label metadata.

**Returns** Named dictionary of embedding modules.

**Return type** Dict[str, EmbeddingBase]

**forward**(**inputs*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.model.Model.Config, feat_config: pytext.config.field_config.FeatureConfig, metadata: pytext.data.data_handler.CommonMetadata*)

**class** pytext.models.**BaseModel**(*stage: pytext.common.constants.Stage = <Stage.TRAIN: 'Training'>*)

Bases: torch.nn.modules.module.Module, *pytext.config.component.Component*

Base model class which inherits from nn.Module. Also has a stage flag to indicate it's in *train*, *eval*, or *test* stage. This is because the built-in train/eval flag in PyTorch can't distinguish eval and test, which is required to support some use cases.

**SUPPORT_FP16_OPTIMIZER = False**

**arrange_caffe2_model_inputs**(*tensor_dict*)

Generate inputs for exported caffe2 model, default behavior is flatten the input tuples

**arrange_model_context**(*tensor_dict*)

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**caffe2_export**(*tensorizers*, *tensor_dict*, *path*, *export_onnx_path=None*)

**contextualize**(*context*)

Add additional context into model. *context* can be anything that helps maintaining/updating state. For example, it is used by DisjointMultitaskModel for changing the task that should be trained with a given iterator.

**eval**(*stage=<Stage.TEST: 'Test'>*)

Override to explicitly maintain the stage (train, eval, test).

**get_loss**(*logit*, *target*, *context*)

**get_num_examples_from_batch**(*batch*)

**get_pred**(*logit*, *target=None*, *context=None*, **args*)

**onnx_trace_input**(*tensor_dict*)

**prepare_for_onnx_export_**(***kwargs*)

Make model exportable via ONNX trace.

**quantize**()

Quantize the model during export.

**save_modules**(*base_path: str = ''*, *suffix: str = ''*)

Save each sub-module in separate files for reusing later.

**trace**(*inputs*)

**train**(*mode=True*)

Override to explicitly maintain the stage (train, eval, test).

**classmethod train_batch**(*model*, *batch*, *state=None*)

**class** pytext.models.**TwoTowerClassificationModel**(*right_encoder*, *left_encoder*, *decoder*, *output_layer*, *stage=<Stage.TRAIN: 'Training'>*)

Bases: *pytext.models.model.BaseModel*

**SUPPORT_FP16_OPTIMIZER = True**

**arrange_model_inputs**(*tensor_dict*)

**arrange_targets**(*tensor_dict*)

**caffe2_export**(*tensorizers*, *tensor_dict*, *path*, *export_onnx_path=None*)

**forward**(*right_encoder_inputs: Tuple[torch.Tensor, ...], left_encoder_inputs: Tuple[torch.Tensor, ...], *args*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from_config**(*config: pytext.models.two_tower_classification_model.TwoTowerClassificationModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**graph_mode_quantize**(*inputs*, *data_loader*, *calibration_num_batches=64*)

Quantize the model during export with graph mode quantization for linformer encoder.

**torchscriptify**(*tensorizers*, *traced_model*)

Using the traced model, create a ScriptModule which has a nicer API that includes generating tensors from simple data types, and returns classified values according to the output layer (eg. as a dict mapping class name to score)

**trace**(*inputs*)

## pytext.optimizer package

### Subpackages

## pytext.optimizer.sparsifiers package

### Submodules

## pytext.optimizer.sparsifiers.blockwise_sparsifier module

**class** pytext.optimizer.sparsifiers.blockwise_sparsifier.**BlockwiseMagnitudeSparsifier**(*sparsity starting_epo frequency, block_si column-wise_bl accumulate_ma layer-wise_pr*

Bases: *pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier*

running blockwise magnitude-based sparsification

**Parameters**

- **block_size** – define the size of each block
- **columnwise_blocking** – define columnwise block if true
- **starting_epoch** – sparsification_condition returns true only after starting_epoch
- **frequency** – sparsification_condition only if number of steps devides frequency
- **accumulate_mask** – if true, the mask after each .sparisfy() will be reused
- **sparsity** – percentage of zeros among the **UNPRUNED** parameters.
- **on how the sparsifier work** (*Examples*) –
- **matrix** (*2D*) –
- **[** – 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
- **]** –
- **3 X 1 block** (*define*) –
- **[** – ***** *** *0 1 2 3 4* ****** *** *5 6 7 8 9* ****** *** *10 11 12 13 14* ****** *** *15 16 17 18 19* ****** *** *20 21 22 23 24* ****** ***
- **]** –
- **l1 norm of each block and sort them. Retain blocks with largest** (*compute*) –
- **values until sparsity threshold is met** (*absolute*) –

**classmethod from_config** (*config: pytext.optimizer.sparsifiers.blockwise_sparsifier.BlockwiseMagnitudeSparsifier.Config*

**get_current_sparsity** (*model*)

**get_masks** (*model: torch.nn.modules.module.Module*, *pre_masks: List[torch.Tensor] = None*) → List[torch.Tensor]

Note: this function returns the masks only but do not sparsify or modify the weights

prune x% of weights among the weights with "1" in pre_masks

**Parameters**

- **model** – Model
- **pre_masks** – list of FloatTensors where "1" means retained the weight and "0" means pruned the weight

**Returns** List[torch.Tensor], intersection of new masks and pre_masks, so that "1" only if the weight is selected after new masking and pre_mask

**Return type** masks

**get_sparsifiable_params** (*model*, *requires_name=False*)

### pytext.optimizer.sparsifiers.sparsifier module

**class** pytext.optimizer.sparsifiers.sparsifier.**CRF_L1_SoftThresholding**(*lambda_l1: float, starting_epoch: int, frequency: int*)

Bases: *pytext.optimizer.sparsifiers.sparsifier.CRF_SparsifierBase*

**implement l1 regularization:** min Loss(x, y, CRFparams) + lambda_l1 * ||CRFparams||_1

and solve the optimiation problem via (stochastic) proximal gradient-based method i.e., soft-thresholding

param_updated = sign(CRFparams) * max ( abs(CRFparams) - lambda_l1, 0)

**classmethod from_config**(*config: pytext.optimizer.sparsifiers.sparsifier.CRF_L1_SoftThresholding.Config*)

**sparsify**(*state*)

**class** pytext.optimizer.sparsifiers.sparsifier.**CRF_MagnitudeThresholding**(*sparsity, starting_epoch, frequency, grouping*)

Bases: *pytext.optimizer.sparsifiers.sparsifier.CRF_SparsifierBase*

magnitude-based (equivalent to projection onto l0 constraint set) sparsification on CRF transition matrix. Preserveing the top-k elements either rowwise or columnwise until sparsity constraint is met.

**classmethod from_config**(*config: pytext.optimizer.sparsifiers.sparsifier.CRF_MagnitudeThresholding.Config*)

**sparsify**(*state*)

**class** pytext.optimizer.sparsifiers.sparsifier.**CRF_SparsifierBase**(*config=None, *args, **kwargs*)

Bases: *pytext.optimizer.sparsifiers.sparsifier.Sparsifier*

**get_sparsifiable_params**(*model: torch.nn.modules.module.Module*)

**get_transition_sparsity**(*transition*)

**sparsification_condition**(*state*)

**class** pytext.optimizer.sparsifiers.sparsifier.**L0_projection_sparsifier**(*sparsity, starting_epoch, frequency, layerwise_pruning=True, accumulate_mask=False*)

Bases: *pytext.optimizer.sparsifiers.sparsifier.Sparsifier*

L0 projection-based (unstructured) sparsification

**Parameters**

- **weights** (*torch.Tensor*) – input weight matrix
- **sparsity** (*float32*) – the desired sparsity [0-1]

**apply_masks**(*model: pytext.models.model.Model, masks: List[torch.Tensor]*)
apply given masks to zero-out learnable weights in model

**classmethod from_config**(*config: pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier.Config*)

**get_masks**(*model: pytext.models.model.Model, pre_masks: List[torch.Tensor] = None*) → List[torch.Tensor]
Note: this function returns the masks only but do not sparsify or modify the weights

prune x% of weights among the weights with "1" in pre_masks

**Parameters**

- **model** – Model
- **pre_masks** – list of FloatTensors where "1" means retained the weight and "0" means pruned the weight

**Returns** List[torch.Tensor], intersection of new masks and pre_masks, so that "1" only if the weight is selected after new masking and pre_mask

**Return type** masks

**get_sparsifiable_params**(*model: pytext.models.model.Model*)

**sparsification_condition**(*state*)

**sparsify**(*state*)
obtain a mask and apply the mask to sparsify

**class** pytext.optimizer.sparsifiers.sparsifier.**SensitivityAnalysisSparsifier**(*pre_train_model_pat analyzed_sparsity, max_analysis_batche max_skipped_weight, pre_analysis_path, sparsity, iterative_pruning, pruning_iterations, start_sparsity_ratio*)

Bases: *pytext.optimizer.sparsifiers.sparsifier.Sparsifier*

**apply_masks**(*model: pytext.models.model.Model, masks: List[torch.Tensor]*)
apply given masks to zero-out learnable weights in model

**find_params_to_prune**(*metric_dict, max_skip_weight_num*)

**classmethod from_config**(*config: pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier.Config*)

**get_current_sparsity**(*model: pytext.models.model.Model*) → float

**get_mask_for_param**(*param*, *sparsity*)
generate the prune mask for one weight tensor.

**get_masks**(*model: pytext.models.model.Model*) → List[torch.Tensor]
Note: this function returns the masks for each weight tensor if that tensor is required to be pruned

prune x% of weights items among the weights with "1" in mask (self._mask) indicate the remained weights, with "0" indicate pruned weights

**Parameters** **model** – Model

**Returns** List[torch.Tensor], the prune mask for the weight of all layers

**Return type** masks

**get_required_sparsifiable_params**(*model: pytext.models.model.Model*)

**get_sparsifiable_params**(*model*)

**increase_sparsity**(*state*)

**initialize**(*trainer*, *state*, *eval_data*, *metric_reporter*, *train_config*)

**layer_wise_analysis**(*param_name*, *param_dict*, *trainer*, *state*, *eval_data*, *metric_reporter*)

**load_analysis_from_path**()

**op_pre_epoch**(*trainer*, *state*)
note: invoke this function at the begin of each pruning iteration. Each pruning iteration contains several epochs. In this function, we will: 1. update the sparsity, 2. reload the best model from the previous iteration, 3. generate the prune mask, and 4. apply the mask to prune the weight of the model with increased sparsity.

**save_model_state_for_all_rank**()

**sensitivity_analysis**(*trainer*, *state*, *eval_data*, *metric_reporter*, *train_config*)
Analysis the sensitivity of each weight tensor to the metric. Prune the weight tensor one by one and evaluate the metric if the correspond weight tensor is pruned. :param trainer: batch iterator of training data :type trainer: trainer :param state: the state of the current training :type state: TrainingState :param eval_data: batch iterator of evaluation data :type eval_data: BatchIterator :param metric_reporter: compute metric based on training :type metric_reporter: MetricReporter :param output and report results to console, file.. etc: :param train_config: training config :type train_config: PyTextConfig

**Returns** a string of each layer sensitivity to metric.

**Return type** analysis_result

**sparsification_condition**(*state*)

**sparsify**(*state*)
apply the mask to sparsify the weight tensor

**class** pytext.optimizer.sparsifiers.sparsifier.**Sparsifier**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.config.component.Component*

**get_current_sparsity**(*model: pytext.models.model.Model*) → float

**get_sparsifiable_params**(**args*, ***kwargs*)

**initialize**(**args*, ***kwargs*)

**op_pre_epoch**(**args*, ***kwargs*)

**save_model_state_for_all_rank**()

**sparsification_condition**(**args*, ***kwargs*)

**sparsify**(**args*, ***kwargs*)

**class** pytext.optimizer.sparsifiers.sparsifier.**State**

Bases: enum.Enum

An enumeration.

**ANALYSIS = 'Analysis'**

**OTHERS = 'Others'**

### Module contents

### Submodules

### pytext.optimizer.activations module

pytext.optimizer.activations.**get_activation**(*name*, *dim=1*)

### pytext.optimizer.adabelief module

**class** pytext.optimizer.adabelief.**AdaBelief**(*params*, *lr=0.001*, *betas=(0.9, 0.999)*, *eps=1e-08*, *weight_decay=0*, *amsgrad=False*, *weight_decouple=False*, *fixed_decay=False*, *rectify=False*)

Bases: *pytext.optimizer.optimizers.Optimizer*, torch.optim.optimizer.Optimizer

*AdaBelief Optimizer, adapting stepsizes by the belief in observed gradients* Paper: https://arxiv.org/abs/2010.07468 Implementation has been copied over from the original author (https://github.com/juntang-zhuang/Adabelief-Optimizer)

**clip_grad_norm**(*max_norm*, *model=None*)

**classmethod from_config**(*config: pytext.optimizer.adabelief.AdaBelief.Config*, *model: torch.nn.modules.module.Module*)

**reset**()

**step**(*closure=None*, ***kwargs*)

Performs a single optimization step.

**Parameters** **closure** (*callable, optional*) – A closure that reevaluates the model and returns the loss.

### pytext.optimizer.fairseq_fp16_utils module

**class** pytext.optimizer.fairseq_fp16_utils.**Fairseq_FP16OptimizerMixin**(**args*, ***kwargs*)

Bases: object

**backward**(*loss*)

Computes the sum of gradients of the given tensor w.r.t. graph leaves.

Compared to fairseq.optim.FairseqOptimizer.backward(), this function additionally dynamically scales the loss to avoid gradient underflow.

**classmethod build_fp32_params**(*params*)

**clip_grad_norm**(*max_norm*)
Clips gradient norm and updates dynamic loss scaler.

**load_state_dict**(*state_dict*, *optimizer_overrides=None*)
Load an optimizer state dict.

In general we should prefer the configuration of the existing optimizer instance (e.g., learning rate) over that found in the state_dict. This allows us to resume training from a checkpoint using a new set of optimizer args.

**multiply_grads**(*c*)
Multiplies grads by a constant `c`.

**state_dict**()
Return the optimizer's state dict.

**step**(*closure=None*)
Performs a single optimization step.

**zero_grad**()
Clears the gradients of all optimized parameters.

**class** pytext.optimizer.fairseq_fp16_utils.**Fairseq_MemoryEfficientFP16OptimizerMixin**(**args*, ***kwargs*)

Bases: `object`

**backward**(*loss*)
Computes the sum of gradients of the given tensor w.r.t. graph leaves.

Compared to `fairseq.optim.FairseqOptimizer.backward()`, this function additionally dynamically scales the loss to avoid gradient underflow.

**clip_grad_norm**(*max_norm*)
Clips gradient norm and updates dynamic loss scaler.

**load_state_dict**(*state_dict*, *optimizer_overrides=None*)
Load an optimizer state dict.

In general we should prefer the configuration of the existing optimizer instance (e.g., learning rate) over that found in the state_dict. This allows us to resume training from a checkpoint using a new set of optimizer args.

**multiply_grads**(*c*)
Multiplies grads by a constant *c*.

**state_dict**()
Return the optimizer's state dict.

**step**(*closure=None*)
Performs a single optimization step.

**zero_grad**()
Clears the gradients of all optimized parameters.

### pytext.optimizer.fp16_optimizer module

**class** pytext.optimizer.fp16_optimizer.**DynamicLossScaler**(*init_scale*, *scale_factor*, *scale_window*)

Bases: `object`

**check_overflow**(*params*)

**check_overflow_**(*grad*)

**unscale**(*grad*)

**unscale_grads**(*param_groups*)

**update_scale**()

According to overflow situation, adjust loss scale.

Once overflow happened, we decrease the scale by scale_factor. Setting tolerance is another approach depending on cases.

If we haven't had overflows for #scale_window times, we should increase the scale by scale_factor.

**upscale**(*loss*)

**class** pytext.optimizer.fp16_optimizer.**FP16Optimizer**(*fp32_optimizer*)

Bases: *pytext.optimizer.optimizers.Optimizer*

**backward**(*loss*)

**clip_grad_norm**(*max_norm*, *model*)

**finalize**() → bool

**load_state_dict**(*state_dict*)

**param_groups**

**pre_export**(*model*)

**state_dict**()

**step**(*closure=None*)

**zero_grad**()

**class** pytext.optimizer.fp16_optimizer.**FP16OptimizerApex**(*fp32_optimizer: pytext.optimizer.optimizers.Optimizer, model: torch.nn.modules.module.Module, opt_level: str, init_loss_scale: Optional[int], min_loss_scale: Optional[float]*)

Bases: *pytext.optimizer.fp16_optimizer.FP16Optimizer*

**backward**(*loss*)

**clip_grad_norm**(*max_norm*, *model*)

**classmethod from_config**(*fp16_config: pytext.optimizer.fp16_optimizer.FP16OptimizerApex.Config, model: torch.nn.modules.module.Module, fp32_config: pytext.optimizer.optimizers.Optimizer.Config, *unused*)

**load_state_dict**(*state_dict*)

**pre_export**(*model*)

**state_dict**()

**step**(*closure=None*)

**zero_grad**()

**class** pytext.optimizer.fp16_optimizer.**FP16OptimizerDeprecated**(*init_optimizer*, *init_scale*, *scale_factor*, *scale_window*)

Bases: object

**finalize**()

**load_state_dict**(*state_dict*)

**scale_loss**(*loss*)

**state_dict**()

**step**()

Realize weights update.

Update the grads from model to master. During iteration for parameters, we check overflow after floating grads and copy. Then do unscaling.

If overflow doesn't happen, call inner optimizer's step() and copy back the updated weights from inner optimizer to model.

Update loss scale according to overflow checking result.

**zero_grad**()

**class** pytext.optimizer.fp16_optimizer.**FP16OptimizerFairseq**(*fp16_params*, *fp32_optimizer*, *init_loss_scale*, *scale_window*, *scale_tolerance*, *threshold_loss_scale*, *min_loss_scale*, *num_accumulated_batches*)

Bases: fairseq.optim.fp16_optimizer._FP16OptimizerMixin, *pytext.optimizer.fp16_optimizer.FP16Optimizer*

Wrap an *optimizer* to support FP16 (mixed precision) training.

**clip_grad_norm**(*max_norm*, *unused_model*)

Clips gradient norm and updates dynamic loss scaler.

**classmethod from_config**(*fp16_config: pytext.optimizer.fp16_optimizer.FP16OptimizerFairseq.Config*, *model: torch.nn.modules.module.Module*, *fp32_config: pytext.optimizer.optimizers.Optimizer.Config*, *num_accumulated_batches: int*)

**pre_export**(*model*)

**class** pytext.optimizer.fp16_optimizer.**GeneratorFP16Optimizer**(*init_optimizer*, *init_scale=65536.0*, *scale_factor=2*, *scale_window=2000*)

Bases: *pytext.optimizer.fp16_optimizer.PureFP16Optimizer*

**load_state_dict**(*state_dict*)

Load an optimizer state dict.

We prefer the configuration of the existing optimizer instance. After we load state dict to inner_optimizer, we create the copy of references of parameters again as in init().

**step**()

Updates weights.

**Effects:** Check overflow, if not, when inner_optimizer supports memory-effcient step, do overall unscale and call memory-efficient step.

If it doesn't support, modify each parameter list in param_groups of inner_optimizer to a generator of the tensors. Call normal step then, data type changing will be added automatically in that function.

No matter whether it is overflow, we need to update scale at the last step.

**class** pytext.optimizer.fp16_optimizer.**MemoryEfficientFP16OptimizerFairseq**(*fp16_params*, *optimizer*, *init_loss_scale*, *scale_window*, *scale_tolerance*, *threshold_loss_scale*, *min_loss_scale*, *num_accumulated_batch*

Bases: `fairseq.optim.fp16_optimizer._MemoryEfficientFP16OptimizerMixin`, *pytext.optimizer.fp16_optimizer.FP16Optimizer*

Wrap the mem efficient *optimizer* to support FP16 (mixed precision) training.

**clip_grad_norm**(*max_norm*, *unused_model*)

Clips gradient norm and updates dynamic loss scaler.

**classmethod from_config**(*fp16_config: pytext.optimizer.fp16_optimizer.MemoryEfficientFP16OptimizerFairseq.Config*, *model: torch.nn.modules.module.Module*, *fp32_config: pytext.optimizer.optimizers.Optimizer.Config*, *num_accumulated_batches: int*)

**pre_export**(*model*)

**class** pytext.optimizer.fp16_optimizer.**PureFP16Optimizer**(*init_optimizer*, *init_scale=65536.0*, *scale_factor=2*, *scale_window=2000*)

Bases: *pytext.optimizer.fp16_optimizer.FP16OptimizerDeprecated*

**load_state_dict**(*state_dict*)

Load an optimizer state dict.

We prefer the configuration of the existing optimizer instance. Realize the same logic as in init() – point the param_groups of outer optimizer to that of the inner_optimizer.

**scale_loss**(*loss*)

Scale the loss.

**Parameters** **loss** (*pytext.Loss*) – loss function object

**step**()

Updates the weights in inner optimizer.

If inner optimizer supports memory efficient, check overflow, unscale and call advanced step.

Otherwise, float weights and grads, check whether grads are overflow during the iteration, if not overflow, unscale grads and call inner optimizer's step; If overflow happens, do nothing, wait to the end to call half weights and grads (grads will be eliminated in zero_grad)

pytext.optimizer.fp16_optimizer.**convert_generator**(*params*, *scale*)

Create the generator for parameter tensors.

For each parameter, we float and unscale it. After the caller calls next(), we realize the half process and start next parameter's processing.

`pytext.optimizer.fp16_optimizer.`**`generate_params`**(*param_groups*)

`pytext.optimizer.fp16_optimizer.`**`initialize`**(*model*, *optimizer*, *opt_level*, *init_scale=65536*, *scale_factor=2.0*, *scale_window=2000*, *memory_efficient=False*)

`pytext.optimizer.fp16_optimizer.`**`master_params`**(*optimizer*)

`pytext.optimizer.fp16_optimizer.`**`scale_loss`**(*loss*, *optimizer*, *delay_unscale=False*)

### pytext.optimizer.lamb module

**class** `pytext.optimizer.lamb.`**`Lamb`**(*params*, *lr=0.001*, *betas=(0.9, 0.999)*, *eps=1e-06*, *weight_decay=0*, *min_trust=None*)

Bases: `pytext.optimizer.optimizers.Optimizer`, `torch.optim.optimizer.Optimizer`

Implements Lamb algorithm. THIS WAS DIRECTLY COPIED OVER FROM pytorch/contrib: https://github.com/cybertronai/pytorch-lamb It has been proposed in *Large Batch Optimization for Deep Learning: Training BERT in 76 minutes*. https://arxiv.org/abs/1904.00962

Has the option for minimum trust LAMB as described in "Single Headed Attention RNN: Stop Thinking With Your Head" section 6.3 https://arxiv.org/abs/1911.11423

**classmethod** **`from_config`**(*config: pytext.optimizer.lamb.Lamb.Config*, *model: torch.nn.modules.module.Module*)

**`step`**(*closure=None*, ***kwargs*)

Performs a single optimization step.

**Parameters** **`closure`** (*`callable, optional`*) – A closure that reevaluates the model and returns the loss.

### pytext.optimizer.madgrad module

**class** `pytext.optimizer.madgrad.`**`MADGRAD`**(*params*, *lr: float = 0.01*, *momentum: float = 0.9*, *weight_decay: float = 0*, *eps: float = 1e-06*, *k: int = 0*)

Bases: `pytext.optimizer.optimizers.Optimizer`, `torch.optim.optimizer.Optimizer`

*MADGRAD Optimizer*: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization. Paper: https://arxiv.org/abs/2101.11075

Implementation has been copied over from the original author (https://github.com/facebookresearch/madgrad/blob/master/madgrad/madgrad.py)

**`add_param_group`**(*param_group*)

Add a param group to the `Optimizer` s *param_groups*.

This can be useful when fine tuning a pre-trained network as frozen layers can be made trainable and added to the `Optimizer` as training progresses.

**Parameters**

- **`param_group`** (*`dict`*) – Specifies what Tensors should be optimized along
- **`group specific optimization options.`** (*`with`*) –

**`clip_grad_norm`**(*max_norm*, *model=None*)

**classmethod from_config**(*config: pytext.optimizer.madgrad.MADGRAD.Config, model: torch.nn.modules.module.Module*)

**initialize_state**()

**reset_param_groups**()

**step**(*closure=None, **kwargs*) → Optional[float]

Performs a single optimization step. :param closure: A closure that reevaluates the model

and returns the loss.

**supports_flat_params**

**supports_memory_efficient_fp16**

### pytext.optimizer.optimizers module

**class** pytext.optimizer.optimizers.**Adagrad**(*parameters, lr, weight_decay*)

Bases: torch.optim.adagrad.Adagrad, *pytext.optimizer.optimizers.Optimizer*

**classmethod from_config**(*config: pytext.optimizer.optimizers.Adagrad.Config, model: torch.nn.modules.module.Module*)

**class** pytext.optimizer.optimizers.**Adam**(*parameters, lr, weight_decay, eps*)

Bases: torch.optim.adam.Adam, *pytext.optimizer.optimizers.Optimizer*

**classmethod from_config**(*config: pytext.optimizer.optimizers.Adam.Config, model: torch.nn.modules.module.Module*)

**class** pytext.optimizer.optimizers.**AdamW**(*parameters, lr, weight_decay, eps*)

Bases: torch.optim.adamw.AdamW, *pytext.optimizer.optimizers.Optimizer*

Adds PyText support for Decoupled Weight Decay Regularization for Adam as done in the paper: https://arxiv.org/abs/1711.05101 for more information read the fast.ai blog on this optimization method here: https://www.fast.ai/2018/07/02/adam-weight-decay/

**classmethod from_config**(*config: pytext.optimizer.optimizers.AdamW.Config, model: torch.nn.modules.module.Module*)

**class** pytext.optimizer.optimizers.**Optimizer**(*config=None, *args, **kwargs*)

Bases: *pytext.config.component.Component*

**backward**(*loss*)

**clip_grad_norm**(*max_norm, model=None*)

**finalize**() → bool

**multiply_grads**(*c*)

Multiplies grads by a constant *c*.

**params**

Return an iterable of the parameters held by the optimizer.

**pre_export**(*model*)

**reset_param_groups**()

**class** pytext.optimizer.optimizers.**SGD**(*parameters, lr, momentum*)

Bases: torch.optim.sgd.SGD, *pytext.optimizer.optimizers.Optimizer*

**classmethod from_config**(*config: pytext.optimizer.optimizers.SGD.Config, model: torch.nn.modules.module.Module*)

pytext.optimizer.optimizers.**learning_rates**(*optimizer*)

### pytext.optimizer.privacy_engine module

**class** pytext.optimizer.privacy_engine.**PrivacyEngine**(*model, optimizer, noise_multiplier, max_grad_norm, batch_size, dataset_size, target_delta, alphas*)

Bases: *pytext.config.component.Component*

A wrapper around PrivacyEngine of Opacus

**attach**(*optimizer*)

**detach**()

**classmethod from_config**(*config: pytext.optimizer.privacy_engine.PrivacyEngine.Config, model, optimizer*)

**get_privacy_spent**()

### pytext.optimizer.radam module

**class** pytext.optimizer.radam.**RAdam**(*params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0*)

Bases: *pytext.optimizer.optimizers.Optimizer*, torch.optim.optimizer.Optimizer

Implements rectified adam as derived in the following paper: "On the Variance of the Adaptive Learning Rate and Beyond" (https://arxiv.org/abs/1908.03265)

This code is mostly a direct copy-paste of the code provided by the authors here: https://github.com/LiyuanLucasLiu/RAdam/blob/master/radam.py

**classmethod from_config**(*config: pytext.optimizer.radam.RAdam.Config, model: torch.nn.modules.module.Module*)

**step**(*closure=None, **kwargs*)

Performs a single optimization step (parameter update).

**Parameters** **closure** (*callable*) – A closure that reevaluates the model and returns the loss. Optional for most optimizers.

**Note:** Unless otherwise specified, this function should not modify the `.grad` field of the parameters.

### pytext.optimizer.scheduler module

**class** pytext.optimizer.scheduler.**BatchScheduler**(*config=None, *args, **kwargs*)

Bases: *pytext.optimizer.scheduler.Scheduler*

**prepare**(*train_iter, total_epochs*)

**class** pytext.optimizer.scheduler.**CosineAnnealingLR**(*optimizer*, *T_max*, *eta_min=0*, *last_epoch=-1*, *verbose=False*)

Bases: torch.optim.lr_scheduler.CosineAnnealingLR, pytext.optimizer.scheduler.BatchScheduler

Wrapper around *torch.optim.lr_scheduler.CosineAnnealingLR* See the original documentation for more details.

**classmethod from_config**(*config: pytext.optimizer.scheduler.CosineAnnealingLR.Config*, *optimizer: pytext.optimizer.optimizers.Optimizer*)

**step_batch**(*metrics=None*, *epoch=None*)

**class** pytext.optimizer.scheduler.**CyclicLR**(*optimizer*, *base_lr*, *max_lr*, *step_size_up=2000*, *step_size_down=None*, *mode='triangular'*, *gamma=1.0*, *scale_fn=None*, *scale_mode='cycle'*, *cycle_momentum=True*, *base_momentum=0.8*, *max_momentum=0.9*, *last_epoch=-1*, *verbose=False*)

Bases: torch.optim.lr_scheduler.CyclicLR, pytext.optimizer.scheduler.BatchScheduler

Wrapper around *torch.optim.lr_scheduler.CyclicLR* See the original documentation for more details

**classmethod from_config**(*config: pytext.optimizer.scheduler.CyclicLR.Config*, *optimizer: pytext.optimizer.optimizers.Optimizer*)

**step_batch**(*metrics=None*, *epoch=None*)

**class** pytext.optimizer.scheduler.**ExponentialLR**(*optimizer*, *gamma*, *last_epoch=-1*, *verbose=False*)

Bases: torch.optim.lr_scheduler.ExponentialLR, pytext.optimizer.scheduler.Scheduler

Wrapper around *torch.optim.lr_scheduler.ExponentialLR* See the original documentation for more details.

**classmethod from_config**(*config: pytext.optimizer.scheduler.ExponentialLR.Config*, *optimizer: pytext.optimizer.optimizers.Optimizer*)

**step_epoch**(*metrics=None*, *epoch=None*)

**class** pytext.optimizer.scheduler.**LmFineTuning**(*optimizer*, *cut_frac=0.1*, *ratio=32*, *non_pretrained_param_groups=2*, *lm_lr_multiplier=1.0*, *lm_use_per_layer_lr=False*, *lm_gradual_unfreezing=True*, *last_epoch=-1*)

Bases: torch.optim.lr_scheduler._LRScheduler, pytext.optimizer.scheduler.BatchScheduler

Fine-tuning methods from the paper "[arXiv:1801.06146]Universal Language Model Fine-tuning for Text Classification".

Specifically, modifies training schedule using slanted triangular learning rates, discriminative fine-tuning (per-layer learning rates), and gradual unfreezing.

**classmethod from_config**(*config: pytext.optimizer.scheduler.LmFineTuning.Config*, *optimizer*)

**get_lr**()

**step_batch**(*metrics=None*, *epoch=None*)

**class** pytext.optimizer.scheduler.**PolynomialDecayScheduler**(*optimizer*, *warmup_steps*, *total_steps*, *end_learning_rate*, *power*)

Bases: torch.optim.lr_scheduler._LRScheduler, *pytext.optimizer.scheduler.BatchScheduler*

Applies a polynomial decay with lr warmup to the learning rate.

It is commonly observed that a monotonically decreasing learning rate, whose degree of change is carefully chosen, results in a better performing model.

This scheduler linearly increase learning rate from 0 to final value at the beginning of training, determined by warmup_steps. Then it applies a polynomial decay function to an optimizer step, given a provided *base_lrs* to reach an *end_learning_rate* after *total_steps*.

**classmethod from_config**(*config: pytext.optimizer.scheduler.PolynomialDecayScheduler.Config*, *optimizer: pytext.optimizer.optimizers.Optimizer*)

**get_lr**()

**prepare**(*train_iter*, *total_epochs*)

**step_batch**()

**class** pytext.optimizer.scheduler.**ReduceLROnPlateau**(*optimizer*, *mode='min'*, *factor=0.1*, *patience=10*, *threshold=0.0001*, *threshold_mode='rel'*, *cooldown=0*, *min_lr=0*, *eps=1e-08*, *verbose=False*)

Bases: torch.optim.lr_scheduler.ReduceLROnPlateau, *pytext.optimizer.scheduler.Scheduler*

Wrapper around *torch.optim.lr_scheduler.ReduceLROnPlateau* See the original documentation for more details.

**classmethod from_config**(*config: pytext.optimizer.scheduler.ReduceLROnPlateau.Config*, *optimizer: pytext.optimizer.optimizers.Optimizer*)

**step_epoch**(*metrics*, *epoch*)

**class** pytext.optimizer.scheduler.**Scheduler**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.config.component.Component*

Schedulers help in adjusting the learning rate during training. Scheduler is a wrapper class over schedulers which can be available in torch library or for custom implementations. There are two kinds of lr scheduling that is supported by this class. Per epoch scheduling and per batch scheduling. In per epoch scheduling, the learning rate is adjusted at the end of each epoch and in per batch scheduling the learning rate is adjusted after the forward and backward pass through one batch during the training.

There are two main methods that needs to be implemented by the Scheduler. step_epoch() is called at the end of each epoch and step_batch() is called at the end of each batch in the training data.

prepare() method can be used by BatchSchedulers to initialize any attributes they may need.

**prepare**(*train_iter*, *total_epochs*)

**step_batch**(***kwargs*) → None

**step_epoch**(***kwargs*) → None

**class** pytext.optimizer.scheduler.**SchedulerWithWarmup**(*optimizer*, *warmup_scheduler*, *scheduler*, *switch_steps*)

Bases: torch.optim.lr_scheduler._LRScheduler, *pytext.optimizer.scheduler.BatchScheduler*

Wraps another scheduler with a warmup phase. After *warmup_steps* defined in warmup_scheduler.warmup_steps, the scheduler will switch to use the specified scheduler in *scheduler*.

*warmup_scheduler*: is the configuration for the WarmupScheduler, that warms up learning rate over *warmup_steps* linearly.

*scheduler*: is the main scheduler that will be applied after the warmup phase (once *warmup_steps* have passed)

**classmethod from_config**(*config: pytext.optimizer.scheduler.SchedulerWithWarmup.Config*, *optimizer: pytext.optimizer.optimizers.Optimizer*)

**get_lr**()

**prepare**(*train_iter*, *total_epochs*)

**step_batch**()

**step_epoch**(*metrics*, *epoch*)

**class** pytext.optimizer.scheduler.**StepLR**(*optimizer*, *step_size*, *gamma=0.1*, *last_epoch=-1*, *verbose=False*)

Bases: torch.optim.lr_scheduler.StepLR, *pytext.optimizer.scheduler.Scheduler*

Wrapper around *torch.optim.lr_scheduler.StepLR* See the original documentation for more details.

**classmethod from_config**(*config: pytext.optimizer.scheduler.StepLR.Config*, *optimizer*)

**step_epoch**(*metrics=None*, *epoch=None*)

**class** pytext.optimizer.scheduler.**WarmupScheduler**(*optimizer*, *warmup_steps*, *inverse_sqrt_decay*)

Bases: torch.optim.lr_scheduler._LRScheduler, *pytext.optimizer.scheduler.BatchScheduler*

Scheduler to linearly increase the learning rate from 0 to its final value over a number of steps:

lr = base_lr * current_step / warmup_steps

After the warm-up phase, the scheduler has the option of decaying the learning rate as the inverse square root of the number of training steps taken:

lr = base_lr * sqrt(warmup_steps) / sqrt(current_step)

**classmethod from_config**(*config: pytext.optimizer.scheduler.WarmupScheduler.Config*, *optimizer: pytext.optimizer.optimizers.Optimizer*)

**get_lr**()

**prepare**(*train_iter*, *total_epochs*)

**step_batch**()

### pytext.optimizer.swa module

**class** pytext.optimizer.swa.**StochasticWeightAveraging**(*optimizer*, *swa_start=None*, *swa_freq=None*, *swa_lr=None*)

Bases: *pytext.optimizer.optimizers.Optimizer*, torch.optim.optimizer.Optimizer

**add_param_group**(*param_group*)
Add a param group to the `Optimizer` s *param_groups*.

This can be useful when fine tuning a pre-trained network as frozen layers can be made trainable and added to the `Optimizer` as training progresses.

**Parameters**

- **param_group** (*dict*) – Specifies what Tensors should be optimized along
- **group specific optimization options.** (*with*) –

**static bn_update**(*loader*, *model*, *device=None*)
Updates BatchNorm running_mean, running_var buffers in the model.

It performs one pass over data in *loader* to estimate the activation statistics for BatchNorm layers in the model.

**Parameters**

- **loader** (*torch.utils.data.DataLoader*) – dataset loader to compute the activation statistics on. Each data batch should be either a tensor, or a list/tuple whose first element is a tensor containing data.
- **model** (*torch.nn.Module*) – model for which we seek to update BatchNorm statistics.
- **device** (*torch.device, optional*) – If set, data will be trasferred to `device` before being passed into `model`.

**finalize**()
Swaps the values of the optimized variables and swa buffers.

It's meant to be called in the end of training to use the collected swa running averages. It can also be used to evaluate the running averages during training; to continue training *swap_swa_sgd* should be called again.

**classmethod from_config**(*config: pytext.optimizer.swa.StochasticWeightAveraging.Config*, *model: torch.nn.modules.module.Module*)

**load_state_dict**(*state_dict*)
Loads the optimizer state.

**Parameters** **state_dict** (*dict*) – SWA optimizer state. Should be an object returned from a call to *state_dict*.

**reset_param_groups**()

**state_dict**()
Returns the state of SWA as a `dict`.

**It contains three entries:**

- **opt_state - a dict holding current optimization state of the base** optimizer. Its content differs between optimizer classes.
- **swa_state - a dict containing current state of SWA. For each** optimized variable it contains swa_buffer keeping the running average of the variable
- param_groups - a dict containing all parameter groups

**step**(*closure=None*, ***kwargs*)
Performs a single optimization step.

In automatic mode also updates SWA running averages.

**update_swa**()
Updates the SWA running averages of all optimized parameters.

**update_swa_group**(*group*)
Updates the SWA running averages for the given parameter group.

**Parameters** **param_group** (*dict*) – Specifies for what parameter group SWA running averages should be updated

### Examples

```
>>> # automatic mode
>>> base_opt = torch.optim.SGD([{'params': [x]},
>>>             {'params': [y], 'lr': 1e-3}], lr=1e-2, momentum=0.9)
>>> opt = torchcontrib.optim.SWA(base_opt)
>>> for i in range(100):
>>>     opt.zero_grad()
>>>     loss_fn(model(input), target).backward()
>>>     opt.step()
>>>     if i > 10 and i % 5 == 0:
>>>         # Update SWA for the second parameter group
>>>         opt.update_swa_group(opt.param_groups[1])
>>> opt.swap_swa_sgd()
```

## Module contents

## pytext.resources package

## Submodules

## pytext.resources.roberta module

## Module contents

## pytext.task package

## Submodules

## pytext.task.accelerator_lowering module

**class** pytext.task.accelerator_lowering.**AcceleratorBiLSTM**(*biLSTM*)
Bases: torch.nn.modules.module.Module

**forward**(*embedded_tokens: torch.Tensor, seq_lengths: torch.Tensor, states: Tuple[torch.Tensor, torch.Tensor]*) → Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]
Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.task.accelerator_lowering.**AcceleratorTransformer**(*transformer*)

Bases: torch.nn.modules.module.Module

**forward**(*tokens: torch.Tensor*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**class** pytext.task.accelerator_lowering.**AcceleratorTransformerLayersInternal**(*layers*)

Bases: torch.nn.modules.module.Module

**forward**(*encoded: torch.Tensor*, *padding_mask: torch.Tensor*) → List[torch.Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

pytext.task.accelerator_lowering.**accelerator_lstm_inputs**(*model: torch.nn.modules.module.Module, trace: torch.jit.ScriptFunction, export_options: pytext.config.pytext_config.ExportConfig, dataset_iterable: Iterable[T_co], module_path*)

pytext.task.accelerator_lowering.**accelerator_transformerLayers_inputs**(*model: torch.nn.modules.module.Mod trace: torch.jit.ScriptFunction, export_options: pytext.config.pytext_config.Expor dataset_iterable: Iterable[T_co], module_path*)

pytext.task.accelerator_lowering.**lower_modules_to_accelerator**(*model: torch.nn.modules.module.Module, trace, export_options: pytext.config.pytext_config.ExportConfig, throughput_optimize=False*)

pytext.task.accelerator_lowering.**swap_modules_for_accelerator**(*model*)

### pytext.task.disjoint_multitask module

**class** pytext.task.disjoint_multitask.**DisjointMultitask**(*target_task_name*, *exporters*, ***kwargs*)

Bases: *pytext.task.task.TaskBase*

Modules which have the same shared_module_key and type share parameters. Only the first instance of such module should be configured in tasks list.

**export**(*multitask_model*, *export_path*, *metric_channels*, *export_onnx_path=None*)

Wrapper method to export PyTorch model to Caffe2 model using `Exporter`.

**Parameters**

- **export_path** (*str*) – file path of exported caffe2 model
- **metric_channels** – output the PyTorch model's execution graph to
- **export_onnx_path** (*str*) – file path of exported onnx model

**classmethod from_config**(*task_config: pytext.task.disjoint_multitask.DisjointMultitask.Config*, *metadata=None*, *model_state=None*, *tensorizers=None*, *rank=0*, *world_size=1*)

Create the task from config, and optionally load metadata/model_state This function will create components including `DataHandler`, `Trainer`, `MetricReporter`, `Exporter`, and wire them up.

**Parameters**

- **task_config** (*Task.Config*) – the config of the current task
- **metadata** – saved global context of this task, e.g: vocabulary, will be generated by `DataHandler` if it's None
- **model_state** – saved model parameters, will be loaded into model when given

**class** pytext.task.disjoint_multitask.**NewDisjointMultitask**(*data: pytext.data.data.Data*, *model: pytext.models.model.BaseModel*, *metric_reporter: Optional[pytext.metric_reporters.metric_reporter.M = None*, *trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: `pytext.task.new_task._NewTask`

Multitask training based on underlying subtasks. To share parameters between modules from different tasks, specify the same shared_module_key. Only the first instance of each shared module should be configured in tasks list. Only the multitask trainer (not the per-task trainers) is used.

**export**(*model*, *export_path*, *metric_channels=None*, *export_onnx_path=None*)

Wrapper method to export PyTorch model to Caffe2 model using `Exporter`.

**Parameters**

- **export_path** (*str*) – file path of exported caffe2 model
- **metric_channels** (*List[Channel]*) – outputs of model's execution graph
- **export_onnx_path** (*str*) – file path of exported onnx model

**classmethod from_config**(*task_config: pytext.task.disjoint_multitask.NewDisjointMultitask.Config, unused_metadata=None, model_state=None, tensorizers=None, rank=0, world_size=1*)

Create the task from config, and optionally load metadata/model_state This function will create components including `DataHandler`, `Trainer`, `MetricReporter`, `Exporter`, and wire them up.

**Parameters**

- **task_config** (*Task.Config*) – the config of the current task
- **metadata** – saved global context of this task, e.g: vocabulary, will be generated by `DataHandler` if it's None
- **model_state** – saved model parameters, will be loaded into model when given

**torchscript_export**(*model, export_path, export_config=None*)

### pytext.task.new_task module

**class** pytext.task.new_task.**NewTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: `pytext.task.new_task._NewTask`

pytext.task.new_task.**create_schema**(*tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], extra_schema: Optional[Dict[str, Type[CT_co]]] = None*) → Dict[str, Type[CT_co]]

pytext.task.new_task.**create_tensorizers**(*model_inputs: Union[pytext.models.model.BaseModel.Config.ModelInput, Dict[str, pytext.data.tensorizers.Tensorizer.Config]]*) → Dict[str, pytext.data.tensorizers.Tensorizer]

pytext.task.new_task.**sort**(*input, dim=-1, descending=False, *, out=None) -> (Tensor, LongTensor*)

Sorts the elements of the `input` tensor along a given dimension in ascending order by value.

If `dim` is not given, the last dimension of the *input* is chosen.

If `descending` is `True` then the elements are sorted in descending order by value.

A namedtuple of (values, indices) is returned, where the *values* are the sorted values and *indices* are the indices of the elements in the original *input* tensor.

**Parameters**

- **input** (*Tensor*) – the input tensor.
- **dim** (*int, optional*) – the dimension to sort along
- **descending** (*bool, optional*) – controls the sorting order (ascending or descending)

**Keyword Arguments** **out** (*tuple, optional*) – the output tuple of (*Tensor*, *LongTensor*) that can be optionally given to be used as output buffers

Example:

```
>>> x = torch.randn(3, 4)
>>> sorted, indices = torch.sort(x)
>>> sorted
```

(continues on next page)

(continued from previous page)

```
tensor([[-0.2162,  0.0608,  0.6719,  2.3332],
        [-0.5793,  0.0061,  0.6058,  0.9497],
        [-0.5071,  0.3343,  0.9553,  1.0960]])
>>> indices
tensor([[ 1,  0,  2,  3],
        [ 3,  1,  0,  2],
        [ 0,  3,  1,  2]])

>>> sorted, indices = torch.sort(x, 0)
>>> sorted
tensor([[-0.5071, -0.2162,  0.6719, -0.5793],
        [ 0.0608,  0.0061,  0.9497,  0.3343],
        [ 0.6058,  0.9553,  1.0960,  2.3332]])
>>> indices
tensor([[ 2,  0,  0,  1],
        [ 0,  1,  1,  2],
        [ 1,  2,  2,  0]])
```

### pytext.task.nop_decorator module

**class** pytext.task.nop_decorator.**accelerator**(*specs*, *inputs_function=None*)

Bases: object

**classmethod get_embedding_module_from_path**(*model*, *submod_path*)

**classmethod get_module_from_path**(*model*, *prefixes*)

**classmethod get_modules**(*model*, *backend*)

### pytext.task.quantize module

pytext.task.quantize.**quantize_fx**(*model*, *inputs*, *data_loader*, *dynamic=True*)

pytext.task.quantize.**quantize_statically**(*model*, *inputs*, *data_loader*, *linear_only=False*, *module_swap=False*)

### pytext.task.serialize module

**class** pytext.task.serialize.**CheckpointManager**

Bases: *pytext.task.serialize.PyTextCheckpointManagerInterface*

**get_latest_checkpoint_path**() → str

Return most recent saved checkpoint path in str Returns: checkpoint_path (str)

**get_post_training_snapshot_path**() → str

**list**() → List[str]

Return all existing checkpoint paths Returns: checkpoint_path_list (List[str]), list elements are in the same order of checkpoint saving

**load**(*load_path: str*)

Loads a checkpoint/snapshot from disk. :param load_path: the file path from which to load :type load_path: str

Returns: De-serialized state (dictionary) that was saved

**save** (*state*, *save_path*)

**save_checkpoint** (*state*, *checkpoint_path*)
Serialize and save checkpoint to given path. State is a dictionary that represents the all data to be saved. :param state: Dictionary containing data to be saved :param checkpoint_path: path of file to save checkpoint

**save_snapshot** (*state*, *snapshot_path*)
Serialize and save post-training model snapshot to given path. State is a dictionary that represents the all data to be saved. Having a separate method for snapshots enables future optimizations like quantization to be applied to snapshots.

**Parameters**

- **state** – Dictionary containing data to be saved
- **snapshot_path** – path of file to save snapshot

**class** pytext.task.serialize.**PyTextCheckpointManagerInterface**
Bases: abc.ABC

CheckpointManager is a class abstraction to manage a training job's checkpoints/snapshots with different IO and storage.

**get_latest_checkpoint_path** () → str
Return most recent saved checkpoint path in str Returns: checkpoint_path (str)

**get_post_training_snapshot_path** () → str

**list** () → List[str]
Return all existing checkpoint paths Returns: checkpoint_path_list (List[str]), list elements are in the same order of checkpoint saving

**load** (*load_path: str*)
Loads a checkpoint/snapshot from disk. :param load_path: the file path from which to load :type load_path: str

Returns: De-serialized state (dictionary) that was saved

**save_checkpoint** (*state*, *checkpoint_path*)
Serialize and save checkpoint to given path. State is a dictionary that represents the all data to be saved. :param state: Dictionary containing data to be saved :param checkpoint_path: path of file to save checkpoint

**save_snapshot** (*state*, *snapshot_path*)
Serialize and save post-training model snapshot to given path. State is a dictionary that represents the all data to be saved. Having a separate method for snapshots enables future optimizations like quantization to be applied to snapshots.

**Parameters**

- **state** – Dictionary containing data to be saved
- **snapshot_path** – path of file to save snapshot

pytext.task.serialize.**generate_checkpoint_path** (*config: pytext.config.pytext_config.PyTextConfig*, *identifier: str*)

pytext.task.serialize.**get_latest_checkpoint_path** (*dir_path: Optional[str] = None*) → str
Get the latest checkpoint path :param dir_path: the dir to scan for existing checkpoint files. Default: if None, :param the latest checkpoint path saved in momery will be returned:

Returns: checkpoint_path

pytext.task.serialize.**get_post_training_snapshot_path**() → str

pytext.task.serialize.**load**(*load_path: str*, *overwrite_config=None*, *rank=0*, *world_size=1*)

Load task, config and training state from a saved snapshot by default, it will construct the task using the saved config then load metadata and model state.

if overwrite_task is specified, it will construct the task using overwrite_task then load metadata and model state.

pytext.task.serialize.**load_checkpoint**(*state*, *overwrite_config=None*, *rank=0*, *world_size=1*)

pytext.task.serialize.**load_v1**(*state*)

pytext.task.serialize.**load_v2**(*state*)

pytext.task.serialize.**load_v3**(*state*, *overwrite_config=None*, *rank=0*, *world_size=1*)

pytext.task.serialize.**register_snapshot_loader**(*version*)

pytext.task.serialize.**save**(*config: pytext.config.pytext_config.PyTextConfig, model: pytext.models.model.Model, meta: Optional[pytext.data.data_handler.CommonMetadata], tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], training_state: Optional[pytext.trainers.training_state.TrainingState] = None, identifier: Optional[str] = None*) → str

Save all stateful information of a training task to a specified file-like object, will save the original config, model state, metadata, training state if training is not completed Args: identifier (str): used to identify a checkpoint within a training job, used as a suffix for save path config (PytextConfig): contains all raw parameter/hyper-parameters for training task model (Model): actual model in training training_state (TrainingState): stateful infomation during training Returns: identifier (str): if identifier is not specified, will save to config.save_snapshot_path to be consistent to post-training snapshot; if specified, will be used to save checkpoint during training, identifier is used to identify checkpoints in the same training

pytext.task.serialize.**save_checkpoint**(*f: io.IOBase, config: pytext.config.pytext_config.PyTextConfig, model: pytext.models.model.Model, meta: Optional[pytext.data.data_handler.CommonMetadata], tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], training_state: Optional[pytext.trainers.training_state.TrainingState] = None*) → str

pytext.task.serialize.**set_checkpoint_manager**(*manager: pytext.task.serialize.PyTextCheckpointManagerInterface*) → None

### pytext.task.task module

**class** pytext.task.task.**TaskBase**(*trainer: pytext.trainers.trainer.Trainer, data_handler: pytext.data.data_handler.DataHandler, model: pytext.models.model.Model, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, exporter: Optional[pytext.exporters.exporter.ModelExporter]*)

Bases: *pytext.config.component.Component*

Task is the central place to define and wire up components for data processing, model training, metric reporting, etc. Task class has a Config class containing the config of each component in a descriptive way.

**export** (*model*, *export_path*, *metric_channels=None*, *export_onnx_path=None*)
Wrapper method to export PyTorch model to Caffe2 model using `Exporter`.

**Parameters**

- **export_path** (*str*) – file path of exported caffe2 model
- **metric_channels** (*List[Channel]*) – outputs of model's execution graph
- **export_onnx_path** (*str*) – file path of exported onnx model

**classmethod format_prediction** (*predictions*, *scores*, *context*, *target_meta*)
Format the prediction and score from model output, by default just return them in a dict

**classmethod from_config** (*task_config*, *metadata=None*, *model_state=None*, *tensorizers=None*, *rank=1*, *world_size=0*)
Create the task from config, and optionally load metadata/model_state This function will create components including `DataHandler`, `Trainer`, `MetricReporter`, `Exporter`, and wire them up.

**Parameters**

- **task_config** (*Task.Config*) – the config of the current task
- **metadata** – saved global context of this task, e.g: vocabulary, will be generated by `DataHandler` if it's None
- **model_state** – saved model parameters, will be loaded into model when given

**predict** (*examples*)
Generates predictions using PyTorch model. The difference with *test()* is that this should be used when the the examples do not have any true label/target.

**Parameters** **examples** – json format examples, input names should match the names specified in this task's features config

**test** (*test_path*)
Wrapper method to compute test metrics on holdout blind test dataset.

**Parameters** **test_path** (*str*) – test data file path

**train** (*train_config*, *rank=0*, *world_size=1*, *training_state=None*)
Wrapper method to train the model using `Trainer` object.

**Parameters**

- **train_config** (*PyTextConfig*) – config for training
- **rank** (*int*) – for distributed training only, rank of the gpu, default is 0
- **world_size** (*int*) – for distributed training only, total gpu to use, default is 1

**class** pytext.task.task.**Task_Deprecated** (*trainer: pytext.trainers.trainer.Trainer, data_handler: pytext.data.data_handler.DataHandler, model: pytext.models.model.Model, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, exporter: Optional[pytext.exporters.exporter.ModelExporter]*)
Bases: *pytext.task.task.TaskBase*

pytext.task.task.**create_task** (*task_config*, *metadata=None*, *model_state=None*, *tensorizers=None*, *rank=0*, *world_size=1*)
Create a task by finding task class in registry and invoking the from_config function of the class, see `from_config()` for more details

### pytext.task.tasks module

**class** pytext.task.tasks.**BertPairRegressionTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.tasks.DocumentRegressionTask*

**class** pytext.task.tasks.**DocumentClassificationTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricRepo = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**classmethod format_prediction**(*predictions, scores, context, target_names*)

Format the prediction and score from model output, by default just return them in a dict

**class** pytext.task.tasks.**DocumentRegressionTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**class** pytext.task.tasks.**EnsembleTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**classmethod example_config**()

**train_single_model**(*train_config, model_id, rank=0, world_size=1*)

**class** pytext.task.tasks.**IntentSlotTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**class** pytext.task.tasks.**LMTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**class** pytext.task.tasks.**MaskedLMTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**class** pytext.task.tasks.**NewBertClassificationTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReport = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.tasks.DocumentClassificationTask*

**class** pytext.task.tasks.**NewBertPairClassificationTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricR = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.tasks.DocumentClassificationTask*

**class** pytext.task.tasks.**PairwiseClassificationForDenseRetrievalTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: pytext.metric_reporters.classificatio trainer: pytext.trainers.trainer.TaskTrainer, trace_both_encoders: bool = True*)

Bases: *pytext.task.tasks.PairwiseClassificationTask*

This task is to implement DPR training in PyText. Code pointer: https://github.com/fairinternal/DPR/tree/master/dpr

**classmethod create_metric_reporter**(*config: pytext.task.tasks.PairwiseClassificationForDenseRetrievalTask.Confi *args, **kwargs*)

**class** pytext.task.tasks.**PairwiseClassificationTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: pytext.metric_reporters.classification_metric_reporter.Classif trainer: pytext.trainers.trainer.TaskTrainer, trace_both_encoders: bool = True*)

Bases: pytext.task.new_task.NewTask

**classmethod from_config**(*config: pytext.task.tasks.PairwiseClassificationTask.Config, unused_metadata=None, model_state=None, tensorizers=None, rank=0, world_size=1*)

Create the task from config, and optionally load metadata/model_state This function will create components including `DataHandler`, `Trainer`, `MetricReporter`, `Exporter`, and wire them up.

**Parameters**

- **task_config** (*Task.Config*) – the config of the current task
- **metadata** – saved global context of this task, e.g: vocabulary, will be generated by `DataHandler` if it's None
- **model_state** – saved model parameters, will be loaded into model when given

**torchscript_export**(*model, export_path=None, export_config=None*)

**class** pytext.task.tasks.**PairwiseRegressionTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: pytext.metric_reporters.classification_metric_reporter.Classificatio trainer: pytext.trainers.trainer.TaskTrainer, trace_both_encoders: bool = True*)

Bases: pytext.task.tasks.PairwiseClassificationTask

**class** pytext.task.tasks.**QueryDocumentPairwiseRankingTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.Me = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: pytext.task.new_task.NewTask

**class** pytext.task.tasks.**RoBERTaNERTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: pytext.task.new_task.NewTask

**classmethod create_metric_reporter**(*config: pytext.task.tasks.RoBERTaNERTask.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

**class** pytext.task.tasks.**SemanticParsingTask**(*data: pytext.data.data.Data, model: pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser, metric_reporter: pytext.metric_reporters.compositional_metric_reporter.CompositionalM trainer: pytext.trainers.hogwild_trainer.HogwildTrainer*)

Bases: pytext.task.new_task.NewTask

**class** pytext.task.tasks.**SeqNNTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**class** pytext.task.tasks.**SequenceLabelingTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**torchscript_export**(*model, export_path=None, export_config=None*)

**class** pytext.task.tasks.**SquadQATask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**class** pytext.task.tasks.**WordTaggingTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

**classmethod create_metric_reporter**(*config: pytext.task.tasks.WordTaggingTask.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

### Module contents

**class** pytext.task.**NewTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.trainers.trainer.TaskTrainer] = None*)

Bases: pytext.task.new_task._NewTask

**class** pytext.task.**Task_Deprecated**(*trainer: pytext.trainers.trainer.Trainer, data_handler: pytext.data.data_handler.DataHandler, model: pytext.models.model.Model, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, exporter: Optional[pytext.exporters.exporter.ModelExporter]*)

Bases: *pytext.task.task.TaskBase*

**class** pytext.task.**TaskBase**(*trainer: pytext.trainers.trainer.Trainer, data_handler: pytext.data.data_handler.DataHandler, model: pytext.models.model.Model, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, exporter: Optional[pytext.exporters.exporter.ModelExporter]*)

Bases: `pytext.config.component.Component`

Task is the central place to define and wire up components for data processing, model training, metric reporting, etc. Task class has a Config class containing the config of each component in a descriptive way.

**export** (*model*, *export_path*, *metric_channels=None*, *export_onnx_path=None*)
Wrapper method to export PyTorch model to Caffe2 model using `Exporter`.

**Parameters**

- **export_path** (*str*) – file path of exported caffe2 model
- **metric_channels** (*List[Channel]*) – outputs of model's execution graph
- **export_onnx_path** (*str*) – file path of exported onnx model

**classmethod format_prediction** (*predictions*, *scores*, *context*, *target_meta*)
Format the prediction and score from model output, by default just return them in a dict

**classmethod from_config** (*task_config*, *metadata=None*, *model_state=None*, *tensorizers=None*, *rank=1*, *world_size=0*)
Create the task from config, and optionally load metadata/model_state This function will create components including `DataHandler`, `Trainer`, `MetricReporter`, `Exporter`, and wire them up.

**Parameters**

- **task_config** (*Task.Config*) – the config of the current task
- **metadata** – saved global context of this task, e.g: vocabulary, will be generated by `DataHandler` if it's None
- **model_state** – saved model parameters, will be loaded into model when given

**predict** (*examples*)
Generates predictions using PyTorch model. The difference with *test()* is that this should be used when the the examples do not have any true label/target.

**Parameters** **examples** – json format examples, input names should match the names specified in this task's features config

**test** (*test_path*)
Wrapper method to compute test metrics on holdout blind test dataset.

**Parameters** **test_path** (*str*) – test data file path

**train** (*train_config*, *rank=0*, *world_size=1*, *training_state=None*)
Wrapper method to train the model using `Trainer` object.

**Parameters**

- **train_config** (*PyTextConfig*) – config for training
- **rank** (*int*) – for distributed training only, rank of the gpu, default is 0
- **world_size** (*int*) – for distributed training only, total gpu to use, default is 1

pytext.task.**save** (*config: pytext.config.pytext_config.PyTextConfig, model: pytext.models.model.Model, meta: Optional[pytext.data.data_handler.CommonMetadata], tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer], training_state: Optional[pytext.trainers.training_state.TrainingState] = None, identifier: Optional[str] = None*) → str
Save all stateful information of a training task to a specified file-like object, will save the original config, model state, metadata, training state if training is not completed Args: identifier (str): used to identify a checkpoint within a training job, used as a suffix for save path config (PytextConfig): contains all raw

parameter/hyper-parameters for training task model (Model): actual model in training training_state (TrainingState): stateful infomation during training Returns: identifier (str): if identifier is not specified, will save to config.save_snapshot_path to be consistent to post-training snapshot; if specified, will be used to save checkpoint during training, identifier is used to identify checkpoints in the same training

`pytext.task.load`(*load_path: str*, *overwrite_config=None*, *rank=0*, *world_size=1*)

Load task, config and training state from a saved snapshot by default, it will construct the task using the saved config then load metadata and model state.

if overwrite_task is specified, it will construct the task using overwrite_task then load metadata and model state.

`pytext.task.create_task`(*task_config*, *metadata=None*, *model_state=None*, *tensorizers=None*, *rank=0*, *world_size=1*)

Create a task by finding task class in registry and invoking the from_config function of the class, see `from_config()` for more details

`pytext.task.get_latest_checkpoint_path`(*dir_path: Optional[str] = None*) → str

Get the latest checkpoint path :param dir_path: the dir to scan for existing checkpoint files. Default: if None, :param the latest checkpoint path saved in momery will be returned:

Returns: checkpoint_path

`pytext.task.quantize_statically`(*model*, *inputs*, *data_loader*, *linear_only=False*, *module_swap=False*)

## pytext.torchscript package

### Subpackages

## pytext.torchscript.seq2seq package

### Submodules

## pytext.torchscript.seq2seq.beam_decode module

**class** `pytext.torchscript.seq2seq.beam_decode.BeamDecode`(*eos_token_id*, *length_penalty*, *nbest*, *beam_size*, *stop_at_eos*)

Bases: `torch.nn.modules.module.Module`

Decodes the output of Beam Search to get the top hypotheses

**forward**(*beam_tokens: torch.Tensor*, *beam_scores: torch.Tensor*, *token_weights: torch.Tensor*, *beam_prev_indices: torch.Tensor*, *num_steps: int*) → List[Tuple[torch.Tensor, float, List[float], torch.Tensor, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

### pytext.torchscript.seq2seq.beam_search module

**class** pytext.torchscript.seq2seq.beam_search.**BeamSearch**(*model_list, tgt_dict_eos, beam_size: int = 2, quantize: bool = False, record_attention: bool = False*)

Bases: torch.nn.modules.module.Module

**forward**(*src_tokens: torch.Tensor, src_lengths: torch.Tensor, num_steps: int, dict_feat: Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]] = None, contextual_token_embedding: Optional[torch.Tensor] = None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

### pytext.torchscript.seq2seq.decoder module

**class** pytext.torchscript.seq2seq.decoder.**DecoderBatchedStepEnsemble**(*models, beam_size, record_attention=False*)

Bases: torch.nn.modules.module.Module

This method should have a common interface such that it can be called after the encoder as well as after the decoder

**beam_search_aggregate_topk**(*log_probs_per_model: List[torch.Tensor], attn_weights_per_model: List[torch.Tensor], prev_scores: torch.Tensor, beam_size: int, record_attention: bool*)

**forward**(*prev_tokens: torch.Tensor, prev_scores: torch.Tensor, timestep: int, decoder_ips: List[Dict[str, torch.Tensor]]*) → Tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor, List[Dict[str, torch.Tensor]]]

Decoder step inputs correspond one-to-one to encoder outputs. HOWEVER: after the first step, encoder outputs (i.e, the first len(self.models) elements of inputs) must be tiled k (beam size) times on the batch dimension (axis 1).

**reset_incremental_states**()

### pytext.torchscript.seq2seq.encoder module

**class** pytext.torchscript.seq2seq.encoder.**EncoderEnsemble**(*models, beam_size*)

Bases: torch.nn.modules.module.Module

This class will call the encoders from all the models in the ensemble. It will process the encoder output to prepare input for each decoder step input

**forward**(*src_tokens: torch.Tensor, src_lengths: torch.Tensor, dict_feat: Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]] = None, contextual_token_embedding: Optional[torch.Tensor] = None*) → List[Dict[str, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**prepare_decoderstep_ip**(*futures: List[Dict[str, torch.Tensor]]*) → List[Dict[str, torch.Tensor]]

### pytext.torchscript.seq2seq.export_model module

**class** pytext.torchscript.seq2seq.export_model.**Seq2SeqJIT**(*src_dict, tgt_dict, sequence_generator, filter_eos_bos, copy_unk_token=False, dictfeat_dict=None*)

Bases: `torch.nn.modules.module.Module`

**forward**(*src_tokens: List[str], dict_feat: Optional[Tuple[List[str], List[float], List[int]]] = None, contextual_token_embedding: Optional[List[float]] = None*) → List[Tuple[List[str], float, List[float]]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**prepare_generator_inputs**(*word_ids: List[int], dict_feat: Optional[Tuple[List[str], List[float], List[int]]] = None, contextual_token_embedding: Optional[List[float]] = None*) → Tuple[torch.Tensor, Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], Optional[torch.Tensor], torch.Tensor]

### pytext.torchscript.seq2seq.scripted_seq2seq_generator module

**class** pytext.torchscript.seq2seq.scripted_seq2seq_generator.**ScriptedSequenceGenerator**(*models, trg_dic, config*)

Bases: *pytext.models.module.Module*

**forward**(*src_tokens: torch.Tensor, dict_feat: Optional[Tuple[torch.Tensor, torch.Tensor, torch.Tensor]], contextual_token_embedding: Optional[torch.Tensor], src_lengths: torch.Tensor*) → List[Tuple[torch.Tensor, float, List[float], torch.Tensor, torch.Tensor]]

Defines the computation performed at every call.

Should be overridden by all subclasses.

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

**classmethod from_config**(*config*, *models*, *trg_dict_eos*)

**generate_hypo**(*tensors: Dict[str, torch.Tensor]*)

### pytext.torchscript.seq2seq.seq2seq_rnn_decoder_utils module

pytext.torchscript.seq2seq.seq2seq_rnn_decoder_utils.**get_src_length**(*models*, *decoder_ip*)

pytext.torchscript.seq2seq.seq2seq_rnn_decoder_utils.**prepare_decoder_ips**(*models*, *decoder_ip*, *model_state_outputs*, *prev_hypos*)

### Module contents

### pytext.torchscript.tensorizer package

### Submodules

### pytext.torchscript.tensorizer.bert module

**class** pytext.torchscript.tensorizer.bert.**ScriptBERTTensorizer**(*tokenizer: torch.jit._script.ScriptModule*, *vocab: pytext.torchscript.vocab.ScriptVocabulary*, *max_seq_len: int*)

Bases: *pytext.torchscript.tensorizer.bert.ScriptBERTTensorizerBase*

**class** pytext.torchscript.tensorizer.bert.**ScriptBERTTensorizerBase**(*tokenizer: torch.jit._script.ScriptModule*, *vocab: pytext.torchscript.vocab.ScriptVocabul*
*max_seq_len: int*)

Bases: *pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer*

### pytext.torchscript.tensorizer.normalizer module

**class** pytext.torchscript.tensorizer.normalizer.**VectorNormalizer**(*dim: int*, *do_normalization: bool = True*)

Bases: torch.nn.modules.module.Module

Performs in-place normalization over all features of a dense feature vector by doing (x - mean)/stddev for each x in the feature vector.

This is a ScriptModule so that the normalize function can be called at training time in the tensorizer, as well as at inference time by using it in your torchscript forward function. To use this in your tensorizer update_meta_data must be called once per row in your initialize function, and then calculate_feature_stats must be called upon the last time it runs. See usage in FloatListTensorizer for an example.

Setting do_normalization=False will make the normalize function an identity function.

**calculate_feature_stats**()

**forward**()

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**normalize**(*vec: List[List[float]]*)

**update_meta_data**(*vec*)

### pytext.torchscript.tensorizer.roberta module

**class** pytext.torchscript.tensorizer.roberta.**ScriptRoBERTaTensorizer**(*tokenizer: torch.jit._script.ScriptModule, vocab: pytext.torchscript.vocab.ScriptVocab, max_seq_len: int*)

Bases: *pytext.torchscript.tensorizer.bert.ScriptBERTTensorizerBase*

**class** pytext.torchscript.tensorizer.roberta.**ScriptRoBERTaTensorizerWithIndices**(*tokenizer: torch.jit._script.S vocab: pytext.torchscript.v max_seq_len: int*)

Bases: *pytext.torchscript.tensorizer.bert.ScriptBERTTensorizerBase*

### pytext.torchscript.tensorizer.tensorizer module

**class** pytext.torchscript.tensorizer.tensorizer.**ScriptFloat1DListTensorizer**

Bases: `torch.jit._script.ScriptModule`

TorchScript implementation of Float1DListTensorizer in pytext/data/tensorizers.py

**torchscriptify**()

**class** pytext.torchscript.tensorizer.tensorizer.**ScriptFloatListSeqTensorizer**(*pad_token*)

Bases: `torch.jit._script.ScriptModule`

TorchScript implementation of ScriptFloatListSeqTensorizer in pytext/data/tensorizers.py

**torchscriptify**()

**class** pytext.torchscript.tensorizer.tensorizer.**ScriptInteger1DListTensorizer**
Bases: torch.jit._script.ScriptModule

TorchScript implementation of Integer1DListTensorizer in pytext/data/tensorizers.py

**torchscriptify**()

**class** pytext.torchscript.tensorizer.tensorizer.**ScriptTensorizer**
Bases: torch.jit._script.ScriptModule

**set_padding_control**(*dimension: str, padding_control: Optional[List[int]]*)
This functions will be called to set a padding style. None - No padding List: first element 0, round seq length to the smallest list element larger than inputs

**class** pytext.torchscript.tensorizer.tensorizer.**VocabLookup**(*vocab: pytext.torchscript.vocab.ScriptVocabulary*)
Bases: torch.jit._script.ScriptModule

TorchScript implementation of lookup_tokens() in pytext/data/tensorizers.py

### pytext.torchscript.tensorizer.xlm module

**class** pytext.torchscript.tensorizer.xlm.**ScriptXLMTensorizer**(*tokenizer: torch.jit._script.ScriptModule, token_vocab: pytext.torchscript.vocab.ScriptVocabulary, language_vocab: pytext.torchscript.vocab.ScriptVocabulary, max_seq_len: int, default_language: str*)
Bases: *pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer*

### Module contents

**class** pytext.torchscript.tensorizer.**ScriptBERTTensorizer**(*tokenizer: torch.jit._script.ScriptModule, vocab: pytext.torchscript.vocab.ScriptVocabulary, max_seq_len: int*)
Bases: *pytext.torchscript.tensorizer.bert.ScriptBERTTensorizerBase*

**class** pytext.torchscript.tensorizer.**ScriptFloat1DListTensorizer**
Bases: torch.jit._script.ScriptModule

TorchScript implementation of Float1DListTensorizer in pytext/data/tensorizers.py

**torchscriptify**()

**class** pytext.torchscript.tensorizer.**ScriptFloatListSeqTensorizer**(*pad_token*)
Bases: torch.jit._script.ScriptModule

TorchScript implementation of ScriptFloatListSeqTensorizer in pytext/data/tensorizers.py

**torchscriptify**()

**class** pytext.torchscript.tensorizer.**ScriptInteger1DListTensorizer**

Bases: torch.jit._script.ScriptModule

TorchScript implementation of Integer1DListTensorizer in pytext/data/tensorizers.py

**torchscriptify**()

**class** pytext.torchscript.tensorizer.**ScriptRoBERTaTensorizer**(*tokenizer: torch.jit._script.ScriptModule, vocab: pytext.torchscript.vocab.ScriptVocabulary, max_seq_len: int*)

Bases: *pytext.torchscript.tensorizer.bert.ScriptBERTTensorizerBase*

**class** pytext.torchscript.tensorizer.**ScriptRoBERTaTensorizerWithIndices**(*tokenizer: torch.jit._script.ScriptModule vocab: pytext.torchscript.vocab.ScriptV max_seq_len: int*)

Bases: *pytext.torchscript.tensorizer.bert.ScriptBERTTensorizerBase*

**class** pytext.torchscript.tensorizer.**ScriptXLMTensorizer**(*tokenizer: torch.jit._script.ScriptModule, token_vocab: pytext.torchscript.vocab.ScriptVocabulary, language_vocab: pytext.torchscript.vocab.ScriptVocabulary, max_seq_len: int, default_language: str*)

Bases: *pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer*

**class** pytext.torchscript.tensorizer.**VectorNormalizer**(*dim: int, do_normalization: bool = True*)

Bases: torch.nn.modules.module.Module

Performs in-place normalization over all features of a dense feature vector by doing (x - mean)/stddev for each x in the feature vector.

This is a ScriptModule so that the normalize function can be called at training time in the tensorizer, as well as at inference time by using it in your torchscript forward function. To use this in your tensorizer update_meta_data must be called once per row in your initialize function, and then calculate_feature_stats must be called upon the last time it runs. See usage in FloatListTensorizer for an example.

Setting do_normalization=False will make the normalize function an identity function.

**calculate_feature_stats**()

**forward**()

Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

**normalize**(*vec: List[List[float]]*)

**update_meta_data**(*vec*)

**class** pytext.torchscript.tensorizer.**ScriptTensorizer**

Bases: torch.jit._script.ScriptModule

**set_padding_control**(*dimension: str, padding_control: Optional[List[int]]*)

This functions will be called to set a padding style. None - No padding List: first element 0, round seq length to the smallest list element larger than inputs

**class** pytext.torchscript.tensorizer.**VocabLookup**(*vocab: pytext.torchscript.vocab.ScriptVocabulary*)

Bases: torch.jit._script.ScriptModule

TorchScript implementation of lookup_tokens() in pytext/data/tensorizers.py

## pytext.torchscript.tokenizer package

### Submodules

### pytext.torchscript.tokenizer.bpe module

**class** pytext.torchscript.tokenizer.bpe.**ScriptBPE**(*vocab: Dict[str, int], eow: str = '_EOW'*)

Bases: torch.jit._script.ScriptModule

Byte-pair encoding implementation in TorchScript.

vocab_file should be a file-like object separated by newlines, where each line consists of a word and a count separated by whitespace. Words in the vocab therefore can't contain space (according to python regex s). The vocab file should be sorted according to the importance of each token, and they will be merged in this priority; the actual score values are irrelevant.

eow_token should be a string that is appended to the last character and token, and that token is used at each step in the process and returned at the end. You should set this to be consistent with the EOW signature used however you generated your ScriptBPE vocab file.

```
>>> import io
>>> vocab_file = io.StringIO('''
hello_EOW 20
world_EOW 18
th  17
is_EOW 16
bpe_EOW 15
! 14
h 13
t 6
s_EOW 2
i -1
ii -2
''')
>>> bpe = ScriptBPE.from_vocab_file(vocab_file)
>>> bpe.tokenize(["hello", "world", "this", "is", "bpe"])
["hello_EOW", "world_EOW", "th", "is_EOW", "is_EOW", "bpe_EOW"]
>>> bpe.tokenize(["iiiis"])
["ii", "i", "is_EOW"]
```

**classmethod from_vocab_file**(*vocab_file: io.IOBase*) → pytext.torchscript.tokenizer.bpe.ScriptBPE

**classmethod from_vocab_filename**(*vocab_filename: str*) → pytext.torchscript.tokenizer.bpe.ScriptBPE

**static load_vocab**(*file: io.IOBase*) → Dict[str, int]

### pytext.torchscript.tokenizer.tokenizer module

**class** pytext.torchscript.tokenizer.tokenizer.**ScriptBPETokenizer**(*bpe: pytext.torchscript.tokenizer.bpe.ScriptBPE*)

Bases: *pytext.torchscript.tokenizer.tokenizer.ScriptTokenizerBase*

**class** pytext.torchscript.tokenizer.tokenizer.**ScriptDoNothingTokenizer**

Bases: *pytext.torchscript.tokenizer.tokenizer.ScriptTokenizerBase*

**class** pytext.torchscript.tokenizer.tokenizer.**ScriptTokenizerBase**

Bases: torch.jit._script.ScriptModule

**class** pytext.torchscript.tokenizer.tokenizer.**ScriptWordTokenizer**(*lowercase=True*)

Bases: *pytext.torchscript.tokenizer.tokenizer.ScriptTokenizerBase*

### Module contents

**class** pytext.torchscript.tokenizer.**ScriptBPE**(*vocab: Dict[str, int], eow: str = '_EOW'*)

Bases: torch.jit._script.ScriptModule

Byte-pair encoding implementation in TorchScript.

vocab_file should be a file-like object separated by newlines, where each line consists of a word and a count separated by whitespace. Words in the vocab therefore can't contain space (according to python regex s). The vocab file should be sorted according to the importance of each token, and they will be merged in this priority; the actual score values are irrelevant.

eow_token should be a string that is appended to the last character and token, and that token is used at each step in the process and returned at the end. You should set this to be consistent with the EOW signature used however you generated your ScriptBPE vocab file.

```
>>> import io
>>> vocab_file = io.StringIO('''
hello_EOW 20
world_EOW 18
th  17
is_EOW 16
bpe_EOW 15
! 14
h 13
t 6
s_EOW 2
i -1
ii -2
''')
>>> bpe = ScriptBPE.from_vocab_file(vocab_file)
>>> bpe.tokenize(["hello", "world", "this", "is", "bpe"])
["hello_EOW", "world_EOW", "th", "is_EOW", "is_EOW", "bpe_EOW"]
>>> bpe.tokenize(["iiiis"])
["ii", "i", "is_EOW"]
```

**classmethod from_vocab_file**(*vocab_file: io.IOBase*) → pytext.torchscript.tokenizer.bpe.ScriptBPE

**classmethod from_vocab_filename**(*vocab_filename: str*) → pytext.torchscript.tokenizer.bpe.ScriptBPE

**static load_vocab**(*file: io.IOBase*) → Dict[str, int]

**class** pytext.torchscript.tokenizer.**ScriptBPETokenizer**(*bpe: pytext.torchscript.tokenizer.bpe.ScriptBPE*)

Bases: *pytext.torchscript.tokenizer.tokenizer.ScriptTokenizerBase*

**class** pytext.torchscript.tokenizer.**ScriptDoNothingTokenizer**

Bases: *pytext.torchscript.tokenizer.tokenizer.ScriptTokenizerBase*

**class** pytext.torchscript.tokenizer.**ScriptTokenizerBase**

Bases: torch.jit._script.ScriptModule

**class** pytext.torchscript.tokenizer.**ScriptWordTokenizer**(*lowercase=True*)

Bases: *pytext.torchscript.tokenizer.tokenizer.ScriptTokenizerBase*

## Submodules

### pytext.torchscript.batchutils module

pytext.torchscript.batchutils.**clip_list**(*input: Optional[List[str]], max_batch: int*) → Optional[List[str]]

pytext.torchscript.batchutils.**clip_listlist**(*input: Optional[List[List[str]]], max_batch: int*) → Optional[List[List[str]]]

pytext.torchscript.batchutils.**clip_listlist_float**(*input: Optional[List[List[float]]], max_batch: int*) → Optional[List[List[float]]]

pytext.torchscript.batchutils.**destructure_any_list**(*client_batch: List[int], result_any_list: List[Any]*)

pytext.torchscript.batchutils.**destructure_dict_list**(*client_batch: List[int], result_input_list: List[Dict[str, float]]*) → List[List[Dict[str, float]]]

pytext.torchscript.batchutils.**destructure_dictlist_list**(*client_batch: List[int], result_input_list: List[List[Dict[str, float]]]*) → List[List[List[Dict[str, float]]]]

pytext.torchscript.batchutils.**destructure_tensor**(*client_batch: List[int], result_tensor: torch.Tensor*) → List[torch.Tensor]

pytext.torchscript.batchutils.**destructure_tensor_list**(*client_batch: List[int], result_tensor_list: List[torch.Tensor]*) → List[List[torch.Tensor]]

pytext.torchscript.batchutils.**input_size**(*texts: Optional[List[str]] = None, multi_texts: Optional[List[List[str]]] = None, tokens: Optional[List[List[str]]] = None*) → int

pytext.torchscript.batchutils.**limit_list**(*input: Optional[List[str]], max_batch: int*) → Optional[List[str]]

pytext.torchscript.batchutils.**limit_listlist**(*input: Optional[List[List[str]]], max_batch: int*) → Optional[List[List[str]]]

pytext.torchscript.batchutils.**limit_listlist_float**(*input: Optional[List[List[float]]], max_batch: int*) → Optional[List[List[float]]]

pytext.torchscript.batchutils.**listify**(*t: torch.Tensor*) → List[torch.Tensor]

pytext.torchscript.batchutils.**make_batch_texts**(*tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer, mega_batch: List[Tuple[List[str], int]], goals: Dict[str, str]*) → List[List[Tuple[List[str], int]]]

pytext.torchscript.batchutils.**make_batch_texts_dense**(*tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer, mega_batch: List[Tuple[List[str], List[List[float]], int]], goals: Dict[str, str]*) → List[List[Tuple[List[str], List[List[float]], int]]]

pytext.torchscript.batchutils.**make_prediction_texts**(*batch: List[Tuple[List[str]]]*) → List[str]

pytext.torchscript.batchutils.**make_prediction_texts_dense**(*batch: List[Tuple[List[str], List[List[float]]]]*) → Tuple[List[str], List[List[float]]]

pytext.torchscript.batchutils.**make_prediction_tokens**(*batch: List[Tuple[List[List[str]]]]*) → List[List[str]]

pytext.torchscript.batchutils.**max_tokens**(*per_sentence_tokens: List[List[Tuple[str, int, int]]]*) → int

receive the tokenize output for a batch per_sentence_tokens, return the max token length of any sentence

pytext.torchscript.batchutils.**nonify_listlist_float**(*input: List[List[float]]*) → Optional[List[List[float]]]

pytext.torchscript.batchutils.**validate_batch_element**(*e: Tuple[Optional[List[str]], Optional[List[List[str]]], Optional[List[List[str]]], Optional[List[str]], Optional[List[List[float]]]]*)

pytext.torchscript.batchutils.**validate_dense_feat**(*batch_element_dense_feat: Optional[List[List[float]]], length: int, uses_dense_feat: bool*) → List[List[float]]

pytext.torchscript.batchutils.**validate_make_prediction_batch_element**(*be: Tuple[Optional[List[str]], Optional[List[List[str]]], Optional[List[List[str]]], Optional[List[str]], Optional[List[List[float]]]]*)

pytext.torchscript.batchutils.**zip_batch_any_list_list**(*zip_batch_list: List[int], result_list_1: List[List[Any]], result_list_2: List[List[Any]]*) → List[List[Any]]

pytext.torchscript.batchutils.**zip_batch_tensor_list**(*zip_batch_list: List[int], result_list_1: List[torch.Tensor], result_list_2: List[torch.Tensor]*) → List[torch.Tensor]

### pytext.torchscript.module module

**class** pytext.torchscript.module.**PyTextEmbeddingModule**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer*)

Bases: `torch.jit._script.ScriptModule`

**class** pytext.torchscript.module.**PyTextEmbeddingModuleIndex**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTens index: int = 0*)

Bases: `pytext.torchscript.module.PyTextEmbeddingModule`

**class** pytext.torchscript.module.**PyTextEmbeddingModuleWithDense**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.Scrip normalizer: pytext.torchscript.tensorizer.normalizer.Vec concat_dense: bool = False*)

Bases: `pytext.torchscript.module.PyTextEmbeddingModule`

**class** pytext.torchscript.module.**PyTextEmbeddingModuleWithDenseIndex**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensoriz normalizer: pytext.torchscript.tensorizer.normali index: int = 0, concat_dense: bool = True*)

Bases: *pytext.torchscript.module.PyTextEmbeddingModuleWithDense*

**class** pytext.torchscript.module.**PyTextLayerModule**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer*)

Bases: *pytext.torchscript.module.PyTextEmbeddingModule*

**class** pytext.torchscript.module.**PyTextLayerModuleWithDense**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTens normalizer: pytext.torchscript.tensorizer.normalizer.VectorNo*

Bases: *pytext.torchscript.module.PyTextEmbeddingModuleWithDense*

**class** pytext.torchscript.module.**PyTextTwoTowerEmbeddingModule**(*model: torch.jit._script.ScriptModule, right_tensorizer: pytext.torchscript.tensorizer.tensorizer.Script left_tensorizer: pytext.torchscript.tensorizer.tensorizer.Script*

Bases: torch.jit._script.ScriptModule

**class** pytext.torchscript.module.**PyTextTwoTowerEmbeddingModuleWithDense**(*model: torch.jit._script.ScriptModule, right_tensorizer: pytext.torchscript.tensorizer.ten, left_tensorizer: pytext.torchscript.tensorizer.ten, right_normalizer: pytext.torchscript.tensorizer.nor, left_normalizer: pytext.torchscript.tensorizer.nor*)

Bases: *pytext.torchscript.module.PyTextTwoTowerEmbeddingModule*

**class** pytext.torchscript.module.**PyTextTwoTowerLayerModule**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, right_tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTenso, left_tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTenso*)

Bases: *pytext.torchscript.module.PyTextTwoTowerEmbeddingModule*

**class** pytext.torchscript.module.**PyTextTwoTowerLayerModuleWithDense**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, right_tensorizer: pytext.torchscript.tensorizer.tensorize, left_tensorizer: pytext.torchscript.tensorizer.tensorize, right_normalizer: pytext.torchscript.tensorizer.normaliz, left_normalizer: pytext.torchscript.tensorizer.normaliz*)

Bases: *pytext.torchscript.module.PyTextTwoTowerLayerModule*

**class** pytext.torchscript.module.**PyTextVariableSizeEmbeddingModule**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.*)

Bases: *pytext.torchscript.module.PyTextEmbeddingModule*

Assumes model returns a tuple of representations and sequence lengths, then slices each example's representation according to length. Returns a list of tensors. The slicing is easier to do outside a traced model.

**class** pytext.torchscript.module.**ScriptPyTextEmbeddingModule**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTen*

Bases: torch.jit._script.ScriptModule

**validate**(*export_conf: pytext.config.pytext_config.ExportConfig*)

**class** pytext.torchscript.module.**ScriptPyTextEmbeddingModuleIndex**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.Sc index: int = 0*)

Bases: *pytext.torchscript.module.ScriptPyTextEmbeddingModule*

**class** pytext.torchscript.module.**ScriptPyTextEmbeddingModuleWithDense**(*model: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensor normalizer: pytext.torchscript.tensorizer.norma concat_dense: bool = False*)

Bases: *pytext.torchscript.module.ScriptPyTextEmbeddingModule*

**class** pytext.torchscript.module.**ScriptPyTextEmbeddingModuleWithDenseIndex**(*model: torch.jit._script.ScriptMo tensorizer: pytext.torchscript.tensorize normalizer: pytext.torchscript.tensorize index: int = 0, concat_dense: bool = True*)

Bases: *pytext.torchscript.module.ScriptPyTextEmbeddingModuleWithDense*

**class** pytext.torchscript.module.**ScriptPyTextModule**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer*)

Bases: *pytext.torchscript.module.ScriptPyTextEmbeddingModule*

**class** pytext.torchscript.module.**ScriptPyTextModuleWithDense**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTen normalizer: pytext.torchscript.tensorizer.normalizer.VectorN*

Bases: *pytext.torchscript.module.ScriptPyTextEmbeddingModuleWithDense*

**class** pytext.torchscript.module.**ScriptPyTextTwoTowerEmbeddingModule**(*model: torch.jit._script.ScriptModule, right_tensorizer: pytext.torchscript.tensorizer.tensoriz left_tensorizer: pytext.torchscript.tensorizer.tensoriz*

Bases: *pytext.torchscript.module.ScriptTwoTowerModule*

**class** pytext.torchscript.module.**ScriptPyTextTwoTowerEmbeddingModuleWithDense**(*model: torch.jit._script.Scri right_tensorizer: pytext.torchscript.tens left_tensorizer: pytext.torchscript.tens right_normalizer: pytext.torchscript.tens left_normalizer: pytext.torchscript.tens*

Bases: *pytext.torchscript.module.ScriptPyTextTwoTowerEmbeddingModule*

**class** pytext.torchscript.module.**ScriptPyTextTwoTowerModule**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, right_tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTens left_tensorizer: pytext.torchscript.tensorizer.tensorizer.ScriptTens*

Bases: *pytext.torchscript.module.ScriptTwoTowerModule*

**class** pytext.torchscript.module.**ScriptPyTextTwoTowerModuleWithDense**(*model: torch.jit._script.ScriptModule, output_layer: torch.jit._script.ScriptModule, right_tensorizer: pytext.torchscript.tensorizer.tensoriz left_tensorizer: pytext.torchscript.tensorizer.tensoriz right_normalizer: pytext.torchscript.tensorizer.normali left_normalizer: pytext.torchscript.tensorizer.normali*)

Bases: *pytext.torchscript.module.ScriptPyTextTwoTowerModule*

**class** pytext.torchscript.module.**ScriptPyTextVariableSizeEmbeddingModule**(*model: torch.jit._script.ScriptModu tensorizer: pytext.torchscript.tensorizer.te*)

Bases: *pytext.torchscript.module.ScriptPyTextEmbeddingModule*

Assumes model returns a tuple of representations and sequence lengths, then slices each example's representation according to length. Returns a list of tensors. The slicing is easier to do outside a traced model.

**class** pytext.torchscript.module.**ScriptTwoTowerModule**

Bases: torch.jit._script.ScriptModule

**validate**(*export_conf: pytext.config.pytext_config.ExportConfig*)

pytext.torchscript.module.**deprecation_warning**(*export_conf: pytext.config.pytext_config.ExportConfig*)

### pytext.torchscript.utils module

**class** pytext.torchscript.utils.**ScriptBatchInput**

Bases: tuple

A batch of inputs for TorchScript Module(bundle of Tensorizer and Model) texts or tokens is required but multually exclusive :param texts: a batch of raw text inputs :param tokens: a batch of pre-tokenized inputs :param languages: language for each input in the batch

**languages**

Alias for field number 2

**texts**

Alias for field number 0

**tokens**

Alias for field number 1

pytext.torchscript.utils.**float_tensor_list1D**(*input_tensor: torch.Tensor*) → List[float]

### pytext.torchscript.vocab module

**class** pytext.torchscript.vocab.**ScriptVocabulary**(*vocab_list, unk_idx: int = 0, pad_idx: int = -1, bos_idx: int = -1, eos_idx: int = -1, mask_idx: int = -1, unk_token: Optional[str] = None*)

Bases: torch.jit._script.ScriptModule

**get_pad_index**()

**get_unk_index**()

### Module contents

### pytext.trainers package

### Submodules

### pytext.trainers.ensemble_trainer module

**class** pytext.trainers.ensemble_trainer.**EnsembleTrainer**(*real_trainers*)

Bases: *pytext.trainers.trainer.TrainerBase*

Trainer for ensemble models

**real_trainer**

the actual trainer to run

**Type** Trainer

**classmethod from_config**(*config: pytext.trainers.ensemble_trainer.EnsembleTrainer.Config, model: torch.nn.modules.module.Module, *args, **kwargs*)

**train**(*train_iter, eval_iter, model, *args, **kwargs*)

### pytext.trainers.hogwild_trainer module

**class** pytext.trainers.hogwild_trainer.**HogwildTrainer**(*real_trainer_config, num_workers, model: torch.nn.modules.module.Module, *args, **kwargs*)

Bases: *pytext.trainers.trainer.Trainer*

**classmethod from_config**(*config: pytext.trainers.hogwild_trainer.HogwildTrainer.Config, model: torch.nn.modules.module.Module, *args, **kwargs*)

**run_epoch**(*state: pytext.trainers.training_state.TrainingState, data_iter: torchtext.legacy.data.iterator.Iterator, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**set_up_training**(*state: pytext.trainers.training_state.TrainingState, training_data*)

**class** pytext.trainers.hogwild_trainer.**HogwildTrainer_Deprecated**(*real_trainer_config, num_workers, model: torch.nn.modules.module.Module, *args, **kwargs*)

Bases: `pytext.trainers.trainer.Trainer`

**classmethod from_config**(*config: pytext.trainers.hogwild_trainer.HogwildTrainer_Deprecated.Config, model: torch.nn.modules.module.Module, *args, **kwargs*)

**run_epoch**(*state: pytext.trainers.training_state.TrainingState, data_iter: torchtext.legacy.data.iterator.Iterator, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**set_up_training**(*state: pytext.trainers.training_state.TrainingState, training_data*)

### pytext.trainers.trainer module

**class** pytext.trainers.trainer.**TaskTrainer**(*config: pytext.trainers.trainer.Trainer.Config, model: torch.nn.modules.module.Module*)

Bases: `pytext.trainers.trainer.Trainer`

**run_step**(*samples: List[Any], state: pytext.trainers.training_state.TrainingState, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, report_metric: bool*)

Our run_step is a bit different, because we're wrapping the model forward call with model.train_batch, which arranges tensors and gets loss, etc.

Whenever "samples" contains more than one mini-batch (sample_size > 1), we want to accumulate gradients locally and only call all-reduce in the last backwards pass.

**class** pytext.trainers.trainer.**Trainer**(*config: pytext.trainers.trainer.Trainer.Config, model: torch.nn.modules.module.Module*)

Bases: `pytext.trainers.trainer.TrainerBase`

**Base Trainer class that provide ways to** 1 Train model, compute metrics against eval set and use the metrics for model selection. 2 Test trained model, compute and publish metrics against a blind test set.

**epochs**

Training epochs

**Type** int

**early_stop_after**

Stop after how many epochs when the eval metric is not improving

**Type** int

**max_clip_norm**

Clip gradient norm if set

**Type** Optional[float]

**report_train_metrics**

Whether metrics on training data should be computed and reported.

**Type** bool

**target_time_limit_seconds**

Target time limit for training in seconds. If the expected time to train another epoch exceeds this limit, stop training.

**Type** float

**backprop**(*state, loss*)

**continue_training**(*state: pytext.trainers.training_state.TrainingState*) → bool

**classmethod from_config**(*config: pytext.trainers.trainer.Trainer.Config, model: torch.nn.modules.module.Module, *args, **kwargs*)

**load_best_model**(*state: pytext.trainers.training_state.TrainingState*)

**move_state_dict_to_cpu**(*state_dict*)

**move_state_dict_to_gpu**(*state_dict*)

**optimizer_step**(*state*)

**run_epoch**(*state: pytext.trainers.training_state.TrainingState, data: pytext.data.data_handler.BatchIterator, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**run_step**(*samples: List[Any], state: pytext.trainers.training_state.TrainingState, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, report_metric: bool*)

**save_checkpoint**(*state: pytext.trainers.training_state.TrainingState, train_config: pytext.config.pytext_config.PyTextConfig*) → str

**set_up_training**(*state: pytext.trainers.training_state.TrainingState, training_data: pytext.data.data_handler.BatchIterator*)

**sparsification_step**(*state*)

**test**(*test_iter, model, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**train**(*training_data: pytext.data.data_handler.BatchIterator, eval_data: pytext.data.data_handler.BatchIterator, model: pytext.models.model.Model, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, train_config: pytext.config.pytext_config.PyTextConfig, rank: int = 0*) → Tuple[torch.nn.modules.module.Module, Any]

Train and eval a model, the model states will be modified. :param train_iter: batch iterator of training data :type train_iter: BatchIterator :param eval_iter: batch iterator of evaluation data :type eval_iter: BatchIterator :param model: model to be trained :type model: Model :param metric_reporter: compute metric based on training :type metric_reporter: MetricReporter :param output and report results to console, file.. etc: :param train_config: training config :type train_config: PyTextConfig :param training_result: only meaningful for Hogwild training. default :type training_result: Optional :param is None: :param rank: only used in distributed training, the rank of the current :type rank: int :param training thread, evaluation will only be done in rank 0:

**Returns** the trained model together with the best metric

**Return type** model, best_metric

**train_from_state**(*state: pytext.trainers.training_state.TrainingState, training_data: pytext.data.data_handler.BatchIterator, eval_data: pytext.data.data_handler.BatchIterator, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, train_config: pytext.config.pytext_config.PyTextConfig*) → Tuple[torch.nn.modules.module.Module, Any]

Train and eval a model from a given training state will be modified. This function iterates epochs specified in config, and for each epoch do:

1. Train model using training data, aggregate and report training results
2. Adjust learning rate if scheduler is specified
3. Evaluate model using evaluation data
4. Calculate metrics based on evaluation results and select best model

**Parameters**

- **training_state**(*TrainingState*) – contrains stateful information to be
- **to restore a training job**(*able*) –

- **train_iter** (*BatchIterator*) – batch iterator of training data
- **eval_iter** (*BatchIterator*) – batch iterator of evaluation data
- **model** (*Model*) – model to be trained
- **metric_reporter** (*MetricReporter*) – compute metric based on training output and report results to console, file.. etc
- **train_config** (*PyTextConfig*) – training config

**Returns** the trained model together with the best metric

**Return type** model, best_metric

**update_best_model**(*state: pytext.trainers.training_state.TrainingState*, *train_config: pytext.config.pytext_config.PyTextConfig*, *eval_metric*)

**zero_grads**(*state*)

**class** pytext.trainers.trainer.**TrainerBase**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.config.component.Component*

pytext.trainers.trainer.**cycle**(*iterator: Iterable[Any]*) → Iterable[Any]

Like itertools.cycle, but will call iter on the original iterable instead. This limits it to not be able to run on say raw generators, but also doesn't store a copy of the iterable in memory for repetition.

pytext.trainers.trainer.**maybe_accumulate_gradients**(*exit_stack*, *model*, *index*, *sample_size*)

### pytext.trainers.training_state module

**class** pytext.trainers.training_state.**TrainingState**(***kwargs*)

Bases: object

**best_model_metric = None**

**best_model_state = None**

**epoch = 0**

**epochs_since_last_improvement = 0**

**rank = 0**

**stage = 'Training'**

**step_counter = 0**

**tensorizers = None**

### Module contents

**class** pytext.trainers.**Trainer**(*config: pytext.trainers.trainer.Trainer.Config*, *model: torch.nn.modules.module.Module*)

Bases: *pytext.trainers.trainer.TrainerBase*

**Base Trainer class that provide ways to** 1 Train model, compute metrics against eval set and use the metrics for model selection. 2 Test trained model, compute and publish metrics against a blind test set.

**epochs**

Training epochs

**Type** int

**early_stop_after**
Stop after how many epochs when the eval metric is not improving

**Type** int

**max_clip_norm**
Clip gradient norm if set

**Type** Optional[float]

**report_train_metrics**
Whether metrics on training data should be computed and reported.

**Type** bool

**target_time_limit_seconds**
Target time limit for training in seconds. If the expected time to train another epoch exceeds this limit, stop training.

**Type** float

**backprop**(*state*, *loss*)

**continue_training**(*state: pytext.trainers.training_state.TrainingState*) → bool

**classmethod from_config**(*config: pytext.trainers.trainer.Trainer.Config*, *model: torch.nn.modules.module.Module*, **args*, ***kwargs*)

**load_best_model**(*state: pytext.trainers.training_state.TrainingState*)

**move_state_dict_to_cpu**(*state_dict*)

**move_state_dict_to_gpu**(*state_dict*)

**optimizer_step**(*state*)

**run_epoch**(*state: pytext.trainers.training_state.TrainingState*, *data: pytext.data.data_handler.BatchIterator*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**run_step**(*samples: List[Any], state: pytext.trainers.training_state.TrainingState, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, report_metric: bool*)

**save_checkpoint**(*state: pytext.trainers.training_state.TrainingState*, *train_config: pytext.config.pytext_config.PyTextConfig*) → str

**set_up_training**(*state: pytext.trainers.training_state.TrainingState*, *training_data: pytext.data.data_handler.BatchIterator*)

**sparsification_step**(*state*)

**test**(*test_iter*, *model*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**train**(*training_data: pytext.data.data_handler.BatchIterator*, *eval_data: pytext.data.data_handler.BatchIterator*, *model: pytext.models.model.Model*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*, *train_config: pytext.config.pytext_config.PyTextConfig*, *rank: int = 0*) → Tuple[torch.nn.modules.module.Module, Any]

Train and eval a model, the model states will be modified. :param train_iter: batch iterator of training data :type train_iter: BatchIterator :param eval_iter: batch iterator of evaluation data :type eval_iter: BatchIterator :param model: model to be trained :type model: Model :param metric_reporter: compute metric based on training :type metric_reporter: MetricReporter :param output and report results to console, file.. etc: :param train_config: training config :type train_config: PyTextConfig :param training_result: only meaningful for Hogwild training. default :type training_result: Optional :param is None: :param rank:

only used in distributed training, the rank of the current :type rank: int :param training thread, evaluation will only be done in rank 0:

**Returns** the trained model together with the best metric

**Return type** model, best_metric

**train_from_state**(*state: pytext.trainers.training_state.TrainingState, training_data: pytext.data.data_handler.BatchIterator, eval_data: pytext.data.data_handler.BatchIterator, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, train_config: pytext.config.pytext_config.PyTextConfig*) → Tuple[torch.nn.modules.module.Module, Any]

Train and eval a model from a given training state will be modified. This function iterates epochs specified in config, and for each epoch do:

1. Train model using training data, aggregate and report training results
2. Adjust learning rate if scheduler is specified
3. Evaluate model using evaluation data
4. Calculate metrics based on evaluation results and select best model

**Parameters**

- **training_state** (*TrainingState*) – contrains stateful information to be
- **to restore a training job** (*able*) –
- **train_iter** (*BatchIterator*) – batch iterator of training data
- **eval_iter** (*BatchIterator*) – batch iterator of evaluation data
- **model** (*Model*) – model to be trained
- **metric_reporter** (*MetricReporter*) – compute metric based on training output and report results to console, file.. etc
- **train_config** (*PyTextConfig*) – training config

**Returns** the trained model together with the best metric

**Return type** model, best_metric

**update_best_model**(*state: pytext.trainers.training_state.TrainingState, train_config: pytext.config.pytext_config.PyTextConfig, eval_metric*)

**zero_grads**(*state*)

**class** pytext.trainers.**TrainingState**(***kwargs*)

Bases: object

**best_model_metric = None**

**best_model_state = None**

**epoch = 0**

**epochs_since_last_improvement = 0**

**rank = 0**

**stage = 'Training'**

**step_counter = 0**

**tensorizers = None**

**class** pytext.trainers.**EnsembleTrainer**(*real_trainers*)

Bases: *pytext.trainers.trainer.TrainerBase*

Trainer for ensemble models

**real_trainer**

the actual trainer to run

**Type** Trainer

**classmethod from_config**(*config: pytext.trainers.ensemble_trainer.EnsembleTrainer.Config, model: torch.nn.modules.module.Module, *args, **kwargs*)

**train**(*train_iter, eval_iter, model, *args, **kwargs*)

**class** pytext.trainers.**HogwildTrainer**(*real_trainer_config, num_workers, model: torch.nn.modules.module.Module, *args, **kwargs*)

Bases: *pytext.trainers.trainer.Trainer*

**classmethod from_config**(*config: pytext.trainers.hogwild_trainer.HogwildTrainer.Config, model: torch.nn.modules.module.Module, *args, **kwargs*)

**run_epoch**(*state: pytext.trainers.training_state.TrainingState, data_iter: torchtext.legacy.data.iterator.Iterator, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**set_up_training**(*state: pytext.trainers.training_state.TrainingState, training_data*)

**class** pytext.trainers.**HogwildTrainer_Deprecated**(*real_trainer_config, num_workers, model: torch.nn.modules.module.Module, *args, **kwargs*)

Bases: *pytext.trainers.trainer.Trainer*

**classmethod from_config**(*config: pytext.trainers.hogwild_trainer.HogwildTrainer_Deprecated.Config, model: torch.nn.modules.module.Module, *args, **kwargs*)

**run_epoch**(*state: pytext.trainers.training_state.TrainingState, data_iter: torchtext.legacy.data.iterator.Iterator, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

**set_up_training**(*state: pytext.trainers.training_state.TrainingState, training_data*)

**class** pytext.trainers.**TaskTrainer**(*config: pytext.trainers.trainer.Trainer.Config, model: torch.nn.modules.module.Module*)

Bases: *pytext.trainers.trainer.Trainer*

**run_step**(*samples: List[Any], state: pytext.trainers.training_state.TrainingState, metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter, report_metric: bool*)

Our run_step is a bit different, because we're wrapping the model forward call with model.train_batch, which arranges tensors and gets loss, etc.

Whenever "samples" contains more than one mini-batch (sample_size > 1), we want to accumulate gradients locally and only call all-reduce in the last backwards pass.

## pytext.utils package

### Submodules

### pytext.utils.ascii_table module

pytext.utils.ascii_table.**ascii_table**(*data*, *human_column_names=None*, *footer=None*, *indentation=''*, *alignments=()*)

pytext.utils.ascii_table.**ascii_table_from_dict**(*dict*, *key_name*, *value_name*, *indentation=''*)

pytext.utils.ascii_table.**ordered_unique**(*sequence*)

### pytext.utils.config_utils module

**class** pytext.utils.config_utils.**MockConfigLoader**(*config_base_path: str*, *reset_paths: {<class 'str'>: <class 'str'>} = None*, *replace_paths: {<class 'str'>: <class 'str'>} = None*)

Bases: object

**disable_cuda**(*config*)

**fix_paths**(*config*)

**make_config**(*config_filename*, *disable_tensorboard=True*)

**make_config_from_dict**(*config*, *disable_tensorboard*)

### pytext.utils.cuda module

pytext.utils.cuda.**FloatTensor**(**args*)

pytext.utils.cuda.**GetTensor**(*tensor*)

pytext.utils.cuda.**LongTensor**(**args*)

pytext.utils.cuda.**Variable**(*data*, **args*, ***kwargs*)

pytext.utils.cuda.**device**()

pytext.utils.cuda.**tensor**(*data*, *dtype*)

pytext.utils.cuda.**var_to_numpy**(*v*)

pytext.utils.cuda.**zerovar**(**size*)

### pytext.utils.data module

**class** pytext.utils.data.**ResultRow**(*name*, *metrics_dict*)

Bases: object

**class** pytext.utils.data.**ResultTable**(*metrics*, *class_names*, *labels*, *preds*)

Bases: object

**class** pytext.utils.data.**Slot**(*label: str*, *start: int*, *end: int*)

Bases: object

**B_LABEL_PREFIX = 'B-'**

**I_LABEL_PREFIX = 'I-'**

**NO_LABEL_SLOT = 'NoLabel'**

**b_label_name**

**i_label_name**

**token_label**(*use_bio_labels*, *token_start*, *token_end*)

**token_overlap**(*token_start*, *token_end*)

pytext.utils.data.**align_slot_labels**(*token_ranges: List[Tuple[int, int]], slots_field: str, use_bio_labels: bool = False*)

pytext.utils.data.**byte_length**(*text: str*) → int
Return the string length in term of byte offset

pytext.utils.data.**char_offset_to_byte_offset**(*text: str*, *char_offset: int*) → int
Convert a char offset to byte offset

pytext.utils.data.**get_substring_from_offsets**(*text: str, start: Optional[int], end: Optional[int], byte_offset: bool = True*) → str
Access substring of a text using byte offset, if the switch is turned on. Otherwise return substring as the usual text[start:end]

pytext.utils.data.**is_number**(*string*)

pytext.utils.data.**merge_token_labels_by_bio**(*token_ranges*, *labels*)

pytext.utils.data.**merge_token_labels_by_label**(*token_ranges*, *labels*)

pytext.utils.data.**merge_token_labels_to_slot**(*token_ranges*, *labels*, *use_bio_label=True*)

pytext.utils.data.**no_tokenize**(*s: Any*) → Any

pytext.utils.data.**parse_and_align_slot_labels_list**(*token_ranges: List[Tuple[int, int]], slots_field: str, use_bio_labels: bool = False*)

pytext.utils.data.**parse_json_array**(*json_text: str*) → List[str]

pytext.utils.data.**parse_slot_string**(*slots_field: str*) → List[pytext.utils.data.Slot]

pytext.utils.data.**parse_token**(*utterance: str, token_range: List[int]*) → List[Tuple[str, Tuple[int, int]]]

pytext.utils.data.**simple_tokenize**(*s: str*) → List[str]

pytext.utils.data.**strip_bio_prefix**(*label*)

pytext.utils.data.**unkify**(*token: str*)

### pytext.utils.distributed module

pytext.utils.distributed.**dist_init**(*distributed_rank: int*, *world_size: int*, *init_method: str*, *device_id: int*, *backend: str = 'nccl'*, *gpu_streams: int = 1*)
1. After spawn process per GPU, we want all workers to call init_process_group around the same time or times out. 2. After dist_init, we want all workers to start calling all_reduce/barrier around the same time or NCCL timeouts.

pytext.utils.distributed.**force_print**(**args*, ***kwargs*)

pytext.utils.distributed.**get_shard_range**(*dataset_size: int*, *rank: int*, *world_size: int*)
In case dataset_size is not evenly divided by world_size, we need to pad one extra example in each shard shard_len = dataset_size // world_size + 1

Case 1 rank < remainder: each shard start position is rank * shard_len

Case 2 rank >= remainder: without padding, each shard start position is rank * (shard_len - 1) + remainder = rank * shard_len - (rank - remainder) But to make sure all shard have same size, we need to pad one extra example when rank >= remainder, so start_position = start_position - 1

For example, dataset_size = 21, world_size = 8 rank 0 to 4: [0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11], [12, 13, 14] rank 5 to 7: [14, 15, 16], [16, 17, 18], [18, 19, 20]

`pytext.utils.distributed.`**`suppress_output`**`()`

### pytext.utils.documentation module

`pytext.utils.documentation.`**`find_config_class`**`(`*class_name*`)`

Return the set of PyText classes matching that name. Handles fully-qualified *class_name* including module.

`pytext.utils.documentation.`**`get_class_members_recursive`**`(`*obj*`)`

Find all the field names for a given class and their default value.

`pytext.utils.documentation.`**`get_config_fields`**`(`*obj*`)`

Return a dict of config help for this object, where: - key: config name - value: (default, type, options)

- default: default value for this key if not specified
- type: type for this config value, as a string
- options: possible values for this config, only if type = Union

If the type is "Union", the options give the lists of class names that are possible, and the default is one of those class names.

`pytext.utils.documentation.`**`get_subclasses`**`(`*klass, stop_classes=(<class 'pytext.models.module.Module'>, <class 'pytext.config.component.Component'>, <class 'torch.nn.modules.module.Module'>)*`)`

`pytext.utils.documentation.`**`pretty_print_config_class`**`(`*obj*`)`

Pretty-print the fields of one object.

`pytext.utils.documentation.`**`replace_components`**`(`*root*, *component*, *base_class*`)`

Recursively look at all fields in config to find where *component* would fit. This is used to change configs so that they don't use default values. Return the chain of field names, from child to parent.

### pytext.utils.embeddings module

**class** `pytext.utils.embeddings.`**`PretrainedEmbedding`**`(`*embeddings_path: str = None, lowercase_tokens: bool = True, skip_header: bool = True, delimiter: str = ' '*`)`

Bases: `object`

Utility class for loading/caching/initializing word embeddings

**`cache_pretrained_embeddings`**`(`*cache_path: str*`)` → None

Cache the processed embedding vectors and vocab to a file for faster loading

**`filter_criteria`**`(`*token: str*`)` → bool

**`initialize_embeddings_weights`**`(`*str_to_idx: Dict[str, int], unk: str, embed_dim: int, init_strategy: pytext.config.field_config.EmbedInitStrategy*`)` → torch.Tensor

Initialize embeddings weights of shape (len(str_to_idx), embed_dim) from the pretrained embeddings

vectors. Words that are not in the pretrained embeddings list will be initialized according to *init_strategy*. :param str_to_idx: a dict that maps words to indices that the model expects :param unk: unknown token :param embed_dim: the embeddings dimension :param init_strategy: method of initializing new tokens :returns: a float tensor of dimension (vocab_size, embed_dim)

**load_cached_embeddings**(*cache_path: str*) → None
Load cached embeddings from file

**load_pretrained_embeddings**(*raw_embeddings_path: str*, *append: bool = False*, *dialect: str = None*, *lowercase_tokens: bool = True*, *skip_header: bool = True*, *delimiter: str = ' '*) → None
Loading raw embeddings vectors from file in the format: num_words dim word_i v0, v1, v2, . . . ., v_dim word_2 v0, v1, v2, . . . ., v_dim . . . . Optionally appends _dialect to every token in the vocabulary (for XLU embeddings).

**normalize_token**(*token: str*) → str
Apply normalizations to the input token for the embedding space

pytext.utils.embeddings.**append_dialect**(*word: str*, *dialect: str*) → str

### pytext.utils.file_io module

pytext.utils.file_io.**chunk_file**(*file_path*, *chunks*, *work_dir*)
Splits a large file by line into number of chunks and writes them into work_dir

pytext.utils.file_io.**register_http_url_handler**()
support reading file from url starting with "http://", "https://", "ftp://"

### pytext.utils.label module

pytext.utils.label.**get_label_weights**(*vocab_dict: Dict[str, int], label_weights: Dict[str, float]*)

### pytext.utils.lazy module

**class** pytext.utils.lazy.**Infer**(*resolve_fn*)
Bases: object

A value which can be inferred from a forward pass. Infer objects should be passed as arguments or keyword arguments to Lazy objects; see Lazy documentation for more details.

**classmethod dimension**(*dim*)
A helper for creating Infer arguments looking at specific dimensions.

**class** pytext.utils.lazy.**Lazy**(*module_class*, **args*, ***kwargs*)
Bases: torch.nn.modules.module.Module

A module which is able to infer some of its parameters from the inputs to its first forward pass. Lazy wraps any other nn.Module, and arguments can be passed that will be used to construct that wrapped Module after the first forward pass. If any of these arguments are Infer objects, those arguments will be replaced by calling the callback of the Infer object on the forward pass input.

For instance, >>> Lazy(nn.Linear, Infer(lambda input: input.size(-1)), 4) Lazy()

takes its in_features dimension from the last dimension of the input to its forward pass. This can be simplified to

```
>>> Lazy(nn.Linear, Infer.dimension(-1), 4)
```

or a partial can be created, for instance

```
>>> LazyLinear = Lazy.partial(nn.Linear, Infer.dimension(-1))
>>> LazyLinear(4)
Lazy()
```

Finally, these Lazy objects explicitly forbid treating themselves normally; they must instead be replaced by calling *init_lazy_modules* on your model before training. For instance,

```
>>> ll = lazy.Linear(4)
>>> seq = nn.Sequential(ll)
>>> seq
Sequential(
    0: Lazy(),
)
>>> init_lazy_modules(seq, torch.rand(1, 2)
Sequential(
    0: Linear(in_features=2, out_features=4, bias=True)
)
```

**forward**(**args*, ***kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod partial**(*module_class*, **args*, ***kwargs*)

**resolve**()

Must make a call to forward before calling this function; returns the full nn.Module object constructed using inferred arguments/dimensions.

**exception** pytext.utils.lazy.**UninitializedLazyModuleError**

Bases: `Exception`

A lazy module was used improperly.

pytext.utils.lazy.**init_lazy_modules**(*module: torch.nn.modules.module.Module, dummy_input: Tuple[torch.Tensor, ...]*) → torch.nn.modules.module.Module

Finalize an nn.Module which has Lazy components. This will both mutate internal modules which have Lazy elements, and return a new non-lazy nn.Module (in case the top-level module itself is Lazy).

**Parameters**

- **module** – An nn.Module which may be lazy or contain Lazy subcomponents
- **dummy_input** – module is called with this input to ensure that Lazy subcomponents have been able to infer any parameters they need

**Returns** The full nn.Module object constructed using inferred arguments/dimensions.

**class** pytext.utils.lazy.**lazy_property**(*fget*)

Bases: `object`

More or less copy-pasta: http://stackoverflow.com/a/6849299 Meant to be used for lazy evaluation of an object attribute. property should represent non-mutable data, as it replaces itself.

`pytext.utils.lazy.replace_lazy_modules(module)`

### pytext.utils.loss module

**class** `pytext.utils.loss.LagrangeMultiplier`

Bases: `torch.autograd.function.Function`

**static backward**(*ctx*, *grad_output*)

Defines a formula for differentiating the operation.

This function is to be overridden by all subclasses.

It must accept a context `ctx` as the first argument, followed by as many outputs did *forward()* return, and it should return as many tensors, as there were inputs to *forward()*. Each argument is the gradient w.r.t the given output, and each returned value should be the gradient w.r.t. the corresponding input.

The context can be used to retrieve tensors saved during the forward pass. It also has an attribute `ctx.needs_input_grad` as a tuple of booleans representing whether each input needs gradient. E.g., *backward()* will have `ctx.needs_input_grad[0] = True` if the first input to *forward()* needs gradient computated w.r.t. the output.

**static forward**(*ctx*, *input*)

Performs the operation.

This function is to be overridden by all subclasses.

It must accept a context ctx as the first argument, followed by any number of arguments (tensors or other types).

The context can be used to store tensors that can be then retrieved during the backward pass.

`pytext.utils.loss.build_class_priors`(*labels*, *class_priors=None*, *weights=None*, *positive_pseudocount=1.0*, *negative_pseudocount=1.0*)

build class priors, if necessary. For each class, the class priors are estimated as (P + sum_i w_i y_i) / (P + N + sum_i w_i), where y_i is the ith label, w_i is the ith weight, P is a pseudo-count of positive labels, and N is a pseudo-count of negative labels.

**Parameters**

- **labels** – A *Tensor* with shape [batch_size, num_classes]. Entries should be in [0, 1].
- **class_priors** – None, or a floating point *Tensor* of shape [C] containing the prior probability of each class (i.e. the fraction of the training data consisting of positive examples). If None, the class priors are computed from *targets* with a moving average.
- **weights** – *Tensor* of shape broadcastable to labels, [N, 1] or [N, C], where *N* = *batch_size*, C = num_classes‘
- **positive_pseudocount** – Number of positive labels used to initialize the class priors.
- **negative_pseudocount** – Number of negative labels used to initialize the class priors.

**Returns**

**A Tensor of shape [num_classes] consisting of the** weighted class priors, after updating with moving average ops if created.

**Return type** class_priors

`pytext.utils.loss.`**`false_postives_upper_bound`**(*labels*, *logits*, *weights*)

false_positives_upper_bound defined in paper: "Scalable Learning of Non-Decomposable Objectives"

**Parameters**

- **`labels`** – A *Tensor* of shape broadcastable to logits.
- **`logits`** – A *Tensor* of shape [N, C] or [N, C, K]. If the third dimension is present, the lower bound is computed on each slice [:, :, k] independently.
- **`weights`** – Per-example loss coefficients, with shape broadcast-compatible with that of *labels*. i.e. [N, 1] or [N, C]

**Returns** A *Tensor* of shape [C] or [C, K].

`pytext.utils.loss.`**`lagrange_multiplier`**(*x*)

`pytext.utils.loss.`**`range_to_anchors_and_delta`**(*precision_range*, *num_anchors*)

Calculates anchor points from precision range.

**Parameters**

- **`precision_range`** – an interval (a, b), where 0.0 <= a <= b <= 1.0
- **`num_anchors`** – int, number of equally spaced anchor points.

**Returns**

**A *Tensor* of [num_anchors] equally spaced values** in the interval precision_range.

delta: The spacing between the values in precision_values.

**Return type** precision_values

**Raises** `ValueError` – If precision_range is invalid.

`pytext.utils.loss.`**`true_positives_lower_bound`**(*labels*, *logits*, *weights*)

true_positives_lower_bound defined in paper: "Scalable Learning of Non-Decomposable Objectives"

**Parameters**

- **`labels`** – A *Tensor* of shape broadcastable to logits.
- **`logits`** – A *Tensor* of shape [N, C] or [N, C, K]. If the third dimension is present, the lower bound is computed on each slice [:, :, k] independently.
- **`weights`** – Per-example loss coefficients, with shape [N, 1] or [N, C]

**Returns** A *Tensor* of shape [C] or [C, K].

`pytext.utils.loss.`**`weighted_hinge_loss`**(*labels*, *logits*, *positive_weights=1.0*, *negative_weights=1.0*)

**Parameters**

- **`labels`** – one-hot representation *Tensor* of shape broadcastable to logits
- **`logits`** – A *Tensor* of shape [N, C] or [N, C, K]
- **`positive_weights`** – Scalar or Tensor
- **`negative_weights`** – same shape as positive_weights

**Returns** 3D Tensor of shape [N, C, K], where K is length of positive weights or 2D Tensor of shape [N, C]

### pytext.utils.meter module

**class** pytext.utils.meter.**Meter**

Bases: object

**avg**

**reset**()

**update**(*val=1*)

**class** pytext.utils.meter.**TimeMeter**

Bases: *pytext.utils.meter.Meter*

Computes the average occurrence of some event per second

**avg**

**elapsed_time**

**reset**()

**update**(*val=1*)

### pytext.utils.mobile_onnx module

pytext.utils.mobile_onnx.**add_feats_numericalize_ops**(*init_net*, *predict_net*, *vocab_map*, *input_names*)

pytext.utils.mobile_onnx.**create_context**(*init_net*)

pytext.utils.mobile_onnx.**create_vocab_index**(*vocab_list*, *net*, *net_workspace*, *index_name*)

pytext.utils.mobile_onnx.**create_vocab_indices_map**(*init_net*, *vocab_map*)

pytext.utils.mobile_onnx.**get_numericalize_net**(*init_net*, *predict_net*, *vocab_map*, *input_names*)

pytext.utils.mobile_onnx.**pytorch_to_caffe2**(*model*, *export_input*, *external_input_names*, *output_names*, *export_path*, *export_onnx_path=None*)

### pytext.utils.model module

pytext.utils.model.**get_mismatched_param**(*models: Iterable[torch.nn.modules.module.Module], rel_epsilon: Optional[float] = None, abs_epsilon: Optional[float] = None*) → str

Return the name of the first mismatched parameter. Return an empty string if all the parameters of the modules are identical.

pytext.utils.model.**to_onehot**(*feat: pytext.utils.cuda.Variable*, *size: int*) → pytext.utils.cuda.Variable

Transform features into one-hot vectors

### pytext.utils.onnx module

pytext.utils.onnx.**add_feats_numericalize_ops**(*c2_prepared*, *vocab_map*, *input_names*)

pytext.utils.onnx.**convert_caffe2_blob_name**(*blob_name*)

pytext.utils.onnx.**create_vocab_index**(*vocab_list*, *net*, *net_workspace*, *index_name*)

pytext.utils.onnx.**create_vocab_indices_map**(*c2_prepared*, *init_net*, *vocab_map*)

pytext.utils.onnx.**export_nets_to_predictor_file**(*c2_prepared*, *input_names*, *output_names*, *predictor_path*, *extra_params=None*)

pytext.utils.onnx.**get_numericalize_net**(*c2_prepared*, *vocab_map*, *input_names*)

pytext.utils.onnx.**pytorch_to_caffe2**(*model*, *export_input*, *external_input_names*, *output_names*, *export_path*, *export_onnx_path=None*)

pytext.utils.onnx.**validate_onnx_export**(*model*)

### pytext.utils.path module

pytext.utils.path.**get_absolute_path**(*file_path: str*) → str

pytext.utils.path.**get_pytext_home**()

pytext.utils.path.**is_absolute_path**(*file_path: str*) → bool

### pytext.utils.precision module

pytext.utils.precision.**delay_unscale**()

pytext.utils.precision.**maybe_float**(*tensor*)

pytext.utils.precision.**maybe_half**(*tensor*)

pytext.utils.precision.**pad_length**(*n*)

pytext.utils.precision.**set_fp16**(*fp16_enabled: bool*)

### pytext.utils.tensor module

### pytext.utils.test module

pytext.utils.test.**import_tests_module**(*packages_to_scan=None*)

### pytext.utils.timing module

**class** pytext.utils.timing.**HierarchicalTimer**

Bases: object

**pop**()

**push**(*label*, *caller_id*)

**snapshot**()

**time**(*label*)

**class** pytext.utils.timing.**Snapshot**

Bases: object

**report**(*report_pep=False*)

**class** pytext.utils.timing.**SnapshotList**

Bases: list

lists are not weakref-able by default.

**class** pytext.utils.timing.**Timings**(*sum: float = 0.0, count: int = 0, max: float = -inf, times: List[T] = None*)

Bases: object

**add**(*time*)

**average**

**p50**

**p90**

**p99**

pytext.utils.timing.**format_time**(*seconds*)

pytext.utils.timing.**report_snapshot**(*fn*)

### pytext.utils.torch module

### pytext.utils.usage module

pytext.utils.usage.**log_accelerator_feature_usage**(*feature*)

pytext.utils.usage.**log_class_usage**(*klass*)

pytext.utils.usage.**log_feature_usage**(*feature*)

pytext.utils.usage.**log_flow_usage**(*flow_name*)

### Module contents

pytext.utils.**cls_vars**(*cls*)

pytext.utils.**recursive_map**(*seq, func*)

This is similar to the build-in map function but works for nested lists. Useful for transforming tensors serialized with .tolist()

pytext.utils.**round_seq**(*seq, ndigits*)

Rounds a nested sequence of floats to ndigits precision. Useful for rounding tensors serialized with .tolist()

pytext.utils.**set_random_seeds**(*seed, use_deterministic_cudnn*)

## 1.22.2 Submodules

## 1.22.3 pytext.builtin_task module

pytext.builtin_task.**add_include**(*path*)

Import tasks (and associated components) from the folder name.

pytext.builtin_task.**register_builtin_tasks**()

### 1.22.4 pytext.main module

**class** pytext.main.**Attrs**
Bases: object

pytext.main.**gen_config_impl**(*task_name*, **args*, ***kwargs*)

pytext.main.**run_single**(*rank: int, config_json: str, world_size: int, dist_init_method: Optional[str], metadata: Union[Dict[str, pytext.data.data_handler.CommonMetadata], pytext.data.data_handler.CommonMetadata, None], metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]]*)

pytext.main.**train_model_distributed**(*config, metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]]*)

### 1.22.5 pytext.workflow module

**class** pytext.workflow.**LogitsWriter**(*results: multiprocessing.context.BaseContext.Queue, output_path: str, use_gzip: bool, ndigits_precision: int*)
Bases: object

Writes model logits to a file.

**The class is designed for use in an asynchronous process spawned by torch.multiprocessing.spawn, e.g.** logits_writer = LogitsWriter(...) logits_writer_ctx = torch.multiprocessing.spawn(logits_writer.run, join=False) logits_writer_ctx.join()

**run**(*process_index*)

pytext.workflow.**batch_predict**(*model_file: str, examples: List[Dict[str, Any]]*)

pytext.workflow.**dict_zip**(**dicts*, *value_only=False*)

pytext.workflow.**export_saved_model_to_caffe2**(*saved_model_path: str, export_caffe2_path: str, output_onnx_path: str = None*) → None

pytext.workflow.**export_saved_model_to_torchscript**(*saved_model_path: str, path: str, export_config: pytext.config.pytext_config.ExportConfig*) → None

pytext.workflow.**get_logits**(*snapshot_path: str, use_cuda_if_available: bool, output_path: Optional[str] = None, test_path: Optional[str] = None, field_names: Optional[List[str]] = None, dump_raw_input: bool = False, batch_size: int = 16, ndigits_precision: int = 0, output_columns: Optional[List[int]] = None, use_gzip: bool = False, device_id: int = 0, fp16: bool = False*)

pytext.workflow.**prepare_task**(*config: pytext.config.pytext_config.PyTextConfig, dist_init_url: str = None, device_id: int = 0, rank: int = 0, world_size: int = 1, metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]] = None, metadata: pytext.data.data_handler.CommonMetadata = None*) → Tuple[pytext.task.task.Task_Deprecated, pytext.trainers.training_state.TrainingState]

pytext.workflow.**prepare_task_metadata**(*config: pytext.config.pytext_config.PyTextConfig*) → pytext.data.data_handler.CommonMetadata
Loading the whole dataset into cpu memory on every single processes could cause OOMs for data parallel

distributed training. To avoid such practice, we move the operations that required loading the whole dataset out of spawn, and pass the context to every single process.

pytext.workflow.**reload_model_for_multi_export**(*config: pytext.config.pytext_config.PyTextConfig*)

pytext.workflow.**save_and_export**(*config: pytext.config.pytext_config.PyTextConfig, task: pytext.task.task.Task_Deprecated, metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]] = None*) → None

pytext.workflow.**save_pytext_snapshot**(*config: pytext.config.pytext_config.PyTextConfig*) → None

pytext.workflow.**test_model**(*test_config: pytext.config.pytext_config.TestConfig, metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]], test_out_path: str*) → Any

pytext.workflow.**test_model_from_snapshot_path**(*snapshot_path: str, use_cuda_if_available: bool, test_path: Optional[str] = None, metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]] = None, test_out_path: str = '', field_names: Optional[List[str]] = None*)

pytext.workflow.**train_model**(*config: pytext.config.pytext_config.PyTextConfig, dist_init_url: str = None, device_id: int = 0, rank: int = 0, world_size: int = 1, metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]] = None, metadata: pytext.data.data_handler.CommonMetadata = None*) → Tuple

### 1.22.6 Module contents

pytext.**batch_predict_caffe2_model**(*pytext_model_file: str, caffe2_model_file: str, db_type: str = 'minidb', data_source: Optional[pytext.data.sources.data_source.DataSource] = None, use_cuda=False, task: Optional[pytext.task.new_task.NewTask] = None, train_config: Optional[pytext.config.pytext_config.PyTextConfig] = None, cache_size: int = 0*)

Gets predictions from caffe2 model from a batch of examples.

**Parameters**

- **pytext_model_file** – Path to pytext model file (required if task and training config is not specified)
- **caffe2_model_file** – Path to caffe2 model file
- **db_type** – DB type to use for caffe2
- **data_source** – Data source for test examples
- **use_cuda** – Whether to turn on cuda processing
- **task** – The pytext task object
- **train_config** – The pytext training config

- **cache_size** – The LRU cache size to use for prediction. 0 = no cache, -1 = boundless cache, [1, inf) = size of cache

`pytext.`**`create_predictor`**(*config: pytext.config.pytext_config.PyTextConfig, model_file: Optional[str] = None, db_type: str = 'minidb', task: Optional[pytext.task.new_task.NewTask] = None, cache_size: int = 0*) → Callable[[Mapping[str, str]], Mapping[str, numpy.array]]

Create a simple prediction API from a training config and an exported caffe2 model file. This model file should be created by calling export on a trained model snapshot.

`pytext.`**`load_config`**(*filename: str*) → pytext.config.pytext_config.PyTextConfig

Load a PyText configuration file from a file path. See pytext.config.pytext_config for more info on configs.

CHAPTER 2

# Indices and tables

- genindex
- search

# Python Module Index

## p


pytext, 772
pytext.builtin_task, 770
pytext.common, 403
pytext.common.constants, 400
pytext.common.utils, 403
pytext.config, 415
pytext.config.component, 403
pytext.config.config_adapter, 405
pytext.config.contextual_intent_slot, 407
pytext.config.doc_classification, 408
pytext.config.field_config, 408
pytext.config.module_config, 409
pytext.config.pair_classification, 411
pytext.config.pytext_config, 411
pytext.config.query_document_pairwise_ranking, 414
pytext.config.serialize, 414
pytext.config.utils, 414
pytext.data, 482
pytext.data.batch_sampler, 440
pytext.data.bert_tensorizer, 442
pytext.data.data, 444
pytext.data.data_handler, 446
pytext.data.data_structures, 418
pytext.data.data_structures.annotation, 415
pytext.data.data_structures.node, 417
pytext.data.dense_retrieval_tensorizer, 451
pytext.data.disjoint_multitask_data, 453
pytext.data.disjoint_multitask_data_handler, 454
pytext.data.dynamic_pooling_batcher, 456
pytext.data.featurizer, 420
pytext.data.featurizer.featurizer, 418
pytext.data.featurizer.simple_featurizer, 419
pytext.data.masked_tensorizer, 457
pytext.data.masked_util, 458
pytext.data.packed_lm_data, 459
pytext.data.roberta_tensorizer, 459
pytext.data.sources, 428
pytext.data.sources.conllu, 421
pytext.data.sources.data_source, 422
pytext.data.sources.dense_retrieval, 424
pytext.data.sources.pandas, 425
pytext.data.sources.session, 426
pytext.data.sources.squad, 426
pytext.data.sources.tsv, 427
pytext.data.squad_for_bert_tensorizer, 460
pytext.data.squad_tensorizer, 461
pytext.data.tensorizers, 462
pytext.data.test, 436
pytext.data.test.batch_sampler_test, 431
pytext.data.test.data_test, 431
pytext.data.test.dynamic_pooling_batcher_test, 432
pytext.data.test.mask_tensorizers_test, 432
pytext.data.test.pandas_data_source_test, 432
pytext.data.test.round_robin_batchiterator_test, 432
pytext.data.test.simple_featurizer_test, 432
pytext.data.test.tensorizers_test, 433
pytext.data.test.tokenizers_test, 435
pytext.data.test.tsv_data_source_test, 435
pytext.data.test.utils_test, 436
pytext.data.token_tensorizer, 478
pytext.data.tokenizers, 438
pytext.data.tokenizers.tokenizer, 436
pytext.data.utils, 480
pytext.data.xlm_constants, 481
pytext.data.xlm_dictionary, 481

pytext.data.xlm_tensorizer, 481
pytext.exporters, 496
pytext.exporters.custom_exporters, 493
pytext.exporters.exporter, 494
pytext.fields, 503
pytext.fields.char_field, 499
pytext.fields.contextual_token_embedding_field, 499
pytext.fields.dict_field, 500
pytext.fields.field, 501
pytext.fields.text_field_with_special_unk, 503
pytext.loss, 510
pytext.loss.loss, 507
pytext.loss.regularized_loss, 509
pytext.loss.regularizer, 509
pytext.loss.structured_loss, 510
pytext.main, 771
pytext.metric_reporters, 533
pytext.metric_reporters.calibration_metric_reporter, 513
pytext.metric_reporters.channel, 514
pytext.metric_reporters.classification_metric_reporter, 517
pytext.metric_reporters.compositional_metric_reporter, 520
pytext.metric_reporters.compositional_utils, 521
pytext.metric_reporters.dense_retrieval_metric_reporter, 521
pytext.metric_reporters.disjoint_multitask_metric_reporter, 522
pytext.metric_reporters.intent_slot_detection_metric_reporter, 523
pytext.metric_reporters.language_model_metric_reporter, 524
pytext.metric_reporters.metric_reporter, 526
pytext.metric_reporters.pairwise_ranking_metric_reporter, 528
pytext.metric_reporters.regression_metric_reporter, 528
pytext.metric_reporters.seq2seq_compositional, 529
pytext.metric_reporters.seq2seq_metric_reporter, 529
pytext.metric_reporters.seq2seq_utils, 530
pytext.metric_reporters.squad_metric_reporter, 530
pytext.metric_reporters.word_tagging_metric_reporter, 532
pytext.metrics, 553
pytext.metrics.calibration_metrics, 544
pytext.metrics.dense_retrieval_metrics, 545
pytext.metrics.intent_slot_metrics, 546
pytext.metrics.language_model_metrics, 551
pytext.metrics.seq2seq_metrics, 552
pytext.metrics.squad_metrics, 552
pytext.models, 704
pytext.models.bert_classification_models, 686
pytext.models.bert_regression_model, 686
pytext.models.crf, 687
pytext.models.decoders, 569
pytext.models.decoders.decoder_base, 565
pytext.models.decoders.intent_slot_model_decoder, 566
pytext.models.decoders.mlp_decoder, 567
pytext.models.decoders.mlp_decoder_query_response, 568
pytext.models.decoders.mlp_decoder_two_tower, 568
pytext.models.decoders.multilabel_decoder, 569
pytext.models.disjoint_multitask_model, 687
pytext.models.distributed_model, 688
pytext.models.doc_model, 689
pytext.models.embeddings, 582
pytext.models.embeddings.char_embedding, 572
pytext.models.embeddings.contextual_token_embedding, 574
pytext.models.embeddings.dict_embedding, 574
pytext.models.embeddings.embedding_base, 576
pytext.models.embeddings.embedding_list, 576
pytext.models.embeddings.mlp_embedding, 577
pytext.models.embeddings.scriptable_embedding_list, 578
pytext.models.embeddings.word_embedding, 579
pytext.models.embeddings.word_seq_embedding, 581
pytext.models.ensembles, 592
pytext.models.ensembles.bagging_doc_ensemble, 589
pytext.models.ensembles.bagging_intent_slot_ensemble, 590
pytext.models.ensembles.ensemble, 591
pytext.models.joint_model, 691
pytext.models.language_models, 596

pytext.models.language_models.lmlstm, 595
pytext.models.masked_lm, 692
pytext.models.masking_utils, 692
pytext.models.model, 693
pytext.models.module, 696
pytext.models.output_layers, 613
pytext.models.output_layers.distance_output_layer, 597
pytext.models.output_layers.doc_classification_output_layer, 599
pytext.models.output_layers.doc_regression_output_layer, 601
pytext.models.output_layers.intent_slot_output_layer, 603
pytext.models.output_layers.lm_output_layer, 604
pytext.models.output_layers.multi_label_classification_layer, 605
pytext.models.output_layers.output_layer_base, 607
pytext.models.output_layers.pairwise_ranking_output_layer, 608
pytext.models.output_layers.squad_output_layer, 609
pytext.models.output_layers.utils, 610
pytext.models.output_layers.word_tagging_output_layer, 611
pytext.models.pair_classification_model, 697
pytext.models.qna, 623
pytext.models.qna.bert_squad_qa, 622
pytext.models.qna.dr_qa, 622
pytext.models.query_document_pairwise_ranking_model, 698
pytext.models.r3f_models, 699
pytext.models.representations, 660
pytext.models.representations.attention, 637
pytext.models.representations.augmented_lstm, 638
pytext.models.representations.bilstm, 641
pytext.models.representations.bilstm_doc_attention, 642
pytext.models.representations.bilstm_doc_slot_attention, 643
pytext.models.representations.bilstm_slot_attn, 644
pytext.models.representations.biseqcnn, 645
pytext.models.representations.contextual_intent_slot_rep, 646
pytext.models.representations.deepcnn, 647
pytext.models.representations.docnn, 649
pytext.models.representations.huggingface_bert_sent 649
pytext.models.representations.huggingface_electra_s 650
pytext.models.representations.jointcnn_rep, 650
pytext.models.representations.ordered_neuron_lstm, 651
pytext.models.representations.pair_rep, 651
pytext.models.representations.pass_through, 652
pytext.models.representations.pooling, 652
pytext.models.representations.pure_doc_attention, 654
pytext.models.representations.representation_base, 654
pytext.models.representations.seq_rep, 655
pytext.models.representations.slot_attention, 655
pytext.models.representations.sparse_transformer_se 656
pytext.models.representations.stacked_bidirectional 656
pytext.models.representations.traced_transformer_en 657
pytext.models.representations.transformer, 631
pytext.models.representations.transformer.multihead 623
pytext.models.representations.transformer.multihead 624
pytext.models.representations.transformer.positiona 625
pytext.models.representations.transformer.represent 626
pytext.models.representations.transformer.residual_ 626
pytext.models.representations.transformer.sentence_ 627
pytext.models.representations.transformer.transform 629
pytext.models.representations.transformer_sentence_ 658
pytext.models.representations.transformer_sentence_ 659
pytext.models.roberta, 700
pytext.models.semantic_parsers, 664
pytext.models.semantic_parsers.rnng, 664
pytext.models.semantic_parsers.rnng.rnng_constant,

660
pytext.models.semantic_parsers.rnng.rnng_data_structures, 660
pytext.models.semantic_parsers.rnng.rnng_parser, 662
pytext.models.seq_models, 685
pytext.models.seq_models.attention, 665
pytext.models.seq_models.base, 666
pytext.models.seq_models.contextual_intent_slot, 667
pytext.models.seq_models.conv_decoder, 668
pytext.models.seq_models.conv_encoder, 671
pytext.models.seq_models.conv_model, 673
pytext.models.seq_models.light_conv, 673
pytext.models.seq_models.mask_generator, 674
pytext.models.seq_models.nar_length, 677
pytext.models.seq_models.nar_modules, 678
pytext.models.seq_models.nar_output_layer, 678
pytext.models.seq_models.positional, 678
pytext.models.seq_models.projection_layer, 680
pytext.models.seq_models.rnn_decoder, 681
pytext.models.seq_models.rnn_encoder, 682
pytext.models.seq_models.rnn_encoder_decoder, 683
pytext.models.seq_models.seq2seq_model, 683
pytext.models.seq_models.seq2seq_output_layer, 684
pytext.models.seq_models.seqnn, 685
pytext.models.seq_models.utils, 685
pytext.models.two_tower_classification_model, 702
pytext.models.utils, 703
pytext.models.word_model, 703
pytext.optimizer, 724
pytext.optimizer.activations, 712
pytext.optimizer.adabelief, 712
pytext.optimizer.fairseq_fp16_utils, 712
pytext.optimizer.fp16_optimizer, 713
pytext.optimizer.lamb, 717
pytext.optimizer.madgrad, 717
pytext.optimizer.optimizers, 718
pytext.optimizer.privacy_engine, 719
pytext.optimizer.radam, 719
pytext.optimizer.scheduler, 719
pytext.optimizer.sparsifiers, 712
pytext.optimizer.sparsifiers.blockwise_sparsifier, 707
pytext.optimizer.sparsifiers.sparsifier, 709
pytext.optimizer.swa, 722
pytext.resources, 724
pytext.resources.roberta, 724
pytext.task, 735
pytext.task.accelerator_lowering, 724
pytext.task.disjoint_multitask, 726
pytext.task.new_task, 727
pytext.task.nop_decorator, 728
pytext.task.quantize, 728
pytext.task.serialize, 728
pytext.task.task, 730
pytext.task.tasks, 732
pytext.torchscript, 754
pytext.torchscript.batchutils, 746
pytext.torchscript.module, 748
pytext.torchscript.seq2seq, 740
pytext.torchscript.seq2seq.beam_decode, 737
pytext.torchscript.seq2seq.beam_search, 738
pytext.torchscript.seq2seq.decoder, 738
pytext.torchscript.seq2seq.encoder, 738
pytext.torchscript.seq2seq.export_model, 739
pytext.torchscript.seq2seq.scripted_seq2seq_generat 739
pytext.torchscript.seq2seq.seq2seq_rnn_decoder_util 740
pytext.torchscript.tensorizer, 742
pytext.torchscript.tensorizer.bert, 740
pytext.torchscript.tensorizer.normalizer, 740
pytext.torchscript.tensorizer.roberta, 741
pytext.torchscript.tensorizer.tensorizer, 741
pytext.torchscript.tensorizer.xlm, 742
pytext.torchscript.tokenizer, 745
pytext.torchscript.tokenizer.bpe, 744
pytext.torchscript.tokenizer.tokenizer, 745
pytext.torchscript.utils, 753
pytext.torchscript.vocab, 754
pytext.trainers, 757
pytext.trainers.ensemble_trainer, 754
pytext.trainers.hogwild_trainer, 754
pytext.trainers.trainer, 755
pytext.trainers.training_state, 757
pytext.utils, 770
pytext.utils.ascii_table, 761

pytext.utils.config_utils, 761
pytext.utils.cuda, 761
pytext.utils.data, 761
pytext.utils.distributed, 762
pytext.utils.documentation, 763
pytext.utils.embeddings, 763
pytext.utils.file_io, 764
pytext.utils.label, 764
pytext.utils.lazy, 764
pytext.utils.loss, 766
pytext.utils.meter, 768
pytext.utils.mobile_onnx, 768
pytext.utils.model, 768
pytext.utils.onnx, 768
pytext.utils.path, 769
pytext.utils.precision, 769
pytext.utils.tensor, 769
pytext.utils.test, 769
pytext.utils.timing, 769
pytext.utils.usage, 770
pytext.workflow, 771

# Index

## A


accelerate (*pytext.config.pytext_config.ExportConfig attribute*), 411
accelerate (*pytext.config.pytext_config.PyTextConfig attribute*), 412
accelerator (*class in pytext.task.nop_decorator*), 728
accelerator_lstm_inputs() (*in module pytext.task.accelerator_lowering*), 725
accelerator_transformerLayers_inputs() (*in module pytext.task.accelerator_lowering*), 725
AcceleratorBiLSTM (*class in pytext.task.accelerator_lowering*), 724
AcceleratorTransformer (*class in pytext.task.accelerator_lowering*), 724
AcceleratorTransformerLayersInternal (*class in pytext.task.accelerator_lowering*), 725
ACCURACY (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 518
ACCURACY (*pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetrievalMetricNames attribute*), 521
accuracy (*pytext.metrics.ClassificationMetrics attribute*), 553
accuracy (*pytext.metrics.dense_retrieval_metrics.DenseRetrievalMetrics attribute*), 545, 546
accuracy (*pytext.metrics.PairwiseRankingMetrics attribute*), 557
ActionField (*class in pytext.fields*), 503
ActionField (*class in pytext.fields.field*), 501
Activation (*class in pytext.config.module_config*), 409
AdaBelief (*class in pytext.optimizer.adabelief*), 712
Adagrad (*class in pytext.optimizer.optimizers*), 718
Adam (*class in pytext.optimizer.optimizers*), 718
AdamW (*class in pytext.optimizer.optimizers*), 718
AdaptiveRegularizer (*class in pytext.loss*), 513
AdaptiveRegularizer (*class in pytext.loss.regularizer*), 509
add() (*pytext.config.component.Registry class method*), 404
add() (*pytext.data.utils.VocabBuilder method*), 480
add() (*pytext.utils.timing.Timings method*), 770
add_all() (*pytext.data.utils.VocabBuilder method*), 480
add_batch_stats() (*pytext.metric_reporters.classification_metric_reporter.ClassificationM method*), 517
add_batch_stats() (*pytext.metric_reporters.ClassificationMetricReporter method*), 536
add_batch_stats() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointM method*), 522
add_batch_stats() (*pytext.metric_reporters.language_model_metric_reporter.LanguageM method*), 524
add_batch_stats() (*pytext.metric_reporters.language_model_metric_reporter.MaskedLM method*), 525
add_batch_stats() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 540
add_batch_stats() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 526
add_batch_stats() (*pytext.metric_reporters.MetricReporter method*), 534
add_batch_stats() (*pytext.metric_reporters.pairwise_ranking_metric_reporter.PairwiseR method*), 528
add_batch_stats() (*pytext.metric_reporters.PairwiseRankingMetricReporter method*), 543
add_batch_stats() (*pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricRep method*), 530
add_batch_stats() (*py-*

*text.metric_reporters.squad_metric_reporter.SquadMetricReporter method*), 531
add_batch_stats() (*pytext.metric_reporters.SquadMetricReporter method*), 541
add_channel() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 522
add_channel() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
add_channel() (*pytext.metric_reporters.MetricReporter method*), 535
add_feats_numericalize_ops() (*in module pytext.utils.mobile_onnx*), 768
add_feats_numericalize_ops() (*in module pytext.utils.onnx*), 768
add_from_file() (*pytext.data.utils.VocabBuilder method*), 480
add_gradients() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
add_gradients() (*pytext.metric_reporters.MetricReporter method*), 535
add_include() (*in module pytext.builtin_task*), 770
add_param_group() (*pytext.optimizer.madgrad.MADGRAD method*), 717
add_param_group() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 722
add_row_indices() (*pytext.data.Data method*), 484
add_row_indices() (*pytext.data.data.Data method*), 445
add_scalars() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 515
add_texts() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 516
aggregate_context() (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method*), 525
aggregate_context() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 540
aggregate_data() (*pytext.metric_reporters.metric_reporter.MetricReporter class method*), 527
aggregate_data() (*pytext.metric_reporters.MetricReporter class method*), 535
aggregate_preds() (*pytext.metric_reporters.calibration_metric_reporter.CalibrationMetricReporter method*), 513
aggregate_preds() (*pytext.metric_reporters.CalibrationMetricReporter method*), 536
aggregate_preds() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 521
aggregate_preds() (*pytext.metric_reporters.DenseRetrievalMetricReporter method*), 544
aggregate_preds() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 523
aggregate_preds() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 539
aggregate_preds() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
aggregate_preds() (*pytext.metric_reporters.MetricReporter method*), 535
aggregate_preds() (*pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter method*), 539
aggregate_preds() (*pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositionalMetricReporter method*), 529
aggregate_preds() (*pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter method*), 530
aggregate_preds() (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter method*), 531
aggregate_preds() (*pytext.metric_reporters.SquadMetricReporter method*), 541
aggregate_preds() (*pytext.metric_reporters.word_tagging_metric_reporter.MultiLabelSeqTaggingMetricReporter method*), 532
aggregate_scores() (*pytext.metric_reporters.calibration_metric_reporter.CalibrationMetricReporter method*), 513
aggregate_scores() (*pytext.metric_reporters.CalibrationMetricReporter method*), 536
aggregate_scores() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 523
aggregate_scores() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 539
aggregate_scores() (*py-*

*text.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method*), 525
aggregate_scores() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 540
aggregate_scores() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
aggregate_scores() (*pytext.metric_reporters.MetricReporter method*), 535
aggregate_scores() (*pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter method*), 539
aggregate_scores() (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter method*), 531
aggregate_scores() (*pytext.metric_reporters.SquadMetricReporter method*), 541
aggregate_scores() (*pytext.metric_reporters.word_tagging_metric_reporter.MultiLabelSequenceTaggingMetricReporter method*), 532
aggregate_src_tokens() (*pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter method*), 530
aggregate_targets() (*pytext.metric_reporters.calibration_metric_reporter.CalibrationMetricReporter method*), 513
aggregate_targets() (*pytext.metric_reporters.CalibrationMetricReporter method*), 536
aggregate_targets() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 523
aggregate_targets() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 539
aggregate_targets() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
aggregate_targets() (*pytext.metric_reporters.MetricReporter method*), 535
aggregate_targets() (*pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter method*), 539
aggregate_targets() (*pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositional method*), 529
aggregate_targets() (*pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter method*), 530
aggregate_targets() (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter method*), 531
aggregate_targets() (*pytext.metric_reporters.SquadMetricReporter method*), 541
aggregate_targets() (*pytext.metric_reporters.word_tagging_metric_reporter.MultiLabelS method*), 532
aggregate_tuple_data() (*pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter method*), 539
aggregate_tuple_data() (*pytext.metric_reporters.word_tagging_metric_reporter.MultiLabelS method*), 532
align_slot_labels() (*in module pytext.utils.data*), 762
align_target_label() (*in module pytext.data.utils*), 480
align_target_labels() (*in module pytext.data.utils*), 480
ALIGNMENT (*pytext.common.constants.DFColumn attribute*), 400
AllCalibrationMetrics (*class in pytext.metrics.calibration_metrics*), 544
AllMetrics (*class in pytext.metrics*), 553
AllMetrics (*class in pytext.metrics.intent_slot_metrics*), 546
AlternatingRandomizedBatchSampler (*class in pytext.data*), 482
AlternatingRandomizedBatchSampler (*class in pytext.data.batch_sampler*), 440
ANALYSIS (*pytext.optimizer.sparsifiers.sparsifier.State attribute*), 712
Annotation (*class in pytext.data.data_structures.annotation*), 415
AnnotationNumberizer (*class in pytext.data.tensorizers*), 462
annotations_and_defaults() (*pytext.config.pytext_config.ConfigBaseMeta method*), 411
ANSWERS_COLUMN (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter attribute*), 531
ANSWERS_COLUMN (*pytext.metric_reporters.SquadMetricReporter attribute*), 541
append_dialect() (*in module pytext.utils.embeddings*), 764
apply_mask() (*pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier method*), 710
apply_mask() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 710

arrange_caffe2_model_inputs() (*pytext.models.BaseModel method*), 706
arrange_caffe2_model_inputs() (*pytext.models.model.BaseModel method*), 693
arrange_model_context() (*pytext.models.BaseModel method*), 706
arrange_model_context() (*pytext.models.disjoint_multitask_model.NewDisjointMultitaskModel method*), 688
arrange_model_context() (*pytext.models.ensembles.ensemble.EnsembleModel method*), 591
arrange_model_context() (*pytext.models.ensembles.EnsembleModel method*), 593
arrange_model_context() (*pytext.models.joint_model.IntentSlotModel method*), 691
arrange_model_context() (*pytext.models.model.BaseModel method*), 693
arrange_model_context() (*pytext.models.word_model.WordTaggingLiteModel method*), 703
arrange_model_context() (*pytext.models.word_model.WordTaggingModel method*), 703
arrange_model_inputs() (*pytext.models.BaseModel method*), 706
arrange_model_inputs() (*pytext.models.disjoint_multitask_model.NewDisjointMultitaskModel method*), 688
arrange_model_inputs() (*pytext.models.doc_model.ByteTokensDocumentModel method*), 689
arrange_model_inputs() (*pytext.models.doc_model.DocModel method*), 690
arrange_model_inputs() (*pytext.models.doc_model.PersonalizedDocModel method*), 691
arrange_model_inputs() (*pytext.models.ensembles.ensemble.EnsembleModel method*), 591
arrange_model_inputs() (*pytext.models.ensembles.EnsembleModel method*), 593
arrange_model_inputs() (*pytext.models.joint_model.IntentSlotModel method*), 691
arrange_model_inputs() (*pytext.models.language_models.lmlstm.LMLSTM method*), 595
arrange_model_inputs() (*pytext.models.masked_lm.MaskedLanguageModel method*), 692
arrange_model_inputs() (*pytext.models.model.BaseModel method*), 693
arrange_model_inputs() (*pytext.models.pair_classification_model.PairwiseModel method*), 697
arrange_model_inputs() (*pytext.models.qna.bert_squad_qa.BertSquadQAModel method*), 622
arrange_model_inputs() (*pytext.models.qna.dr_qa.DrQAModel method*), 622
arrange_model_inputs() (*pytext.models.query_document_pairwise_ranking_model.QueryDoc method*), 698
arrange_model_inputs() (*pytext.models.roberta.RoBERTaWordTaggingModel method*), 701
arrange_model_inputs() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 662
arrange_model_inputs() (*pytext.models.seq_models.contextual_intent_slot.ContextualIntentSl method*), 668
arrange_model_inputs() (*pytext.models.seq_models.seq2seq_model.Seq2SeqModel method*), 683
arrange_model_inputs() (*pytext.models.seq_models.seqnn.SeqNNModel method*), 685
arrange_model_inputs() (*pytext.models.two_tower_classification_model.TwoTowerClassificat method*), 702
arrange_model_inputs() (*pytext.models.TwoTowerClassificationModel method*), 706
arrange_model_inputs() (*pytext.models.word_model.WordTaggingLiteModel method*), 703
arrange_model_inputs() (*pytext.models.word_model.WordTaggingModel method*), 703
arrange_targets() (*pytext.models.BaseModel method*), 706
arrange_targets() (*pytext.models.disjoint_multitask_model.NewDisjointMultitaskMode method*), 688
arrange_targets() (*pytext.models.doc_model.DocModel method*), 690
arrange_targets() (*pytext.models.ensembles.ensemble.EnsembleModel method*), 591
arrange_targets() (*py-*

*text.models.ensembles.EnsembleModel method*), 593
arrange_targets() (*pytext.models.joint_model.IntentSlotModel method*), 691
arrange_targets() (*pytext.models.language_models.lmlstm.LMLSTM method*), 595
arrange_targets() (*pytext.models.masked_lm.MaskedLanguageModel method*), 692
arrange_targets() (*pytext.models.model.BaseModel method*), 693
arrange_targets() (*pytext.models.pair_classification_model.PairwiseModel method*), 698
arrange_targets() (*pytext.models.qna.bert_squad_qa.BertSquadQAModel method*), 622
arrange_targets() (*pytext.models.qna.dr_qa.DrQAModel method*), 622
arrange_targets() (*pytext.models.query_document_pairwise_ranking_model.QueryDocPairwiseRankingModel method*), 698
arrange_targets() (*pytext.models.roberta.RoBERTaWordTaggingModel method*), 701
arrange_targets() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 662
arrange_targets() (*pytext.models.seq_models.seq2seq_model.Seq2SeqModel method*), 683
arrange_targets() (*pytext.models.two_tower_classification_model.TwoTowerClassificationModel method*), 702
arrange_targets() (*pytext.models.TwoTowerClassificationModel method*), 706
arrange_targets() (*pytext.models.word_model.WordTaggingModel method*), 703
ascii_table() (*in module pytext.utils.ascii_table*), 761
ascii_table_from_dict() (*in module pytext.utils.ascii_table*), 761
assign_id() (*pytext.models.seq_models.base.PyTextSeq2SeqModule method*), 667
attach() (*pytext.optimizer.privacy_engine.PrivacyEngine method*), 719
attention (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute*), 642
attention (*pytext.models.representations.bilstm_slot_attention.BiLSTMSlotAttention attribute*), 645
Attrs (*class in pytext.main*), 771
AUCPRHingeLoss (*class in pytext.loss*), 510
AUCPRHingeLoss (*class in pytext.loss.loss*), 507
AugmentedLSTM (*class in pytext.models.representations.augmented_lstm*), 638
AugmentedLSTMCell (*class in pytext.models.representations.augmented_lstm*), 639
AugmentedLSTMUnidirectional (*class in pytext.models.representations.augmented_lstm*), 639
auto_resume_from_snapshot (*pytext.config.pytext_config.PyTextConfig attribute*), 412
average (*pytext.utils.timing.Timings attribute*), 770
average_label_precision (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 555
average_label_recall (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 555
average_label_roc_auc (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 555
average_overall_auc (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 555
average_overall_precision (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556
average_overall_recall (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556
average_precision (*pytext.metrics.SoftClassificationMetrics attribute*), 559
average_precision_score() (*in module pytext.metrics*), 559
average_rank (*pytext.metrics.dense_retrieval_metrics.DenseRetrievalM attribute*), 545, 546
average_score_difference (*pytext.metrics.PairwiseRankingMetrics attribute*), 557
AVERAGE_TOKEN_LPROB (*pytext.models.seq_models.mask_generator.BeamRankingAlgorithm attribute*), 674
avg (*pytext.utils.meter.Meter attribute*), 768
avg (*pytext.utils.meter.TimeMeter attribute*), 768
AVG_CONCAT_LAST_4_LAYERS (*pytext.models.representations.transformer_sentence_encoder_base. attribute*), 659
AVG_LAST_LAYER (*pytext.models.representation*

text.models.representations.transformer_sentence_encoder_base.PoolingMethod attribute), 659
AVG_RANK (pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetrievalMetricNames attribute), 521
AVG_SECOND_TO_LAST_LAYER (pytext.models.representations.transformer_sentence_encoder_base.PoolingMethod attribute), 659
AVG_SUM_LAST_4_LAYERS (pytext.models.representations.transformer_sentence_encoder_base.PoolingMethod attribute), 659
avg_token_lprob() (in module pytext.models.seq_models.mask_generator), 675

## B

b_label_name (pytext.utils.data.Slot attribute), 761
B_LABEL_PREFIX (pytext.utils.data.Slot attribute), 761
backprop() (pytext.trainers.Trainer method), 758
backprop() (pytext.trainers.trainer.Trainer method), 755
backward() (pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16OptimizerMixin method), 712
backward() (pytext.optimizer.fairseq_fp16_utils.Fairseq_MemoryEfficientFP16OptimizerMixin method), 713
backward() (pytext.optimizer.fp16_optimizer.FP16Optimizer method), 714
backward() (pytext.optimizer.fp16_optimizer.FP16OptimizerApex method), 714
backward() (pytext.optimizer.optimizers.Optimizer method), 718
backward() (pytext.utils.loss.LagrangeMultiplier static method), 766
backward_layers (pytext.models.representations.augmented_lstm.AugmentedLSTM attribute), 638
BaggingDocEnsembleModel (class in pytext.models.ensembles), 592
BaggingDocEnsembleModel (class in pytext.models.ensembles.bagging_doc_ensemble), 589
BaggingIntentSlotEnsembleModel (class in pytext.models.ensembles), 592
BaggingIntentSlotEnsembleModel (class in pytext.models.ensembles.bagging_intent_slot_ensemble), 590
BaseBatchSampler (class in pytext.data), 483
BaseBatchSampler (class in pytext.data.batch_sampler), 440
BaseModel (class in pytext.models), 705
BaseModel (class in pytext.models.model), 693
BasePairwiseModel (class in pytext.models.pair_classification_model), 697
batch_context() (pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method), 537
batch_context() (pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method), 542
batch_context() (pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetrieval method), 521
batch_context() (pytext.metric_reporters.DenseRetrievalMetricReporter method), 544
batch_context() (pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultiTaskMetricReporter method), 522
batch_context() (pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotDetectionMetricReporter method), 523
batch_context() (pytext.metric_reporters.IntentSlotMetricReporter method), 539
batch_context() (pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method), 525
batch_context() (pytext.metric_reporters.LanguageModelMetricReporter method), 540
batch_context() (pytext.metric_reporters.metric_reporter.MetricReporter method), 527
batch_context() (pytext.metric_reporters.MetricReporter method), 535
batch_context() (pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter method), 539
batch_context() (pytext.metric_reporters.NERMetricReporter method), 544
batch_context() (pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositional method), 529
batch_context() (pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter method), 530
batch_context() (pytext.metric_reporters.SequenceTaggingMetricReporter method), 543

batch_context() (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter method*), 531
batch_context() (*pytext.metric_reporters.SquadMetricReporter method*), 541
batch_context() (*pytext.metric_reporters.word_tagging_metric_reporter.MultiLabelSeqTaggingMetricReporter method*), 532
batch_context() (*pytext.metric_reporters.word_tagging_metric_reporter.NERMetricReporter method*), 532
batch_context() (*pytext.metric_reporters.word_tagging_metric_reporter.SequenceTaggingMetricReporter method*), 532
batch_padding_control (*pytext.config.pytext_config.ExportConfig attribute*), 411
batch_padding_control (*pytext.config.pytext_config.PyTextConfig attribute*), 412
batch_predict() (*in module pytext.workflow*), 771
batch_predict_caffe2_model() (*in module pytext*), 772
BATCH_SAMPLER (*pytext.config.component.ComponentType attribute*), 403
batch_size (*pytext.config.pytext_config.LogitsConfig attribute*), 411
batch_size() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 475
BatchContext (*class in pytext.common.constants*), 400
BatchData (*class in pytext.data.data*), 444
Batcher (*class in pytext.data*), 482
Batcher (*class in pytext.data.data*), 444
BATCHER (*pytext.config.component.ComponentType attribute*), 403
BatcherSchedulerConfig (*class in pytext.data.dynamic_pooling_batcher*), 456
BatcherTest (*class in pytext.data.test.data_test*), 431
batches() (*pytext.data.Data method*), 484
batches() (*pytext.data.data.Data method*), 445
batches() (*pytext.data.disjoint_multitask_data.DisjointMultitaskDataset method*), 454
batches() (*pytext.data.DisjointMultitaskData method*), 488
batchify() (*pytext.data.AlternatingRandomizedBatchSampler method*), 482
batchify() (*pytext.data.BaseBatchSampler method*), 483
batchify() (*pytext.data.batch_sampler.AlternatingRandomizedBatchSampler method*), 440
batchify() (*pytext.data.batch_sampler.BaseBatchSampler method*), 440
batchify() (*pytext.data.batch_sampler.EvalBatchSampler method*), 440
batchify() (*pytext.data.batch_sampler.NaturalBatchSampler method*), 440
batchify() (*pytext.data.batch_sampler.RandomizedBatchSampler method*), 441
batchify() (*pytext.data.batch_sampler.RoundRobinBatchSampler method*), 441
batchify() (*pytext.data.Batcher method*), 483
batchify() (*pytext.data.data.Batcher method*), 444
batchify() (*pytext.data.data.PoolingBatcher method*), 446
batchify() (*pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher method*), 456
batchify() (*pytext.data.DynamicPoolingBatcher method*), 489
batchify() (*pytext.data.EvalBatchSampler method*), 489
batchify() (*pytext.data.NaturalBatchSampler method*), 491
batchify() (*pytext.data.PoolingBatcher method*), 490
batchify() (*pytext.data.RandomizedBatchSampler method*), 491
batchify() (*pytext.data.RoundRobinBatchSampler method*), 491
BatchIterator (*class in pytext.data*), 483
BatchIterator (*class in pytext.data.data_handler*), 446
BatchSamplerTest (*class in pytext.data.test.batch_sampler_test*), 431
BatchScheduler (*class in pytext.optimizer.scheduler*), 719
beam_search_aggregate_topk() (*pytext.torchscript.seq2seq.decoder.DecoderBatchedStepEnsemble method*), 738
BeamDecode (*class in pytext.torchscript.seq2seq.beam_decode*), 737
BeamRankingAlgorithm (*class in pytext.models.seq_models.mask_generator*), 674
BeamSearch (*class in pytext.torchscript.seq2seq.beam_search*), 738
BERTCDatatextTensorizerForDenseRetrieval (*class in pytext.data.dense_retrieval_tensorizer*), 451
BERTInitialTokenizer (*class in pytext.data.tokenizers.tokenizer*), 436
BertPairRegressionTask (*class in pytext.task.tasks*), 732
BertPairwiseModel (*class in pytext.models.bert_classification_models*), 686
BertPairwiseRegressionModel (*class in py-*

*text.models.bert_regression_model*), 686
BertSquadQAModel (*class in pytext.models.qna.bert_squad_qa*), 622
BERTTensorizer (*class in pytext.data.bert_tensorizer*), 442
BERTTensorizerBase (*class in pytext.data.bert_tensorizer*), 442
BERTTensorizerBaseScriptImpl (*class in pytext.data.bert_tensorizer*), 443
BERTTensorizerScriptImpl (*class in pytext.data.bert_tensorizer*), 443
BERTTensorizerTest (*class in pytext.data.test.tensorizers_test*), 433
best_model_metric (*pytext.trainers.training_state.TrainingState attribute*), 757
best_model_metric (*pytext.trainers.TrainingState attribute*), 759
best_model_state (*pytext.trainers.training_state.TrainingState attribute*), 757
best_model_state (*pytext.trainers.TrainingState attribute*), 759
bidirectional (*pytext.models.representations.augmented_lstm.AugmentedLSTM.Config attribute*), 227
bidirectional (*pytext.models.representations.bilstm.BiLSTM.Config attribute*), 228
bidirectional (*pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN.Config attribute*), 250
BiLSTM (*class in pytext.models.representations.bilstm*), 641
BiLSTM (*class in pytext.models.seq_models.rnn_encoder*), 682
BiLSTMDocAttention (*class in pytext.models.representations.bilstm_doc_attention*), 642
BiLSTMDocSlotAttention (*class in pytext.models.representations.bilstm_doc_slot_attention*), 643
BiLSTMSlotAttention (*class in pytext.models.representations.bilstm_slot_attn*), 644
BinaryClassificationOutputLayer (*class in pytext.models.output_layers.doc_classification_output_layer*), 599
BinaryCrossEntropyLoss (*class in pytext.loss*), 511
BinaryCrossEntropyLoss (*class in pytext.loss.loss*), 508
BinaryCrossEntropyWithLogitsLoss (*class in pytext.loss*), 511
BinaryCrossEntropyWithLogitsLoss (*class in pytext.loss.loss*), 508
BIT_4 (*pytext.models.seq_models.mask_generator.EmbedQuantizeType attribute*), 674
BIT_8 (*pytext.models.seq_models.mask_generator.EmbedQuantizeType attribute*), 674
bleu (*pytext.metrics.seq2seq_metrics.Seq2SeqMetrics attribute*), 552
BlockShardedTSV (*class in pytext.data.sources.tsv*), 427
BlockShardedTSVDataSource (*class in pytext.data.sources.tsv*), 427
BlockShardedTSVDataSourceTest (*class in pytext.data.test.tsv_data_source_test*), 435
BlockwiseMagnitudeSparsifier (*class in pytext.optimizer.sparsifiers.blockwise_sparsifier*), 707
bn_update() (*pytext.optimizer.swa.StochasticWeightAveraging static method*), 723
BOL (*pytext.common.constants.SpecialTokens attribute*), 402
BOS (*pytext.common.constants.SpecialTokens attribute*), 402
BottleneckPooling.Config (*class in pytext.models.representations.pooling*), 652
bracket_metrics (*pytext.metrics.intent_slot_metrics.AllMetrics attribute*), 546
BSeqCNNRepresentation (*class in pytext.models.representations.biseqcnn*), 645
build_class_priors() (*in module pytext.utils.loss*), 766
build_fairseq_vocab() (*in module pytext.data.bert_tensorizer*), 444
build_fp32_params() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16OptimizerMixin class method*), 712
build_from_data (*pytext.data.tensorizers.VocabConfig attribute*), 477
build_noise_sampler() (*in module pytext.models.r3f_models*), 700
build_positional_embedding() (*in module pytext.models.seq_models.positional*), 679
build_subclass_dict() (*in module pytext.config.serialize*), 414
build_tree() (*pytext.data.data_structures.annotation.Annotation method*), 415
build_vocab() (*pytext.fields.char_field.CharFeatureField method*), 499
build_vocab() (*pytext.fields.CharFeatureField method*), 503

build_vocab() (*pytext.fields.dict_field.DictFeatureField method*), 500
build_vocab() (*pytext.fields.DictFeatureField method*), 505
build_vocab() (*pytext.fields.text_field_with_special_unk.TextFeatureFieldWithSpecialUnk method*), 503
build_vocab() (*pytext.fields.TextFeatureFieldWithSpecialUnk method*), 507
BYTE_BOS (*pytext.common.constants.SpecialTokens attribute*), 402
BYTE_EOS (*pytext.common.constants.SpecialTokens attribute*), 402
byte_length() (*in module pytext.utils.data*), 762
BYTE_SPACE (*pytext.common.constants.SpecialTokens attribute*), 402
ByteTensorizer (*class in pytext.data.tensorizers*), 463
ByteTokensDocumentModel (*class in pytext.models.doc_model*), 689
ByteTokenTensorizer (*class in pytext.data.tensorizers*), 463

## C

cache() (*pytext.data.Data method*), 484
cache() (*pytext.data.data.Data method*), 445
cache_pretrained_embeddings() (*pytext.utils.embeddings.PretrainedEmbedding method*), 763
caffe2_export() (*pytext.models.BaseModel method*), 706
caffe2_export() (*pytext.models.doc_model.DocModel method*), 690
caffe2_export() (*pytext.models.joint_model.IntentSlotModel method*), 691
caffe2_export() (*pytext.models.language_models.lmlstm.LMLSTM method*), 595
caffe2_export() (*pytext.models.model.BaseModel method*), 693
caffe2_export() (*pytext.models.two_tower_classification_model.TwoTowerClassificationModel method*), 702
caffe2_export() (*pytext.models.TwoTowerClassificationModel method*), 706
calculate_error() (*in module pytext.metrics.calibration_metrics*), 545
calculate_feature_stats() (*pytext.torchscript.tensorizer.normalizer.VectorNormalizer method*), 741
calculate_feature_stats() (*pytext.torchscript.tensorizer.VectorNormalizer method*), 743
calculate_loss() (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method*), 525
calculate_loss() (*pytext.metric_reporters.language_model_metric_reporter.MaskedLM method*), 526
calculate_loss() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 540
calculate_loss() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
calculate_loss() (*pytext.metric_reporters.MetricReporter method*), 535
calculate_loss() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTaggingMetricReporter method*), 533
calculate_loss() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 542
calculate_metric() (*pytext.metric_reporters.calibration_metric_reporter.CalibrationMetricReporter method*), 513
calculate_metric() (*pytext.metric_reporters.CalibrationMetricReporter method*), 536
calculate_metric() (*pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method*), 518
calculate_metric() (*pytext.metric_reporters.classification_metric_reporter.MultiLabelClassificationMetricReporter method*), 519
calculate_metric() (*pytext.metric_reporters.ClassificationMetricReporter method*), 537
calculate_metric() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method*), 520
calculate_metric() (*pytext.metric_reporters.CompositionalMetricReporter method*), 542
calculate_metric() (*pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetrievalMetricReporter method*), 522
calculate_metric() (*pytext.metric_reporters.DenseRetrievalMetricReporter method*), 544
calculate_metric() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotDetectionMetricReporter*

*method*), 523
calculate_metric() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 539
calculate_metric() (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method*), 525
calculate_metric() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 540
calculate_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
calculate_metric() (*pytext.metric_reporters.metric_reporter.PureLossMetricReporter method*), 528
calculate_metric() (*pytext.metric_reporters.MetricReporter method*), 535
calculate_metric() (*pytext.metric_reporters.MultiLabelClassificationMetricReporter method*), 538
calculate_metric() (*pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter method*), 539
calculate_metric() (*pytext.metric_reporters.NERMetricReporter method*), 544
calculate_metric() (*pytext.metric_reporters.pairwise_ranking_metric_reporter.PairwiseRankingMetricReporter method*), 528
calculate_metric() (*pytext.metric_reporters.PairwiseRankingMetricReporter method*), 543
calculate_metric() (*pytext.metric_reporters.PureLossMetricReporter method*), 543
calculate_metric() (*pytext.metric_reporters.regression_metric_reporter.RegressionMetricReporter method*), 528
calculate_metric() (*pytext.metric_reporters.RegressionMetricReporter method*), 539
calculate_metric() (*pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositionalMetricReporter method*), 529
calculate_metric() (*pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter method*), 530
calculate_metric() (*pytext.metric_reporters.SequenceTaggingMetricReporter method*), 543
calculate_metric() (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter method*), 531
calculate_metric() (*pytext.metric_reporters.SquadMetricReporter method*), 541
calculate_metric() (*pytext.metric_reporters.text_tagging_metric_reporter.MultiLabelS method*), 532
calculate_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.NERMetricR method*), 532
calculate_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.SequenceTag method*), 532
calculate_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTagging method*), 533
calculate_metric() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 542
calculate_perplexity() (*pytext.models.output_layers.lm_output_layer.LMOutputLayer static method*), 605
calibration_metrics (*pytext.metrics.calibration_metrics.AllCalibrationMetrics attribute*), 544
CalibrationMetricReporter (*class in pytext.metric_reporters*), 536
CalibrationMetricReporter (*class in pytext.metric_reporters.calibration_metric_reporter*), 513
CalibrationMetrics (*class in pytext.metrics.calibration_metrics*), 544
camel_str() (*in module pytext.config.utils*), 414
causal (*pytext.config.module_config.CNNParams attribute*), 410
cell (*pytext.models.representations.augmented_lstm.AugmentedLSTMUni attribute*), 640
Channel (*class in pytext.metric_reporters*), 533
Channel (*class in pytext.metric_reporters.channel*), 514
channels (*pytext.metric_reporters.metric_reporter.MetricReporter attribute*), 526
channels (*pytext.metric_reporters.MetricReporter attribute*), 534
CHAR (*pytext.config.contextual_intent_slot.ModelInput attribute*), 407
char_embed (*pytext.models.embeddings.char_embedding.CharacterEmb attribute*), 572
char_embed (*pytext.models.embeddings.CharacterEmbedding attribute*), 586
char_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 407
CHAR_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 408
char_feat (*pytext.config.doc_classification.ModelInputConfig*

*attribute*), 408
char_feat (*pytext.config.field_config.FeatureConfig attribute*), 409
CHAR_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
char_offset_to_byte_offset() (*in module pytext.utils.data*), 762
character_tokenize() (*pytext.data.tensorizers.CharacterVocabTokenTensorizer method*), 465
CharacterEmbedding (*class in pytext.models.embeddings*), 585
CharacterEmbedding (*class in pytext.models.embeddings.char_embedding*), 572
characters (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 419
characters (*pytext.data.featurizer.OutputRecord attribute*), 420
CharacterTokenTensorizer (*class in pytext.data.tensorizers*), 464
CharacterVocabTokenTensorizer (*class in pytext.data.tensorizers*), 464
CharacterVocabTokenTensorizerScriptImpl (*class in pytext.data.tensorizers*), 465
CharacterVocabTokenTensorizerTest (*class in pytext.data.test.tensorizers_test*), 433
CharFeatConfig (*in module pytext.config.field_config*), 408
CharFeatureField (*class in pytext.fields*), 503
CharFeatureField (*class in pytext.fields.char_field*), 499
check_overflow() (*pytext.optimizer.fp16_optimizer.DynamicLossScaler method*), 713
check_overflow_() (*pytext.optimizer.fp16_optimizer.DynamicLossScaler method*), 714
check_state_keys() (*in module pytext.models.representations.transformer.sentence_encoder*), 628
check_valid() (*pytext.data.xlm_dictionary.Dictionary method*), 481
CheckpointManager (*class in pytext.task.serialize*), 728
checkTokenConfig() (*pytext.models.language_models.lmlstm.LMLSTM class method*), 595
children (*pytext.data.data_structures.node.Node attribute*), 417
children (*pytext.metrics.intent_slot_metrics.Node attribute*), 548
children_flat_str_spans() (*pytext.data.data_structures.annotation.Node method*), 415
chunk_file() (*in module pytext.utils.file_io*), 764
classification_metrics (*pytext.metrics.squad_metrics.SquadMetrics attribute*), 552
ClassificationMetricReporter (*class in pytext.metric_reporters*), 536
ClassificationMetricReporter (*class in pytext.metric_reporters.classification_metric_reporter*), 517
ClassificationMetrics (*class in pytext.metrics*), 553
ClassificationOutputLayer (*class in pytext.models.output_layers*), 616
ClassificationOutputLayer (*class in pytext.models.output_layers.doc_classification_output_layer*), 599
ClassificationScores (*class in pytext.models.output_layers.doc_classification_output_layer*), 600
clean_eos_bos() (*pytext.data.masked_util.TreeMask method*), 458
clip_grad_norm() (*pytext.optimizer.adabelief.AdaBelief method*), 712
clip_grad_norm() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16OptimizerMixin method*), 713
clip_grad_norm() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_MemoryEfficientFP16O method*), 713
clip_grad_norm() (*pytext.optimizer.fp16_optimizer.FP16Optimizer method*), 714
clip_grad_norm() (*pytext.optimizer.fp16_optimizer.FP16OptimizerApex method*), 714
clip_grad_norm() (*pytext.optimizer.fp16_optimizer.FP16OptimizerFairseq method*), 715
clip_grad_norm() (*pytext.optimizer.fp16_optimizer.MemoryEfficientFP16OptimizerFai method*), 716
clip_grad_norm() (*pytext.optimizer.madgrad.MADGRAD method*), 717
clip_grad_norm() (*pytext.optimizer.optimizers.Optimizer method*), 718
clip_list() (*in module pytext.torchscript.batchutils*), 746
clip_listlist() (*in module py-*

*text.torchscript.batchutils*), 746
clip_listlist_float() (*in module pytext.torchscript.batchutils*), 746
close() (*pytext.metric_reporters.Channel method*), 534
close() (*pytext.metric_reporters.channel.Channel method*), 514
close() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 516
CLS_TOKEN (*pytext.models.representations.transformer_sentence_encoder_base.PoolingMethod attribute*), 659
cls_vars() (*in module pytext.utils*), 770
CNNModel (*class in pytext.models.seq_models.conv_model*), 673
CNNParams (*class in pytext.config.module_config*), 410
COLUMN (*pytext.config.component.ComponentType attribute*), 403
column_schema (*pytext.data.bert_tensorizer.BERTTensorizerBase attribute*), 442
column_schema (*pytext.data.roberta_tensorizer.RoBERTaTokenLevelTensorizer attribute*), 459
column_schema (*pytext.data.Tensorizer attribute*), 492
column_schema (*pytext.data.tensorizers.AnnotationNumberizer attribute*), 462
column_schema (*pytext.data.tensorizers.ByteTensorizer attribute*), 463
column_schema (*pytext.data.tensorizers.ByteTokenTensorizer attribute*), 463
column_schema (*pytext.data.tensorizers.CharacterVocabTokenTensorizer attribute*), 465
column_schema (*pytext.data.tensorizers.Float1DListTensorizer attribute*), 466
column_schema (*pytext.data.tensorizers.FloatListSeqTensorizer attribute*), 467
column_schema (*pytext.data.tensorizers.FloatListTensorizer attribute*), 467
column_schema (*pytext.data.tensorizers.FloatTensorizer attribute*), 467
column_schema (*pytext.data.tensorizers.GazetteerTensorizer attribute*), 468
column_schema (*pytext.data.tensorizers.Integer1DListTensorizer attribute*), 469
column_schema (*pytext.data.tensorizers.LabelListRankTensorizer attribute*), 469
column_schema (*pytext.data.tensorizers.LabelListTensorizer attribute*), 470
column_schema (*pytext.data.tensorizers.LabelTensorizer attribute*), 470
column_schema (*pytext.data.tensorizers.NumericLabelTensorizer attribute*), 471
column_schema (*pytext.data.tensorizers.SeqTokenTensorizer attribute*), 472
column_schema (*pytext.data.tensorizers.SlotLabelTensorizer attribute*), 472
column_schema (*pytext.data.tensorizers.SoftLabelTensorizer attribute*), 473
column_schema (*pytext.data.tensorizers.String2DListTensorizer attribute*), 473
column_schema (*pytext.data.tensorizers.Tensorizer attribute*), 475
column_schema (*pytext.data.tensorizers.TokenTensorizer attribute*), 476
column_schema (*pytext.data.tensorizers.UidTensorizer attribute*), 477
column_schema (*pytext.data.token_tensorizer.ScriptBasedTokenTensorizer attribute*), 478
column_schema (*pytext.data.xlm_tensorizer.XLMTensorizer attribute*), 481
combine_pos_embed (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
combine_pos_embed (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute*), 671
combine_title_text_id() (*in module pytext.data.sources.dense_retrieval*), 425
CommonMetadata (*class in pytext.data*), 483
CommonMetadata (*class in pytext.data.data_handler*), 447
ComparableClassificationMetric (*class in pytext.metric_reporters.classification_metric_reporter*), 518
compare_frames() (*in module py-*

*text.metrics.intent_slot_metrics*), 548
compare_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
compare_metric() (*pytext.metric_reporters.MetricReporter method*), 535
Component (*class in pytext.config.component*), 403
component_config_type_from_type_name() (*in module pytext.config.serialize*), 414
ComponentMeta (*class in pytext.config.component*), 403
ComponentType (*class in pytext.config.component*), 403
compose_embedding() (*pytext.models.Model class method*), 705
compose_embedding() (*pytext.models.model.Model class method*), 695
CompositionalMetricReporter (*class in pytext.metric_reporters*), 542
CompositionalMetricReporter (*class in pytext.metric_reporters.compositional_metric_reporter*), 520
CompositionalNN (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 660
CompositionalSeq2SeqFileChannel (*class in pytext.metric_reporters.seq2seq_compositional*), 529
CompositionalSummationNN (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 660
compute_adaptive_loss() (*pytext.loss.AdaptiveRegularizer method*), 513
compute_adaptive_loss() (*pytext.loss.regularizer.AdaptiveRegularizer method*), 509
compute_all_metrics() (*in module pytext.metrics.intent_slot_metrics*), 549
compute_average_recall() (*in module pytext.metrics*), 559
compute_calibration() (*in module pytext.metrics.calibration_metrics*), 545
compute_classification_metrics() (*in module pytext.metrics*), 559
compute_dynamic_batch_size() (*pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher method*), 456
compute_dynamic_batch_size() (*pytext.data.dynamic_pooling_batcher.ExponentialDynamicPoolingBatcher method*), 457
compute_dynamic_batch_size() (*pytext.data.dynamic_pooling_batcher.LinearDynamicPoolingBatcher method*), 457
compute_dynamic_batch_size() (*pytext.data.DynamicPoolingBatcher method*), 489
compute_f1() (*in module pytext.metrics.seq2seq_metrics*), 552
compute_frame_accuracies_by_depth() (*in module pytext.metrics.intent_slot_metrics*), 550
compute_frame_accuracy() (*in module pytext.metrics.intent_slot_metrics*), 550
compute_frame_accuracy_top_k() (*in module pytext.metrics.intent_slot_metrics*), 550
compute_intent_slot_metrics() (*in module pytext.metrics.intent_slot_metrics*), 550
compute_language_model_metric() (*in module pytext.metrics.language_model_metrics*), 552
compute_macro_avg() (*in module pytext.metrics*), 560
compute_matthews_correlation_coefficients() (*in module pytext.metrics*), 560
compute_metric_at_k() (*in module pytext.metrics.intent_slot_metrics*), 551
compute_metrics() (*pytext.metrics.AllConfusions method*), 553
compute_metrics() (*pytext.metrics.Confusions method*), 554
compute_metrics() (*pytext.metrics.PerLabelConfusions method*), 558
compute_multi_label_classification_metrics() (*in module pytext.metrics*), 560
compute_multi_label_full_vector_classification_metr
(*in module pytext.metrics*), 561
compute_multi_label_multi_class_soft_metrics() (*in module pytext.metrics*), 562
compute_multi_label_soft_full_vector_metrics() (*in module pytext.metrics*), 562
compute_multi_label_soft_metrics() (*in module pytext.metrics*), 563
compute_pairwise_ranking_metrics() (*in module pytext.metrics*), 563
compute_prf1() (*in module pytext.metrics*), 563
compute_prf1_metrics() (*in module pytext.metrics.intent_slot_metrics*), 551
compute_regression_metrics() (*in module pytext.metrics*), 563
compute_roc_auc() (*in module pytext.metrics*), 563
compute_roc_auc_given_sorted_positives() (*in module pytext.metrics*), 564
compute_score() (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method*), 525
compute_scores() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 535

*text.metric_reporters.LanguageModelMetricReporter method*), 540
compute_soft_metrics() (*in module pytext.metrics*), 564
compute_symmetric_kl() (*in module pytext.models.r3f_models*), 700
compute_top_intent_accuracy() (*in module pytext.metrics.intent_slot_metrics*), 551
CONCAT (*pytext.config.module_config.SlotAttentionType attribute*), 410
concat (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 576
concat (*pytext.models.embeddings.EmbeddingList attribute*), 583
CONCAT (*pytext.models.seq_models.positional.PostionalEmbedCombine attribute*), 679
concat_layers (*pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN.Config attribute*), 250
config_from_json() (*in module pytext.config.serialize*), 414
config_to_json() (*in module pytext.config.serialize*), 414
ConfigBase (*class in pytext.config.pytext_config*), 411
ConfigBaseMeta (*class in pytext.config.pytext_config*), 411
ConfigParseError, 414
configs() (*pytext.config.component.Registry class method*), 404
Confusions (*class in pytext.metrics*), 553
confusions (*pytext.metrics.AllConfusions attribute*), 553
CoNLLUNERDataSource (*class in pytext.data.sources*), 430
CoNLLUNERDataSource (*class in pytext.data.sources.conllu*), 421
CoNLLUNERFile (*class in pytext.data.sources.conllu*), 421
CoNLLUPOSDataSource (*class in pytext.data.sources.conllu*), 421
ConsoleChannel (*class in pytext.metric_reporters.channel*), 514
CONTEXT_SEQUENCE (*pytext.common.constants.DFColumn attribute*), 400
CONTEXTUAL_TOKEN_EMBEDDING (*pytext.common.constants.DatasetFieldName attribute*), 401
CONTEXTUAL_TOKEN_EMBEDDING (*pytext.config.contextual_intent_slot.ModelInput attribute*), 407
contextual_token_embedding (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 407
CONTEXTUAL_TOKEN_EMBEDDING (*pytext.config.doc_classification.ModelInput attribute*), 408
contextual_token_embedding (*pytext.config.doc_classification.ModelInputConfig attribute*), 408
contextual_token_embedding (*pytext.config.field_config.FeatureConfig attribute*), 409
contextual_token_embedding (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 419
contextual_token_embedding (*pytext.data.featurizer.OutputRecord attribute*), 420
ContextualIntentSlotModel (*class in pytext.models.seq_models.contextual_intent_slot*), 667
ContextualIntentSlotRepresentation (*class in pytext.models.representations.contextual_intent_slot_rep*), 646
contextualize() (*pytext.models.BaseModel method*), 706
contextualize() (*pytext.models.disjoint_multitask_model.DisjointMultitaskModel method*), 688
contextualize() (*pytext.models.model.BaseModel method*), 693
contextualize() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserBase method*), 663
ContextualTokenEmbedding (*class in pytext.models.embeddings*), 587
ContextualTokenEmbedding (*class in pytext.models.embeddings.contextual_token_embedding*), 574
ContextualTokenEmbeddingConfig (*in module pytext.config.field_config*), 408
ContextualTokenEmbeddingField (*class in pytext.fields*), 504
ContextualTokenEmbeddingField (*class in pytext.fields.contextual_token_embedding_field*), 499
ContextualWordConvolution (*class in pytext.models.representations.biseqcnn*), 646
continue_training() (*pytext.trainers.Trainer method*), 758
continue_training() (*pytext.trainers.trainer.Trainer method*), 755
conv_and_pool() (*pytext.models.representations.docnn.DocNNRepresentation method*), 649
ConvDecoderConfig (*class in py-*

*text.models.seq_models.conv_decoder*), 668
ConvEncoderConfig (*class in pytext.models.seq_models.conv_encoder*), 671
convert_bio_to_spans() (*in module pytext.metric_reporters.word_tagging_metric_reporter*), 533
convert_caffe2_blob_name() (*in module pytext.utils.onnx*), 768
convert_generator() (*in module pytext.optimizer.fp16_optimizer*), 716
ConvLengthPredictionModule (*class in pytext.models.seq_models.nar_length*), 677
convs (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 572
convs (*pytext.models.embeddings.CharacterEmbedding attribute*), 586
copy() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.ParseState method*), 661
copy() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLSTM method*), 661
CosineAnnealingLR (*class in pytext.optimizer.scheduler*), 719
CosineEmbeddingLoss (*class in pytext.loss*), 511
CosineEmbeddingLoss (*class in pytext.loss.loss*), 508
CostFunctionType (*class in pytext.loss*), 512
CostFunctionType (*class in pytext.loss.structured_loss*), 510
CppProcessorMixin (*class in pytext.data.tokenizers*), 439
CppProcessorMixin (*class in pytext.data.tokenizers.tokenizer*), 437
cpu() (*pytext.models.distributed_model.DistributedModel method*), 688
cpu() (*pytext.models.language_models.lmlstm.LMLSTM method*), 595
create_component() (*in module pytext.config.component*), 404
create_context() (*in module pytext.utils.mobile_onnx*), 768
create_conv_package() (*in module pytext.models.representations.deepcnn*), 648
create_data_handler() (*in module pytext.config.component*), 404
create_decoder() (*pytext.models.doc_model.DocModel class method*), 690
create_embedding() (*pytext.models.doc_model.ByteTokensDocumentModel class method*), 689
create_embedding() (*pytext.models.doc_model.DocModel class method*), 690
create_embedding() (*pytext.models.joint_model.IntentSlotModel class method*), 691
create_embedding() (*pytext.models.Model class method*), 705
create_embedding() (*pytext.models.model.Model class method*), 695
create_embedding() (*pytext.models.qna.dr_qa.DrQAModel class method*), 622
create_embedding() (*pytext.models.seq_models.contextual_intent_slot.ContextualIntentSl class method*), 668
create_embedding() (*pytext.models.word_model.WordTaggingLiteModel class method*), 703
create_embedding() (*pytext.models.word_model.WordTaggingModel class method*), 703
create_encoder() (*in module pytext.config.component*), 404
create_featurizer() (*in module pytext.config.component*), 404
create_fields() (*in module pytext.fields*), 503
create_fields() (*in module pytext.fields.field*), 503
create_frame() (*in module pytext.metric_reporters.intent_slot_detection_metric_reporter*), 524
create_frame_prediction_pairs() (*pytext.metric_reporters.compositional_metric_reporter.Composition method*), 520
create_frame_prediction_pairs() (*pytext.metric_reporters.CompositionalMetricReporter method*), 542
create_frame_prediction_pairs() (*pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositio method*), 529
create_label_fields() (*in module pytext.fields*), 503
create_label_fields() (*in module pytext.fields.field*), 503
create_loss() (*in module pytext.config.component*), 404
create_metric_reporter() (*in module pytext.config.component*), 404
create_metric_reporter() (*pytext.task.tasks.PairwiseClassificationForDenseRetrievalTask class method*), 733
create_metric_reporter() (*pytext.task.tasks.RoBERTaNERTask class method*), 734
create_metric_reporter() (*pytext.task.tasks.WordTaggingTask class method*), 735
create_model() (*in module py-*

*text.config.component*), 404
create_module() (*in module pytext.models.module*), 696
create_optimizer() (*in module pytext.config.component*), 404
create_output_layer() (*pytext.models.doc_model.DocModel class method*), 690
create_parameter() (*in module pytext.config.config_adapter*), 405
create_predictor() (*in module pytext*), 773
create_predictor() (*in module pytext.config.component*), 404
create_privacy_engine() (*in module pytext.config.component*), 405
create_scheduler() (*in module pytext.config.component*), 405
create_schema() (*in module pytext.task.new_task*), 727
create_sparsifier() (*in module pytext.config.component*), 405
create_src_lengths_mask() (*in module pytext.models.seq_models.attention*), 666
create_sub_embs() (*pytext.models.Model class method*), 705
create_sub_embs() (*pytext.models.model.Model class method*), 695
create_task() (*in module pytext.task*), 737
create_task() (*in module pytext.task.task*), 731
create_tensorizers() (*in module pytext.task.new_task*), 727
create_trainer() (*in module pytext.config.component*), 405
create_vocab_index() (*in module pytext.utils.mobile_onnx*), 768
create_vocab_index() (*in module pytext.utils.onnx*), 768
create_vocab_indices_map() (*in module pytext.utils.mobile_onnx*), 768
create_vocab_indices_map() (*in module pytext.utils.onnx*), 769
CRF (*class in pytext.models.crf*), 687
CRF_L1_SoftThresholding (*class in pytext.optimizer.sparsifiers.sparsifier*), 709
CRF_MagnitudeThresholding (*class in pytext.optimizer.sparsifiers.sparsifier*), 709
CRF_SparsifierBase (*class in pytext.optimizer.sparsifiers.sparsifier*), 709
CRFOutputLayer (*class in pytext.models.output_layers*), 615
CRFOutputLayer (*class in pytext.models.output_layers.word_tagging_output_layer*), 611
CRFWordTaggingScores (*class in pytext.models.output_layers.word_tagging_output_layer*), 612
CrossEntropyLoss (*class in pytext.loss*), 511
CrossEntropyLoss (*class in pytext.loss.loss*), 508
current_model (*pytext.models.disjoint_multitask_model.DisjointMultitaskModel attribute*), 688
cycle() (*in module pytext.trainers.trainer*), 757
cycle() (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatchI class method*), 456
CyclicLR (*class in pytext.optimizer.scheduler*), 720

## D

Data (*class in pytext.data*), 483
Data (*class in pytext.data.data*), 444
data_dict (*pytext.data.disjoint_multitask_data.DisjointMultitaskData attribute*), 453
data_dict (*pytext.data.DisjointMultitaskData attribute*), 488
DATA_HANDLER (*pytext.config.component.ComponentType attribute*), 403
data_handlers (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH attribute*), 454
data_handlers (*pytext.data.DisjointMultitaskDataHandler attribute*), 488
DATA_SOURCE (*pytext.config.component.ComponentType attribute*), 403
DATA_SOURCE_TYPES (*pytext.data.sources.data_source.RootDataSource attribute*), 423
DATA_TYPE (*pytext.config.component.ComponentType attribute*), 403
DataHandler (*class in pytext.data*), 484
DataHandler (*class in pytext.data.data_handler*), 447
DatasetFieldName (*class in pytext.common.constants*), 401
DataSource (*class in pytext.data.sources*), 428
DataSource (*class in pytext.data.sources.data_source*), 422
DataTest (*class in pytext.data.test.data_test*), 431
debug_path (*pytext.config.pytext_config.PyTextConfig attribute*), 412
decision_thresh_at_precision (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556
decision_thresh_at_precision (*pytext.metrics.SoftClassificationMetrics attribute*), 559
decision_thresh_at_recall (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556

decision_thresh_at_recall (*pytext.metrics.SoftClassificationMetrics attribute*), 559
decode() (*pytext.data.tokenizers.GPT2BPETokenizer method*), 438
decode() (*pytext.data.tokenizers.Tokenizer method*), 439
decode() (*pytext.data.tokenizers.tokenizer.GPT2BPETokenizer method*), 437
decode() (*pytext.data.tokenizers.tokenizer.Tokenizer method*), 438
decode() (*pytext.models.crf.CRF method*), 687
decoder (*pytext.models.Model attribute*), 705
decoder (*pytext.models.model.Model attribute*), 695
decoder_embed_dim (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
decoder_input_dim (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
decoder_learned_pos (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
decoder_normalize_before (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
decoder_output_dim (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
DecoderBase (*class in pytext.models.decoders*), 569
DecoderBase (*class in pytext.models.decoders.decoder_base*), 565
DecoderBatchedStepEnsemble (*class in pytext.torchscript.seq2seq.decoder*), 738
DecoderWithLinearOutputProjection (*class in pytext.models.seq_models.projection_layers*), 680
DecoderWithLinearOutputProjection (*class in pytext.models.seq_models.rnn_decoder*), 681
DecoupledCNNModel (*class in pytext.models.seq_models.conv_model*), 673
DecoupledDecoderHead (*class in pytext.models.seq_models.projection_layers*), 680
DecoupledMultiheadAttention (*class in pytext.models.seq_models.attention*), 665
DeepCNNRepresentation (*class in pytext.models.representations.deepcnn*), 647
DEFAULT_LABEL_PAD_IDX (*pytext.common.constants.Padding attribute*), 402
DEFAULT_SCHEMA (*pytext.data.sources.dense_retrieval.DenseRetrievalDataSource attribute*), 424
DEFAULT_SCHEMA (*pytext.data.sources.DenseRetrievalDataSource attribute*), 430
DEFAULT_SCHEMA (*pytext.data.sources.squad.SquadDataSource attribute*), 426
DEFAULT_SCHEMA (*pytext.data.sources.SquadDataSource attribute*), 429
delay_unscale() (*in module pytext.utils.precision*), 769
delete_parameter() (*in module pytext.config.config_adapter*), 405
DENSE (*pytext.config.contextual_intent_slot.ModelInput attribute*), 407
dense (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAt attribute*), 642
dense (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttentio attribute*), 645
DENSE_FEAT (*pytext.common.constants.DFColumn attribute*), 400
dense_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 407
DENSE_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 408
dense_feat (*pytext.config.doc_classification.ModelInputConfig attribute*), 408
dense_feat (*pytext.config.field_config.FeatureConfig attribute*), 409
dense_feats (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 419
dense_feats (*pytext.data.featurizer.OutputRecord attribute*), 420
DENSE_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
DenseFeatureExporter (*class in pytext.exporters*), 497
DenseFeatureExporter (*class in pytext.exporters.custom_exporters*), 493
DenseRetrievalDataSource (*class in pytext.data.sources*), 430
DenseRetrievalDataSource (*class in pytext.data.sources.dense_retrieval*), 424
DenseRetrievalMetricNames (*class in pytext.metric_reporters.dense_retrieval_metric_reporter*), 521
DenseRetrievalMetricReporter (*class in pytext.metric_reporters*), 544
DenseRetrievalMetricReporter (*class in pytext.metric_reporters.dense_retrieval_metric_reporter*), 521
DenseRetrievalMetrics (*class in py-*

*text.metrics.dense_retrieval_metrics*), 545
DenseRetrievalOutputLayer (*class in pytext.models.output_layers*), 620
DenseRetrievalOutputLayer (*class in pytext.models.output_layers.distance_output_layer*), 597
deprecate() (*in module pytext.config.config_adapter*), 405
deprecation_warning() (*in module pytext.torchscript.module*), 753
depth() (*pytext.data.data_structures.annotation.Tree method*), 416
destructure_any_list() (*in module pytext.torchscript.batchutils*), 746
destructure_dict_list() (*in module pytext.torchscript.batchutils*), 746
destructure_dictlist_list() (*in module pytext.torchscript.batchutils*), 746
destructure_tensor() (*in module pytext.torchscript.batchutils*), 746
destructure_tensor_list() (*in module pytext.torchscript.batchutils*), 746
detach() (*pytext.optimizer.privacy_engine.PrivacyEngine method*), 719
device() (*in module pytext.utils.cuda*), 761
DFColumn (*class in pytext.common.constants*), 400
DICT (*pytext.config.contextual_intent_slot.ModelInput attribute*), 407
DICT_FEAT (*pytext.common.constants.DFColumn attribute*), 400
dict_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 407
DICT_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 408
dict_feat (*pytext.config.doc_classification.ModelInputConfig attribute*), 408
dict_feat (*pytext.config.field_config.FeatureConfig attribute*), 409
DICT_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
dict_zip() (*in module pytext.workflow*), 771
DictEmbedding (*class in pytext.models.embeddings*), 584
DictEmbedding (*class in pytext.models.embeddings.dict_embedding*), 574
DictFeatConfig (*in module pytext.config.field_config*), 408
DictFeatureField (*class in pytext.fields*), 505
DictFeatureField (*class in pytext.fields.dict_field*), 500
Dictionary (*class in pytext.data.xlm_dictionary*), 481
dilated (*pytext.config.module_config.CNNParams attribute*), 410
dim (*pytext.config.field_config.FloatVectorConfig attribute*), 409
dim_error_check (*pytext.config.field_config.FloatVectorConfig attribute*), 409
dimension() (*pytext.utils.lazy.Infer class method*), 764
disable_cuda() (*pytext.utils.config_utils.MockConfigLoader method*), 761
disable_sort_in_jit (*pytext.models.representations.bilstm.BiLSTM.Config attribute*), 228
DisjointMultitask (*class in pytext.task.disjoint_multitask*), 726
DisjointMultitaskData (*class in pytext.data*), 487
DisjointMultitaskData (*class in pytext.data.disjoint_multitask_data*), 453
DisjointMultitaskDataHandler (*class in pytext.data*), 488
DisjointMultitaskDataHandler (*class in pytext.data.disjoint_multitask_data_handler*), 454
DisjointMultitaskMetricReporter (*class in pytext.metric_reporters.disjoint_multitask_metric_reporter*), 522
DisjointMultitaskModel (*class in pytext.models.disjoint_multitask_model*), 687
dist_init() (*in module pytext.utils.distributed*), 762
distributed_world_size (*pytext.config.pytext_config.PyTextConfig attribute*), 412
DistributedModel (*class in pytext.models.distributed_model*), 688
doc_attention (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDo attribute*), 643
DOC_COLUMN (*pytext.metric_reporters.squad_metric_reporter.SquadMetri attribute*), 531
DOC_COLUMN (*pytext.metric_reporters.SquadMetricReporter attribute*), 541
doc_decoder (*pytext.models.decoders.intent_slot_model_decoder.IntentS attribute*), 566
doc_decoder (*pytext.models.decoders.IntentSlotModelDecoder attribute*), 571
DOC_LABEL (*pytext.common.constants.DFColumn attribute*), 400
DOC_LABEL (*pytext.config.field_config.Target attribute*), 409
DOC_LABEL_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
doc_model_deprecated() (*in module py-*

text.config.config_adapter), 405
doc_output (pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer attribute), 603
DOC_WEIGHT (pytext.common.constants.DFColumn attribute), 400
DOC_WEIGHT (pytext.config.contextual_intent_slot.ExtraField attribute), 407
DOC_WEIGHT_FIELD (pytext.common.constants.DatasetFieldName attribute), 401
DocLabelConfig (class in pytext.config.field_config), 408
DocLabelField (class in pytext.fields), 505
DocLabelField (class in pytext.fields.field), 501
DocModel (class in pytext.models.doc_model), 690
DocNNRepresentation (class in pytext.models.representations.docnn), 649
DocRegressionModel (class in pytext.models.doc_model), 690
DocumentClassificationTask (class in pytext.task.tasks), 732
DocumentRegressionTask (class in pytext.task.tasks), 732
DoNothingTokenizer (class in pytext.data.tokenizers), 439
DoNothingTokenizer (class in pytext.data.tokenizers.tokenizer), 437
DOT (pytext.config.module_config.SlotAttentionType attribute), 410
DotAttention (class in pytext.models.seq_models.attention), 665
DotProductSelfAttention (class in pytext.models.representations.attention), 637
downgrade_one_version() (in module pytext.config.config_adapter), 405
dropout (pytext.models.representations.augmented_lstm.AugmentedLSTM.Config attribute), 226
dropout (pytext.models.representations.bilstm.BiLSTM attribute), 641
dropout (pytext.models.representations.bilstm.BiLSTM.Config attribute), 228
dropout (pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute), 642
dropout (pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention.Config attribute), 229
dropout (pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute), 643
dropout (pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttention attribute), 645
dropout (pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN attribute), 657
dropout (pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN.Config attribute), 250
dropout (pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute), 668
dropout (pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute), 671
DrQAModel (class in pytext.models.qna.dr_qa), 622
dummy_model_input (pytext.exporters.exporter.ModelExporter attribute), 494
dummy_model_input (pytext.exporters.ModelExporter attribute), 496
dummy_model_input (pytext.fields.char_field.CharFeatureField attribute), 499
dummy_model_input (pytext.fields.CharFeatureField attribute), 503
dummy_model_input (pytext.fields.dict_field.DictFeatureField attribute), 500
dummy_model_input (pytext.fields.DictFeatureField attribute), 505
dummy_model_input (pytext.fields.field.SeqFeatureField attribute), 502
dummy_model_input (pytext.fields.field.TextFeatureField attribute), 502
dummy_model_input (pytext.fields.SeqFeatureField attribute), 507
dummy_model_input (pytext.fields.TextFeatureField attribute), 506
dump_raw_input (pytext.config.pytext_config.LogitsConfig attribute), 411
DynamicLossScaler (class in pytext.optimizer.fp16_optimizer), 713
DynamicPoolingBatcher (class in pytext.data), 489
DynamicPoolingBatcher (class in pytext.data.dynamic_pooling_batcher), 456
DynamicPoolingBatcherTest (class in pytext.data.test.dynamic_pooling_batcher_test), 432


## E


early_stop_after (pytext.trainers.Trainer attribute), 758
early_stop_after (pytext.trainers.trainer.Trainer attribute), 755
elapsed_time (pytext.utils.meter.TimeMeter attribute), 768
Element (class in pytext.models.semantic_parsers.rnng.rnng_data_structures), 666
element_from_top() (pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLS

*method*), 661
embedding (*pytext.models.Model attribute*), 705
embedding (*pytext.models.model.Model attribute*), 695
embedding() (*pytext.models.semantic_parsers.rnng.rnng_data.StackLSTM method*), 661
embedding_dim (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 573
embedding_dim (*pytext.models.embeddings.CharacterEmbedding attribute*), 586
embedding_dim (*pytext.models.embeddings.embedding_base.EmbeddingBase attribute*), 576
embedding_dim (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 576
embedding_dim (*pytext.models.embeddings.EmbeddingBase attribute*), 582
embedding_dim (*pytext.models.embeddings.EmbeddingList attribute*), 583
embedding_dim (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute*), 671
EmbeddingBase (*class in pytext.models.embeddings*), 582
EmbeddingBase (*class in pytext.models.embeddings.embedding_base*), 576
EmbeddingList (*class in pytext.models.embeddings*), 582
EmbeddingList (*class in pytext.models.embeddings.embedding_list*), 576
EMBEDDINGS (*pytext.models.pair_classification_model.PairwiseModel attribute*), 697
EmbedInitStrategy (*class in pytext.config.field_config*), 408
EmbedQuantizeType (*class in pytext.models.seq_models.mask_generator*), 674
encode_relations (*pytext.models.pair_classification_model.PairwiseModel.Config attribute*), 210
encoder_embed_dim (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute*), 671
encoder_learned_pos (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute*), 671
encoder_normalize_before (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute*), 671
EncoderEnsemble (*class in pytext.torchscript.seq2seq.encoder*), 738
end (*pytext.data.structures.structures.node.Span attribute*), 418
end (*pytext.data.tokenizers.Token attribute*), 438
end (*pytext.data.tokenizers.tokenizer.Token attribute*), 437
end (*pytext.metric_reporters.word_tagging_metric_reporter.Span attribute*), 533
end_batch_size (*pytext.data.dynamic_pooling_batcher.BatcherSchedulerConfig attribute*), 456
end_of_scheduler() (*pytext.data.test.dynamic_pooling_batcher_test.DynamicPoolingBatc method*), 432
ensemble_task_deprecated() (*in module pytext.config.config_adapter*), 405
EnsembleModel (*class in pytext.models.ensembles*), 593
EnsembleModel (*class in pytext.models.ensembles.ensemble*), 591
EnsembleTask (*class in pytext.task.tasks*), 732
EnsembleTrainer (*class in pytext.trainers*), 760
EnsembleTrainer (*class in pytext.trainers.ensemble_trainer*), 754
EntropyRegularizer (*class in pytext.loss*), 513
EntropyRegularizer (*class in pytext.loss.regularizer*), 509
EnumTypeError, 414
EOL (*pytext.common.constants.SpecialTokens attribute*), 402
EOS (*pytext.common.constants.SpecialTokens attribute*), 402
EOS (*pytext.config.module_config.PerplexityType attribute*), 410
EOS_MODEL (*pytext.common.constants.VocabMeta attribute*), 403
EOS_TOKEN (*pytext.common.constants.VocabMeta attribute*), 403
epoch (*pytext.trainers.training_state.TrainingState attribute*), 757
epoch (*pytext.trainers.TrainingState attribute*), 759
epoch_period (*pytext.data.dynamic_pooling_batcher.BatcherSchedulerC attribute*), 456
epochs (*pytext.trainers.Trainer attribute*), 757
epochs (*pytext.trainers.trainer.Trainer attribute*), 755
epochs_since_last_improvement (*pytext.trainers.training_state.TrainingState attribute*), 757
epochs_since_last_improvement (*pytext.trainers.TrainingState attribute*), 759
eprint() (*in module pytext.common.utils*), 403
EpsilonOptions (*pytext.models.r3f_models.R3FConfigOptions at-*

*tribute*), 699
escape_brackets() (*in module pytext.data.data_structures.annotation*), 417
EVAL (*pytext.common.constants.Stage attribute*), 402
eval (*pytext.data.sources.data_source.DataSource attribute*), 422
eval (*pytext.data.sources.data_source.RootDataSource attribute*), 423
eval (*pytext.data.sources.DataSource attribute*), 428
eval (*pytext.data.sources.dense_retrieval.DenseRetrievalDataSource attribute*), 424
eval (*pytext.data.sources.DenseRetrievalDataSource attribute*), 430
eval (*pytext.data.sources.squad.SquadDataSource attribute*), 426
eval (*pytext.data.sources.SquadDataSource attribute*), 429
eval (*pytext.data.sources.tsv.MultilingualTSVDataSource attribute*), 427
eval() (*pytext.models.BaseModel method*), 706
eval() (*pytext.models.distributed_model.DistributedModel method*), 689
eval() (*pytext.models.model.BaseModel method*), 694
eval_batch_size (*pytext.data.data_handler.DataHandler attribute*), 448
eval_batch_size (*pytext.data.DataHandler attribute*), 485
eval_path (*pytext.data.data_handler.DataHandler attribute*), 448
eval_path (*pytext.data.DataHandler attribute*), 485
EvalBatchSampler (*class in pytext.data*), 489
EvalBatchSampler (*class in pytext.data.batch_sampler*), 440
exact_match (*pytext.metrics.seq2seq_metrics.Seq2SeqMetrics attribute*), 552
exact_matches (*pytext.metrics.squad_metrics.SquadMetrics attribute*), 552
example_config() (*pytext.task.tasks.EnsembleTask class method*), 732
expected_error (*pytext.metrics.calibration_metrics.CalibrationMetrics attribute*), 544
expected_frame (*pytext.metrics.intent_slot_metrics.FramePredictionPair attribute*), 547
expected_label (*pytext.metrics.LabelListPrediction attribute*), 554
expected_label (*pytext.metrics.LabelPrediction attribute*), 554
expected_nodes (*pytext.metrics.intent_slot_metrics.NodesPredictionPair attribute*), 548
expected_numberized (*pytext.data.test.tensorizers_test.String2DListTensorizerTest attribute*), 434
expected_tensorized (*pytext.data.test.tensorizers_test.String2DListTensorizerTest attribute*), 434
ExponentialBatcherSchedulerConfig (*class in pytext.data.dynamic_pooling_batcher*), 457
ExponentialDynamicPoolingBatcher (*class in pytext.data.dynamic_pooling_batcher*), 457
ExponentialLR (*class in pytext.optimizer.scheduler*), 720
export (*pytext.config.pytext_config.PyTextConfig attribute*), 412
export() (*pytext.metric_reporters.Channel method*), 534
export() (*pytext.metric_reporters.channel.Channel method*), 514
export() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 516
export() (*pytext.task.disjoint_multitask.DisjointMultitask method*), 726
export() (*pytext.task.disjoint_multitask.NewDisjointMultitask method*), 726
export() (*pytext.task.task.TaskBase method*), 730
export() (*pytext.task.TaskBase method*), 736
export_caffe2_path (*pytext.config.pytext_config.ExportConfig attribute*), 411
export_caffe2_path (*pytext.config.pytext_config.PyTextConfig attribute*), 412
export_check() (*pytext.config.pytext_config.PyTextConfig method*), 412
export_input_names (*pytext.config.field_config.FloatVectorConfig attribute*), 409
export_list (*pytext.config.pytext_config.PyTextConfig attribute*), 412
export_lite_path (*pytext.config.pytext_config.ExportConfig attribute*), 411
export_nets_to_predictor_file() (*in module pytext.utils.onnx*), 769
export_onnx_path (*pytext.config.pytext_config.ExportConfig attribute*), 411
export_onnx_path (*pytext.config.pytext_config.PyTextConfig attribute*), 412
export_output_names (*pytext.config.field_config.DocLabelConfig attribute*), 408

export_output_names (*pytext.config.field_config.WordLabelConfig attribute*), 409
export_saved_model_to_caffe2() (*in module pytext.workflow*), 771
export_saved_model_to_torchscript() (*in module pytext.workflow*), 771
export_to_caffe2() (*pytext.exporters.custom_exporters.InitPredictNetExporter method*), 493
export_to_caffe2() (*pytext.exporters.exporter.ModelExporter method*), 494
export_to_caffe2() (*pytext.exporters.InitPredictNetExporter method*), 498
export_to_caffe2() (*pytext.exporters.ModelExporter method*), 496
export_to_caffe2() (*pytext.models.crf.CRF method*), 687
export_to_caffe2() (*pytext.models.output_layers.CRFOutputLayer method*), 615
export_to_caffe2() (*pytext.models.output_layers.doc_classification_output_layer.BinaryClassificationOutputLayer method*), 599
export_to_caffe2() (*pytext.models.output_layers.doc_classification_output_layer.MultiClassificationOutputLayer method*), 601
export_to_caffe2() (*pytext.models.output_layers.doc_classification_output_layer.MultiLabelOutputLayer method*), 600
export_to_caffe2() (*pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer method*), 603
export_to_caffe2() (*pytext.models.output_layers.multi_label_classification_layer.MultiLabelClassificationLayer method*), 606
export_to_caffe2() (*pytext.models.output_layers.output_layer_base.OutputLayerBase method*), 607
export_to_caffe2() (*pytext.models.output_layers.OutputLayerBase method*), 613
export_to_caffe2() (*pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer method*), 611
export_to_caffe2() (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method*), 612
export_to_caffe2() (*pytext.models.output_layers.WordTaggingOutputLayer method*), 617
export_to_metrics() (*pytext.exporters.exporter.ModelExporter method*), 495
export_to_metrics() (*pytext.exporters.ModelExporter method*), 496
export_torchscript_path (*pytext.config.pytext_config.ExportConfig attribute*), 411
export_torchscript_path (*pytext.config.pytext_config.PyTextConfig attribute*), 412
ExportConfig (*class in pytext.config.pytext_config*), 411
EXPORTER (*pytext.config.component.ComponentType attribute*), 403
ExporterType (*class in pytext.config.module_config*), 410
ExportType (*class in pytext.models.decoders.mlp_decoder_two_tower*), 568
extra_fields (*pytext.data.data_handler.DataHandler attribute*), 448
extra_fields (*pytext.data.DataHandler attribute*), 485
extra_repr() (*pytext.models.seq_models.conv_decoder.LightConvDecoderModel method*), 670
extra_repr() (*pytext.models.seq_models.conv_encoder.LightConvEncoder method*), 671
extra_repr() (*pytext.models.seq_models.conv_encoder.LightConvEncoderModel method*), 672
extra_repr() (*pytext.models.seq_models.light_conv.LightweightConvDecoderLayer method*), 673
extract_beam_subtrees() (*in module pytext.metric_reporters.compositional_utils*), 501
extract_features() (*pytext.models.representations.transformer.PostEncoder method*), 626
extract_features() (*pytext.models.representations.transformer.sentence_encoder.PostEncoder method*), 627
extract_features() (*pytext.models.representations.transformer.sentence_encoder.SentenceEncoder method*), 628
extract_features() (*pytext.models.representations.transformer.SentenceEncoder method*), 634
extract_iterator_properties() (*in module pytext.data.batch_sampler*), 441
extract_tagging_output_lab() (*in module pytext.models.seq_models.utils*), 685
extract_subtree() (*in module pytext.metric_reporters.compositional_utils*), 521
ExtraField (*class in py-*

*text.config.contextual_intent_slot*), 407
ExtraField (*class in pytext.config.doc_classification*), 408
ExtraField (*class in pytext.config.pair_classification*), 411

## F

f1 (*pytext.metrics.MacroPRF1Scores attribute*), 555
f1 (*pytext.metrics.PRF1Scores attribute*), 557
f1 (*pytext.metrics.seq2seq_metrics.Seq2SeqMetrics attribute*), 552
f1_score (*pytext.metrics.squad_metrics.SquadMetrics attribute*), 552
Fairseq_FP16OptimizerMixin (*class in pytext.optimizer.fairseq_fp16_utils*), 712
Fairseq_MemoryEfficientFP16OptimizerMixin (*class in pytext.optimizer.fairseq_fp16_utils*), 713
false_negatives (*pytext.metrics.PRF1Scores attribute*), 556, 557
false_positives (*pytext.metrics.PRF1Scores attribute*), 556, 557
false_postives_upper_bound() (*in module pytext.utils.loss*), 766
FeatureConfig (*class in pytext.config.field_config*), 409
features (*pytext.data.data_handler.DataHandler attribute*), 448
features (*pytext.data.DataHandler attribute*), 485
featurize() (*pytext.data.featurizer.Featurizer method*), 420
featurize() (*pytext.data.featurizer.featurizer.Featurizer method*), 418
featurize() (*pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method*), 419
featurize() (*pytext.data.featurizer.SimpleFeaturizer method*), 421
featurize_batch() (*pytext.data.featurizer.Featurizer method*), 420
featurize_batch() (*pytext.data.featurizer.featurizer.Featurizer method*), 418
featurize_batch() (*pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method*), 419
featurize_batch() (*pytext.data.featurizer.SimpleFeaturizer method*), 421
Featurizer (*class in pytext.data.featurizer*), 420
Featurizer (*class in pytext.data.featurizer.featurizer*), 418
FEATURIZER (*pytext.config.component.ComponentType attribute*), 403
featurizer (*pytext.data.data_handler.DataHandler attribute*), 447
featurizer (*pytext.data.DataHandler attribute*), 485
Field (*class in pytext.fields*), 505
Field (*class in pytext.fields.field*), 501
field_names (*pytext.config.pytext_config.TestConfig attribute*), 413
FieldMeta (*class in pytext.fields*), 506
FieldMeta (*class in pytext.fields.field*), 501
FileChannel (*class in pytext.metric_reporters.channel*), 515
filepath (*pytext.data.tensorizers.VocabFileConfig attribute*), 477
filter_criteria() (*pytext.utils.embeddings.PretrainedEmbedding method*), 763
filter_invalid_beams() (*in module pytext.metric_reporters.compositional_utils*), 521
finalize() (*pytext.optimizer.fp16_optimizer.FP16Optimizer method*), 714
finalize() (*pytext.optimizer.fp16_optimizer.FP16OptimizerDeprecated method*), 715
finalize() (*pytext.optimizer.optimizers.Optimizer method*), 718
finalize() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 723
finalize_tree() (*pytext.data.data_structures.annotation.TreeBuilder method*), 417
find_and_replace() (*pytext.models.embeddings.dict_embedding.DictEmbedding method*), 575
find_and_replace() (*pytext.models.embeddings.DictEmbedding method*), 585
find_config_class() (*in module pytext.utils.documentation*), 763
find_dicts_containing_key() (*in module pytext.config.config_adapter*), 405
find_param() (*in module pytext.config.utils*), 414
find_parameter() (*in module pytext.config.config_adapter*), 405
find_params_to_prune() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 710
finished() (*pytext.models.semantic_parsers.rnng.rnng_data_structures method*), 661
finished_dynamic() (*pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher method*), 457
finished_dynamic() (*pytext.data.dynamic_pooling_batcher.ExponentialDynamicPoolingB method*), 457

finished_dynamic() (pytext.data.DynamicPoolingBatcher method), 489
fix_fl_local_optimizer_and_trainer() (in module pytext.config.config_adapter), 405
fix_paths() (pytext.utils.config_utils.MockConfigLoader method), 761
fixed_generation_vocab_expanded (pytext.models.seq_models.projection_layers.DecoupledDecoderHead attribute), 680
flat_str() (pytext.data.data_structures.annotation.Node method), 415
flat_str() (pytext.data.data_structures.annotation.Tree method), 416
flatten_deprecated_ensemble_config() (in module pytext.config.config_adapter), 405
Float1DListTensorizer (class in pytext.data.tensorizers), 466
float_tensor_list1D() (in module pytext.torchscript.utils), 753
FloatField (class in pytext.fields), 506
FloatField (class in pytext.fields.field), 501
FloatListSeqTensorizer (class in pytext.data.tensorizers), 466
FloatListTensorizer (class in pytext.data.tensorizers), 467
FloatTensor() (in module pytext.utils.cuda), 761
FloatTensorizer (class in pytext.data.tensorizers), 467
FloatVectorConfig (class in pytext.config.field_config), 409
FloatVectorField (class in pytext.fields), 506
FloatVectorField (class in pytext.fields.field), 501
FN (pytext.metrics.Confusions attribute), 554
force_print() (in module pytext.utils.distributed), 762
format_prediction() (pytext.task.task.TaskBase class method), 731
format_prediction() (pytext.task.TaskBase class method), 736
format_prediction() (pytext.task.tasks.DocumentClassificationTask class method), 732
format_time() (in module pytext.utils.timing), 770
forward() (pytext.data.bert_tensorizer.BERTTensorizerBaseScriptImpl method), 443
forward() (pytext.data.tensorizers.CharacterVocabTokenTensorizerScriptImpl method), 465
forward() (pytext.data.tensorizers.String2DListTensorizerScriptImpl method), 474
forward() (pytext.data.token_tensorizer.TokenTensorizerScriptImpl method), 479
forward() (pytext.data.xlm_tensorizer.XLMTensorizerScriptImpl method), 482
forward() (pytext.loss.AUCPRHingeLoss method), 511
forward() (pytext.loss.loss.AUCPRHingeLoss method), 507
forward() (pytext.models.crf.CRF method), 687
forward() (pytext.models.decoders.decoder_base.DecoderBase method), 565
forward() (pytext.models.decoders.DecoderBase method), 570
forward() (pytext.models.decoders.intent_slot_model_decoder.IntentSlot method), 566
forward() (pytext.models.decoders.IntentSlotModelDecoder method), 571
forward() (pytext.models.decoders.mlp_decoder.MLPDecoder method), 567
forward() (pytext.models.decoders.mlp_decoder_query_response.MLPD method), 568
forward() (pytext.models.decoders.mlp_decoder_two_tower.MLPDecod method), 568
forward() (pytext.models.decoders.MLPDecoder method), 570
forward() (pytext.models.decoders.multilabel_decoder.MultiLabelDecod method), 569
forward() (pytext.models.disjoint_multitask_model.DisjointMultitaskMo method), 688
forward() (pytext.models.doc_model.PersonalizedDocModel method), 691
forward() (pytext.models.embeddings.char_embedding.CharacterEmbed method), 573
forward() (pytext.models.embeddings.char_embedding.Highway method), 573
forward() (pytext.models.embeddings.CharacterEmbedding method), 586
forward() (pytext.models.embeddings.contextual_token_embedding.Con method), 574
forward() (pytext.models.embeddings.ContextualTokenEmbedding method), 587
forward() (pytext.models.embeddings.dict_embedding.DictEmbedding method), 575
forward() (pytext.models.embeddings.DictEmbedding method), 585
forward() (pytext.models.embeddings.embedding_list.EmbeddingList method), 576
forward() (pytext.models.embeddings.EmbeddingList method), 583
forward() (pytext.models.embeddings.mlp_embedding.MLPEmbeddingScriptImpl method), 577
forward() (pytext.models.embeddings.MLPEmbedding method), 588
forward() (pytext.models.embeddings.scriptable_embedding_list.Scripta method), 578
forward() (pytext.models.embeddings.scriptable_embedding_list.Scripta method), 578
forward() (pytext.models.embeddings.scriptable_embedding_list.Scripta

*method*), 578
forward() (*pytext.models.embeddings.word_embedding.WordEmbedding method*), 579
forward() (*pytext.models.embeddings.word_seq_embedding.WordSeqEmbedding method*), 581
forward() (*pytext.models.embeddings.WordEmbedding method*), 584
forward() (*pytext.models.embeddings.WordSeqEmbedding method*), 587
forward() (*pytext.models.ensembles.bagging_doc_ensemble.BaggingDocEnsembleModel method*), 589
forward() (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsembleModel method*), 590
forward() (*pytext.models.ensembles.BaggingDocEnsembleModel method*), 592
forward() (*pytext.models.ensembles.BaggingIntentSlotEnsembleModel method*), 592
forward() (*pytext.models.ensembles.ensemble.EnsembleModel method*), 591
forward() (*pytext.models.ensembles.EnsembleModel method*), 593
forward() (*pytext.models.language_models.lmlstm.LMLSTM method*), 595
forward() (*pytext.models.masked_lm.MaskedLanguageModel method*), 692
forward() (*pytext.models.Model method*), 705
forward() (*pytext.models.model.Model method*), 696
forward() (*pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer method*), 604
forward() (*pytext.models.output_layers.multi_label_classification_output_layer.MultiLabelClassificationOutputLayer method*), 606
forward() (*pytext.models.output_layers.word_tagging_output_layer.CRFWordTaggingOutputLayer method*), 612
forward() (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method*), 613
forward() (*pytext.models.pair_classification_model.BasePairwiseModel method*), 697
forward() (*pytext.models.pair_classification_model.PairwiseModel method*), 698
forward() (*pytext.models.qna.bert_squad_qa.BertSquadQAModel method*), 622
forward() (*pytext.models.qna.dr_qa.DrQAModel method*), 622
forward() (*pytext.models.query_document_pairwise_ranking.QueryDocumentPairwiseRankingModel method*), 698
forward() (*pytext.models.r3f_models.R3FPyTextMixin method*), 699
forward() (*pytext.models.representations.attention.DotProductSelfAttention method*), 637
forward() (*pytext.models.representations.attention.MultiplicativeAttention method*), 637
forward() (*pytext.models.representations.attention.SequenceAlignedAttention method*), 637
forward() (*pytext.models.representations.augmented_lstm.AugmentedLSTM method*), 638
forward() (*pytext.models.representations.augmented_lstm.AugmentedLSTMCell method*), 639
forward() (*pytext.models.representations.augmented_lstm.AugmentedLSTMStack method*), 640
forward() (*pytext.models.representations.bilstm.BiLSTM method*), 641
forward() (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention method*), 642
forward() (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention method*), 644
forward() (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttn method*), 645
forward() (*pytext.models.representations.biseqcnn.BSeqCNNRepresentation method*), 646
forward() (*pytext.models.representations.biseqcnn.ContextualWordConv method*), 646
forward() (*pytext.models.representations.contextual_intent_slot_rep.ContextualIntentSlotRep method*), 647
forward() (*pytext.models.representations.deepcnn.DeepCNNRepresentation method*), 647
forward() (*pytext.models.representations.deepcnn.SeparableConv1d method*), 648
forward() (*pytext.models.representations.deepcnn.Trim1d method*), 648
forward() (*pytext.models.representations.docnn.DocNNRepresentation method*), 649
forward() (*pytext.models.representations.jointcnn_rep.JointCNNRepresentation method*), 650
forward() (*pytext.models.representations.jointcnn_rep.SharedCNNRepresentation method*), 650
forward() (*pytext.models.representations.ordered_neuron_lstm.OrderedNeuronLSTM method*), 651
forward() (*pytext.models.representations.ordered_neuron_lstm.OrderedNeuronLSTMStack method*), 651
forward() (*pytext.models.representations.pair_rep.PairRepresentation method*), 652
forward() (*pytext.models.representations.pass_through.PassThroughRep method*), 652
forward() (*pytext.models.representations.pooling.BoundaryPool method*), 652
forward() (*pytext.models.representations.pooling.LastTimestepPool method*), 653
forward() (*pytext.models.representations.pooling.MaxPool method*), 653
forward() (*pytext.models.representations.pooling.MeanPool method*), 653
forward() (*pytext.models.representations.pooling.NoPool method*), 653
forward() (*pytext.models.representations.pooling.SelfAttention method*), 654
forward() (*pytext.models.representations.pure_doc_attention.PureDocAttention method*), 654
forward() (*pytext.models.representations.representation_base.RepresentationBase*

*method*), 654
forward() (*pytext.models.representations.seq_rep.SeqRepresentation*
*method*), 655
forward() (*pytext.models.representations.slot_attention.SlotAttention*
*method*), 655
forward() (*pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN*
*method*), 657
forward() (*pytext.models.representations.traced_transformer_encoder.TracedBertSelfAttention*
*method*), 657
forward() (*pytext.models.representations.traced_transformer_encoder.TracedTransformerEncoder*
*method*), 658
forward() (*pytext.models.representations.transformer.multihead_attention.MultiheadSelfAttention*
*method*), 623
forward() (*pytext.models.representations.transformer.multihead_linear_attention.MultiheadLinearAttention*
*method*), 624
forward() (*pytext.models.representations.transformer.MultiheadLinearAttention*
*method*), 632
forward() (*pytext.models.representations.transformer.MultiheadSelfAttention*
*method*), 633
forward() (*pytext.models.representations.transformer.positional_embedding.PositionalEmbedding*
*method*), 625
forward() (*pytext.models.representations.transformer.PositionalEmbedding*
*method*), 633
forward() (*pytext.models.representations.transformer.PostEncoder*
*method*), 635
forward() (*pytext.models.representations.transformer.representation.TransformerRepresentation*
*method*), 626
forward() (*pytext.models.representations.transformer.residual_mlp.GELU*
*method*), 626
forward() (*pytext.models.representations.transformer.residual_mlp.ResidualMLP*
*method*), 627
forward() (*pytext.models.representations.transformer.ResidualMLP*
*method*), 634
forward() (*pytext.models.representations.transformer.SELFETransformer*
*method*), 635
forward() (*pytext.models.representations.transformer.sentence_encoder.PostEncoder*
*method*), 627
forward() (*pytext.models.representations.transformer.sentence_encoder.SentenceEncoder*
*method*), 628
forward() (*pytext.models.representations.transformer.SentenceEncoder*
*method*), 634
forward() (*pytext.models.representations.transformer.Transformer*
*method*), 636
forward() (*pytext.models.representations.transformer.transformer_SELFETransformer*
*method*), 629
forward() (*pytext.models.representations.transformer.transformer_Transformer*
*method*), 630
forward() (*pytext.models.representations.transformer.transformer_TransformerLayer*
*method*), 631
forward() (*pytext.models.representations.transformer.TransformerLayer*
*method*), 636
forward() (*pytext.models.representations.transformer.TransformerRepresentation*
*method*), 636
forward() (*pytext.models.representations.transformer_sentence_encoder.TransformerSentenceEncoder*
*method*), 660
forward() (*pytext.models.roberta.RoBERTaEncoder*
*method*), 700
forward() (*pytext.models.roberta.RoBERTaR3F*
*method*), 701
forward() (*pytext.models.roberta.RoBERTaWordTaggingModel*
*method*), 701
forward() (*pytext.models.roberta.TorchBertWSELFE*
*method*), 702
forward() (*pytext.utils.torchscriptify.parsers.rnng.rnng_parser.RNNGParser*
*method*), 663
forward() (*pytext.utils.torchscriptify.seq_models.attention.DecoupledMultiheadAttention*
*method*), 665
forward() (*pytext.utils.torchscriptify.seq_models.attention.DotAttention*
*method*), 665
forward() (*pytext.utils.torchscriptify.seq_models.attention.MultiheadAttention*
*method*), 666
forward() (*pytext.utils.torchscriptify.seq_models.base.PlaceholderAttentionIdentity*
*method*), 666
forward() (*pytext.utils.torchscriptify.seq_models.base.PlaceholderIdentity*
*method*), 667
forward() (*pytext.utils.torchscriptify.seq_models.conv_decoder.LightConvDecoder*
*method*), 668
forward() (*pytext.utils.torchscriptify.seq_models.conv_decoder.LightConvDecoderL*
*method*), 670
forward() (*pytext.utils.torchscriptify.seq_models.conv_decoder.LightConvDecoupled*
*method*), 671
forward() (*pytext.utils.torchscriptify.seq_models.conv_encoder.LightConvEncoder*
*method*), 672
forward() (*pytext.utils.torchscriptify.seq_models.conv_encoder.LightConvEncoderL*
*method*), 672
forward() (*pytext.utils.torchscriptify.seq_models.conv_model.CNNModel*
*method*), 673
forward() (*pytext.utils.torchscriptify.seq_models.light_conv.LightweightConv*
*method*), 673
forward() (*pytext.utils.torchscriptify.seq_models.mask_generator.MaskedSequenceG*
*method*), 674
forward() (*pytext.utils.torchscriptify.seq_models.nar_length.ConvLengthPrediction*
*method*), 677
forward() (*pytext.utils.torchscriptify.seq_models.nar_length.MaskedLengthPredicti*
*method*), 677
forward() (*pytext.utils.torchscriptify.seq_models.positional.LearnedPositionalEmbe*
*method*), 679
forward() (*pytext.utils.torchscriptify.seq_models.positional.SinusoidalPositionalEm*
*method*), 679
forward() (*pytext.utils.torchscriptify.seq_models.projection_layers.DecoderWithLin*
*method*), 680
forward() (*pytext.utils.torchscriptify.seq_models.projection_layers.DecoupledDeco*
*method*), 680
forward() (*pytext.utils.torchscriptify.seq_models.rnn_decoder.DecoderWithLinearC*
*method*), 681
forward() (*pytext.utils.torchscriptify.seq_models.rnn_encoder.BiLSTM*
*method*), 682
forward() (*pytext.utils.torchscriptify.seq_models.rnn_encoder.BiLSTMSequenceEnco*

method), 682
forward() (pytext.models.seq_models.rnn_encoder_decoder.RNNModel method), 683
forward() (pytext.models.seq_models.seq2seq_model.Seq2SeqModel method), 683
forward() (pytext.models.two_tower_classification_model.TwoTowerClassificationModel method), 702
forward() (pytext.models.TwoTowerClassificationModel method), 706
forward() (pytext.task.accelerator_lowering.AcceleratorBiLSTM method), 724
forward() (pytext.task.accelerator_lowering.AcceleratorTransformer method), 725
forward() (pytext.task.accelerator_lowering.AcceleratorTransformerLayersInstead method), 725
forward() (pytext.torchscript.seq2seq.beam_decode.BeamDecode method), 737
forward() (pytext.torchscript.seq2seq.beam_search.BeamSearch method), 738
forward() (pytext.torchscript.seq2seq.decoder.DecoderBatchedStepEnsemble method), 738
forward() (pytext.torchscript.seq2seq.encoder.EncoderEnsemble method), 738
forward() (pytext.torchscript.seq2seq.export_model.Seq2SeqJIT method), 739
forward() (pytext.torchscript.seq2seq.scripted_seq2seq_generator.ScriptedSeq2SeqGenerator method), 739
forward() (pytext.torchscript.tensorizer.normalizer.VectorNormalizer method), 741
forward() (pytext.torchscript.tensorizer.VectorNormalizer method), 743
forward() (pytext.utils.lazy.Lazy method), 765
forward() (pytext.utils.loss.LagrangeMultiplier static method), 766
forward_layers (pytext.models.representations.augmented_lstm.AugmentedLSTM attribute), 638
forward_unprojected() (pytext.models.seq_models.conv_decoder.LightConvDecoderBase method), 669
forward_unprojected() (pytext.models.seq_models.rnn_decoder.DecoderWithLinearOutputProjection method), 681
forward_unprojected() (pytext.models.seq_models.rnn_decoder.RNNDecoderBase method), 681
forward_with_noise() (pytext.models.r3f_models.R3FPyTextMixin method), 699
FP (pytext.metrics.Confusions attribute), 554
fp16 (pytext.config.pytext_config.LogitsConfig attribute), 411
FP16Optimizer (class in pytext.optimizer.fp16_optimizer), 714
FP16OptimizerApex (class in pytext.optimizer.fp16_optimizer), 714
FP16OptimizerDeprecated (class in pytext.optimizer.fp16_optimizer), 714
FP16OptimizerFairseq (class in pytext.optimizer.fp16_optimizer), 715
frame_accuracies_by_depth (pytext.metrics.intent_slot_metrics.AllMetrics attribute), 546
frame_accuracy (pytext.metrics.intent_slot_metrics.AllMetrics attribute), 546
frame_accuracy (pytext.metrics.intent_slot_metrics.FrameAccuracy attribute), 547
frame_accuracy_top_k (pytext.metrics.intent_slot_metrics.AllMetrics attribute), 546
frame_to_str() (in module pytext.ensemble_reporters.intent_slot_detection_metric_reporter), 524
FrameAccuraciesByDepth (in module pytext.metrics.intent_slot_metrics), 547
FrameAccuracy (class in pytext.metrics.intent_slot_metrics), 547
FramesDecodedSequence (class in pytext.metrics.intent_slot_metrics), 547
freeze() (pytext.models.embeddings.word_embedding.WordEmbedding method), 580
freeze() (pytext.models.embeddings.word_seq_embedding.WordSeqEmb method), 582
freeze() (pytext.models.embeddings.WordEmbedding method), 584
freeze() (pytext.models.embeddings.WordSeqEmbedding method), 588
freezeLSTM() (pytext.models.module.Module method), 696
FREQUENCY (pytext.models.masking_utils.MaskingStrategy attribute), 693
frequency_based_masking() (in module pytext.models.masking_utils), 693
from_config() (pytext.config.component.Component class method), 403
from_config() (pytext.data.AlternatingRandomizedBatchSampler class method), 482
from_config() (pytext.data.BaseBatchSampler class method), 483
from_config() (pytext.data.batch_sampler.AlternatingRandomizedBatchSampler class method), 440
from_config() (pytext.data.batch_sampler.BaseBatchSampler class method), 440
from_config() (py-

*text.data.batch_sampler.NaturalBatchSampler class method*), 440
from_config() (*pytext.data.batch_sampler.RandomizedBatchSampler class method*), 441
from_config() (*pytext.data.batch_sampler.RoundRobinBatchSampler class method*), 441
from_config() (*pytext.data.Batcher class method*), 483
from_config() (*pytext.data.bert_tensorizer.BERTTensorizer class method*), 442
from_config() (*pytext.data.Data class method*), 484
from_config() (*pytext.data.data.Batcher class method*), 444
from_config() (*pytext.data.data.Data class method*), 445
from_config() (*pytext.data.data.PoolingBatcher class method*), 446
from_config() (*pytext.data.dense_retrieval_tensorizer.RoBERTaContextTensorizerForDenseRetrieval class method*), 453
from_config() (*pytext.data.disjoint_multitask_data.DisjointMultitaskData class method*), 454
from_config() (*pytext.data.DisjointMultitaskData class method*), 488
from_config() (*pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher class method*), 457
from_config() (*pytext.data.DynamicPoolingBatcher class method*), 489
from_config() (*pytext.data.featurizer.Featurizer class method*), 420
from_config() (*pytext.data.featurizer.featurizer.Featurizer class method*), 418
from_config() (*pytext.data.masked_tensorizer.MaskedTokenTensorizer class method*), 457
from_config() (*pytext.data.masked_util.MaskingFunction class method*), 458
from_config() (*pytext.data.masked_util.NoOpMaskingFunction class method*), 458
from_config() (*pytext.data.masked_util.RandomizedMaskingFunction class method*), 458
from_config() (*pytext.data.masked_util.TreeMask class method*), 458
from_config() (*pytext.data.NaturalBatchSampler class method*), 491
from_config() (*pytext.data.packed_lm_data.PackedLMData class method*), 459
from_config() (*pytext.data.PoolingBatcher class method*), 490
from_config() (*pytext.data.RandomizedBatchSampler class method*), 491
from_config() (*pytext.data.roberta_tensorizer.RoBERTaTensorizer class method*), 459
from_config() (*pytext.data.roberta_tensorizer.RoBERTaTokenLevelTensorizer class method*), 459
from_config() (*pytext.data.RoundRobinBatchSampler class method*), 491
from_config() (*pytext.data.sources.conllu.CoNLLUPOSDataSource class method*), 421
from_config() (*pytext.data.sources.dense_retrieval.DenseRetrievalDataSource class method*), 424
from_config() (*pytext.data.sources.DenseRetrievalDataSource class method*), 430
from_config() (*pytext.data.sources.pandas.PandasDataSource class method*), 425
from_config() (*pytext.data.sources.PandasDataSource class method*), 430
from_config() (*pytext.data.sources.squad.SquadDataSource class method*), 426
from_config() (*pytext.data.sources.SquadDataSource class method*), 429
from_config() (*pytext.data.sources.tsv.TSVDataSource class method*), 428
from_config() (*pytext.data.sources.TSVDataSource class method*), 429
from_config() (*pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizer class method*), 460
from_config() (*pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizerFor class method*), 460
from_config() (*pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizer class method*), 461
from_config() (*pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizer

class method), 461
from_config() (pytext.data.squad_tensorizer.SquadTensorizer class method), 462
from_config() (pytext.data.squad_tensorizer.SquadTensorizerForKD class method), 462
from_config() (pytext.data.Tensorizer class method), 492
from_config() (pytext.data.tensorizers.AnnotationNumberizer class method), 463
from_config() (pytext.data.tensorizers.ByteTensorizer class method), 463
from_config() (pytext.data.tensorizers.ByteTokenTensorizer class method), 464
from_config() (pytext.data.tensorizers.CharacterVocabTokenTensorizer class method), 465
from_config() (pytext.data.tensorizers.Float1DListTensorizer class method), 466
from_config() (pytext.data.tensorizers.FloatListSeqTensorizer class method), 467
from_config() (pytext.data.tensorizers.FloatListTensorizer class method), 467
from_config() (pytext.data.tensorizers.FloatTensorizer class method), 468
from_config() (pytext.data.tensorizers.GazetteerTensorizer class method), 468
from_config() (pytext.data.tensorizers.Integer1DListTensorizer class method), 469
from_config() (pytext.data.tensorizers.LabelListRankTensorizer class method), 470
from_config() (pytext.data.tensorizers.LabelListTensorizer class method), 470
from_config() (pytext.data.tensorizers.LabelTensorizer class method), 470
from_config() (pytext.data.tensorizers.MetricTensorizer class method), 471
from_config() (pytext.data.tensorizers.NumericLabelTensorizer class method), 471
from_config() (pytext.data.tensorizers.SeqTokenTensorizer class method), 472
from_config() (pytext.data.tensorizers.SlotLabelTensorizer class method), 473
from_config() (pytext.data.tensorizers.SoftLabelTensorizer class method), 473
from_config() (pytext.data.tensorizers.String2DListTensorizer class method), 474
from_config() (pytext.data.tensorizers.Tensorizer class method), 475
from_config() (pytext.data.tensorizers.TokenTensorizer class method), 476
from_config() (pytext.data.tensorizers.UidTensorizer class method), 477
from_config() (pytext.data.token_tensorizer.ScriptBasedTokenTensorizer class method), 478
from_config() (pytext.data.tokenizers.DoNothingTokenizer class method), 439
from_config() (pytext.data.tokenizers.GPT2BPETokenizer class method), 438
from_config() (pytext.data.tokenizers.SentencePieceTokenizer class method), 439
from_config() (pytext.data.tokenizers.Tokenizer class method), 439
from_config() (pytext.data.tokenizers.tokenizer.BERTInitialTokenizer class method), 437
from_config() (pytext.data.tokenizers.tokenizer.DoNothingTokenizer class method), 437
from_config() (pytext.data.tokenizers.tokenizer.GPT2BPETokenizer class method), 437
from_config() (pytext.data.tokenizers.tokenizer.SentencePieceTokenizer class method), 437
from_config() (pytext.data.tokenizers.tokenizer.Tokenizer class method), 438
from_config() (pytext.data.tokenizers.tokenizer.WordPieceTokenizer class method), 438
from_config() (pytext.data.tokenizers.WordPieceTokenizer

class method), 439
from_config() (pytext.data.xlm_tensorizer.XLMTensorizer class method), 482
from_config() (pytext.exporters.exporter.ModelExporter class method), 495
from_config() (pytext.exporters.ModelExporter class method), 496
from_config() (pytext.fields.Field class method), 505
from_config() (pytext.fields.field.Field class method), 501
from_config() (pytext.metric_reporters.calibration_metric_reporter.CalibrationMetricReporter class method), 513
from_config() (pytext.metric_reporters.CalibrationMetricReporter class method), 536
from_config() (pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter class method), 518
from_config() (pytext.metric_reporters.ClassificationMetricReporter class method), 537
from_config() (pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter class method), 520
from_config() (pytext.metric_reporters.CompositionalMetricReporter class method), 542
from_config() (pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetrievalMetricReporter class method), 522
from_config() (pytext.metric_reporters.DenseRetrievalMetricReporter class method), 544
from_config() (pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter class method), 523
from_config() (pytext.metric_reporters.IntentSlotMetricReporter class method), 540
from_config() (pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter class method), 525
from_config() (pytext.metric_reporters.language_model_metric_reporter.MaskedLMMetricReporter class method), 526
from_config() (pytext.metric_reporters.LanguageModelMetricReporter class method), 540
from_config() (pytext.metric_reporters.metric_reporter.PureLossMetricReporter class method), 528
from_config() (pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter class method), 539
from_config() (pytext.metric_reporters.NERMetricReporter class method), 544
from_config() (pytext.metric_reporters.pairwise_ranking_metric_reporter.PairwiseRankingMetricReporter class method), 528
from_config() (pytext.metric_reporters.PairwiseRankingMetricReporter class method), 543
from_config() (pytext.metric_reporters.PureLossMetricReporter class method), 543
from_config() (pytext.metric_reporters.regression_metric_reporter.RegressionMetricReporter class method), 528
from_config() (pytext.metric_reporters.RegressionMetricReporter class method), 539
from_config() (pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositionalMetricReporter class method), 529
from_config() (pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter class method), 530
from_config() (pytext.metric_reporters.SequenceTaggingMetricReporter class method), 543
from_config() (pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter class method), 531
from_config() (pytext.metric_reporters.SquadMetricReporter class method), 541
from_config() (pytext.metric_reporters.word_tagging_metric_reporter.MultiLabelSequenceTaggingMetricReporter class method), 532
from_config() (pytext.metric_reporters.word_tagging_metric_reporter.NERMetricReporter class method), 532
from_config() (pytext.metric_reporters.word_tagging_metric_reporter.SequenceTaggingMetricReporter class method), 533
from_config() (pytext.metric_reporters.word_tagging_metric_reporter.WordTaggingMetricReporter class method), 533
from_config() (pytext.metric_reporters.WordTaggingMetricReporter class method), 542
from_config() (pytext.models.bert_regression_model.NewBertRegressionModel

class method), 686
from_config() (pytext.models.doc_model.DocModel class method), 690
from_config() (pytext.models.doc_model.DocRegressionModel class method), 690
from_config() (pytext.models.doc_model.PersonalizedDocModel class method), 691
from_config() (pytext.models.embeddings.char_embedding.CharacterEmbedding class method), 573
from_config() (pytext.models.embeddings.CharacterEmbedding class method), 586
from_config() (pytext.models.embeddings.contextual_token_embedding.ContextualTokenEmbedding class method), 574
from_config() (pytext.models.embeddings.ContextualTokenEmbedding class method), 587
from_config() (pytext.models.embeddings.dict_embedding.DictEmbedding class method), 575
from_config() (pytext.models.embeddings.DictEmbedding class method), 585
from_config() (pytext.models.embeddings.mlp_embedding.MLPEmbedding class method), 577
from_config() (pytext.models.embeddings.MLPEmbedding class method), 588
from_config() (pytext.models.embeddings.word_embedding.WordEmbedding class method), 580
from_config() (pytext.models.embeddings.word_seq_embedding.WordSeqEmbedding class method), 582
from_config() (pytext.models.embeddings.WordEmbedding class method), 584
from_config() (pytext.models.embeddings.WordSeqEmbedding class method), 588
from_config() (pytext.models.ensembles.ensemble.EnsembleModel class method), 591
from_config() (pytext.models.ensembles.EnsembleModel class method), 594
from_config() (pytext.models.joint_model.IntentSlotModel class method), 692
from_config() (pytext.models.language_models.lmlstm.LMLSTM class method), 595
from_config() (pytext.models.masked_lm.MaskedLanguageModel class method), 692
from_config() (pytext.models.Model class method), 705
from_config() (pytext.models.model.Model class method), 696
from_config() (pytext.models.output_layers.ClassificationOutputLayer class method), 616
from_config() (pytext.models.output_layers.CRFOutputLayer class method), 615
from_config() (pytext.models.output_layers.distance_output_layer.PairwiseCosineL class method), 598
from_config() (pytext.models.output_layers.doc_classification_output_layer.Classif class method), 600
from_config() (pytext.models.output_layers.doc_regression_output_layer.PairwiseC class method), 601
from_config() (pytext.models.output_layers.doc_regression_output_layer.Regressio class method), 602
from_config() (pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutp class method), 603
from_config() (pytext.models.output_layers.lm_output_layer.LMOutputLayer class method), 605
from_config() (pytext.models.output_layers.multi_label_classification_layer.MultiL class method), 606
from_config() (pytext.models.output_layers.pairwise_ranking_output_layer.Pairwis class method), 608
from_config() (pytext.models.output_layers.PairwiseCosineDistanceOutputLayer class method), 619
from_config() (pytext.models.output_layers.PairwiseCosineRegressionOutputLayer class method), 620
from_config() (pytext.models.output_layers.PairwiseRankingOutputLayer class method), 618
from_config() (pytext.models.output_layers.RegressionOutputLayer class method), 616
from_config() (pytext.models.output_layers.squad_output_layer.SquadOutputLayer

class method), 609
from_config() (pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer class method), 611
from_config() (pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer class method), 612
from_config() (pytext.models.output_layers.WordTaggingOutputLayer class method), 617
from_config() (pytext.models.pair_classification_model.BasePairwiseModel class method), 697
from_config() (pytext.models.pair_classification_model.PairwiseModel class method), 698
from_config() (pytext.models.qna.bert_squad_qa.BertSquadQAModel class method), 622
from_config() (pytext.models.qna.dr_qa.DrQAModel class method), 623
from_config() (pytext.models.query_document_pairwise_ranking_model.QueryDocumentPairwiseRankingModel class method), 699
from_config() (pytext.models.representations.attention.DotProductSelfAttention class method), 637
from_config() (pytext.models.representations.attention.MultiplicativeAttention class method), 637
from_config() (pytext.models.representations.attention.SequenceAlignedAttention class method), 638
from_config() (pytext.models.representations.transformer_sentence_encoder_base.TransformerSentenceEncoderBase class method), 660
from_config() (pytext.models.roberta.RoBERTaWordTaggingModel class method), 702
from_config() (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserBase class method), 664
from_config() (pytext.models.seq_models.attention.MultiheadAttention class method), 666
from_config() (pytext.models.seq_models.conv_decoder.LightConvDecoderBase class method), 669
from_config() (pytext.models.seq_models.conv_decoder.LightConvDecoderLayer class method), 670
from_config() (pytext.models.seq_models.conv_decoder.LightConvDecoupledDecoder class method), 671
from_config() (pytext.models.seq_models.conv_encoder.LightConvEncoder class method), 672
from_config() (pytext.models.seq_models.conv_encoder.LightConvEncoderLayer class method), 673
from_config() (pytext.models.seq_models.conv_model.CNNModel class method), 673
from_config() (pytext.models.seq_models.light_conv.LightweightConv class method), 674
from_config() (pytext.models.seq_models.mask_generator.MaskedSequenceGenera class method), 675
from_config() (pytext.models.seq_models.nar_length.ConvLengthPredictionModule class method), 677
from_config() (pytext.models.seq_models.nar_length.MaskedLengthPredictionMod class method), 677
from_config() (pytext.models.seq_models.seq2seq_output_layer.NARSeq2SeqOutputLay class method), 678
from_config() (pytext.models.seq_models.rnn_decoder.RNNDecoderBase class method), 682
from_config() (pytext.models.seq_models.rnn_encoder.LSTMSequenceEncoder class method), 682
from_config() (pytext.models.seq_models.rnn_encoder_decoder.RNNModel class method), 683
from_config() (pytext.models.seq_models.seq2seq_encoder_base.Seq2SeqModel class method), 684
from_config() (pytext.models.seq_models.seq2seq_output_layer.Seq2SeqOutputLay class method), 684
from_config() (pytext.models.two_tower_classification_model.TwoTowerClassificat class method), 702
from_config() (pytext.models.TwoTowerClassificationModel class method), 707
from_config() (pytext.models.word_model.WordTaggingModel class method), 703
from_config() (pytext.optimizer.adabelief.AdaBelief class method), 712
from_config() (pytext.optimizer.fp16_optimizer.FP16OptimizerApex class method), 714

from_config() (pytext.optimizer.fp16_optimizer.FP16OptimizerFairseq class method), 715
from_config() (pytext.optimizer.fp16_optimizer.MemoryEfficientFP16OptimizerFairseq class method), 716
from_config() (pytext.optimizer.lamb.Lamb class method), 717
from_config() (pytext.optimizer.madgrad.MADGRAD class method), 717
from_config() (pytext.optimizer.optimizers.Adagrad class method), 718
from_config() (pytext.optimizer.optimizers.Adam class method), 718
from_config() (pytext.optimizer.optimizers.AdamW class method), 718
from_config() (pytext.optimizer.optimizers.SGD class method), 718
from_config() (pytext.optimizer.privacy_engine.PrivacyEngine class method), 719
from_config() (pytext.optimizer.radam.RAdam class method), 719
from_config() (pytext.optimizer.scheduler.CosineAnnealingLR class method), 720
from_config() (pytext.optimizer.scheduler.CyclicLR class method), 720
from_config() (pytext.optimizer.scheduler.ExponentialLR class method), 720
from_config() (pytext.optimizer.scheduler.LmFineTuning class method), 720
from_config() (pytext.optimizer.scheduler.PolynomialDecayScheduler class method), 721
from_config() (pytext.optimizer.scheduler.ReduceLROnPlateau class method), 721
from_config() (pytext.optimizer.scheduler.SchedulerWithWarmup class method), 722
from_config() (pytext.optimizer.scheduler.StepLR class method), 722
from_config() (pytext.optimizer.scheduler.WarmupScheduler class method), 722
from_config() (pytext.optimizer.sparsifiers.blockwise_sparsifier.BlockwiseMagnitudeSparsifier class method), 708
from_config() (pytext.optimizer.sparsifiers.sparsifier.CRF_L1_SoftThresholding class method), 709
from_config() (pytext.optimizer.sparsifiers.sparsifier.CRF_MagnitudeThresholding class method), 709
from_config() (pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier class method), 710
from_config() (pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier class method), 710
from_config() (pytext.optimizer.swa.StochasticWeightAveraging class method), 723
from_config() (pytext.task.disjoint_multitask.DisjointMultitask class method), 726
from_config() (pytext.task.disjoint_multitask.NewDisjointMultitask class method), 726
from_config() (pytext.task.task.TaskBase class method), 731
from_config() (pytext.task.TaskBase class method), 736
from_config() (pytext.task.tasks.PairwiseClassificationTask class method), 734
from_config() (pytext.torchscript.seq2seq.scripted_seq2seq_generator.ScriptedSequ class method), 740
from_config() (pytext.trainers.ensemble_trainer.EnsembleTrainer class method), 754
from_config() (pytext.trainers.EnsembleTrainer class method), 760
from_config() (pytext.trainers.hogwild_trainer.HogwildTrainer class method), 754
from_config() (pytext.trainers.hogwild_trainer.HogwildTrainer_Deprecated class method), 755
from_config() (pytext.trainers.HogwildTrainer class method), 760
from_config() (pytext.trainers.HogwildTrainer_Deprecated class method), 760
from_config() (pytext.trainers.Trainer class method), 758
from_config() (pytext.trainers.trainer.Trainer class method), 755
from_config_and_label_names() (pytext.metric_reporters.classification_metric_reporter.ClassificationM class method), 518
from_config_and_label_names() (pytext.metric_reporters.ClassificationMetricReporter

class method), 537
from_vocab_file() (pytext.torchscript.tokenizer.bpe.ScriptBPE class method), 744
from_vocab_file() (pytext.torchscript.tokenizer.ScriptBPE class method), 745
from_vocab_filename() (pytext.torchscript.tokenizer.bpe.ScriptBPE class method), 744
from_vocab_filename() (pytext.torchscript.tokenizer.ScriptBPE class method), 746

## G

gamma (pytext.data.dynamic_pooling_batcher.ExponentialBatcherSchedulerConfig attribute), 457
gazetteer_feat_lengths (pytext.data.featurizer.featurizer.OutputRecord attribute), 419
gazetteer_feat_lengths (pytext.data.featurizer.OutputRecord attribute), 420
gazetteer_feat_weights (pytext.data.featurizer.featurizer.OutputRecord attribute), 419
gazetteer_feat_weights (pytext.data.featurizer.OutputRecord attribute), 420
gazetteer_feats (pytext.data.featurizer.featurizer.OutputRecord attribute), 419
gazetteer_feats (pytext.data.featurizer.OutputRecord attribute), 420
GazetteerTensorizer (class in pytext.data.tensorizers), 468
GeLU (class in pytext.models.representations.transformer.residual_mlp), 626
GELU (pytext.config.module_config.Activation attribute), 409
gen_additional_blobs() (pytext.models.output_layers.OutputLayerUtils static method), 621
gen_additional_blobs() (pytext.models.output_layers.utils.OutputLayerUtils static method), 610
gen_config_impl() (in module pytext.main), 771
gen_content() (pytext.metric_reporters.channel.FileChannel method), 515
gen_content() (pytext.metric_reporters.language_model_metric_reporter.LanguageModelChannel method), 524
gen_content() (pytext.metric_reporters.seq2seq_compositional.CompositionalSeq2S method), 529
gen_content() (pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqFileChann method), 529
gen_content() (pytext.metric_reporters.squad_metric_reporter.SquadFileChannel method), 530
gen_dataset() (pytext.data.data_handler.DataHandler method), 449
gen_dataset() (pytext.data.DataHandler method), 486
gen_dataset_from_path() (pytext.data.data_handler.DataHandler method), 449
gen_dataset_from_path() (pytext.data.DataHandler method), 486
gen_extra_context() (pytext.metric_reporters.compositional_metric_reporter.Compositior method), 520
gen_extra_context() (pytext.metric_reporters.CompositionalMetricReporter method), 542
gen_extra_context() (pytext.metric_reporters.metric_reporter.MetricReporter method), 527
gen_extra_context() (pytext.metric_reporters.MetricReporter method), 535
gen_extra_context() (pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricRep method), 530
gen_masked_source_target() (pytext.data.masked_util.MaskEverything method), 458
gen_masked_source_target() (pytext.data.masked_util.MaskingFunction method), 458
gen_masked_source_target() (pytext.data.masked_util.NoOpMaskingFunction method), 458
gen_masked_source_target() (pytext.data.masked_util.RandomizedMaskingFunction method), 458
gen_masked_source_target() (pytext.data.masked_util.TreeMask method), 458
gen_masked_tree() (pytext.data.masked_util.MaskEverything method), 458
gen_masked_tree() (pytext.data.masked_util.TreeMask method),

458
generate_checkpoint_path() (*in module pytext.task.serialize*), 729
generate_hypo() (*pytext.models.seq_models.mask_generator.MaskedSequenceGenerator method*), 675
generate_hypo() (*pytext.torchscript.seq2seq.scripted_seq2seq_generator.ScriptedSeq2SeqGenerator method*), 740
generate_non_autoregressive() (*pytext.models.seq_models.mask_generator.MaskedSequenceGenerator method*), 675
generate_params() (*in module pytext.optimizer.fp16_optimizer*), 717
generator_iterator() (*in module pytext.data*), 489
generator_iterator() (*in module pytext.data.data*), 446
generator_property (*in module pytext.data.sources.data_source*), 424
GeneratorFP16Optimizer (*class in pytext.optimizer.fp16_optimizer*), 715
GeneratorIterator (*class in pytext.data.sources.data_source*), 422
GeneratorMethodProperty (*class in pytext.data.sources.data_source*), 422
get() (*pytext.config.component.Registry class method*), 404
get_absolute_path() (*in module pytext.utils.path*), 769
get_activation() (*in module pytext.optimizer.activations*), 712
get_annotation_from_string() (*pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositionalMetricReporter method*), 529
get_batch_size() (*pytext.data.data.PoolingBatcher method*), 446
get_batch_size() (*pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher method*), 457
get_batch_size() (*pytext.data.DynamicPoolingBatcher method*), 489
get_batch_size() (*pytext.data.PoolingBatcher method*), 490
get_beam_ranking_function() (*in module pytext.models.seq_models.mask_generator*), 675
get_bos_index() (*pytext.data.utils.Vocabulary method*), 480
get_bucket_accuracy() (*in module pytext.metrics.calibration_metrics*), 545
get_bucket_confidence() (*in module pytext.metrics.calibration_metrics*), 545
get_bucket_scores() (*in module pytext.metrics.calibration_metrics*), 545
get_class_members_recursive() (*in module pytext.utils.documentation*), 763
get_clip_length() (*pytext.models.seq_models.mask_generator.MaskedSequenceGenerator method*), 675
get_component_name() (*in module pytext.config.component*), 405
get_compressed_projection() (*pytext.models.representations.transformer.multihead_linear_attention method*), 624
get_compressed_projection() (*pytext.models.representations.transformer.multihead_linear_attention method*), 625
get_compressed_projection() (*pytext.models.representations.transformer.MultiheadLinearAttention method*), 632
get_compressed_projection() (*pytext.models.representations.transformer.QuantizedMultiheadLinear method*), 633
get_config_fields() (*in module pytext.utils.documentation*), 763
get_cost_fn() (*in module pytext.loss.structured_loss*), 510
get_current_sparsity() (*pytext.optimizer.sparsifiers.blockwise_sparsifier.BlockwiseMagnitude method*), 708
get_current_sparsity() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 710
get_current_sparsity() (*pytext.optimizer.sparsifiers.sparsifier.Sparsifier method*), 711
get_decoder() (*pytext.models.decoders.decoder_base.DecoderBase method*), 565
get_decoder() (*pytext.models.decoders.DecoderBase method*), 570
get_decoder() (*pytext.models.decoders.intent_slot_model_decoder.IntentSlotModel method*), 567
get_decoder() (*pytext.models.decoders.IntentSlotModelDecoder method*), 572
get_decoder() (*pytext.models.decoders.mlp_decoder.MLPDecoder method*), 567
get_decoder() (*pytext.models.decoders.mlp_decoder_query_response.MLPDecoder method*), 568
get_decoder() (*pytext.models.decoders.mlp_decoder_two_tower.MLPDecoderTwoT method*), 569

get_decoder() (pytext.models.decoders.MLPDecoder method), 571
get_decoder() (pytext.models.decoders.multilabel_decoder.MultiLabelDecoder method), 569
get_depth() (pytext.data.data_structures.node.Node method), 417
get_dropout_mask() (pytext.models.representations.augmented_lstm.AugmentedLSTMUnidirectional method), 641
get_embedding_module() (pytext.models.r3f_models.R3FPyTextMixin method), 699
get_embedding_module() (pytext.models.roberta.RoBERTaR3F method), 701
get_embedding_module() (pytext.models.seq_models.conv_model.CNNModel method), 673
get_embedding_module_from_path() (pytext.task.nop_decorator.accelerator class method), 728
get_encoder_out() (pytext.models.seq_models.mask_generator.MaskedSequenceGenerator method), 675
get_eos_index() (pytext.data.utils.Vocabulary method), 480
get_eval_iter() (pytext.data.data_handler.DataHandler method), 449
get_eval_iter() (pytext.data.DataHandler method), 486
get_eval_iter() (pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler method), 455
get_eval_iter() (pytext.data.DisjointMultitaskDataHandler method), 488
get_export_accelerate() (pytext.config.pytext_config.PyTextConfig method), 412
get_export_batch_padding_control() (pytext.config.pytext_config.PyTextConfig method), 412
get_export_caffe2_path() (pytext.config.pytext_config.PyTextConfig method), 412
get_export_inference_interface() (pytext.config.pytext_config.PyTextConfig method), 412
get_export_input_names() (pytext.models.doc_model.ByteTokensDocumentModel method), 690
get_export_input_names() (pytext.models.doc_model.DocModel method), 690
get_export_input_names() (pytext.models.doc_model.PersonalizedDocModel method), 691
get_export_input_names() (pytext.models.ensembles.ensemble.EnsembleModel method), 591
get_export_input_names() (pytext.models.ensembles.EnsembleModel method), 594
get_export_input_names() (pytext.models.joint_model.IntentSlotModel method), 692
get_export_input_names() (pytext.models.language_models.lmlstm.LMLSTM method), 595
get_export_input_names() (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method), 662
get_export_input_names() (pytext.models.seq_models.contextual_intent_slot.ContextualIntentSl method), 668
get_export_input_names() (pytext.models.word_model.WordTaggingLiteModel method), 703
get_export_input_names() (pytext.models.word_model.WordTaggingModel method), 703
get_export_onnx_path() (pytext.config.pytext_config.PyTextConfig method), 412
get_export_output_names() (pytext.models.doc_model.DocModel method), 690
get_export_output_names() (pytext.models.ensembles.ensemble.EnsembleModel method), 591
get_export_output_names() (pytext.models.ensembles.EnsembleModel method), 594
get_export_output_names() (pytext.models.joint_model.IntentSlotModel method), 692
get_export_output_names() (pytext.models.language_models.lmlstm.LMLSTM method), 595
get_export_output_names() (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method), 662
get_export_output_names() (pytext.models.word_model.WordTaggingModel method), 703

get_export_paths() (*pytext.exporters.custom_exporters.InitPredictNetExporter method*), 493
get_export_paths() (*pytext.exporters.InitPredictNetExporter method*), 498
get_export_seq_padding_control() (*pytext.config.pytext_config.PyTextConfig method*), 412
get_export_target() (*pytext.config.pytext_config.PyTextConfig method*), 412
get_export_torchscript_path() (*pytext.config.pytext_config.PyTextConfig method*), 412
get_export_torchscript_quantize() (*pytext.config.pytext_config.PyTextConfig method*), 412
get_exporter() (*in module pytext.exporters.custom_exporters*), 494
get_extra_params() (*pytext.exporters.exporter.ModelExporter method*), 495
get_extra_params() (*pytext.exporters.ModelExporter method*), 497
get_feature_metadata() (*pytext.exporters.custom_exporters.DenseFeatureExporter class method*), 493
get_feature_metadata() (*pytext.exporters.DenseFeatureExporter class method*), 497
get_feature_metadata() (*pytext.exporters.exporter.ModelExporter class method*), 495
get_feature_metadata() (*pytext.exporters.ModelExporter class method*), 497
get_first_config() (*pytext.config.pytext_config.PyTextConfig method*), 412
get_gradients() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
get_gradients() (*pytext.metric_reporters.MetricReporter method*), 535
get_in_dim() (*pytext.models.decoders.decoder_base.DecoderBase method*), 565
get_in_dim() (*pytext.models.decoders.DecoderBase method*), 570
get_incremental_state() (*pytext.models.seq_models.base.PyTextIncrementalDecoderComponent method*), 667
get_info() (*pytext.data.data_structures.annotation.Node method*), 415
get_json_config_iterator() (*in module pytext.config.config_adapter*), 405
get_label_weights() (*in module pytext.utils.label*), 764
get_lang_id() (*pytext.data.xlm_tensorizer.XLMTensorizer method*), 482
get_latest_checkpoint_path() (*in module pytext.task*), 737
get_latest_checkpoint_path() (*in module pytext.task.serialize*), 729
get_latest_checkpoint_path() (*pytext.task.serialize.CheckpointManager method*), 728
get_latest_checkpoint_path() (*pytext.task.serialize.PyTextCheckpointManagerInterface method*), 729
get_logits() (*in module pytext.workflow*), 771
get_loss() (*pytext.models.BaseModel method*), 706
get_loss() (*pytext.models.disjoint_multitask_model.DisjointMultitaskM method*), 688
get_loss() (*pytext.models.model.BaseModel method*), 694
get_loss() (*pytext.models.output_layers.CRFOutputLayer method*), 615
get_loss() (*pytext.models.output_layers.DenseRetrievalOutputLayer method*), 621
get_loss() (*pytext.models.output_layers.distance_output_layer.DenseR method*), 597
get_loss() (*pytext.models.output_layers.distance_output_layer.Pairwis method*), 598
get_loss() (*pytext.models.output_layers.doc_regression_output_layer.P method*), 601
get_loss() (*pytext.models.output_layers.doc_regression_output_layer.R method*), 602
get_loss() (*pytext.models.output_layers.intent_slot_output_layer.Intent method*), 603
get_loss() (*pytext.models.output_layers.lm_output_layer.LMOutputLay method*), 605
get_loss() (*pytext.models.output_layers.multi_label_classification_laye method*), 606
get_loss() (*pytext.models.output_layers.output_layer_base.OutputLaye method*), 608
get_loss() (*pytext.models.output_layers.OutputLayerBase method*), 614
get_loss() (*pytext.models.output_layers.PairwiseCosineDistanceOutpu method*), 619
get_loss() (*pytext.models.output_layers.PairwiseCosineRegressionOut method*), 620
get_loss() (*pytext.models.output_layers.RegressionOutputLayer method*), 616
get_loss() (*pytext.models.output_layers.squad_output_layer.SquadOut method*), 609

get_loss() (pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer method), 611
get_loss() (pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method), 612
get_loss() (pytext.models.output_layers.WordTaggingOutputLayer method), 617
get_loss() (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserBase method), 664
get_loss() (pytext.models.seq_models.nar_output_layer.NARSeq2SeqOutputLayer method), 678
get_loss() (pytext.models.seq_models.seq2seq_output_layer.Seq2SeqOutputLayer method), 684
get_lr() (pytext.optimizer.scheduler.LmFineTuning method), 720
get_lr() (pytext.optimizer.scheduler.PolynomialDecayScheduler method), 721
get_lr() (pytext.optimizer.scheduler.SchedulerWithWarmup method), 722
get_lr() (pytext.optimizer.scheduler.WarmupScheduler method), 722
get_mask_for_param() (pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method), 711
get_mask_index() (pytext.data.utils.Vocabulary method), 480
get_masks() (pytext.optimizer.sparsifiers.blockwise_sparsifier.BlockwiseMagnitudeSparsifier method), 708
get_masks() (pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier method), 710
get_masks() (pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method), 711
get_max_steps() (pytext.data.dynamic_pooling_batcher.ExponentialDynamicPoolingBatcher method), 457
get_meta() (pytext.fields.Field method), 505
get_meta() (pytext.fields.field.Field method), 501
get_meta() (pytext.fields.field.NestedField method), 501
get_meta() (pytext.fields.field.RawField method), 501
get_meta() (pytext.fields.field.WordLabelField method), 503
get_meta() (pytext.fields.NestedField method), 506
get_meta() (pytext.fields.RawField method), 506
get_meta() (pytext.fields.WordLabelField method), 506
get_meta() (pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method), 518
get_meta() (pytext.metric_reporters.ClassificationMetricReporter method), 537
get_meta() (pytext.metric_reporters.metric_reporter.MetricReporter method), 527
get_meta() (pytext.metric_reporters.MetricReporter method), 535
get_mismatched_param() (in module pytext.GRU.output_layer), 568
get_mlp() (pytext.models.decoders.mlp_decoder_query_response.MLPDecoderWithMultiOutputLayer static method), 568
get_mlp() (pytext.models.decoders.mlp_decoder_two_tower.MLPDecoderTwoTower static method), 569
get_mlp() (pytext.models.decoders.multilabel_decoder.MultiLabelDecoder static method), 569
get_model_select_metric() (pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method), 518
get_model_select_metric() (pytext.metric_reporters.ClassificationMetricReporter method), 537
get_model_select_metric() (pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method), 520
get_model_select_metric() (pytext.metric_reporters.CompositionalMetricReporter method), 542
get_model_select_metric() (pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetrievalMetricReporter method), 522
get_model_select_metric() (pytext.metric_reporters.DenseRetrievalMetricReporter method), 544
get_model_select_metric() (pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method), 522
get_model_select_metric() (pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method), 523
get_model_select_metric() (pytext.metric_reporters.IntentSlotMetricReporter method), 540
get_model_select_metric() (pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method), 525
get_model_select_metric() (pytext.metric_reporters.LanguageModelMetricReporter method), 541
get_model_select_metric() (pytext.metric_reporters.metric_reporter.MetricReporter method), 527
get_model_select_metric() (pytext.metric_reporters.MetricReporter method), 535
get_model_select_metric() (pytext.metric_reporters.MultiLabelSequenceTaggingMetricReporter static method), 539
get_model_select_metric() (pytext.metric_reporters.NERMetricReporter static method), 544
get_model_select_metric() (pytext.metric_reporters.pairwise_ranking_metric_reporter.PairwiseRanking

*static method*), 528
get_model_select_metric() (*pytext.metric_reporters.PairwiseRankingMetricReporter static method*), 543
get_model_select_metric() (*pytext.metric_reporters.regression_metric_reporter.RegressionMetricReporter method*), 528
get_model_select_metric() (*pytext.metric_reporters.RegressionMetricReporter method*), 539
get_model_select_metric() (*pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter method*), 530
get_model_select_metric() (*pytext.metric_reporters.SequenceTaggingMetricReporter static method*), 543
get_model_select_metric() (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter method*), 531
get_model_select_metric() (*pytext.metric_reporters.SquadMetricReporter method*), 541
get_model_select_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.MultiLabelSeq static method*), 532
get_model_select_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.NERMetricReporter static method*), 532
get_model_select_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.SequenceTaggingMetricReporter static method*), 533
get_model_select_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTaggingMetricReporter method*), 533
get_model_select_metric() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 542
get_module_from_path() (*pytext.task.nop_decorator.accelerator class method*), 728
get_modules() (*pytext.task.nop_decorator.accelerator class method*), 728
get_name_from_options() (*in module pytext.config.config_adapter*), 405
get_norm_cosine_scores() (*in module pytext.models.output_layers.distance_output_layer*), 599
get_normalized_probs() (*pytext.models.seq_models.conv_model.CNNModel method*), 673
get_normalized_probs() (*pytext.models.seq_models.rnn_decoder.RNNDecoderBase method*), 682
get_normalized_probs() (*pytext.models.seq_models.rnn_encoder_decoder.RNNModel method*), 683
get_num_examples_from_batch() (*pytext.models.BaseModel method*), 706
get_num_examples_from_batch() (*pytext.models.doc_model.DocModel method*), 690
get_num_examples_from_batch() (*pytext.models.language_models.lmlstm.LMLSTM method*), 595
get_num_examples_from_batch() (*pytext.models.model.BaseModel method*), 694
get_num_examples_from_batch() (*pytext.models.query_document_pairwise_ranking_model.QueryDoc method*), 699
get_numericalize_net() (*in module pytext.utils.mobile_onnx*), 768
get_numericalize_net() (*in module pytext.utils.onnx*), 769
get_out_dim() (*pytext.models.decoders.decoder_base.DecoderBase method*), 565
get_output_dim() (*pytext.models.decoders.DecoderBase method*), 570
get_pad_index() (*pytext.data.utils.Vocabulary method*), 480
get_pad_index() (*pytext.utils.torchscriptify.ScriptVocabulary method*), 754
get_parent() (*pytext.data.data_structures.annotation.Node_Info method*), 416
get_parent() (*pytext.data.data_structures.annotation.Token_Info method*), 416
get_perplexity_func() (*in module pytext.metric_reporters.language_model_metric_reporter*), 526
get_pointer_src_tokens() (*pytext.models.seq_models.projection_layers.DecoupledDecoderHea method*), 680
get_position_preds() (*pytext.models.output_layers.squad_output_layer.SquadOutputLayer method*), 610
get_post_training_snapshot_path() (*in module pytext.task.serialize*), 730
get_post_training_snapshot_path() (*pytext.task.serialize.CheckpointManager method*), 728
get_post_training_snapshot_path() (*pytext.task.serialize.PyTextCheckpointManagerInterface method*), 729
get_pred() (*pytext.models.BaseModel method*), 706
get_pred() (*pytext.models.disjoint_multitask_model.DisjointMultitaskM*

method), 688
get_pred() (pytext.models.model.BaseModel method), 694
get_pred() (pytext.models.output_layers.ClassificationOutputLayer method), 616
get_pred() (pytext.models.output_layers.CRFOutputLayer method), 615
get_pred() (pytext.models.output_layers.DenseRetrievalOutputLayer method), 621
get_pred() (pytext.models.output_layers.distance_output_layer.DenseRetrievalOutputLayer method), 597
get_pred() (pytext.models.output_layers.distance_output_layer.PairwiseCosineDistanceOutputLayer method), 599
get_pred() (pytext.models.output_layers.doc_classification_output_layer.BinaryClassificationOutputLayer method), 599
get_pred() (pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer method), 600
get_pred() (pytext.models.output_layers.doc_classification_output_layer.MultiLabelOutputLayer method), 601
get_pred() (pytext.models.output_layers.doc_classification_output_layer.MultiLabelSoftmaxOutputLayer method), 601
get_pred() (pytext.models.output_layers.doc_regression_output_layer.PairwiseCosineRegressionOutputLayer method), 602
get_pred() (pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer method), 602
get_pred() (pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer method), 604
get_pred() (pytext.models.output_layers.lm_output_layer.LMOutputLayer method), 605
get_pred() (pytext.models.output_layers.multi_label_classification_output_layer.MultiLabelClassificationLayer method), 606
get_pred() (pytext.models.output_layers.output_layer_base.OutputLayerBase method), 608
get_pred() (pytext.models.output_layers.OutputLayerBase method), 614
get_pred() (pytext.models.output_layers.pairwise_ranking_output_layer.PairwiseRankingOutputLayer method), 608
get_pred() (pytext.models.output_layers.PairwiseCosineDistanceOutputLayer method), 619
get_pred() (pytext.models.output_layers.PairwiseCosineRegressionOutputLayer method), 620
get_pred() (pytext.models.output_layers.PairwiseRankingOutputLayer method), 618
get_pred() (pytext.models.output_layers.RegressionOutputLayer method), 617
get_pred() (pytext.models.output_layers.squad_output_layer.SquadOutputLayer method), 610
get_pred() (pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer method), 611
get_pred() (pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method), 613
get_pred() (pytext.models.output_layers.WordTaggingOutputLayer method), 617
get_pred() (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserBase method), 664
get_pred() (pytext.models.seq_models.seq2seq_model.Seq2SeqModel method), 684
get_predict_iter() (pytext.data.data_handler.DataHandler method), 449
get_predict_iter() (pytext.data.DataHandler method), 486
get_privacy_spent() (pytext.optimizer.optimizer.PrivacyEngine method), 719
get_pred() (pytext.models.seq_models.tasks_decoder.LightConvDecoderTask method), 669
get_pred() (pytext.models.seq_models.tasks_decoder.LightConvDecoderTask method), 669
get_pred() (pytext.models.seq_models.tasks_decoder.projection_layers.DecoderWithOutputLayer method), 680
get_pred() (pytext.models.seq_models.tasks_decoder.projection_layers.DecoupledDecoderWithOutputLayer method), 680
get_path() (pytext.utils.path module), 769
get_r3f_layer() (pytext.models.r3f_models.R3FPyTextMixin method), 699
get_r3f_model_output() (pytext.models.r3f_models.R3FPyTextMixin method), 699
get_raw_str() (pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method), 540
get_representation_dim() (pytext.models.representations.representation_base.RepresentationBase method), 654
get_required_sparsifiable_params() (pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method), 711
get_root() (pytext.data.data_structures.annotation.Node_Info method), 416
get_sample_size() (pytext.models.r3f_models.R3FPyTextMixin method), 699
get_sentence() (pytext.models.roberta.RoBERTaR3F method), 701
get_sentence_markers() (pytext.data.featurizer.WordTaggingOutputFeaturizer method), 420
get_sentence_markers() (pytext.data.featurizer.featurizer.Featurizer method), 418
get_sentence_markers() (py-

*text.data.featurizer.simple_featurizer.SimpleFeaturizer* method), 419
get_sentence_markers() (*pytext.data.featurizer.SimpleFeaturizer* method), 421
get_shard_range() (*in module pytext.utils.distributed*), 762
get_sigmoid_scores() (*in module pytext.models.output_layers.distance_output_layer*), 599
get_similarities() (*pytext.loss.loss.PairwiseRankingLoss* static method), 509
get_similarities() (*pytext.loss.PairwiseRankingLoss* static method), 512
get_single_pred() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserModel* method), 664
get_sinusoidal_embedding() (*in module pytext.models.seq_models.positional*), 679
get_slots() (*in module pytext.metric_reporters.word_tagging_metric_reporter*), 533
get_sparsifiable_params() (*pytext.optimizer.sparsifiers.blockwise_sparsifier.BlockwiseMagnitudeSparsifier* method), 708
get_sparsifiable_params() (*pytext.optimizer.sparsifiers.sparsifier.CRF_SparsifierBase* method), 709
get_sparsifiable_params() (*pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier* method), 710
get_sparsifiable_params() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier* method), 711
get_sparsifiable_params() (*pytext.optimizer.sparsifiers.sparsifier.Sparsifier* method), 711
get_src_length() (*in module pytext.torchscript.seq2seq.seq2seq_rnn_decoder_utils*), 740
get_subclasses() (*in module pytext.utils.documentation*), 763
get_substring_from_offsets() (*in module pytext.utils.data*), 762
get_test_iter() (*pytext.data.data_handler.DataHandler* method), 449
get_test_iter() (*pytext.data.DataHandler* method), 486
get_test_iter() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler* method), 455
get_test_iter() (*pytext.data.DisjointMultitaskDataHandler* method), 488
get_test_iter_from_path() (*pytext.data.data_handler.DataHandler* method), 449
get_test_iter_from_path() (*pytext.data.DataHandler* method), 486
get_test_iter_from_raw_data() (*pytext.data.data_handler.DataHandler* method), 449
get_test_iter_from_raw_data() (*pytext.data.DataHandler* method), 486
get_texts_by_index() (*pytext.data.tensorizers.CharacterVocabTokenTensorizerScriptImpl* method), 465
get_texts_by_index() (*pytext.data.tensorizers.TensorizerScriptImpl* method), 475
get_texts_by_index() (*pytext.data.token_tensorizer.TokenTensorizerScriptImpl* method), 479
get_title() (*pytext.metric_reporters.channel.FileChannel* method), 515
get_title() (*pytext.metric_reporters.language_model_metric_reporter.* method), 524
get_title() (*pytext.metric_reporters.seq2seq_compositional.Composit* method), 529
get_title() (*pytext.metric_reporters.seq2seq_metric_reporter.Seq2Seq* method), 530
get_title() (*pytext.metric_reporters.squad_metric_reporter.SquadFile* method), 531
get_token_indices() (*pytext.data.data_structures.annotation.Node* method), 415
get_token_span() (*pytext.data.data_structures.annotation.Node* method), 415
get_tokens_by_index() (*pytext.data.tensorizers.CharacterVocabTokenTensorizerScriptImpl* method), 465
get_tokens_by_index() (*pytext.data.tensorizers.TensorizerScriptImpl* method), 475
get_tokens_by_index() (*pytext.data.token_tensorizer.TokenTensorizerScriptImpl* method), 479
get_train_iter() (*pytext.data.data_handler.DataHandler* method), 449
get_train_iter() (*pytext.data.DataHandler* method), 486
get_train_iter() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH*

*method*), 455
get_train_iter() (*pytext.data.DisjointMultitaskDataHandler method*), 488
get_train_iter_from_path() (*pytext.data.data_handler.DataHandler method*), 449
get_train_iter_from_path() (*pytext.data.DataHandler method*), 486
get_train_iter_from_raw_data() (*pytext.data.data_handler.DataHandler method*), 449
get_train_iter_from_raw_data() (*pytext.data.DataHandler method*), 486
get_transition_sparsity() (*pytext.optimizer.sparsifiers.sparsifier.CRF_SparsifierBase method*), 709
get_transitions() (*pytext.models.crf.CRF method*), 687
get_unk_index() (*pytext.data.utils.Vocabulary method*), 480
get_unk_index() (*pytext.torchscript.vocab.ScriptVocabulary method*), 754
get_weights_context() (*pytext.models.joint_model.IntentSlotModel method*), 692
GetTensor() (*in module pytext.utils.cuda*), 761
GLU (*pytext.config.module_config.Activation attribute*), 409
GPT2BPETest (*class in pytext.data.test.tokenizers_test*), 435
GPT2BPETokenizer (*class in pytext.data.tokenizers*), 438
GPT2BPETokenizer (*class in pytext.data.tokenizers.tokenizer*), 437
gpu_streams_for_distributed_training (*pytext.config.pytext_config.PyTextConfig attribute*), 412
gpus (*pytext.config.pytext_config.LogitsConfig attribute*), 412
graph_mode_quantize() (*pytext.models.roberta.RoBERTa method*), 700
graph_mode_quantize() (*pytext.models.two_tower_classification_model.TwoTowerClassificationModel method*), 702
graph_mode_quantize() (*pytext.models.TwoTowerClassificationModel method*), 707
GRU (*pytext.models.representations.stacked_bidirectional_rnn.RnnType attribute*), 656

## H

HAMMING (*pytext.loss.CostFunctionType attribute*), 512
HAMMING (*pytext.loss.structured_loss.CostFunctionType attribute*), 510
hamming_distance() (*in module pytext.loss.structured_loss*), 510
has_added_tokens() (*pytext.data.utils.VocabBuilder method*), 480
hidden_dims (*pytext.models.decoders.mlp_decoder.MLPDecoder attribute*), 567
hidden_dims (*pytext.models.decoders.mlp_decoder.MLPDecoder.Config attribute*), 156
hidden_dims (*pytext.models.decoders.MLPDecoder attribute*), 570
hidden_size (*pytext.models.representations.stacked_bidirectional_rnn.S attribute*), 250
HierarchicalTimer (*class in pytext.utils.timing*), 769
Highway (*class in pytext.models.embeddings.char_embedding*), 573
highway_layers (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 573
highway_layers (*pytext.models.embeddings.CharacterEmbedding attribute*), 586
HingeLoss (*class in pytext.loss*), 511
HingeLoss (*class in pytext.loss.loss*), 508
HogwildTrainer (*class in pytext.trainers*), 760
HogwildTrainer (*class in pytext.trainers.hogwild_trainer*), 754
HogwildTrainer_Deprecated (*class in pytext.trainers*), 760
HogwildTrainer_Deprecated (*class in pytext.trainers.hogwild_trainer*), 754
HuggingFaceBertSentenceEncoder (*class in pytext.models.representations.huggingface_bert_sentence_encoder*) 649
HuggingFaceElectraSentenceEncoder (*class in pytext.models.representations.huggingface_electra_sentence_encod* 650
HYBRID (*pytext.models.seq_models.positional.PostionalEmbedType attribute*), 679

## I

i_label_name (*pytext.utils.data.Slot attribute*), 762
I_LABEL_PREFIX (*pytext.utils.data.Slot attribute*), 761
IGNORE_LOSS (*pytext.common.constants.BatchContext attribute*), 400
ignore_loss_for_unsupported (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGConstraint attribute*), 270

import_tests_module() (*in module pytext.utils.test*), 769
in_dim (*pytext.models.decoders.decoder_base.DecoderBase attribute*), 565
in_dim (*pytext.models.decoders.DecoderBase attribute*), 569
include_dirs (*pytext.config.pytext_config.PyTextConfig attribute*), 412
IncorrectTypeError, 414
increase_sparsity() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
INDEX (*pytext.common.constants.BatchContext attribute*), 400
index() (*pytext.data.xlm_dictionary.Dictionary method*), 481
index_data() (*pytext.data.xlm_dictionary.Dictionary static method*), 481
Infer (*class in pytext.utils.lazy*), 764
inference_interface (*pytext.config.pytext_config.ExportConfig attribute*), 411
inference_interface (*pytext.config.pytext_config.PyTextConfig attribute*), 412
init_feature_metadata() (*pytext.data.data_handler.DataHandler method*), 449
init_feature_metadata() (*pytext.data.DataHandler method*), 486
init_hidden() (*pytext.models.language_models.lmlstm.LMLSTM method*), 595
init_lazy_modules() (*in module pytext.utils.lazy*), 765
init_metadata() (*pytext.data.data_handler.DataHandler method*), 449
init_metadata() (*pytext.data.DataHandler method*), 486
init_metadata() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler method*), 455
init_metadata() (*pytext.data.DisjointMultitaskDataHandler method*), 488
init_metadata_from_path() (*pytext.data.data_handler.DataHandler method*), 449
init_metadata_from_path() (*pytext.data.DataHandler method*), 486
init_metadata_from_raw_data() (*pytext.data.data_handler.DataHandler method*), 449
init_metadata_from_raw_data() (*pytext.data.DataHandler method*), 486
init_params() (*in module pytext.models.roberta*), 702
INIT_PREDICT (*pytext.config.module_config.ExporterType attribute*), 410
init_rows (*pytext.data.test.tensorizers_test.String2DListTensorizerTest attribute*), 434
INIT_SEQ (*pytext.common.constants.VocabMeta attribute*), 403
init_target_metadata() (*pytext.data.data_handler.DataHandler method*), 449
init_target_metadata() (*pytext.data.DataHandler method*), 487
INIT_TOKEN (*pytext.common.constants.VocabMeta attribute*), 403
init_weights() (*pytext.models.representations.pooling.SelfAttention method*), 654
initialize() (*in module pytext.optimizer.fp16_optimizer*), 717
initialize() (*pytext.data.bert_tensorizer.BERTTensorizerBase method*), 442
initialize() (*pytext.data.squad_tensorizer.SquadTensorizer method*), 462
initialize() (*pytext.data.Tensorizer method*), 492
initialize() (*pytext.data.tensorizers.AnnotationNumberizer method*), 463
initialize() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 464
initialize() (*pytext.data.tensorizers.CharacterVocabTokenTensorizer method*), 465
initialize() (*pytext.data.tensorizers.Float1DListTensorizer method*), 466
initialize() (*pytext.data.tensorizers.FloatListTensorizer method*), 467
initialize() (*pytext.data.tensorizers.GazetteerTensorizer method*), 468
initialize() (*pytext.data.tensorizers.Integer1DListTensorizer method*), 469
initialize() (*pytext.data.tensorizers.LabelListRankTensorizer method*), 470
initialize() (*pytext.data.tensorizers.LabelTensorizer method*), 470
initialize() (*pytext.data.tensorizers.SeqTokenTensorizer method*), 472
initialize() (*pytext.data.tensorizers.SlotLabelTensorizer method*), 473
initialize() (*pytext.data.tensorizers.String2DListTensorizer method*), 474
initialize() (*pytext.data.tensorizers.Tensorizer method*), 475
initialize() (*pytext.data.tensorizers.TokenTensorizer

*method*), 476
initialize() (*pytext.data.tensorizers.UidTensorizer method*), 477
initialize() (*pytext.data.token_tensorizer.ScriptBasedTokenTensorizer method*), 478
initialize() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
initialize() (*pytext.optimizer.sparsifiers.sparsifier.Sparsifier method*), 711
initialize_embeddings_weights() (*pytext.utils.embeddings.PretrainedEmbedding method*), 763
initialize_state() (*pytext.optimizer.madgrad.MADGRAD method*), 718
initialize_tensorizers() (*in module pytext.data.tensorizers*), 477
InitPredictNetExporter (*class in pytext.exporters*), 497
InitPredictNetExporter (*class in pytext.exporters.custom_exporters*), 493
input_linearity (*pytext.models.representations.augmented_lstm.AugmentedLSTMCell attribute*), 639
input_names (*pytext.exporters.exporter.ModelExporter attribute*), 494
input_names (*pytext.exporters.ModelExporter attribute*), 496
input_size() (*in module pytext.torchscript.batchutils*), 746
input_start_indices (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 576
input_start_indices (*pytext.models.embeddings.EmbeddingList attribute*), 583
InputRecord (*class in pytext.data.featurizer*), 420
InputRecord (*class in pytext.data.featurizer.featurizer*), 418
INPUTS_PAIR (*pytext.models.pair_classification_model.PairwiseModel attribute*), 697
instance_id (*pytext.models.seq_models.base.PyTextSeq2SeqModule attribute*), 667
Integer1DListTensorizer (*class in pytext.data.tensorizers*), 469
Intent (*class in pytext.data.data_structures.annotation*), 415
intent_confusions (*pytext.metrics.intent_slot_metrics.IntentSlotConfusions attribute*), 547
intent_metrics (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics attribute*), 547
intent_slot_nesting (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGConstraint attribute*), 270
intents (*pytext.metrics.intent_slot_metrics.IntentsAndSlots attribute*), 548
IntentsAndSlots (*class in pytext.metrics.intent_slot_metrics*), 548
IntentSlotConfusions (*class in pytext.metrics.intent_slot_metrics*), 547
IntentSlotMetricReporter (*class in pytext.metric_reporters*), 539
IntentSlotMetricReporter (*class in pytext.metric_reporters.intent_slot_detection_metric_reporter*), 523
IntentSlotMetrics (*class in pytext.metrics.intent_slot_metrics*), 547
IntentSlotModel (*class in pytext.models.joint_model*), 691
IntentSlotModelDecoder (*class in pytext.models.decoders*), 571
IntentSlotModelDecoder (*class in pytext.models.decoders.intent_slot_model_decoder*), 566
IntentSlotOutputLayer (*class in pytext.models.output_layers.intent_slot_output_layer*), 603
IntentSlotScores (*class in pytext.models.output_layers.intent_slot_output_layer*), 604
IntentSlotTask (*class in pytext.task.tasks*), 732
InvalidMethodInvocation, 411
is_absolute_path() (*in module pytext.utils.path*), 769
is_component_class() (*in module pytext.config.utils*), 415
is_intent_nonterminal() (*in module pytext.data.data_structures.annotation*), 417
is_number() (*in module pytext.utils.data*), 762
is_slot_nonterminal() (*in module pytext.data.data_structures.annotation*), 417
is_type_specifier() (*in module pytext.config.config_adapter*), 405
is_unsupported() (*in module pytext.data.data_structures.annotation*), 417
is_valid_nonterminal() (*in module pytext.data.data_structures.annotation*), 417
is_valid_tree() (*in module pytext.metric_reporters.compositional_utils*), 521
items() (*pytext.config.pytext_config.ConfigBase method*), 411
iter_to_set_epoch (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatchIterator attribute*), 456
iterators (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatchIterator*

*attribute*), 456

## J

JointCNNRepresentation (*class in pytext.models.representations.jointcnn_rep*), 650

## K

kernel_num (*pytext.config.module_config.CNNParams attribute*), 410
kernel_sizes (*pytext.config.module_config.CNNParams attribute*), 410
KLDivergenceBCELoss (*class in pytext.loss*), 511
KLDivergenceBCELoss (*class in pytext.loss.loss*), 508
KLDivergenceCELoss (*class in pytext.loss*), 511
KLDivergenceCELoss (*class in pytext.loss.loss*), 508

## L

L0_projection_sparsifier (*class in pytext.optimizer.sparsifiers.sparsifier*), 709
label (*pytext.data.data_structures.node.Node attribute*), 417
label (*pytext.metric_reporters.word_tagging_metric_reporter.Span attribute*), 533
label (*pytext.metrics.intent_slot_metrics.Node attribute*), 548
label_accuracy (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556
LABEL_AVG_PRECISION (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 518
label_confusions_map (*pytext.metrics.PerLabelConfusions attribute*), 557, 558
LABEL_F1 (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 518
LABEL_ROC_AUC (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 518
label_scores (*pytext.metrics.LabelListPrediction attribute*), 554
label_scores (*pytext.metrics.LabelPrediction attribute*), 554
label_weights (*pytext.config.field_config.DocLabelConfig attribute*), 408
LabelListPrediction (*class in pytext.metrics*), 554
LabelListRankTensorizer (*class in pytext.data.tensorizers*), 469
LabelListTensorizer (*class in pytext.data.tensorizers*), 470
LabelPrediction (*class in pytext.metrics*), 554
labels (*pytext.data.data_handler.DataHandler attribute*), 448
labels (*pytext.data.DataHandler attribute*), 485
LABELS_COLUMN (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter attribute*), 524
LABELS_COLUMN (*pytext.metric_reporters.LanguageModelMetricReporter attribute*), 540
LabelSmoothedCrossEntropyLoss (*class in pytext.loss*), 512
LabelSmoothedCrossEntropyLoss (*class in pytext.loss.loss*), 508
LabelSmoothingLoss (*class in pytext.loss*), 512
LabelSmoothingLoss (*class in pytext.loss.regularized_loss*), 509
LabelTensorizer (*class in pytext.data.tensorizers*), 470
lagrange_multiplier() (*in module pytext.utils.loss*), 767
LagrangeMultiplier (*class in pytext.utils.loss*), 766
Lamb (*class in pytext.optimizer.lamb*), 717
LANGUAGE_ID (*pytext.common.constants.DFColumn attribute*), 401
LANGUAGE_ID_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
LanguageModelChannel (*class in pytext.metric_reporters.language_model_metric_reporter*), 524
LanguageModelMetric (*class in pytext.metrics.language_model_metrics*), 551
LanguageModelMetricReporter (*class in pytext.metric_reporters*), 540
LanguageModelMetricReporter (*class in pytext.metric_reporters.language_model_metric_reporter*), 524
languages (*pytext.torchscript.utils.ScriptBatchInput attribute*), 753
LastTimestepPool (*class in pytext.models.representations.pooling*), 652
layer_wise_analysis() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
Lazy (*class in pytext.utils.lazy*), 764
lazy_property (*class in pytext.utils.lazy*), 765
LEAKYRELU (*pytext.config.module_config.Activation attribute*), 409
LEARNED (*pytext.models.seq_models.positional.PostionalEmbedType attribute*), 679
LearnedPositionalEmbedding (*class in pytext.models.seq_models.positional*), 678
learning_rates() (*in module pytext.optimizer.optimizers*), 719

LEFT (*pytext.models.decoders.mlp_decoder_two_tower.ExpitTower attribute*), 568
LEN_ONLY (*pytext.models.seq_models.mask_generator.BeamRankingAlgorithm attribute*), 674
length() (*in module pytext.models.seq_models.mask_generator*), 675
LENGTH_CONDITIONED_AVERAGE_TOKEN_LPROB (*pytext.models.seq_models.mask_generator.BeamRankingAlgorithm attribute*), 674
LENGTH_CONDITIONED_AVERAGE_TOKEN_LPROB_MULTIPLIED (*pytext.models.seq_models.mask_generator.BeamRankingAlgorithm attribute*), 674
length_conditioned_avg_lprob_rank() (*in module pytext.models.seq_models.mask_generator*), 675
length_conditioned_avg_lprob_rank_mul() (*in module pytext.models.seq_models.mask_generator*), 676
LENGTH_CONDITIONED_RANK (*pytext.models.seq_models.mask_generator.BeamRankingAlgorithm attribute*), 674
length_conditioned_rank() (*in module pytext.models.seq_models.mask_generator*), 676
LENGTH_CONDITIONED_RANK_MUL (*pytext.models.seq_models.mask_generator.BeamRankingAlgorithm attribute*), 674
length_conditioned_rank_mul() (*in module pytext.models.seq_models.mask_generator*), 676
LightConvDecoder (*class in pytext.models.seq_models.conv_decoder*), 668
LightConvDecoderBase (*class in pytext.models.seq_models.conv_decoder*), 669
LightConvDecoderLayer (*class in pytext.models.seq_models.conv_decoder*), 669
LightConvDecoupledDecoder (*class in pytext.models.seq_models.conv_decoder*), 670
LightConvEncoder (*class in pytext.models.seq_models.conv_encoder*), 671
LightConvEncoderLayer (*class in pytext.models.seq_models.conv_encoder*), 672
LightweightConv (*class in pytext.models.seq_models.light_conv*), 673
limit_list() (*in module pytext.torchscript.batchutils*), 746
limit_listlist() (*in module pytext.torchscript.batchutils*), 746
limit_listlist_float() (*in module pytext.torchscript.batchutils*), 747
Linear() (*in module pytext.models.seq_models.utils*), 685
LinearDynamicPoolingBatcher (*class in pytext.data.dynamic_pooling_batcher*), 457
list() (*pytext.task.serialize.CheckpointManager method*), 728
list() (*pytext.task.serialize.PyTextCheckpointManagerInterface method*), 729
list_ancestors() (*pytext.data.data_structures.annotation.Node method*), 415
list_from_actions() (*in module pytext.data.data_structures.annotation*), 417
list_nonterminals() (*pytext.data.data_structures.annotation.Node method*), 415
list_terminals() (*pytext.data.data_structures.annotation.Node method*), 415
list_tokens() (*pytext.data.data_structures.annotation.Node method*), 416
list_tokens() (*pytext.data.data_structures.annotation.Tree method*), 416
listify() (*in module pytext.torchscript.batchutils*), 747
ListTensorizersTest (*class in pytext.data.test.tensorizers_test*), 433
lm_add_deprecated() (*in module pytext.config.config_adapter*), 405
LmFineTuning (*class in pytext.optimizer.scheduler*), 720
LMLSTM (*class in pytext.models.language_models.lmlstm*), 595
LMOutputLayer (*class in pytext.models.output_layers.lm_output_layer*), 604
LMTask (*class in pytext.task.tasks*), 732
load() (*in module pytext.task*), 737
load() (*in module pytext.task.serialize*), 730
load() (*pytext.data.sources.data_source.RootDataSource method*), 423
load() (*pytext.task.serialize.CheckpointManager method*), 728
load() (*pytext.task.serialize.PyTextCheckpointManagerInterface method*), 729
load_analysis_from_path() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
load_best_model() (*pytext.trainers.Trainer method*), 758
load_best_model() (*pytext.trainers.trainer.Trainer method*), 755
load_cached_embeddings() (*pytext.utils.embeddings.PretrainedEmbedding*

*method*), 764
load_checkpoint() (*in module pytext.task.serialize*), 730
load_config() (*in module pytext*), 773
load_float() (*in module pytext.data.sources.data_source*), 424
load_float_list() (*in module pytext.data.sources.data_source*), 424
load_int() (*in module pytext.data.sources.data_source*), 424
load_json() (*in module pytext.data.sources.data_source*), 424
load_json_string() (*in module pytext.data.sources.data_source*), 424
load_meta() (*pytext.fields.Field method*), 505
load_meta() (*pytext.fields.field.Field method*), 501
load_meta() (*pytext.fields.field.NestedField method*), 501
load_meta() (*pytext.fields.NestedField method*), 506
load_metadata() (*pytext.data.data_handler.DataHandler method*), 449
load_metadata() (*pytext.data.DataHandler method*), 487
load_metadata() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler method*), 455
load_metadata() (*pytext.data.DisjointMultitaskDataHandler method*), 489
load_pretrained_embeddings() (*pytext.utils.embeddings.PretrainedEmbedding method*), 764
load_roberta_state_dict() (*pytext.models.representations.transformer.sentence_encoder.SentenceEncoder method*), 628
load_roberta_state_dict() (*pytext.models.representations.transformer.SentenceEncoder method*), 635
load_slots() (*in module pytext.data.sources.data_source*), 424
load_snapshot_path (*pytext.config.pytext_config.PyTextConfig attribute*), 412
load_state_dict() (*pytext.models.distributed_model.DistributedModel method*), 689
load_state_dict() (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsemble method*), 590
load_state_dict() (*pytext.models.ensembles.BaggingIntentSlotEnsembleModel method*), 593
load_state_dict() (*pytext.models.representations.transformer_sentence_encoder.TransformerSentenceEncoder method*), 659
load_state_dict() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16OptimizerMixin method*), 713
load_state_dict() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_MemoryEfficientFP16Optimizer method*), 713
load_state_dict() (*pytext.optimizer.fp16_optimizer.FP16Optimizer method*), 714
load_state_dict() (*pytext.optimizer.fp16_optimizer.FP16OptimizerApex method*), 714
load_state_dict() (*pytext.optimizer.fp16_optimizer.FP16OptimizerDeprecated method*), 715
load_state_dict() (*pytext.optimizer.fp16_optimizer.GeneratorFP16Optimizer method*), 715
load_state_dict() (*pytext.optimizer.fp16_optimizer.PureFP16Optimizer method*), 716
load_state_dict() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 723
load_text() (*in module pytext.data.sources.data_source*), 424
load_v1() (*in module pytext.task.serialize*), 730
load_v2() (*in module pytext.task.serialize*), 730
load_v3() (*in module pytext.task.serialize*), 730
load_vocab() (*pytext.data.data_handler.DataHandler method*), 449
load_vocab() (*pytext.data.DataHandler method*), 487
load_vocab() (*pytext.data.tokenizers.tokenizer.WordPieceTokenizer static method*), 438
load_vocab() (*pytext.data.tokenizers.WordPieceTokenizer static method*), 439
load_vocab() (*pytext.torchscript.tokenizer.bpe.ScriptBPE static method*), 745
load_vocab() (*pytext.torchscript.tokenizer.ScriptBPE static method*), 746
locale (*pytext.data.featurizer.featurizer.InputRecord attribute*), 418
locale (*pytext.data.featurizer.InputRecord attribute*), 420
log_accelerator_feature_usage() (*in module pytext.utils.usage*), 770
log_class_usage() (*in module pytext.utils.usage*), 770
log_feature_usage() (*in module pytext.utils.usage*), 770
log_flow_usage() (*in module pytext.utils.usage*),

770
log_gradient (pytext.metric_reporters.metric_reporter.MetricReporter attribute), 527
log_gradient (pytext.metric_reporters.MetricReporter attribute), 535
log_loss() (pytext.metric_reporters.channel.TensorBoardChannel method), 516
LOG_PROBS (pytext.loss.loss.SourceType attribute), 509
LOG_PROBS (pytext.loss.SourceType attribute), 512
log_vector() (pytext.metric_reporters.channel.TensorBoardChannel method), 516
LOGITS (pytext.loss.loss.SourceType attribute), 509
LOGITS (pytext.loss.SourceType attribute), 512
LogitsConfig (class in pytext.config.pytext_config), 411
LogitsWriter (class in pytext.workflow), 771
LOGSUMEXP (pytext.config.module_config.PoolingType attribute), 410
LongTensor() (in module pytext.utils.cuda), 761
lookup_all() (pytext.data.utils.Vocabulary method), 480
lookup_all_internal() (pytext.data.utils.Vocabulary method), 480
lookup_tokens() (in module pytext.data.tensorizers), 477
LookupTokensTest (class in pytext.data.test.tensorizers_test), 433
Loss (class in pytext.loss), 511
Loss (class in pytext.loss.loss), 508
LOSS (pytext.config.component.ComponentType attribute), 403
loss (pytext.metrics.ClassificationMetrics attribute), 553
loss (pytext.metrics.intent_slot_metrics.AllMetrics attribute), 546
loss (pytext.metrics.seq2seq_metrics.Seq2SeqMetrics attribute), 552
loss_fn (pytext.models.output_layers.ClassificationOutputLayer attribute), 616
loss_fn (pytext.models.output_layers.doc_classification_output_layer.DocClassificationOutputLayer attribute), 600
loss_fn (pytext.models.output_layers.lm_output_layer.LMOutputLayer attribute), 605
loss_fn (pytext.models.output_layers.output_layer_base.OutputLayerBase attribute), 607
loss_fn (pytext.models.output_layers.OutputLayerBase attribute), 613
loss_fn (pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer attribute), 612
loss_fn (pytext.models.output_layers.WordTaggingOutputLayer attribute), 617
lotv_str() (pytext.data.data_structures.annotation.Tree method), 416
lower_is_better (pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMetricReporter attribute), 522
lower_is_better (pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter attribute), 525
lower_is_better (pytext.metric_reporters.LanguageModelMetricReporter attribute), 541
lower_is_better (pytext.metric_reporters.metric_reporter.MetricReporter attribute), 526, 527
lower_is_better (pytext.metric_reporters.metric_reporter.PureLossMetricReporter attribute), 528
lower_is_better (pytext.metric_reporters.MetricReporter attribute), 534, 535
lower_is_better (pytext.metric_reporters.PureLossMetricReporter attribute), 543
lower_is_better (pytext.metric_reporters.regression_metric_reporter.RegressionMetricReporter attribute), 529
lower_is_better (pytext.metric_reporters.RegressionMetricReporter attribute), 539
lower_is_better (pytext.metric_reporters.seq2seq_metric_reporter.Seq2SeqMetricReporter attribute), 530
lower_modules_to_accelerator() (in module pytext.task.accelerator_lowering), 725
lowercase_tokens (pytext.data.tensorizers.VocabFileConfig attribute), 477
lstm (pytext.models.representations.bilstm.BiLSTM attribute), 641
lstm (pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute), 642
lstm (pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute), 229
lstm (pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute), 643
lstm (pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttention attribute), 645
LSTM (pytext.models.representations.stacked_bidirectional_rnn.RnnType attribute), 656
lstm (pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN attribute), 657
LSTM() (pytext.models.seq_models.rnn_encoder.BiLSTM static method), 682
lstm_dim (pytext.models.representations.augmented_lstm.AugmentedLSTM attribute), 226
lstm_dim (pytext.models.representations.bilstm.BiLSTM.Config attribute), 228

LSTMSequenceEncoder (*class in pytext.models.seq_models.rnn_encoder*), 682

## M

MACRO_F1 (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 518
macro_prf1_metrics (*pytext.metrics.ClassificationMetrics attribute*), 553
macro_scores (*pytext.metrics.MacroPRF1Metrics attribute*), 555
macro_scores (*pytext.metrics.PRF1Metrics attribute*), 556
MacroPRF1Metrics (*class in pytext.metrics*), 555
MacroPRF1Scores (*class in pytext.metrics*), 555
MADGRAD (*class in pytext.optimizer.madgrad*), 717
MAELoss (*class in pytext.loss*), 511
MAELoss (*class in pytext.loss.loss*), 508
make_batch_texts() (*in module pytext.torchscript.batchutils*), 747
make_batch_texts_dense() (*in module pytext.torchscript.batchutils*), 747
make_config() (*pytext.utils.config_utils.MockConfigLoader method*), 761
make_config_from_dict() (*pytext.utils.config_utils.MockConfigLoader method*), 761
make_positions() (*in module pytext.models.representations.transformer.positional_embedding*), 625
make_positions() (*in module pytext.models.seq_models.utils*), 685
make_prediction_texts() (*in module pytext.torchscript.batchutils*), 747
make_prediction_texts_dense() (*in module pytext.torchscript.batchutils*), 747
make_prediction_tokens() (*in module pytext.torchscript.batchutils*), 747
make_vocab() (*pytext.data.utils.VocabBuilder method*), 480
MASK (*pytext.common.constants.SpecialTokens attribute*), 402
mask_and_tensorize() (*pytext.data.masked_tensorizer.MaskedTokenTensorizer method*), 458
masked_softmax() (*in module pytext.models.seq_models.attention*), 666
MaskedLanguageModel (*class in pytext.models.masked_lm*), 692
MaskedLengthPredictionModule (*class in pytext.models.seq_models.nar_length*), 677
MaskedLMMetricReporter (*class in pytext.metric_reporters.language_model_metric_reporter*), 525
MaskedLMTask (*class in pytext.task.tasks*), 732
MaskedSequenceGenerator (*class in pytext.models.seq_models.mask_generator*), 674
MaskedTokenTensorizer (*class in pytext.data.masked_tensorizer*), 457
MaskedVocabBuilder (*class in pytext.data.masked_util*), 458
MaskEverything (*class in pytext.data.masked_util*), 458
MASKING_FUNCTION (*pytext.config.component.ComponentType attribute*), 404
MaskingFunction (*class in pytext.data.masked_util*), 458
MaskingStrategy (*class in pytext.models.masking_utils*), 692
MaskTensorizersTest (*class in pytext.data.test.mask_tensorizers_test*), 432
master_params() (*in module pytext.optimizer.fp16_optimizer*), 717
MAX (*pytext.config.module_config.PerplexityType attribute*), 410
MAX (*pytext.config.module_config.PoolingType attribute*), 410
max_clip_norm (*pytext.trainers.Trainer attribute*), 758
max_clip_norm (*pytext.trainers.trainer.Trainer attribute*), 755
max_decoder_positions() (*pytext.models.seq_models.conv_model.CNNModel method*), 673
max_decoder_positions() (*pytext.models.seq_models.rnn_encoder_decoder.RNNModel method*), 683
max_decoder_positions() (*pytext.models.seq_models.seq2seq_model.Seq2SeqModel method*), 684
max_error (*pytext.metrics.calibration_metrics.CalibrationMetrics attribute*), 545
max_positions() (*pytext.models.representations.transformer.positional_embedding.Po method*), 625
max_positions() (*pytext.models.representations.transformer.PositionalEmbedding method*), 633
max_positions() (*pytext.models.seq_models.conv_decoder.LightConvDecoderBase method*), 669
max_positions() (*pytext.models.seq_models.conv_encoder.LightConvEncoder method*), 672
max_positions() (*py-*

*text.models.seq_models.positional.SinusoidalPositionalEmbedding* *method*), 679
max_positions() (*pytext.models.seq_models.rnn_decoder.RNNDecoderBase* *method*), 682
max_positions() (*pytext.models.seq_models.rnn_encoder.LSTMSequenceEncoder* *method*), 682
max_seq_len (*pytext.data.data_handler.DataHandler* *attribute*), 448
max_seq_len (*pytext.data.DataHandler* *attribute*), 486
max_source_positions (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig* *attribute*), 671
max_target_positions (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig* *attribute*), 668
max_target_positions (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig* *attribute*), 671
max_tokens() (*in module pytext.torchscript.batchutils*), 747
max_vocab() (*pytext.data.xlm_dictionary.Dictionary* *method*), 481
MaxPool (*class in pytext.models.representations.pooling*), 653
maybe_accumulate_gradients() (*in module pytext.trainers.trainer*), 757
maybe_float() (*in module pytext.utils.precision*), 769
maybe_half() (*in module pytext.utils.precision*), 769
maybe_layer_norm() (*pytext.models.seq_models.conv_decoder.LightConvDecoderLayer* *method*), 670
maybe_layer_norm() (*pytext.models.seq_models.conv_encoder.LightConvEncoderLayer* *method*), 673
maybe_log_normalize() (*in module pytext.loss.loss*), 509
MCC (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric* *attribute*), 518
mcc (*pytext.metrics.ClassificationMetrics attribute*), 553
MEAN (*pytext.config.module_config.PerplexityType* *attribute*), 410
MEAN (*pytext.config.module_config.PoolingType* *attribute*), 410
mean() (*in module pytext.models.seq_models.nar_length*), 677
MEAN_RECIPROCAL_RANK (*pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetrievalMetricNames* *attribute*), 521
mean_reciprocal_rank (*pytext.metrics.dense_retrieval_metrics.DenseRetrievalMetrics* *attribute*), 546
MeanEmbedding (*class in pytext.models.representations.pooling*), 653
MEDIAN (*pytext.config.module_config.PerplexityType* *attribute*), 410
MemoryEfficientFP16OptimizerFairseq (*class in pytext.optimizer.fp16_optimizer*), 716
merge_input_projection() (*in module pytext.models.representations.transformer.sentence_encoder*), 628
merge_session() (*pytext.data.sources.session.SessionDataSource* *method*), 426
merge_sub_models() (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntent* *method*), 590
merge_sub_models() (*pytext.models.ensembles.BaggingIntentSlotEnsembleModel* *method*), 593
merge_sub_models() (*pytext.models.ensembles.ensemble.EnsembleModel* *method*), 591
merge_sub_models() (*pytext.models.ensembles.EnsembleModel* *method*), 594
merge_token_labels_by_bio() (*in module pytext.utils.data*), 762
merge_token_labels_by_label() (*in module pytext.utils.data*), 762
merge_token_labels_to_slot() (*in module pytext.utils.data*), 762
metadata_to_save() (*pytext.data.data_handler.DataHandler* *method*), 450
metadata_to_save() (*pytext.data.DataHandler* *method*), 487
metadata_to_save() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH* *method*), 455
metadata_to_save() (*pytext.data.DisjointMultitaskDataHandler* *method*), 489
Meter (*class in pytext.utils.meter*), 768
metric_name (*pytext.metric_reporters.channel.TensorBoardChannel* *attribute*), 515
METRIC_REPORTER (*pytext.config.component.ComponentType* *attribute*), 404
MetricReporter (*class in pytext.metric_reporters*), 534
MetricReporterMetricNames (*class in pytext.metric_reporters.metric_reporter*), 526
MetricTensorizer (*class in pytext.data.tensorizers*), 470
micro_scores (*pytext.metrics.PRF1Metrics* *at-*

*tribute*), 556
migrate_to_new_data_handler() (*in module pytext.config.config_adapter*), 405
MIN (*pytext.config.module_config.PerplexityType attribute*), 410
min_count() (*pytext.data.xlm_dictionary.Dictionary method*), 481
min_counts (*pytext.data.tensorizers.VocabConfig attribute*), 477
MissingValueError, 414
mlp (*pytext.models.decoders.mlp_decoder.MLPDecoder attribute*), 567
mlp (*pytext.models.decoders.MLPDecoder attribute*), 570
mlp_decoder (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention.Config attribute*), 229
MLPDecoder (*class in pytext.models.decoders*), 570
MLPDecoder (*class in pytext.models.decoders.mlp_decoder*), 567
MLPDecoderQueryResponse (*class in pytext.models.decoders.mlp_decoder_query_response*), 568
MLPDecoderTwoTower (*class in pytext.models.decoders.mlp_decoder_two_tower*), 568
MLPEmbedding (*class in pytext.models.embeddings*), 588
MLPEmbedding (*class in pytext.models.embeddings.mlp_embedding*), 577
MLPFeatConfig (*in module pytext.config.field_config*), 409
MockConfigLoader (*class in pytext.utils.config_utils*), 761
Model (*class in pytext.models*), 704
Model (*class in pytext.models.model*), 694
MODEL (*pytext.config.component.ComponentType attribute*), 404
MODEL2 (*pytext.config.component.ComponentType attribute*), 404
MODEL_FEATS (*pytext.common.constants.DFColumn attribute*), 401
ModelExporter (*class in pytext.exporters*), 496
ModelExporter (*class in pytext.exporters.exporter*), 494
ModelInput (*class in pytext.config.contextual_intent_slot*), 407
ModelInput (*class in pytext.config.doc_classification*), 408
ModelInput (*class in pytext.config.pair_classification*), 411
ModelInput (*class in pytext.config.query_document_pairwise_ranking*), 414
ModelInputBase (*class in pytext.models.model*), 696
ModelInputConfig (*class in pytext.config.contextual_intent_slot*), 407
ModelInputConfig (*class in pytext.config.doc_classification*), 408
ModelInputConfig (*class in pytext.config.pair_classification*), 411
ModelInputConfig (*class in pytext.config.query_document_pairwise_ranking*), 414
ModelInputMeta (*class in pytext.models.model*), 696
models (*pytext.models.ensembles.bagging_doc_ensemble.BaggingDocEns attribute*), 182
models (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingI attribute*), 182
models (*pytext.models.ensembles.ensemble.EnsembleModel attribute*), 591
models (*pytext.models.ensembles.EnsembleModel attribute*), 593
Module (*class in pytext.models.module*), 696
MODULE (*pytext.config.component.ComponentType attribute*), 404
ModuleConfig (*in module pytext.config.module_config*), 410
modules_save_dir (*pytext.config.pytext_config.PyTextConfig attribute*), 412
move_epoch_size() (*in module pytext.config.config_adapter*), 405
move_state_dict_to_cpu() (*pytext.trainers.Trainer method*), 758
move_state_dict_to_cpu() (*pytext.trainers.trainer.Trainer method*), 756
move_state_dict_to_gpu() (*pytext.trainers.Trainer method*), 758
move_state_dict_to_gpu() (*pytext.trainers.trainer.Trainer method*), 756
mse (*pytext.metrics.RegressionMetrics attribute*), 558, 559
MSELoss (*class in pytext.loss*), 511
MSELoss (*class in pytext.loss.loss*), 508
MulticlassOutputLayer (*class in pytext.models.output_layers.doc_classification_output_layer*), 601
MultiheadAttention (*class in pytext.models.seq_models.attention*), 665
MultiheadLinearAttention (*class in pytext.models.representations.transformer*), 631
MultiheadLinearAttention (*class in pytext.models.representations.transformer.multihead_linear_attentio*, 624
MultiheadSelfAttention (*class in pytext.models.representations.transformer*),

633
MultiheadSelfAttention (*class in pytext.models.representations.transformer.multihead_attention*), 623
MultiLabelClassificationLayer (*class in pytext.models.output_layers.multi_label_classification_layer*), 605
MultiLabelClassificationMetricReporter (*class in pytext.metric_reporters*), 537
MultiLabelClassificationMetricReporter (*class in pytext.metric_reporters.classification_metric_reporter*), 518
MultiLabelClassificationScores (*class in pytext.models.output_layers.multi_label_classification_layer*), 606
MultiLabelDecoder (*class in pytext.models.decoders.multilabel_decoder*), 569
MultiLabelOutputLayer (*class in pytext.models.output_layers.doc_classification_output_layer*), 600
MultiLabelSequenceTaggingMetricReporter (*class in pytext.metric_reporters*), 538
MultiLabelSequenceTaggingMetricReporter (*class in pytext.metric_reporters.word_tagging_metric_reporter*), 532
MultiLabelSoftClassificationMetrics (*class in pytext.metrics*), 555
MultiLabelSoftMarginLoss (*class in pytext.loss*), 511
MultiLabelSoftMarginLoss (*class in pytext.loss.loss*), 508
MultilingualTSVDataSource (*class in pytext.data.sources.tsv*), 427
MultiplicativeAttention (*class in pytext.models.representations.attention*), 637
MULTIPLY (*pytext.config.module_config.SlotAttentionType attribute*), 410
multiply_grads() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16OptimizerMixin method*), 713
multiply_grads() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_MemoryEfficientFP16OptimizerMixin method*), 713
multiply_grads() (*pytext.optimizer.optimizers.Optimizer method*), 718

## N

NAREncoderUtility (*class in pytext.models.seq_models.nar_modules*), 678
NARSamplewiseSequenceLoss (*class in pytext.loss*), 512
NARSamplewiseSequenceLoss (*class in pytext.loss.regularized_loss*), 509
NARSeq2SeqOutputLayer (*class in pytext.models.seq_models.nar_output_layer*), 678
NARSequenceLoss (*class in pytext.loss*), 512
NARSequenceLoss (*class in pytext.loss.regularized_loss*), 509
NaturalBatchSampler (*class in pytext.data*), 491
NaturalBatchSampler (*class in pytext.data.batch_sampler*), 440
ndigits_precision (*pytext.config.pytext_config.LogitsConfig attribute*), 412
NEG_RESPONSE (*pytext.config.query_document_pairwise_ranking.ModelI attribute*), 414
neg_response (*pytext.config.query_document_pairwise_ranking.ModelI attribute*), 414
NEGATIVE_LOSS (*pytext.metric_reporters.classification_metric_reporter.ComparableC attribute*), 518
NEGATIVE_LOSS (*pytext.metric_reporters.dense_retrieval_metric_reporter.DenseRetri attribute*), 521
NERMetricReporter (*class in pytext.metric_reporters*), 543
NERMetricReporter (*class in pytext.metric_reporters.word_tagging_metric_reporter*), 532
NestedField (*class in pytext.fields*), 506
NestedField (*class in pytext.fields.field*), 501
new_tasks_rename() (*in module pytext.config.config_adapter*), 405
NewBertClassificationTask (*class in pytext.task.tasks*), 733
NewBertModel (*class in pytext.models.bert_classification_models*), 686
NewBertPairClassificationTask (*class in pytext.task.tasks*), 733
NewBertRegressionModel (*class in pytext.models.bert_regression_model*), 686
NewDisjointMultitaskOptimizer (*class in pytext.task.disjoint_multitask*), 726
NewDisjointMultitaskModel (*class in pytext.models.disjoint_multitask_model*), 688
NewTask (*class in pytext.task*), 735
NewTask (*class in pytext.task.new_task*), 727
NLLLoss (*class in pytext.loss*), 511
NLLLoss (*class in pytext.loss.loss*), 508
NO_ATTENTION (*pytext.config.module_config.SlotAttentionType attribute*), 410

NO_LABEL_SLOT (*pytext.utils.data.Slot attribute*), 761
NO_POOL (*pytext.models.representations.transformer_sentence_encoder_base.PoolingMethod attribute*), 659
no_slots_inside_unsupported (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGConstraints attribute*), 270
no_token_positional_embeddings (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
no_token_positional_embeddings (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute*), 671
no_tokenize() (*in module pytext.utils.data*), 762
Node (*class in pytext.data.data_structures.annotation*), 415
Node (*class in pytext.data.data_structures.node*), 417
Node (*class in pytext.metrics.intent_slot_metrics*), 548
Node_Info (*class in pytext.data.data_structures.annotation*), 416
node_to_metrics_node() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter static method*), 520
node_to_metrics_node() (*pytext.metric_reporters.CompositionalMetricReporter static method*), 542
NodesPredictionPair (*class in pytext.metrics.intent_slot_metrics*), 548
noise_type (*pytext.models.r3f_models.R3FConfigOptions attribute*), 699
NONE (*pytext.config.module_config.PoolingType attribute*), 410
NONE (*pytext.models.decoders.mlp_decoder_two_tower.ExportType attribute*), 568
NONE (*pytext.models.seq_models.mask_generator.EmbedQuantizeType attribute*), 674
nonify_listlist_float() (*in module pytext.torchscript.batchutils*), 747
NoOpMaskingFunction (*class in pytext.data.masked_util*), 458
NoPool (*class in pytext.models.representations.pooling*), 653
norm_cosine (*pytext.models.output_layers.distance_output_layer.OutputSelector attribute*), 598
NORMAL (*pytext.models.r3f_models.R3FNoiseType attribute*), 699
normalize() (*pytext.torchscript.tensorizer.normalizer.VectorNormalizer method*), 741
normalize() (*pytext.torchscript.tensorizer.VectorNormalizer method*), 743
normalize_embeddings() (*in module pytext.models.utils*), 703
normalize_token() (*pytext.utils.embeddings.PretrainedEmbedding method*), 764
NtokensTensorizer (*class in pytext.data.tensorizers*), 471
NUM (*pytext.data.tensorizers.ByteTensorizer attribute*), 463
NUM (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 116
NUM_BYTES (*pytext.data.tensorizers.ByteTokenTensorizer attribute*), 463
num_classes (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 116
num_emb_modules (*pytext.models.embeddings.embedding_base.EmbeddingBase attribute*), 576
num_emb_modules (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 576
num_emb_modules (*pytext.models.embeddings.EmbeddingBase attribute*), 582
num_emb_modules (*pytext.models.embeddings.EmbeddingList attribute*), 583
num_examples (*pytext.metrics.dense_retrieval_metrics.DenseRetrievalM attribute*), 545, 546
num_examples (*pytext.metrics.PairwiseRankingMetrics attribute*), 557
num_examples (*pytext.metrics.RegressionMetrics attribute*), 558, 559
num_examples (*pytext.metrics.squad_metrics.SquadMetrics attribute*), 552
num_label (*pytext.metrics.MacroPRF1Scores attribute*), 555
num_labels (*pytext.metrics.MacroPRF1Scores attribute*), 555
num_layers (*pytext.models.representations.augmented_lstm.AugmentedL attribute*), 227
num_layers (*pytext.models.representations.bilstm.BiLSTM.Config attribute*), 228
num_layers (*pytext.models.representations.stacked_bidirectional_rnn.St attribute*), 250
num_samples (*pytext.metrics.intent_slot_metrics.FrameAccuracy attribute*), 547
num_tags (*pytext.models.output_layers.CRFOutputLayer attribute*), 615
num_tags (*pytext.models.output_layers.word_tagging_output_layer.CRF attribute*), 611
NUM_TOKENS (*pytext.common.constants.DatasetFieldName attribute*), 401
numberize() (*pytext.data.bert_tensorizer.BERTTensorizerBase method*), 442
numberize() (*pytext.data.bert_tensorizer.BERTTensorizerBaseScriptImp method*), 443
numberize() (*pytext.data.dense_retrieval_tensorizer.BERTContextTenso method*), 451

numberize() (*pytext.data.dense_retrieval_tensorizer.PositiveLabelTensorizerForDenseRetrieval method*), 452
numberize() (*pytext.data.roberta_tensorizer.RoBERTaTokenLevelTensorizer method*), 460
numberize() (*pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizer method*), 460
numberize() (*pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizerForKD method*), 460
numberize() (*pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizer method*), 461
numberize() (*pytext.data.squad_tensorizer.SquadTensorizer method*), 462
numberize() (*pytext.data.squad_tensorizer.SquadTensorizerForKD method*), 462
numberize() (*pytext.data.Tensorizer method*), 492
numberize() (*pytext.data.tensorizers.AnnotationNumberizer method*), 463
numberize() (*pytext.data.tensorizers.ByteTensorizer method*), 463
numberize() (*pytext.data.tensorizers.ByteTokenTensorizer method*), 464
numberize() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 464
numberize() (*pytext.data.tensorizers.CharacterVocabTokenTensorizer method*), 465
numberize() (*pytext.data.tensorizers.CharacterVocabTokenTensorizerScriptImpl method*), 465
numberize() (*pytext.data.tensorizers.Float1DListTensorizer method*), 466
numberize() (*pytext.data.tensorizers.FloatListSeqTensorizer method*), 467
numberize() (*pytext.data.tensorizers.FloatListTensorizer method*), 467
numberize() (*pytext.data.tensorizers.FloatTensorizer method*), 468
numberize() (*pytext.data.tensorizers.GazetteerTensorizer method*), 468
numberize() (*pytext.data.tensorizers.Integer1DListTensorizer method*), 469
numberize() (*pytext.data.tensorizers.LabelListRankTensorizer method*), 470
numberize() (*pytext.data.tensorizers.LabelListTensorizer method*), 470
numberize() (*pytext.data.tensorizers.LabelTensorizer method*), 470
numberize() (*pytext.data.tensorizers.MetricTensorizer method*), 471
numberize() (*pytext.data.tensorizers.NumericLabelTensorizer method*), 471
numberize() (*pytext.data.tensorizers.SeqTokenTensorizer method*), 472
numberize() (*pytext.data.tensorizers.SlotLabelTensorizer method*), 473
numberize() (*pytext.data.tensorizers.SoftLabelTensorizer method*), 473
numberize() (*pytext.data.tensorizers.String2DListTensorizer method*), 474
numberize() (*pytext.data.tensorizers.String2DListTensorizerScriptImpl method*), 474
numberize() (*pytext.data.tensorizers.Tensorizer method*), 475
numberize() (*pytext.data.tensorizers.TensorizerForKD method*), 475
numberize() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 475
numberize() (*pytext.data.tensorizers.TokenTensorizer method*), 476
numberize() (*pytext.data.tensorizers.UidTensorizer method*), 477
numberize() (*pytext.data.token_tensorizer.ScriptBasedTokenTensorizer method*), 478
numberize() (*pytext.data.token_tensorizer.TokenTensorizerScriptImpl method*), 479
numberize() (*pytext.data.xlm_tensorizer.XLMTensorizer method*), 482
numberize() (*pytext.data.xlm_tensorizer.XLMTensorizerScriptImpl method*), 482
numberize_rows() (*pytext.data.Data method*), 484
numberize_rows() (*pytext.data.data.Data method*), 445
numberize_rows() (*pytext.data.packed_lm_data.PackedLMData method*), 459
numberized (*pytext.data.data.BatchData attribute*), 444
numberized (*pytext.data.data.RowData attribute*), 446
numericalize() (*pytext.fields.char_field.CharFeatureField method*), 499
numericalize() (*pytext.fields.CharFeatureField method*), 504
numericalize() (*pytext.fields.contextual_token_embedding_field.ContextualTokenEm method*), 499
numericalize() (*pytext.fields.ContextualTokenEmbeddingField method*), 504
numericalize() (*pytext.fields.dict_field.DictFeatureField method*), 500
numericalize() (*pytext.fields.DictFeatureField method*), 505
numericalize() (*pytext.fields.text_field_with_special_unk.TextFeatureFieldWithSpeci method*), 503
numericalize() (*pytext.fields.TextFeatureFieldWithSpecialUnk method*), 507
NumericLabelTensorizer (*class in pytext.data.tensorizers*), 471

## O


old_tasks_deprecated() (*in module pytext.config.config_adapter*), 406
onnx_trace_input() (*pytext.models.BaseModel method*), 706
onnx_trace_input() (*pytext.models.model.BaseModel method*), 694
op_pre_epoch() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
op_pre_epoch() (*pytext.optimizer.sparsifiers.sparsifier.Sparsifier method*), 711
Optimizer (*class in pytext.optimizer.optimizers*), 718
OPTIMIZER (*pytext.config.component.ComponentType attribute*), 404
optimizer_step() (*pytext.trainers.Trainer method*), 758
optimizer_step() (*pytext.trainers.trainer.Trainer method*), 756
ordered_unique() (*in module pytext.utils.ascii_table*), 761
OrderedNeuronLSTM (*class in pytext.models.representations.ordered_neuron_lstm*), 651
OrderedNeuronLSTMLayer (*class in pytext.models.representations.ordered_neuron_lstm*), 651
original_forward() (*pytext.models.r3f_models.R3FPyTextMixin method*), 700
original_forward() (*pytext.models.roberta.RoBERTaR3F method*), 701
OTHERS (*pytext.common.constants.Stage attribute*), 402
OTHERS (*pytext.optimizer.sparsifiers.sparsifier.State attribute*), 712
out_dim (*pytext.models.decoders.decoder_base.DecoderBase attribute*), 565
out_dim (*pytext.models.decoders.DecoderBase attribute*), 570
out_dim (*pytext.models.decoders.mlp_decoder.MLPDecoder attribute*), 567
out_dim (*pytext.models.decoders.MLPDecoder attribute*), 570
output_columns (*pytext.config.pytext_config.LogitsConfig attribute*), 412
output_layer (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsembleModel attribute*), 590
output_layer (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsembleModel.Config attribute*), 182
output_layer (*pytext.models.ensembles.BaggingIntentSlotEnsembleModel attribute*), 592
output_layer (*pytext.models.ensembles.ensemble.EnsembleModel attribute*), 591
output_layer (*pytext.models.ensembles.EnsembleModel attribute*), 593
output_layer (*pytext.models.Model attribute*), 705
output_layer (*pytext.models.model.Model attribute*), 695
output_names (*pytext.exporters.exporter.ModelExporter attribute*), 494
output_names (*pytext.exporters.ModelExporter attribute*), 496
OutputLayerBase (*class in pytext.models.output_layers*), 613
OutputLayerBase (*class in pytext.models.output_layers.output_layer_base*), 607
OutputLayerUtils (*class in pytext.models.output_layers*), 621
OutputLayerUtils (*class in pytext.models.output_layers.utils*), 610
OutputRecord (*class in pytext.data.featurizer*), 420
OutputRecord (*class in pytext.data.featurizer.featurizer*), 418
OutputScore (*class in pytext.models.output_layers.distance_output_layer*), 598
overall_metrics (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics attribute*), 547, 548


## P


p50 (*pytext.utils.timing.Timings attribute*), 770
p90 (*pytext.utils.timing.Timings attribute*), 770
p99 (*pytext.utils.timing.Timings attribute*), 770
PackageFileName (*class in pytext.common.constants*), 402
PackedLMData (*class in pytext.data.packed_lm_data*), 459
PAD (*pytext.common.constants.SpecialTokens attribute*), 402
pad() (*in module pytext.data.utils*), 480
pad() (*pytext.fields.char_field.CharFeatureField method*), 499
pad() (*pytext.fields.CharFeatureField method*), 504
pad() (*pytext.fields.contextual_token_embedding_field.ContextualTokenEmbeddingField method*), 500
pad() (*pytext.fields.ContextualTokenEmbeddingField method*), 504
pad() (*pytext.fields.dict_field.DictFeatureField method*), 501
pad() (*pytext.fields.DictFeatureField method*), 505
pad_and_tensorize() (*in module pytext.data.utils*), 481

pad_and_tensorize_batches() (*in module pytext.data.data*), 446
PAD_BYTE (*pytext.data.tensorizers.ByteTensorizer attribute*), 463
pad_length() (*in module pytext.utils.precision*), 769
pad_length() (*pytext.fields.Field method*), 505
pad_length() (*pytext.fields.field.Field method*), 501
PAD_SEQ (*pytext.common.constants.VocabMeta attribute*), 403
PAD_TOKEN (*pytext.common.constants.VocabMeta attribute*), 403
Padding (*class in pytext.common.constants*), 402
padding_value (*pytext.models.representations.augmented_lstm.AugmentedLSTM attribute*), 638
padding_value (*pytext.models.representations.bilstm.BiLSTM attribute*), 641
padding_value (*pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN attribute*), 657
PaddingTest (*class in pytext.data.test.utils_test*), 436
PairRepresentation (*class in pytext.models.representations.pair_rep*), 651
PairwiseClassificationForDenseRetrievalTask (*class in pytext.task.tasks*), 733
PairwiseClassificationTask (*class in pytext.task.tasks*), 733
PairwiseCosineDistanceOutputLayer (*class in pytext.models.output_layers*), 618
PairwiseCosineDistanceOutputLayer (*class in pytext.models.output_layers.distance_output_layer*), 598
PairwiseCosineRegressionOutputLayer (*class in pytext.models.output_layers*), 620
PairwiseCosineRegressionOutputLayer (*class in pytext.models.output_layers.doc_regression_output_layer*), 601
PairwiseModel (*class in pytext.models.pair_classification_model*), 697
PairwiseRankingLoss (*class in pytext.loss*), 512
PairwiseRankingLoss (*class in pytext.loss.loss*), 509
PairwiseRankingMetricReporter (*class in pytext.metric_reporters*), 543
PairwiseRankingMetricReporter (*class in pytext.metric_reporters.pairwise_ranking_metric_reporter*), 528
PairwiseRankingMetrics (*class in pytext.metrics*), 557
PairwiseRankingOutputLayer (*class in pytext.models.output_layers*), 618
PairwiseRankingOutputLayer (*class in pytext.models.output_layers.pairwise_ranking_output_layer*), 608
PairwiseRegressionTask (*class in pytext.task.tasks*), 734
PandasDataSource (*class in pytext.data.sources*), 429
PandasDataSource (*class in pytext.data.sources.pandas*), 425
PandasDataSourceTest (*class in pytext.data.test.pandas_data_source_test*), 432
param_groups (*pytext.optimizer.fp16_optimizer.FP16Optimizer attribute*), 714
params (*pytext.optimizer.optimizers.Optimizer attribute*), 718
parse_and_align_slot_labels_list() (*in module pytext.utils.data*), 762
parse_config() (*in module pytext.config.serialize*), 41
parse_json_array() (*in module pytext.utils.data*), 762
parse_slot_string() (*in module pytext.utils.data*), 762
parse_token() (*in module pytext.utils.data*), 762
ParserState (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 661
partial() (*pytext.utils.lazy.Lazy class method*), 765
pass_index (*pytext.data.data_handler.DataHandler attribute*), 448
pass_index (*pytext.data.DataHandler attribute*), 486
PassThroughRepresentation (*class in pytext.models.representations.pass_through*), 652
pearson_correlation (*pytext.metrics.RegressionMetrics attribute*), 558, 559
per_label_confusions (*pytext.metrics.AllConfusions attribute*), 553
per_label_scores (*pytext.metrics.MacroPRF1Metrics attribute*), 555
per_label_scores (*pytext.metrics.PRF1Metrics attribute*), 556
per_label_soft_scores (*pytext.metrics.ClassificationMetrics attribute*), 553
PerLabelConfusions (*class in pytext.metrics*), 557
perplexity_per_word (*pytext.metrics.language_model_metrics.LanguageModelMetric attribute*), 551, 552
PerplexityType (*class in pytext.config.module_config*), 410

PersonalizedDocModel (*class in pytext.models.doc_model*), 690
PickleableGPT2BPEEncoder (*class in pytext.data.tokenizers.tokenizer*), 437
PlaceHolder (*class in pytext.config.pytext_config*), 412
PlaceholderAttentionIdentity (*class in pytext.models.seq_models.base*), 666
PlaceholderIdentity (*class in pytext.models.seq_models.base*), 667
PlaceholderIdentity.Config (*class in pytext.models.seq_models.base*), 667
PolynomialDecayScheduler (*class in pytext.optimizer.scheduler*), 720
pool() (*in module pytext.models.representations.deepcnn*), 649
pool() (*in module pytext.models.seq_models.nar_length*), 678
pooling (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttentionConfig attribute*), 229
pooling_type (*pytext.models.embeddings.dict_embedding.DictEmbedding attribute*), 575
pooling_type (*pytext.models.embeddings.DictEmbedding attribute*), 585
PoolingBatcher (*class in pytext.data*), 489
PoolingBatcher (*class in pytext.data.data*), 445
PoolingMethod (*class in pytext.models.representations.transformer_sentence_encoder_base*), 659
PoolingType (*class in pytext.config.module_config*), 410
pop() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLSTM method*), 661
pop() (*pytext.utils.timing.HierarchicalTimer method*), 769
pos_embed() (*pytext.models.seq_models.conv_decoder.LightConvDecoder method*), 669
pos_embed() (*pytext.models.seq_models.conv_encoder.LightConvEncoder method*), 672
POS_RESPONSE (*pytext.config.query_document_pairwise_ranking.ModelInputConfig attribute*), 414
pos_response (*pytext.config.query_document_pairwise_ranking.ModelInputConfig attribute*), 414
positional_embedding_type (*pytext.models.seq_models.conv_decoder.ConvDecoderConfig attribute*), 668
positional_embedding_type (*pytext.models.seq_models.conv_encoder.ConvEncoderConfig attribute*), 671
PositionalEmbedding (*class in pytext.models.representations.transformer*), 633
PositionalEmbedding (*class in pytext.models.representations.transformer.positional_embedding*), 625
PositiveLabelTensorizerForDenseRetrieval (*class in pytext.data.dense_retrieval_tensorizer*), 451
PostEncoder (*class in pytext.models.representations.transformer*), 635
PostEncoder (*class in pytext.models.representations.transformer.sentence_encoder*), 627
PostionalEmbedCombine (*class in pytext.models.seq_models.positional*), 679
PostionalEmbedType (*class in pytext.models.seq_models.positional*), 679
postprocess_output() (*pytext.exporters.custom_exporters.InitPredictNetExporter method*), 493
postprocess_output() (*pytext.exporters.custom_exporters.ModelExporter method*), 495
postprocess_output() (*pytext.exporters.InitPredictNetExporter method*), 498
postprocess_output() (*pytext.exporters.ModelExporter method*), 497
pre_export() (*pytext.optimizer.fp16_optimizer.FP16Optimizer method*), 714
pre_export() (*pytext.optimizer.fp16_optimizer.FP16OptimizerApex method*), 714
pre_export() (*pytext.optimizer.fp16_optimizer.FP16OptimizerFairseq method*), 715
pre_export() (*pytext.optimizer.fp16_optimizer.MemoryEfficientFP16Optimizer method*), 716
pre_export() (*pytext.optimizer.optimizers.Optimizer method*), 718
precision (*pytext.metrics.MacroPRF1Scores attribute*), 555
precision (*pytext.metrics.PRF1Scores attribute*), 557
precision_at_recall (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556
precision_at_recall (*pytext.metrics.SoftClassificationMetrics attribute*), 559
precision_at_recall() (*in module pytext.metrics*), 564
PRECISION_AT_RECALL_THRESHOLDS (*in module pytext.metrics*), 556
precision_range_lower (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 116
precision_range_upper (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 116

predict() (*pytext.task.task.TaskBase method*), 731
predict() (*pytext.task.TaskBase method*), 736
predicted_frame (*pytext.metrics.intent_slot_metrics.FramePredictionPair attribute*), 547
predicted_label (*pytext.metrics.LabelListPrediction attribute*), 554
predicted_label (*pytext.metrics.LabelPrediction attribute*), 554
predicted_nodes (*pytext.metrics.intent_slot_metrics.NodesPredictionPair attribute*), 548
predictions_to_report() (*pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method*), 518
predictions_to_report() (*pytext.metric_reporters.classification_metric_reporter.MultiLabelClassificationMetricReporter method*), 520
predictions_to_report() (*pytext.metric_reporters.ClassificationMetricReporter method*), 537
predictions_to_report() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method*), 520
predictions_to_report() (*pytext.metric_reporters.CompositionalMetricReporter method*), 542
predictions_to_report() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 523
predictions_to_report() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 540
predictions_to_report() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
predictions_to_report() (*pytext.metric_reporters.MetricReporter method*), 535
predictions_to_report() (*pytext.metric_reporters.MultiLabelClassificationMetricReporter method*), 538
PREDICTOR (*pytext.config.component.ComponentType attribute*), 404
PREDICTOR (*pytext.config.module_config.ExporterType attribute*), 410
prepare() (*pytext.optimizer.scheduler.BatchScheduler method*), 719
prepare() (*pytext.optimizer.scheduler.PolynomialDecayScheduler method*), 721
prepare() (*pytext.optimizer.scheduler.Scheduler method*), 721
prepare() (*pytext.optimizer.scheduler.SchedulerWithWarmup method*), 722
prepare() (*pytext.optimizer.scheduler.WarmupScheduler method*), 722
prepare_decoder_ips() (*in module pytext.torchscript.seq2seq.seq2seq_rnn_decoder_utils*), 740
prepare_decoderstep_ip() (*pytext.torchscript.seq2seq.encoder.EncoderEnsemble method*), 739
prepare_for_nar_inference() (*pytext.models.seq_models.nar_modules.NAREncoderUtility method*), 678
prepare_for_onnx_export_() (*pytext.models.BaseModel method*), 706
prepare_for_onnx_export_() (*pytext.models.model.BaseModel method*), 694
prepare_full_key() (*in module pytext.data.masks.seq_masks*), 685
prepare_generator_inputs() (*pytext.torchscript.seq2seq.export_model.Seq2SeqJIT method*), 739
prepare_input() (*pytext.data.Tensorizer method*), 492
prepare_input() (*pytext.data.tensorizers.SeqTokenTensorizer method*), 472
prepare_input() (*pytext.data.tensorizers.Tensorizer method*), 475
prepare_input() (*pytext.data.token_tensorizer.TokenTensorizer method*), 476
prepare_input() (*pytext.data.token_tensorizer.ScriptBasedTokenTensorizer method*), 479
prepare_masked_target_for_lengths() (*in module pytext.models.seq_models.mask_generator*), 676
prepare_task() (*in module pytext.workflow*), 771
prepare_task_metadata() (*in module pytext.workflow*), 771
prepend_operators() (*pytext.exporters.custom_exporters.InitPredictNetExporter method*), 493
prepend_operators() (*pytext.exporters.exporter.ModelExporter method*), 495
prepend_operators() (*pytext.exporters.InitPredictNetExporter method*), 498
prepend_operators() (*pytext.exporters.ModelExporter method*), 497
preprocess() (*pytext.data.data_handler.DataHandler method*), 450

preprocess() (*pytext.data.DataHandler method*), 487
preprocess_row() (*pytext.data.data_handler.DataHandler method*), 450
preprocess_row() (*pytext.data.DataHandler method*), 487
PretrainedEmbedding (*class in pytext.utils.embeddings*), 763
pretty_print_config_class() (*in module pytext.utils.documentation*), 763
PRF1Metrics (*class in pytext.metrics*), 556
PRF1Scores (*class in pytext.metrics*), 556
print_metrics() (*pytext.metrics.calibration_metrics.AllCalibrationMetrics method*), 544
print_metrics() (*pytext.metrics.calibration_metrics.CalibrationMetrics method*), 545
print_metrics() (*pytext.metrics.ClassificationMetrics method*), 553
print_metrics() (*pytext.metrics.dense_retrieval_metrics.DenseRetrievalMetrics method*), 546
print_metrics() (*pytext.metrics.intent_slot_metrics.AllMetrics method*), 546
print_metrics() (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics method*), 548
print_metrics() (*pytext.metrics.language_model_metrics.LanguageModelMetrics method*), 552
print_metrics() (*pytext.metrics.MacroPRF1Metrics method*), 555
print_metrics() (*pytext.metrics.PairwiseRankingMetrics method*), 557
print_metrics() (*pytext.metrics.PRF1Metrics method*), 556
print_metrics() (*pytext.metrics.RegressionMetrics method*), 559
print_metrics() (*pytext.metrics.seq2seq_metrics.Seq2SeqMetrics method*), 552
print_metrics() (*pytext.metrics.seq2seq_metrics.Seq2SeqTopKMetrics method*), 552
print_metrics() (*pytext.metrics.squad_metrics.SquadMetrics method*), 552
print_pep() (*pytext.metrics.ClassificationMetrics method*), 553
print_tree() (*pytext.data.data_structures.annotation.Tree method*), 416
PRIVACY_ENGINE (*pytext.config.component.ComponentType attribute*), 404
PrivacyEngine (*class in pytext.optimizer.privacy_engine*), 719
PROBS (*pytext.loss.loss.SourceType attribute*), 509
PROBS (*pytext.loss.SourceType attribute*), 512
process_file() (*pytext.data.sources.dense_retrieval.DenseRetrievalDataSource method*), 425
process_file() (*pytext.data.sources.DenseRetrievalDataSource method*), 430
process_file() (*pytext.data.sources.squad.SquadDataSource method*), 426
process_file() (*pytext.data.sources.SquadDataSource method*), 429
process_pred() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTagging method*), 533
process_pred() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 542
process_squad_json() (*pytext.data.sources.squad.SquadDataSource method*), 427
process_squad_json() (*pytext.data.sources.SquadDataSource method*), 429
process_squad_tsv() (*pytext.data.sources.squad.SquadDataSource method*), 427
process_squad_tsv() (*pytext.data.sources.squad.SquadDataSourceForKD method*), 427
process_squad_tsv() (*pytext.data.sources.SquadDataSource method*), 429
projection (*pytext.models.embeddings.char_embedding.CharacterEmb attribute*), 573
projection (*pytext.models.embeddings.CharacterEmbedding attribute*), 586
projection_d (*pytext.models.representations.bilstm_doc_slot_attention attribute*), 644
projection_w (*pytext.models.representations.bilstm_doc_slot_attention attribute*), 644
prune_multi_heads() (*pytext.models.representations.transformer.multihead_attention.Mult method*), 623

prune_multi_heads() (*pytext.models.representations.transformer.MultiheadSelfAttention method*), 633
prune_multi_linear_heads() (*pytext.models.representations.transformer.multihead_linear_attention.MultiheadLinearAttention method*), 624
prune_multi_linear_heads() (*pytext.models.representations.transformer.MultiheadLinearAttention method*), 632
PureDocAttention (*class in pytext.models.representations.pure_doc_attention*), 654
PureFP16Optimizer (*class in pytext.optimizer.fp16_optimizer*), 716
PureLossMetricReporter (*class in pytext.metric_reporters*), 543
PureLossMetricReporter (*class in pytext.metric_reporters.metric_reporter*), 527
push() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLSTM method*), 661
push() (*pytext.utils.timing.HierarchicalTimer method*), 769
push_action() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserBase method*), 664
pytext (*module*), 772
pytext.builtin_task (*module*), 770
pytext.common (*module*), 403
pytext.common.constants (*module*), 400
pytext.common.utils (*module*), 403
pytext.config (*module*), 415
pytext.config.component (*module*), 403
pytext.config.config_adapter (*module*), 405
pytext.config.contextual_intent_slot (*module*), 407
pytext.config.doc_classification (*module*), 408
pytext.config.field_config (*module*), 408
pytext.config.module_config (*module*), 409
pytext.config.pair_classification (*module*), 411
pytext.config.pytext_config (*module*), 411
pytext.config.query_document_pairwise_ranking (*module*), 414
pytext.config.serialize (*module*), 414
pytext.config.utils (*module*), 414
pytext.data (*module*), 482
pytext.data.batch_sampler (*module*), 440
pytext.data.bert_tensorizer (*module*), 442
pytext.data.data (*module*), 444
pytext.data.data_handler (*module*), 446
pytext.data.data_structures (*module*), 418
pytext.data.data_structures.annotation (*module*), 415
pytext.data.data_structures.node (*module*), 417
pytext.data.dense_retrieval_tensorizer (*module*), 451
pytext.data.disjoint_multitask_data (*module*), 453
pytext.data.disjoint_multitask_data_handler (*module*), 454
pytext.data.dynamic_pooling_batcher (*module*), 456
pytext.data.featurizer (*module*), 420
pytext.data.featurizer.featurizer (*module*), 418
pytext.data.featurizer.simple_featurizer (*module*), 419
pytext.data.masked_tensorizer (*module*), 457
pytext.data.masked_util (*module*), 458
pytext.data.masked_lm_data (*module*), 459
pytext.data.roberta_tensorizer (*module*), 459
pytext.data.sources (*module*), 428
pytext.data.sources.conllu (*module*), 421
pytext.data.sources.data_source (*module*), 422
pytext.data.sources.dense_retrieval (*module*), 424
pytext.data.sources.pandas (*module*), 425
pytext.data.sources.session (*module*), 426
pytext.data.sources.squad (*module*), 426
pytext.data.sources.tsv (*module*), 427
pytext.data.squad_for_bert_tensorizer (*module*), 460
pytext.data.squad_tensorizer (*module*), 461
pytext.data.tensorizers (*module*), 462
pytext.data.test (*module*), 436
pytext.data.test.batch_sampler_test (*module*), 431
pytext.data.test.data_test (*module*), 431
pytext.data.test.dynamic_pooling_batcher_test (*module*), 432
pytext.data.test.mask_tensorizers_test (*module*), 432
pytext.data.test.pandas_data_source_test (*module*), 432
pytext.data.test.round_robin_batchiterator_test (*module*), 432
pytext.data.test.simple_featurizer_test (*module*), 432
pytext.data.test.tensorizers_test (*module*), 433
pytext.data.test.tokenizers_test (*module*), 435
pytext.data.test.tsv_data_source_test

(*module*), 435
pytext.data.test.utils_test (*module*), 436
pytext.data.token_tensorizer (*module*), 478
pytext.data.tokenizers (*module*), 438
pytext.data.tokenizers.tokenizer (*module*), 436
pytext.data.utils (*module*), 480
pytext.data.xlm_constants (*module*), 481
pytext.data.xlm_dictionary (*module*), 481
pytext.data.xlm_tensorizer (*module*), 481
pytext.exporters (*module*), 496
pytext.exporters.custom_exporters (*module*), 493
pytext.exporters.exporter (*module*), 494
pytext.fields (*module*), 503
pytext.fields.char_field (*module*), 499
pytext.fields.contextual_token_embedding_field (*module*), 499
pytext.fields.dict_field (*module*), 500
pytext.fields.field (*module*), 501
pytext.fields.text_field_with_special_unk (*module*), 503
pytext.loss (*module*), 510
pytext.loss.loss (*module*), 507
pytext.loss.regularized_loss (*module*), 509
pytext.loss.regularizer (*module*), 509
pytext.loss.structured_loss (*module*), 510
pytext.main (*module*), 771
pytext.metric_reporters (*module*), 533
pytext.metric_reporters.calibration_metric_reporter (*module*), 513
pytext.metric_reporters.channel (*module*), 514
pytext.metric_reporters.classification_metric_reporter (*module*), 517
pytext.metric_reporters.compositional_metric_reporter (*module*), 520
pytext.metric_reporters.compositional_utils (*module*), 521
pytext.metric_reporters.dense_retrieval_metric_reporter (*module*), 521
pytext.metric_reporters.disjoint_multitask_metric_reporter (*module*), 522
pytext.metric_reporters.intent_slot_detection_metric_reporter (*module*), 523
pytext.metric_reporters.language_model_metric_reporter (*module*), 524
pytext.metric_reporters.metric_reporter (*module*), 526
pytext.metric_reporters.pairwise_ranking_metric_reporter (*module*), 528
pytext.metric_reporters.regression_metric_reporter (*module*), 528
pytext.metric_reporters.seq2seq_compositional (*module*), 529
pytext.metric_reporters.seq2seq_metric_reporter (*module*), 529
pytext.metric_reporters.seq2seq_utils (*module*), 530
pytext.metric_reporters.squad_metric_reporter (*module*), 530
pytext.metric_reporters.word_tagging_metric_reporter (*module*), 532
pytext.metrics (*module*), 553
pytext.metrics.calibration_metrics (*module*), 544
pytext.metrics.dense_retrieval_metrics (*module*), 545
pytext.metrics.intent_slot_metrics (*module*), 546
pytext.metrics.language_model_metrics (*module*), 551
pytext.metrics.seq2seq_metrics (*module*), 552
pytext.metrics.squad_metrics (*module*), 552
pytext.models (*module*), 704
pytext.models.bert_classification_models (*module*), 686
pytext.models.bert_regression_model (*module*), 686
pytext.models.crf (*module*), 687
pytext.models.decoders (*module*), 569
pytext.models.decoders.decoder_base (*module*), 565
pytext.models.decoders.intent_slot_model_decoder (*module*), 566
pytext.models.decoders.mlp_decoder (*module*), 567
pytext.models.decoders.mlp_decoder_query_response (*module*), 568
pytext.models.decoders.mlp_decoder_two_tower (*module*), 568
pytext.models.decoders.multilabel_decoder (*module*), 569
pytext.models.disjoint_multitask_model (*module*), 687
pytext.models.distributed_model (*module*), 688
pytext.models.doc_model (*module*), 689
pytext.models.embeddings (*module*), 582
pytext.models.embeddings.char_embedding (*module*), 572
pytext.models.embeddings.contextual_token_embedding (*module*), 574
pytext.models.embeddings.dict_embedding (*module*), 574
pytext.models.embeddings.embedding_base (*module*), 576

pytext.models.embeddings.embedding_list (*module*), 576
pytext.models.embeddings.mlp_embedding (*module*), 577
pytext.models.embeddings.scriptable_embedding_list (*module*), 578
pytext.models.embeddings.word_embedding (*module*), 579
pytext.models.embeddings.word_seq_embedding (*module*), 581
pytext.models.ensembles (*module*), 592
pytext.models.ensembles.bagging_doc_ensemble (*module*), 589
pytext.models.ensembles.bagging_intent_slot_ensemble (*module*), 590
pytext.models.ensembles.ensemble (*module*), 591
pytext.models.joint_model (*module*), 691
pytext.models.language_models (*module*), 596
pytext.models.language_models.lmlstm (*module*), 595
pytext.models.masked_lm (*module*), 692
pytext.models.masking_utils (*module*), 692
pytext.models.model (*module*), 693
pytext.models.module (*module*), 696
pytext.models.output_layers (*module*), 613
pytext.models.output_layers.distance_output_layer (*module*), 597
pytext.models.output_layers.doc_classification_output_layer (*module*), 599
pytext.models.output_layers.doc_regression_output_layer (*module*), 601
pytext.models.output_layers.intent_slot_output_layer (*module*), 603
pytext.models.output_layers.lm_output_layer (*module*), 604
pytext.models.output_layers.multi_label_classification_output_layer (*module*), 605
pytext.models.output_layers.output_layer_base (*module*), 607
pytext.models.output_layers.pairwise_ranking_output_layer (*module*), 608
pytext.models.output_layers.squad_output_layer (*module*), 609
pytext.models.output_layers.utils (*module*), 610
pytext.models.output_layers.word_tagging_output_layer (*module*), 611
pytext.models.pair_classification_model (*module*), 697
pytext.models.qna (*module*), 623
pytext.models.qna.bert_squad_qa (*module*), 622
pytext.models.qna.dr_qa (*module*), 622
pytext.models.query_document_pairwise_ranking_model (*module*), 698
pytext.models.r3f_models (*module*), 699
pytext.models.representations (*module*), 660
pytext.models.representations.attention (*module*), 637
pytext.models.representations.augmented_lstm (*module*), 638
pytext.models.representations.bilstm (*module*), 641
pytext.models.representations.bilstm_doc_attention (*module*), 642
pytext.models.representations.bilstm_doc_slot_attention (*module*), 643
pytext.models.representations.bilstm_slot_attn (*module*), 644
pytext.models.representations.biseqcnn (*module*), 645
pytext.models.representations.contextual_intent_slot (*module*), 646
pytext.models.representations.deepcnn (*module*), 647
pytext.models.representations.docnn (*module*), 649
pytext.models.representations.huggingface_bert_sentence (*module*), 649
pytext.models.representations.huggingface_electra_sentence (*module*), 650
pytext.models.representations.jointcnn_rep (*module*), 650
pytext.models.representations.ordered_neuron_lstm (*module*), 651
pytext.models.representations.pair_rep (*module*), 651
pytext.models.representations.pass_through (*module*), 652
pytext.models.representations.pooling (*module*), 652
pytext.models.representations.pure_doc_attention (*module*), 654
pytext.models.representations.representation_base (*module*), 654
pytext.models.representations.seq_rep (*module*), 655
pytext.models.representations.slot_attention (*module*), 655
pytext.models.representations.sparse_transformer_seq (*module*), 656
pytext.models.representations.stacked_bidirectional (*module*), 656
pytext.models.representations.traced_transformer_encoder (*module*), 657

pytext.models.representations.transformer (module), 631
pytext.models.representations.transformer.multi_head_attention (module), 623
pytext.models.representations.transformer.multi_head_linear (module), 624
pytext.models.representations.transformer.positional_embedding (module), 625
pytext.models.representations.transformer.representation (module), 626
pytext.models.representations.transformer.residual_ffn (module), 626
pytext.models.representations.transformer.transformer_encoder (module), 627
pytext.models.representations.transformer.transformer_layer (module), 629
pytext.models.representations.transformer.utils (module), 658
pytext.models.representations.transformer.xlm_decoder (module), 659
pytext.models.roberta (module), 700
pytext.models.semantic_parsers (module), 664
pytext.models.semantic_parsers.rnng (module), 664
pytext.models.semantic_parsers.rnng.rnng_constituent (module), 660
pytext.models.semantic_parsers.rnng.rnng_data_structures (module), 660
pytext.models.semantic_parsers.rnng.rnng_parser (module), 662
pytext.models.seq_models (module), 685
pytext.models.seq_models.attention (module), 665
pytext.models.seq_models.base (module), 666
pytext.models.seq_models.contextual_intent_slot (module), 667
pytext.models.seq_models.conv_decoder (module), 668
pytext.models.seq_models.conv_encoder (module), 671
pytext.models.seq_models.conv_model (module), 673
pytext.models.seq_models.light_conv (module), 673
pytext.models.seq_models.mask_generator (module), 674
pytext.models.seq_models.nar_length (module), 677
pytext.models.seq_models.nar_modules (module), 678
pytext.models.seq_models.nar_output_layer (module), 678
pytext.models.seq_models.positional (module), 678
pytext.models.seq_models.projection_layers (module), 680
pytext.models.seq_models.rnn_decoder (module), 681
pytext.models.seq_models.rnn_encoder (module), 682
pytext.models.seq_models.rnn_encoder_decoder (module), 683
pytext.models.seq_models.seq2seq_model (module), 683
pytext.models.seq_models.seq2seq_output_layer (module), 684
pytext.models.seq_models.seqnn (module), 685
pytext.models.seq_models.utils (module), 685
pytext.models.text_pair_classification_model (module), 702
pytext.models.utils (module), 703
pytext.models.word_model (module), 703
pytext.optimizer (module), 724
pytext.optimizer.activations (module), 712
pytext.optimizer.adabelief (module), 712
pytext.optimizer.fairseq_fp16_utils (module), 712
pytext.optimizer.fp16_optimizer (module), 713
pytext.optimizer.lamb (module), 717
pytext.optimizer.madgrad (module), 717
pytext.optimizer.optimizers (module), 718
pytext.optimizer.privacy_engine (module), 719
pytext.optimizer.radam (module), 719
pytext.optimizer.scheduler (module), 719
pytext.optimizer.sparsifiers (module), 712
pytext.optimizer.sparsifiers.blockwise_sparsifier (module), 707
pytext.optimizer.sparsifiers.sparsifier (module), 709
pytext.optimizer.swa (module), 722
pytext.resources (module), 724
pytext.resources.roberta (module), 724
pytext.task (module), 735
pytext.task.accelerator_lowering (module), 724
pytext.task.disjoint_multitask (module), 726
pytext.task.new_task (module), 727
pytext.task.nop_decorator (module), 728
pytext.task.quantize (module), 728
pytext.task.serialize (module), 728
pytext.task.task (module), 730

pytext.task.tasks (*module*), 732
pytext.torchscript (*module*), 754
pytext.torchscript.batchutils (*module*), 746
pytext.torchscript.module (*module*), 748
pytext.torchscript.seq2seq (*module*), 740
pytext.torchscript.seq2seq.beam_decode (*module*), 737
pytext.torchscript.seq2seq.beam_search (*module*), 738
pytext.torchscript.seq2seq.decoder (*module*), 738
pytext.torchscript.seq2seq.encoder (*module*), 738
pytext.torchscript.seq2seq.export_model (*module*), 739
pytext.torchscript.seq2seq.scripted_seq2seq_generator (*module*), 739
pytext.torchscript.seq2seq.seq2seq_rnn_decoder (*module*), 740
pytext.torchscript.tensorizer (*module*), 742
pytext.torchscript.tensorizer.bert (*module*), 740
pytext.torchscript.tensorizer.normalizer (*module*), 740
pytext.torchscript.tensorizer.roberta (*module*), 741
pytext.torchscript.tensorizer.tensorizer (*module*), 741
pytext.torchscript.tensorizer.xlm (*module*), 742
pytext.torchscript.tokenizer (*module*), 745
pytext.torchscript.tokenizer.bpe (*module*), 744
pytext.torchscript.tokenizer.tokenizer (*module*), 745
pytext.torchscript.utils (*module*), 753
pytext.torchscript.vocab (*module*), 754
pytext.trainers (*module*), 757
pytext.trainers.ensemble_trainer (*module*), 754
pytext.trainers.hogwild_trainer (*module*), 754
pytext.trainers.trainer (*module*), 755
pytext.trainers.training_state (*module*), 757
pytext.utils (*module*), 770
pytext.utils.ascii_table (*module*), 761
pytext.utils.config_utils (*module*), 761
pytext.utils.cuda (*module*), 761
pytext.utils.data (*module*), 761
pytext.utils.distributed (*module*), 762
pytext.utils.documentation (*module*), 763
pytext.utils.embeddings (*module*), 763
pytext.utils.file_io (*module*), 764
pytext.utils.label (*module*), 764
pytext.utils.lazy (*module*), 764
pytext.utils.loss (*module*), 766
pytext.utils.meter (*module*), 768
pytext.utils.mobile_onnx (*module*), 768
pytext.utils.model (*module*), 768
pytext.utils.onnx (*module*), 768
pytext.utils.path (*module*), 769
pytext.utils.precision (*module*), 769
pytext.utils.tensor (*module*), 769
pytext.utils.test (*module*), 769
pytext.utils.timing (*module*), 769
pytext.utils.usage (*module*), 770
pytext.workflow (*module*), 771
pytext_config_from_json() (*in module pytext.config.serialize*), 414
PyTextCheckpointManagerInterface (*class in pytext.task.serialize*), 729
PyTextConfig (*class in pytext.config.pytext_config*), 412
PyTextEmbeddingModule (*class in pytext.torchscript.module*), 748
PyTextEmbeddingModuleIndex (*class in pytext.torchscript.module*), 748
PyTextEmbeddingModuleWithDense (*class in pytext.torchscript.module*), 748
PyTextEmbeddingModuleWithDenseIndex (*class in pytext.torchscript.module*), 748
PyTextIncrementalDecoderComponent (*class in pytext.models.seq_models.base*), 667
PyTextLayerModule (*class in pytext.torchscript.module*), 749
PyTextLayerModuleWithDense (*class in pytext.torchscript.module*), 749
PyTextSeq2SeqModule (*class in pytext.models.seq_models.base*), 667
PyTextTwoTowerEmbeddingModule (*class in pytext.torchscript.module*), 749
PyTextTwoTowerEmbeddingModuleWithDense (*class in pytext.torchscript.module*), 749
PyTextTwoTowerLayerModule (*class in pytext.torchscript.module*), 750
PyTextTwoTowerLayerModuleWithDense (*class in pytext.torchscript.module*), 750
PyTextVariableSizeEmbeddingModule (*class in pytext.torchscript.module*), 750
pytorch_to_caffe2() (*in module pytext.utils.mobile_onnx*), 768
pytorch_to_caffe2() (*in module pytext.utils.onnx*), 769

## Q


quantize() (*pytext.models.BaseModel method*), 706
quantize() (*pytext.models.model.BaseModel method*), 694
quantize_fx() (*in module pytext.task.quantize*), 728
quantize_statically() (*in module pytext.task*), 737
quantize_statically() (*in module pytext.task.quantize*), 728
QuantizedMultiheadLinearAttention (*class in pytext.models.representations.transformer*), 632
QuantizedMultiheadLinearAttention (*class in pytext.models.representations.transformer.multihead_linear_attention*), 624
QUERY (*pytext.config.query_document_pairwise_ranking.ModelInput attribute*), 414
query (*pytext.config.query_document_pairwise_ranking.ModelInputConfig attribute*), 414
query_word_reprs() (*in module pytext.models.output_layers.utils*), 610
QueryDocPairwiseRankingModel (*class in pytext.models.query_document_pairwise_ranking_model*), 698
QueryDocumentPairwiseRankingTask (*class in pytext.task.tasks*), 734
QUES_COLUMN (*pytext.metric_reporters.squad_metric_reporter.SquadMetricReporter attribute*), 531
QUES_COLUMN (*pytext.metric_reporters.SquadMetricReporter attribute*), 541


## R


r3f_default_lambda (*pytext.models.r3f_models.R3FConfigOptions attribute*), 699
r3f_lambda_by_loss (*pytext.models.r3f_models.R3FConfigOptions attribute*), 699
R3FConfigOptions (*class in pytext.models.r3f_models*), 699
R3FNoiseContextManager (*class in pytext.models.r3f_models*), 699
R3FNoiseType (*class in pytext.models.r3f_models*), 699
R3FPyTextMixin (*class in pytext.models.r3f_models*), 699
RAdam (*class in pytext.optimizer.radam*), 719
RANDOM (*pytext.config.field_config.EmbedInitStrategy attribute*), 408
RANDOM (*pytext.models.masking_utils.MaskingStrategy attribute*), 693
random_masking() (*in module pytext.models.masking_utils*), 693
random_seed (*pytext.config.pytext_config.PyTextConfig attribute*), 412
RandomizedBatchSampler (*class in pytext.data*), 490
RandomizedBatchSampler (*class in pytext.data.batch_sampler*), 440
RandomizedMaskingFunction (*class in pytext.data.masked_util*), 458
range_to_anchors_and_delta() (*in module pytext.utils.loss*), 767
rank (*pytext.trainers.training_state.TrainingState attribute*), 757
rank (*pytext.trainers.TrainingState attribute*), 759
raw_columns (*pytext.data.data_handler.DataHandler attribute*), 447
raw_columns (*pytext.data.DataHandler attribute*), 485
raw_cosine (*pytext.models.output_layers.distance_output_layer.OutputS attribute*), 598
raw_data (*pytext.data.data.BatchData attribute*), 444
raw_data (*pytext.data.data.RowData attribute*), 446
RAW_DICT_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
RAW_EMBED (*pytext.common.constants.PackageFileName attribute*), 402
raw_eval_data_generator() (*pytext.data.sources.conllu.CoNLLUPOSDataSource method*), 421
raw_eval_data_generator() (*pytext.data.sources.data_source.RootDataSource method*), 423
raw_eval_data_generator() (*pytext.data.sources.pandas.PandasDataSource method*), 425
raw_eval_data_generator() (*pytext.data.sources.PandasDataSource method*), 430
raw_eval_data_generator() (*pytext.data.sources.tsv.TSVDataSource method*), 428
raw_eval_data_generator() (*pytext.data.sources.TSVDataSource method*), 429
RAW_FEATS (*pytext.common.constants.DFColumn attribute*), 401
raw_gazetteer_feats (*pytext.data.featurizer.featurizer.InputRecord attribute*), 418
raw_gazetteer_feats (*pytext.data.featurizer.InputRecord attribute*), 420
raw_generator() (*pytext.data.sources.pandas.PandasDataSource*

*static method*), 425
raw_generator() (*pytext.data.sources.PandasDataSource static method*), 430
RAW_SEQUENCE (*pytext.common.constants.DatasetFieldName attribute*), 401
raw_test_data_generator() (*pytext.data.sources.conllu.CoNLLUPOSDataSource method*), 421
raw_test_data_generator() (*pytext.data.sources.data_source.RootDataSource method*), 423
raw_test_data_generator() (*pytext.data.sources.pandas.PandasDataSource method*), 425
raw_test_data_generator() (*pytext.data.sources.PandasDataSource method*), 430
raw_test_data_generator() (*pytext.data.sources.tsv.TSVDataSource method*), 428
raw_test_data_generator() (*pytext.data.sources.TSVDataSource method*), 429
RAW_TEXT (*pytext.config.doc_classification.ExtraField attribute*), 408
raw_text (*pytext.data.featurizer.featurizer.InputRecord attribute*), 418
raw_text (*pytext.data.featurizer.InputRecord attribute*), 420
RAW_TEXT_COLUMN (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter attribute*), 524
RAW_TEXT_COLUMN (*pytext.metric_reporters.LanguageModelMetricReporter attribute*), 540
raw_train_data_generator() (*pytext.data.sources.conllu.CoNLLUPOSDataSource method*), 422
raw_train_data_generator() (*pytext.data.sources.data_source.RootDataSource method*), 423
raw_train_data_generator() (*pytext.data.sources.pandas.PandasDataSource method*), 425
raw_train_data_generator() (*pytext.data.sources.PandasDataSource method*), 430
raw_train_data_generator() (*pytext.data.sources.tsv.TSVDataSource method*), 428
raw_train_data_generator() (*pytext.data.sources.TSVDataSource method*), 429
RAW_WORD_LABEL (*pytext.common.constants.DatasetFieldName attribute*), 401
RAW_WORD_LABEL (*pytext.config.contextual_intent_slot.ExtraField attribute*), 407
RawExample (*class in pytext.data.sources*), 428
RawExample (*class in pytext.data.sources.data_source*), 422
RawExampleFieldName (*class in pytext.common.constants*), 402
RawExampleTest (*class in pytext.data.test.data_test*), 431
RawField (*class in pytext.fields*), 506
RawField (*class in pytext.fields.field*), 501
read_chunk_size (*pytext.config.pytext_config.PyTextConfig attribute*), 412
read_file() (*pytext.data.sources.dense_retrieval.DenseRetrievalDataSource method*), 425
read_file() (*pytext.data.sources.DenseRetrievalDataSource method*), 430
read_from_file() (*pytext.data.data_handler.DataHandler method*), 450
read_from_file() (*pytext.data.DataHandler method*), 487
read_vocab() (*pytext.data.xlm_dictionary.Dictionary static method*), 481
real_trainer (*pytext.trainers.ensemble_trainer.EnsembleTrainer attribute*), 754
real_trainer (*pytext.trainers.EnsembleTrainer attribute*), 760
RealtimeMetrics (*class in pytext.metrics*), 558
recall (*pytext.metrics.MacroPRF1Scores attribute*), 555
recall (*pytext.metrics.PRF1Scores attribute*), 557
recall_at_precision (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556
recall_at_precision (*pytext.metrics.SoftClassificationMetrics attribute*), 559
recall_at_precision() (*in module pytext.metrics*), 564
recursive_map() (*in module pytext.utils*), 770
recursive_validation() (*pytext.data.data_structures.annotation.Tree method*), 416
ReduceLROnPlateau (*class in pytext.optimizer.scheduler*), 721
register_adapter() (*in module pytext.config.config_adapter*), 406
register_builtin_tasks() (*in module py-*

*text.builtin_task*), 770
register_down_grade_adapter() (*in module pytext.config.config_adapter*), 406
register_http_url_handler() (*in module pytext.utils.file_io*), 764
register_snapshot_loader() (*in module pytext.task.serialize*), 730
register_tasks() (*in module pytext.config.component*), 405
register_type() (*pytext.data.sources.data_source.RootDataSource class method*), 423
Registry (*class in pytext.config.component*), 404
RegistryError, 404
RegressionMetricReporter (*class in pytext.metric_reporters*), 539
RegressionMetricReporter (*class in pytext.metric_reporters.regression_metric_reporter*), 528
RegressionMetrics (*class in pytext.metrics*), 558
RegressionOutputLayer (*class in pytext.models.output_layers*), 616
RegressionOutputLayer (*class in pytext.models.output_layers.doc_regression_output_layer*), 602
RegressionScores (*class in pytext.models.output_layers.doc_regression_output_layer*), 603
Regularizer (*class in pytext.loss.regularizer*), 510
reload_model_for_multi_export() (*in module pytext.workflow*), 772
RELU (*pytext.config.module_config.Activation attribute*), 409
relu (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 643
remove() (*pytext.data.data_structures.annotation.Token method*), 416
remove_docclassificationtask_deprecated() (*in module pytext.config.config_adapter*), 406
remove_lmtask_deprecated() (*in module pytext.config.config_adapter*), 406
remove_state_keys() (*in module pytext.models.representations.transformer.sentence_encoder*), 628
rename() (*in module pytext.config.config_adapter*), 406
rename_bitransformer_inputs() (*in module pytext.config.config_adapter*), 406
rename_component_from_root() (*in module pytext.models.representations.transformer.sentence_encoder*), 628
rename_fl_task() (*in module pytext.config.config_adapter*), 406
rename_parameter() (*in module pytext.config.config_adapter*), 406
rename_state_keys() (*in module pytext.models.representations.transformer.sentence_encoder*), 628
rename_tensorizer_vocab_params() (*in module pytext.config.config_adapter*), 406
reorder_encoder_out() (*pytext.models.seq_models.conv_encoder.LightConvEncoder method*), 672
reorder_incremental_state() (*pytext.models.seq_models.attention.MultiheadAttention method*), 666
reorder_incremental_state() (*pytext.models.seq_models.base.PlaceholderAttentionIdentity method*), 667
reorder_incremental_state() (*pytext.models.seq_models.base.PyTextIncrementalDecoderComponent method*), 667
reorder_incremental_state() (*pytext.models.seq_models.conv_decoder.LightConvDecoderBase method*), 669
reorder_incremental_state() (*pytext.models.seq_models.conv_decoder.LightConvDecoderLayer method*), 670
reorder_incremental_state() (*pytext.models.seq_models.light_conv.LightweightConv method*), 674
reorder_incremental_state() (*pytext.models.seq_models.rnn_decoder.RNNDecoderBase method*), 682
repackage_hidden() (*in module pytext.models.language_models.lmlstm*), 596
replace_components() (*in module pytext.utils.torchscriptify.utils.utils*), 763
replace_lazy_modules() (*in module pytext.utils.lazy*), 766
replace_param() (*in module pytext.config.utils*), 415
replace_tokens() (*pytext.data.utils.Vocabulary method*), 480
report() (*pytext.metric_reporters.Channel method*), 534
report() (*pytext.metric_reporters.channel.Channel method*), 514
report() (*pytext.metric_reporters.channel.ConsoleChannel method*), 514
report() (*pytext.metric_reporters.channel.FileChannel method*), 515
report() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 516
report() (*pytext.utils.timing.Snapshot method*), 769
report_eval_results (*pytext.config.pytext_config.PyTextConfig attribute*), 413
report_metric() (*py-*

*text.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 522
report_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
report_metric() (*pytext.metric_reporters.MetricReporter method*), 535
report_realtime_metric() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 523
report_realtime_metric() (*pytext.metric_reporters.language_model_metric_reporter.MaskedLMMetricReporter method*), 526
report_realtime_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
report_realtime_metric() (*pytext.metric_reporters.MetricReporter method*), 536
report_snapshot() (*in module pytext.utils.timing*), 770
report_test_results (*pytext.config.pytext_config.PyTextConfig attribute*), 413
report_train_metrics (*pytext.trainers.Trainer attribute*), 758
report_train_metrics (*pytext.trainers.trainer.Trainer attribute*), 755
representation (*pytext.models.Model attribute*), 705
representation (*pytext.models.model.Model attribute*), 695
representation_dim (*pytext.models.representations.augmented_lstm.AugmentedLSTM attribute*), 638
representation_dim (*pytext.models.representations.bilstm.BiLSTM attribute*), 641
representation_dim (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute*), 642
representation_dim (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 644
representation_dim (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttention attribute*), 645
representation_dim (*pytext.models.representations.stacked_bidirectional_rnn.StackedBiRNN attribute*), 657
RepresentationBase (*class in pytext.models.representations.representation_base*), 654
reset() (*pytext.optimizer.AdaBelief method*), 712
reset() (*pytext.utils.meter.Meter method*), 768
reset() (*pytext.utils.meter.TimeMeter method*), 768
reset_incremental_states() (*pytext.torchscript.seq2seq.decoder.DecoderBatchedStepEnsemble method*), 738
reset_param_groups() (*pytext.optimizer.madgrad.MADGRAD method*), 718
reset_param_groups() (*pytext.optimizer.optimizers.Optimizer method*), 718
reset_param_groups() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 723
reset_parameters() (*pytext.models.crf.CRF method*), 687
reset_parameters() (*pytext.models.embeddings.char_embedding.Highway method*), 574
reset_parameters() (*pytext.models.representations.augmented_lstm.AugmentedLSTMCell method*), 639
reset_parameters() (*pytext.models.seq_models.light_conv.LightweightConv method*), 674
reset_parameters() (*pytext.models.seq_models.projection_layers.DecoderWithLinearOut method*), 680
reset_parameters() (*pytext.models.seq_models.rnn_decoder.DecoderWithLinearOutputP method*), 681
ResidualMLP (*class in pytext.models.representations.transformer*), 634
ResidualMLP (*class in pytext.models.representations.transformer.residual_mlp*), 626
resolve() (*pytext.utils.lazy.Lazy method*), 765
resolve_conditional() (*in module pytext.config.utils*), 415
ResultRow (*class in pytext.utils.data*), 761
ResultRowSchema (*class in pytext.utils.data*), 761
RIGHT (*pytext.models.decoders.mlp_decoder_two_tower.ExportType attribute*), 568
RNN (*pytext.models.representations.stacked_bidirectional_rnn.RnnType attribute*), 656
rnn_type (*pytext.models.representations.stacked_bidirectional_rnn.StackedBiRNN attribute*), 250
RNNDecoder (*class in pytext.models.seq_models.rnn_decoder*), 681
RNNDecoderBase (*class in pytext.models.seq_models.rnn_decoder*), 681

RNNGParser (*class in pytext.models.semantic_parsers.rnng.rnng_parser*), 662
RNNGParserBase (*class in pytext.models.semantic_parsers.rnng.rnng_parser*), 662
RNNModel (*class in pytext.models.seq_models.rnn_encoder_decoder*), 683
RnnType (*class in pytext.models.representations.stacked_bidirectional_rnn*), 656
RoBERTa (*class in pytext.models.roberta*), 700
RoBERTaContextTensorizerForDenseRetrieval (*class in pytext.data.dense_retrieval_tensorizer*), 452
RoBERTaEncoder (*class in pytext.models.roberta*), 700
RoBERTaEncoderBase (*class in pytext.models.roberta*), 700
RoBERTaEncoderJit (*class in pytext.models.roberta*), 700
RoBERTaNERTask (*class in pytext.task.tasks*), 734
RoBERTaR3F (*class in pytext.models.roberta*), 701
RoBERTaRegression (*class in pytext.models.roberta*), 701
RoBERTaTensorizer (*class in pytext.data.roberta_tensorizer*), 459
RobertaTensorizerTest (*class in pytext.data.test.tensorizers_test*), 433
RoBERTaTokenLevelTensorizer (*class in pytext.data.roberta_tensorizer*), 459
RoBERTaWordTaggingModel (*class in pytext.models.roberta*), 701
ROC_AUC (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 518
roc_auc (*pytext.metrics.ClassificationMetrics attribute*), 553
roc_auc (*pytext.metrics.MultiLabelSoftClassificationMetrics attribute*), 556
roc_auc (*pytext.metrics.SoftClassificationMetrics attribute*), 559
Root (*class in pytext.data.data_structures.annotation*), 416
RootDataSource (*class in pytext.data.sources.data_source*), 422
round_seq() (*in module pytext.utils*), 770
RoundRobinBatchIterator (*class in pytext.data.disjoint_multitask_data_handler*), 455
RoundRobinBatchIteratorTest (*class in pytext.data.test.round_robin_batchiterator_test*), 432
RoundRobinBatchSampler (*class in pytext.data*), 491
RoundRobinBatchSampler (*class in pytext.data.batch_sampler*), 441
ROW_INDEX (*pytext.common.constants.RawExampleFieldName attribute*), 402
ROW_INDEX (*pytext.metric_reporters.squad_metric_reporter.SquadMetric attribute*), 531
ROW_INDEX (*pytext.metric_reporters.SquadMetricReporter attribute*), 541
row_size() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 476
RowData (*class in pytext.data.data*), 446
RowShardedDataSource (*class in pytext.data.sources.data_source*), 423
run() (*pytext.workflow.LogitsWriter method*), 771
run_epoch() (*pytext.trainers.hogwild_trainer.HogwildTrainer method*), 754
run_epoch() (*pytext.trainers.hogwild_trainer.HogwildTrainer_Deprecat method*), 755
run_epoch() (*pytext.trainers.HogwildTrainer method*), 760
run_epoch() (*pytext.trainers.HogwildTrainer_Deprecated method*), 760
run_epoch() (*pytext.trainers.Trainer method*), 758
run_epoch() (*pytext.trainers.trainer.Trainer method*), 756
run_single() (*in module pytext.main*), 771
run_step() (*pytext.trainers.TaskTrainer method*), 760
run_step() (*pytext.trainers.Trainer method*), 758
run_step() (*pytext.trainers.trainer.TaskTrainer method*), 755
run_step() (*pytext.trainers.trainer.Trainer method*), 756

## S

safe_division() (*in module pytext.metrics*), 565
SafeFileWrapper (*class in pytext.data.sources.data_source*), 423
samples (*pytext.metrics.RealtimeMetrics attribute*), 558
SamplewiseLabelSmoothingLoss (*class in pytext.loss*), 512
SamplewiseLabelSmoothingLoss (*class in pytext.loss.regularized_loss*), 509
save() (*in module pytext.task*), 736
save() (*in module pytext.task.serialize*), 730
save() (*pytext.task.serialize.CheckpointManager method*), 728
save_all_checkpoints (*pytext.config.pytext_config.PyTextConfig attribute*), 413
save_and_export() (*in module pytext.workflow*), 772

save_caffe2_pb_net() (*in module pytext.exporters.custom_exporters*), 494
save_checkpoint() (*in module pytext.task.serialize*), 730
save_checkpoint() (*pytext.task.serialize.CheckpointManager method*), 729
save_checkpoint() (*pytext.task.serialize.PyTextCheckpointManagerInterface method*), 729
save_checkpoint() (*pytext.trainers.Trainer method*), 758
save_checkpoint() (*pytext.trainers.trainer.Trainer method*), 756
save_model_state_for_all_rank() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
save_model_state_for_all_rank() (*pytext.optimizer.sparsifiers.sparsifier.Sparsifier method*), 711
save_module_checkpoints (*pytext.config.pytext_config.PyTextConfig attribute*), 413
save_modules() (*pytext.models.BaseModel method*), 706
save_modules() (*pytext.models.disjoint_multitask_model.DisjointMultiTaskModel method*), 688
save_modules() (*pytext.models.ensembles.ensemble.EnsembleModel method*), 592
save_modules() (*pytext.models.ensembles.EnsembleModel method*), 594
save_modules() (*pytext.models.model.BaseModel method*), 694
save_modules() (*pytext.models.pair_classification_model.BasePairwiseModel method*), 697
save_modules() (*pytext.models.pair_classification_model.PairwiseModel method*), 698
save_modules() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserModel method*), 664
save_pytext_snapshot() (*in module pytext.workflow*), 772
save_snapshot() (*pytext.task.serialize.CheckpointManager method*), 729
save_snapshot() (*pytext.task.serialize.PyTextCheckpointManagerInterface method*), 729
save_snapshot_path (*pytext.config.pytext_config.PyTextConfig attribute*), 413
scale_loss() (*in module pytext.optimizer.fp16_optimizer*), 717
scale_loss() (*pytext.optimizer.fp16_optimizer.FP16OptimizerDeprecat method*), 715
scale_loss() (*pytext.optimizer.fp16_optimizer.PureFP16Optimizer method*), 716
scheduler (*class in pytext.optimizer.scheduler*), 721
SCHEDULER (*pytext.config.component.ComponentType attribute*), 404
SchedulerWithWarmup (*class in pytext.optimizer.scheduler*), 721
ScriptableEmbeddingList (*class in pytext.models.embeddings.scriptable_embedding_list*), 578
ScriptableEmbeddingList.Wrapper1 (*class in pytext.models.embeddings.scriptable_embedding_list*), 578
ScriptableEmbeddingList.Wrapper3 (*class in pytext.models.embeddings.scriptable_embedding_list*), 578
ScriptBasedTokenTensorizer (*class in pytext.data.token_tensorizer*), 478
ScriptBatchInput (*class in pytext.torchscript.utils*), 753
ScriptBERTTensorizer (*class in pytext.torchscript.tensorizer*), 742
ScriptBERTTensorizer (*class in pytext.torchscript.tensorizer.bert*), 740
ScriptBERTTensorizerBase (*class in pytext.torchscript.tensorizer.bert*), 740
ScriptBPE (*class in pytext.torchscript.tokenizer*), 745
ScriptBPE (*class in pytext.torchscript.tokenizer.bpe*), 744
ScriptBPETokenizer (*class in pytext.torchscript.tokenizer*), 746
ScriptBPETokenizer (*class in pytext.torchscript.tokenizer.tokenizer*), 745
ScriptDoNothingTokenizer (*class in pytext.torchscript.tokenizer*), 746
ScriptDoNothingTokenizer (*class in pytext.torchscript.tokenizer.tokenizer*), 745
ScriptedSequenceGenerator (*class in pytext.torchscript.seq2seq.scripted_seq2seq_generator*), 739
ScriptFloat1DListTensorizer (*class in pytext.torchscript.tensorizer*), 742
ScriptFloat1DListTensorizer (*class in pytext.torchscript.tensorizer.tensorizer*), 741
ScriptFloatListSeqTensorizer (*class in pytext.torchscript.tensorizer*), 742

ScriptFloatListSeqTensorizer (*class in pytext.torchscript.tensorizer.tensorizer*), 741
ScriptInteger1DListTensorizer (*class in pytext.torchscript.tensorizer*), 742
ScriptInteger1DListTensorizer (*class in pytext.torchscript.tensorizer.tensorizer*), 742
ScriptPyTextEmbeddingModule (*class in pytext.torchscript.module*), 750
ScriptPyTextEmbeddingModuleIndex (*class in pytext.torchscript.module*), 751
ScriptPyTextEmbeddingModuleWithDense (*class in pytext.torchscript.module*), 751
ScriptPyTextEmbeddingModuleWithDenseIndex (*class in pytext.torchscript.module*), 751
ScriptPyTextModule (*class in pytext.torchscript.module*), 751
ScriptPyTextModuleWithDense (*class in pytext.torchscript.module*), 752
ScriptPyTextTwoTowerEmbeddingModule (*class in pytext.torchscript.module*), 752
ScriptPyTextTwoTowerEmbeddingModuleWithDense (*class in pytext.torchscript.module*), 752
ScriptPyTextTwoTowerModule (*class in pytext.torchscript.module*), 752
ScriptPyTextTwoTowerModuleWithDense (*class in pytext.torchscript.module*), 752
ScriptPyTextVariableSizeEmbeddingModule (*class in pytext.torchscript.module*), 753
ScriptRoBERTaTensorizer (*class in pytext.torchscript.tensorizer*), 743
ScriptRoBERTaTensorizer (*class in pytext.torchscript.tensorizer.roberta*), 741
ScriptRoBERTaTensorizerWithIndices (*class in pytext.torchscript.tensorizer*), 743
ScriptRoBERTaTensorizerWithIndices (*class in pytext.torchscript.tensorizer.roberta*), 741
ScriptTensorizer (*class in pytext.torchscript.tensorizer*), 744
ScriptTensorizer (*class in pytext.torchscript.tensorizer.tensorizer*), 742
ScriptTokenizerBase (*class in pytext.torchscript.tokenizer*), 746
ScriptTokenizerBase (*class in pytext.torchscript.tokenizer.tokenizer*), 745
ScriptTwoTowerModule (*class in pytext.torchscript.module*), 753
ScriptVocabulary (*class in pytext.torchscript.vocab*), 754
ScriptWordTokenizer (*class in pytext.torchscript.tokenizer*), 746
ScriptWordTokenizer (*class in pytext.torchscript.tokenizer.tokenizer*), 745
ScriptXLMTensorizer (*class in pytext.torchscript.tensorizer*), 743
ScriptXLMTensorizer (*class in pytext.torchscript.tensorizer.xlm*), 742
select_key_and_batch() (*in module pytext.data.batch_sampler*), 441
SelfAttention (*class in pytext.models.representations.pooling*), 654
SELFIE (*class in pytext.models.roberta*), 702
SELFIE_RAW_IMAGE (*pytext.common.constants.SpecialTokens attribute*), 402
SELFIETransformer (*class in pytext.models.representations.transformer*), 635
SELFIETransformer (*class in pytext.models.representations.transformer.transformer*), 629
SemanticParsingTask (*class in pytext.task.tasks*), 734
sensitivity_analysis() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
SensitivityAnalysisSparsifier (*class in pytext.optimizer.sparsifiers.sparsifier*), 710
SentenceEncoder (*class in pytext.models.representations.transformer*), 634
SentenceEncoder (*class in pytext.models.representations.transformer.sentence_encoder*), 627
SentencePieceTokenizer (*class in pytext.data.tokenizers*), 439
SentencePieceTokenizer (*class in pytext.data.tokenizers.tokenizer*), 437
SentencePieceTokenizerTest (*class in pytext.data.test.tokenizers_test*), 435
SeparableConv1d (*class in pytext.models.representations.deepcnn*), 647
SEQ (*pytext.config.contextual_intent_slot.ModelInput attribute*), 407
Seq2SeqCompositionalMetricReporter (*class in pytext.metric_reporters.seq2seq_compositional*), 529
Seq2SeqFileChannel (*class in pytext.metric_reporters.seq2seq_metric_reporter*), 529
Seq2SeqJIT (*class in pytext.torchscript.seq2seq.export_model*), 739
Seq2SeqMetricReporter (*class in pytext.metric_reporters.seq2seq_metric_reporter*), 530
Seq2SeqMetrics (*class in pytext.metrics.seq2seq_metrics*), 552
Seq2SeqModel (*class in py-*

*text.models.seq_models.seq2seq_model*), 683
Seq2SeqOutputLayer (*class in pytext.models.seq_models.seq2seq_output_layer*), 684
Seq2SeqTopKMetrics (*class in pytext.metrics.seq2seq_metrics*), 552
SEQ_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
SEQ_LENS (*pytext.common.constants.DatasetFieldName attribute*), 401
SEQ_LENS (*pytext.config.doc_classification.ModelInput attribute*), 408
seq_padding_control (*pytext.config.pytext_config.ExportConfig attribute*), 411
seq_padding_control (*pytext.config.pytext_config.PyTextConfig attribute*), 413
seq_word_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 407
seq_word_feat (*pytext.config.field_config.FeatureConfig attribute*), 409
SeqFeatureField (*class in pytext.fields*), 507
SeqFeatureField (*class in pytext.fields.field*), 501
SEQLOGICAL (*pytext.common.constants.DFColumn attribute*), 401
SeqNNModel (*class in pytext.models.seq_models.seqnn*), 685
SeqNNModel_Deprecated (*class in pytext.models.seq_models.seqnn*), 685
SeqNNTask (*class in pytext.task.tasks*), 734
SeqRepresentation (*class in pytext.models.representations.seq_rep*), 655
SeqTokenTensorizer (*class in pytext.data.tensorizers*), 471
SequenceAlignedAttention (*class in pytext.models.representations.attention*), 637
SequenceLabelingTask (*class in pytext.task.tasks*), 735
SequenceTaggingMetricReporter (*class in pytext.metric_reporters*), 543
SequenceTaggingMetricReporter (*class in pytext.metric_reporters.word_tagging_metric_reporter*), 532
SERIALIZED_EMBED (*pytext.common.constants.PackageFileName attribute*), 402
SessionDataSource (*class in pytext.data.sources.session*), 426
SessionPandasDataSource (*class in pytext.data.sources.pandas*), 425
SessionTSVDataSource (*class in pytext.data.sources.tsv*), 427
SessionTSVDataSourceTest (*class in pytext.data.test.tsv_data_source_test*), 435
set_checkpoint_manager() (*in module pytext.task.serialize*), 730
set_device() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 476
set_export_accelerate() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_batch_padding_control() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_caffe2_path() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_inference_interface() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_onnx_path() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_seq_padding_control() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_target() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_torchscript_path() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_export_torchscript_quantize() (*pytext.config.pytext_config.PyTextConfig method*), 413
set_fp16() (*in module pytext.utils.precision*), 769
set_incremental_state() (*pytext.models.seq_models.base.PyTextIncrementalDecoderComponent method*), 667
set_padding_control() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 476
set_padding_control() (*pytext.torchscript.tensorizer.ScriptTensorizer method*), 744
set_padding_control() (*pytext.torchscript.tensorizer.tensorizer.ScriptTensorizer method*), 742
set_random_seeds() (*in module pytext.utils*), 770
set_transitions() (*pytext.models.crf.CRF method*), 687
set_up_training() (*pytext.trainers.hogwild_trainer.HogwildTrainer method*), 754

set_up_training() (*pytext.trainers.hogwild_trainer.HogwildTrainer_Deprecated method*), 755
set_up_training() (*pytext.trainers.HogwildTrainer method*), 760
set_up_training() (*pytext.trainers.HogwildTrainer_Deprecated method*), 760
set_up_training() (*pytext.trainers.Trainer method*), 758
set_up_training() (*pytext.trainers.trainer.Trainer method*), 756
setUp() (*pytext.data.test.batch_sampler_test.BatchSamplerTest method*), 431
setUp() (*pytext.data.test.data_test.DataTest method*), 431
setUp() (*pytext.data.test.mask_tensorizers_test.MaskTensorizersTest method*), 432
setUp() (*pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method*), 432
setUp() (*pytext.data.test.tensorizers_test.ListTensorizersTest method*), 433
setUp() (*pytext.data.test.tensorizers_test.SquadTensorizerTest method*), 433
setUp() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 434
setUp() (*pytext.data.test.tsv_data_source_test.SessionTSVDataSourceTest method*), 435
setUp() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 436
SGD (*class in pytext.optimizer.optimizers*), 718
shard() (*in module pytext.data.utils*), 481
ShardedDataSource (*class in pytext.data.sources.data_source*), 424
SharedCNNRepresentation (*class in pytext.models.representations.jointcnn_rep*), 650
should_iter() (*in module pytext.data.utils*), 481
should_mask() (*pytext.data.masked_util.MaskingFunction method*), 458
should_mask() (*pytext.data.masked_util.TreeMask method*), 458
shuffle (*pytext.data.data_handler.DataHandler attribute*), 448
shuffle (*pytext.data.DataHandler attribute*), 485
sigmoid_cosine (*pytext.models.output_layers.distance_output_layer.OutputScore attribute*), 598
simple_tokenize() (*in module pytext.utils.data*), 762
SimpleFeaturizer (*class in pytext.data.featurizer*), 420
SimpleFeaturizer (*class in pytext.data.featurizer.simple_featurizer*), 419
SimpleFeaturizerTest (*class in pytext.data.test.simple_featurizer_test*), 432
SINUSOIDAL (*pytext.models.seq_models.positional.PostionalEmbedType attribute*), 679
SinusoidalPositionalEmbedding (*class in pytext.models.seq_models.positional*), 679
size_from_data (*pytext.data.tensorizers.VocabConfig attribute*), 477
size_limit (*pytext.data.tensorizers.VocabFileConfig attribute*), 477
skip_header_line (*pytext.data.tensorizers.VocabFileConfig attribute*), 477
Slot (*class in pytext.data.data_structures.annotation*), 416
Slot (*class in pytext.utils.data*), 761
slot_confusions (*pytext.metrics.intent_slot_metrics.IntentSlotConfusions attribute*), 547
slot_metrics (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics attribute*), 547, 548
SlotAttention (*class in pytext.models.representations.slot_attention*), 655
SlotAttentionType (*class in pytext.config.module_config*), 410
SlotLabelTensorizer (*class in pytext.data.tensorizers*), 472
SlotLabelTensorizerExpansible (*class in pytext.data.tensorizers*), 473
slots (*pytext.metrics.intent_slot_metrics.IntentsAndSlots attribute*), 548
Snapshot (*class in pytext.utils.timing*), 769
snapshot() (*pytext.utils.timing.HierarchicalTimer method*), 769
SnapshotList (*class in pytext.utils.timing*), 769
SoftClassificationMetrics (*class in pytext.metrics*), 559
SoftLabelTensorizer (*class in pytext.data.tensorizers*), 473
sort() (*in module pytext.task.new_task*), 727
sort_by_score() (*in module pytext.metrics*), 565
sort_key() (*pytext.data.bert_tensorizer.BERTTensorizerBase method*), 442
sort_key() (*pytext.data.data_handler.DataHandler method*), 450
sort_key() (*pytext.data.DataHandler method*), 487
sort_key() (*pytext.data.squad_tensorizer.SquadTensorizer method*), 462
sort_key() (*pytext.data.Tensorizer method*), 492
sort_key() (*pytext.data.tensorizers.ByteTensorizer method*), 463

sort_key() (*pytext.data.tensorizers.ByteTokenTensorizer method*), 464
sort_key() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 464
sort_key() (*pytext.data.tensorizers.LabelListRankTensorizer method*), 470
sort_key() (*pytext.data.tensorizers.LabelListTensorizer method*), 470
sort_key() (*pytext.data.tensorizers.SeqTokenTensorizer method*), 472
sort_key() (*pytext.data.tensorizers.Tensorizer method*), 475
sort_key() (*pytext.data.tensorizers.TokenTensorizer method*), 477
sort_key() (*pytext.data.token_tensorizer.ScriptBasedTokenTensorizer method*), 479
sort_within_batch (*pytext.data.data_handler.DataHandler attribute*), 448
sort_within_batch (*pytext.data.DataHandler attribute*), 485
SOURCE_FEATS (*pytext.common.constants.DFColumn attribute*), 401
SOURCE_SEQ_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
SOURCE_SEQUENCE (*pytext.common.constants.DFColumn attribute*), 401
SourceType (*class in pytext.loss*), 512
SourceType (*class in pytext.loss.loss*), 509
Span (*class in pytext.data.data_structures.node*), 417
Span (*class in pytext.metric_reporters.word_tagging_metric_reporter*), 533
span (*pytext.data.data_structures.node.Node attribute*), 417
span (*pytext.metrics.intent_slot_metrics.Node attribute*), 548
SPAN_PAD_IDX (*pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizer attribute*), 460
SPAN_PAD_IDX (*pytext.data.squad_tensorizer.SquadTensorizer attribute*), 462
SPAN_PAD_IDX (*pytext.data.tensorizers.Integer1DListTensorizer attribute*), 469
SparseTransformerSentenceEncoder (*class in pytext.models.representations.sparse_transformer_sentence_encoder*), 656
sparsification_condition() (*pytext.optimizer.sparsifiers.sparsifier.CRF_SparsifierBase method*), 709
sparsification_condition() (*pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier method*), 710
sparsification_condition() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
sparsification_condition() (*pytext.optimizer.sparsifiers.sparsifier.Sparsifier method*), 712
sparsification_step() (*pytext.trainers.Trainer method*), 758
sparsification_step() (*pytext.trainers.trainer.Trainer method*), 756
Sparsifier (*class in pytext.optimizer.sparsifiers.sparsifier*), 711
SPARSIFIER (*pytext.config.component.ComponentType attribute*), 404
sparsify() (*pytext.optimizer.sparsifiers.sparsifier.CRF_L1_SoftThreshold method*), 709
sparsify() (*pytext.optimizer.sparsifiers.sparsifier.CRF_MagnitudeThreshold method*), 709
sparsify() (*pytext.optimizer.sparsifiers.sparsifier.L0_projection_sparsifier method*), 710
sparsify() (*pytext.optimizer.sparsifiers.sparsifier.SensitivityAnalysisSparsifier method*), 711
sparsify() (*pytext.optimizer.sparsifiers.sparsifier.Sparsifier method*), 712
SpecialTokens (*class in pytext.common.constants*), 402
SquadDataSource (*class in pytext.data.sources*), 428
SquadDataSource (*class in pytext.data.sources.squad*), 426
SquadDataSourceForKD (*class in pytext.data.sources.squad*), 427
SquadFileChannel (*class in pytext.metric_reporters.squad_metric_reporter*), 530
SquadForBERTTensorizer (*class in pytext.data.squad_for_bert_tensorizer*), 460
SquadForBERTTensorizerForKD (*class in pytext.data.squad_for_bert_tensorizer*), 460
SquadForBERTTensorizerTest (*class in pytext.data.test.tensorizers_test*), 433
SquadForRoBERTaTensorizer (*class in pytext.data.squad_for_bert_tensorizer*), 460
SquadForRoBERTaTensorizerForKD (*class in pytext.data.squad_for_bert_tensorizer*), 461
SquadForRobertaTensorizerTest (*class in pytext.data.test.tensorizers_test*), 433
SquadMetricReporter (*class in pytext.metric_reporters*), 541
SquadMetricReporter (*class in pytext.metric_reporters.squad_metric_reporter*), 531
SquadMetrics (*class in pytext.metrics.squad_metrics*), 552
SquadOutputLayer (*class in py-*

*text.models.output_layers.squad_output_layer*), 609
SquadQATask (*class in pytext.task.tasks*), 735
SquadTensorizer (*class in pytext.data.squad_tensorizer*), 461
SquadTensorizerForKD (*class in pytext.data.squad_tensorizer*), 462
SquadTensorizerTest (*class in pytext.data.test.tensorizers_test*), 433
StackedBidirectionalRNN (*class in pytext.models.representations.stacked_bidirectional_rnn*), 656
StackLSTM (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 661
Stage (*class in pytext.common.constants*), 402
stage (*pytext.trainers.training_state.TrainingState attribute*), 757
stage (*pytext.trainers.TrainingState attribute*), 759
stages (*pytext.metric_reporters.Channel attribute*), 534
stages (*pytext.metric_reporters.channel.Channel attribute*), 514
start (*pytext.data.data_structures.node.Span attribute*), 418
start (*pytext.data.tokenizers.Token attribute*), 438
start (*pytext.data.tokenizers.tokenizer.Token attribute*), 438
start (*pytext.metric_reporters.word_tagging_metric_reporter.Span attribute*), 533
start_batch_size (*pytext.data.dynamic_pooling_batcher.BatcherSchedulerConfig attribute*), 456
State (*class in pytext.optimizer.sparsifiers.sparsifier*), 712
state_dict() (*pytext.models.distributed_model.DistributedModel method*), 689
state_dict() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16OptimizerMixin method*), 713
state_dict() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_MemoryEfficientFP16OptimizerMixin method*), 713
state_dict() (*pytext.optimizer.fp16_optimizer.FP16Optimizer method*), 714
state_dict() (*pytext.optimizer.fp16_optimizer.FP16OptimizerApex method*), 714
state_dict() (*pytext.optimizer.fp16_optimizer.FP16OptimizerDeprecated method*), 715
state_dict() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 723
state_linearity (*pytext.models.representations.augmented_lstm.AugmentedLSTMCell attribute*), 639
step() (*pytext.optimizer.adabelief.AdaBelief method*), 712
step() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16OptimizerMixin method*), 713
step() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_MemoryEfficientFP16 method*), 713
step() (*pytext.optimizer.fp16_optimizer.FP16Optimizer method*), 714
step() (*pytext.optimizer.fp16_optimizer.FP16OptimizerApex method*), 714
step() (*pytext.optimizer.fp16_optimizer.FP16OptimizerDeprecated method*), 715
step() (*pytext.optimizer.fp16_optimizer.GeneratorFP16Optimizer method*), 715
step() (*pytext.optimizer.fp16_optimizer.PureFP16Optimizer method*), 716
step() (*pytext.optimizer.lamb.Lamb method*), 717
step() (*pytext.optimizer.madgrad.MADGRAD method*), 718
step() (*pytext.optimizer.radam.RAdam method*), 719
step() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 723
step_batch() (*pytext.optimizer.scheduler.CosineAnnealingLR method*), 720
step_batch() (*pytext.optimizer.scheduler.CyclicLR method*), 720
step_batch() (*pytext.optimizer.scheduler.LmFineTuning method*), 720
step_batch() (*pytext.optimizer.scheduler.PolynomialDecayScheduler method*), 721
step_batch() (*pytext.optimizer.scheduler.Scheduler method*), 721
step_batch() (*pytext.optimizer.scheduler.SchedulerWithWarmup method*), 722
step_batch() (*pytext.optimizer.scheduler.WarmupScheduler method*), 722
step_counter (*pytext.trainers.training_state.TrainingState attribute*), 757
step_counter (*pytext.trainers.TrainingState attribute*), 759
step_epoch() (*pytext.data.dynamic_pooling_batcher.DynamicPoolingBatcher method*), 457
step_epoch() (*pytext.data.DynamicPoolingBatcher method*), 489
step_epoch() (*pytext.optimizer.scheduler.ExponentialLR method*), 720
step_epoch() (*pytext.optimizer.scheduler.ReduceLROnPlateau method*), 721
step_epoch() (*pytext.optimizer.scheduler.Scheduler method*), 721
step_epoch() (*pytext.optimizer.scheduler.SchedulerWithWarmup method*), 722
step_epoch() (*pytext.optimizer.scheduler.StepLR method*), 722
step_size (*pytext.data.dynamic_pooling_batcher.BatcherSchedulerConfig attribute*), 456

StepLR (*class in pytext.optimizer.scheduler*), 722
StochasticWeightAveraging (*class in pytext.optimizer.swa*), 722
String2DListTensorizer (*class in pytext.data.tensorizers*), 473
String2DListTensorizerScriptImpl (*class in pytext.data.tensorizers*), 474
String2DListTensorizerTest (*class in pytext.data.test.tensorizers_test*), 434
stringify() (*in module pytext.metric_reporters.seq2seq_utils*), 530
stringify() (*pytext.data.Tensorizer method*), 492
stringify() (*pytext.data.tensorizers.Tensorizer method*), 475
stringify_annotation_tree() (*pytext.metric_reporters.seq2seq_compositional.Seq2SeqCompositionalMetricReporter method*), 529
strip_bio_prefix() (*in module pytext.utils.data*), 762
StructuredLoss (*class in pytext.loss*), 512
StructuredLoss (*class in pytext.loss.structured_loss*), 510
StructuredMarginLoss (*class in pytext.loss*), 512
StructuredMarginLoss (*class in pytext.loss.structured_loss*), 510
subconfigs() (*pytext.config.component.Registry class method*), 404
SUM (*pytext.models.seq_models.positional.PostionalEmbedCombine attribute*), 679
summary_writer (*pytext.metric_reporters.channel.TensorBoardChannel attribute*), 515
SUPPORT_FP16_OPTIMIZER (*pytext.models.BaseModel attribute*), 706
SUPPORT_FP16_OPTIMIZER (*pytext.models.masked_lm.MaskedLanguageModel attribute*), 692
SUPPORT_FP16_OPTIMIZER (*pytext.models.model.BaseModel attribute*), 693
SUPPORT_FP16_OPTIMIZER (*pytext.models.two_tower_classification_model.TwoTowerClassificationModel attribute*), 702
SUPPORT_FP16_OPTIMIZER (*pytext.models.TwoTowerClassificationModel attribute*), 706
supports_flat_params (*pytext.optimizer.madgrad.MADGRAD attribute*), 718
supports_memory_efficient_fp16 (*pytext.optimizer.madgrad.MADGRAD attribute*), 718
suppress_output() (*in module pytext.utils.distributed*), 763
swap_modules_for_accelerator() (*in module pytext.task.accelerator_lowering*), 725


## T


TANH (*pytext.config.module_config.Activation attribute*), 409
Target (*class in pytext.config.field_config*), 409
target (*pytext.config.pytext_config.ExportConfig attribute*), 411
target (*pytext.config.pytext_config.PyTextConfig attribute*), 413
TARGET_LABEL_FIELD (*pytext.config.field_config.Target attribute*), 409
TARGET_LABELS (*pytext.common.constants.DFColumn attribute*), 401
TARGET_LOGITS (*pytext.common.constants.DFColumn attribute*), 401
TARGET_LOGITS_FIELD (*pytext.config.field_config.Target attribute*), 409
target_prob (*pytext.config.field_config.DocLabelConfig attribute*), 408
TARGET_PROB_FIELD (*pytext.config.field_config.Target attribute*), 409
TARGET_PROBS (*pytext.common.constants.DFColumn attribute*), 401
TARGET_SEQ_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 401
TARGET_SEQ_LENS (*pytext.common.constants.DatasetFieldName attribute*), 401
TARGET_SEQUENCE (*pytext.common.constants.DFColumn attribute*), 401
target_task_name (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH attribute*), 434
target_task_name (*pytext.data.DisjointMultitaskDataHandler attribute*), 488
target_time_limit_seconds (*pytext.trainers.Trainer attribute*), 758
target_time_limit_seconds (*pytext.trainers.trainer.Trainer attribute*), 755
TARGET_TOKENS (*pytext.common.constants.DFColumn attribute*), 401
targets_to_report() (*pytext.metric_reporters.classification_metric_reporter.Classificatio*

*method*), 518
targets_to_report() (*pytext.metric_reporters.classification_metric_reporter.MultiLabelClassificationMetricReporter method*), 520
targets_to_report() (*pytext.metric_reporters.ClassificationMetricReporter method*), 537
targets_to_report() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method*), 520
targets_to_report() (*pytext.metric_reporters.CompositionalMetricReporter method*), 542
targets_to_report() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 523
targets_to_report() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 540
targets_to_report() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 527
targets_to_report() (*pytext.metric_reporters.MetricReporter method*), 536
targets_to_report() (*pytext.metric_reporters.MultiLabelClassificationMetricReporter method*), 538
TargetTest (*class in pytext.data.test.utils_test*), 436
TASK (*pytext.config.component.ComponentType attribute*), 404
Task_Deprecated (*class in pytext.task*), 735
Task_Deprecated (*class in pytext.task.task*), 731
TASK_NAME (*pytext.common.constants.BatchContext attribute*), 400
TaskBase (*class in pytext.task*), 735
TaskBase (*class in pytext.task.task*), 730
TaskTrainer (*class in pytext.trainers*), 760
TaskTrainer (*class in pytext.trainers.trainer*), 755
temperature (*pytext.models.decoders.mlp_decoder.MLPDecoder.Config attribute*), 156
tensor() (*in module pytext.utils.cuda*), 761
TensorBoardChannel (*class in pytext.metric_reporters.channel*), 515
tensorize() (*pytext.data.bert_tensorizer.BERTTensorizerBase method*), 443
tensorize() (*pytext.data.bert_tensorizer.BERTTensorizerBaseScriptImpl method*), 443
tensorize() (*pytext.data.dense_retrieval_tensorizer.BERTContextNoDocTensorizerForDenseRetrieval method*), 451
tensorize() (*pytext.data.dense_retrieval_tensorizer.PositiveLabelMaskTensorizerForDenseRetrieval method*), 452
tensorize() (*pytext.data.masked_tensorizer.MaskedTokenTensorizer method*), 458
tensorize() (*pytext.data.roberta_tensorizer.RoBERTaTokenLevelTensorizer method*), 460
tensorize() (*pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizer method*), 460
tensorize() (*pytext.data.squad_for_bert_tensorizer.SquadForBERTTensorizerForKD method*), 460
tensorize() (*pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizer method*), 461
tensorize() (*pytext.data.squad_tensorizer.SquadTensorizer method*), 462
tensorize() (*pytext.data.squad_tensorizer.SquadTensorizerForKD method*), 462
tensorize() (*pytext.data.Tensorizer method*), 492
tensorize() (*pytext.data.tensorizers.AnnotationNumberizer method*), 463
tensorize() (*pytext.data.tensorizers.ByteTensorizer method*), 463
tensorize() (*pytext.data.tensorizers.ByteTokenTensorizer method*), 464
tensorize() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 464
tensorize() (*pytext.data.tensorizers.CharacterVocabTokenTensorizer method*), 465
tensorize() (*pytext.data.tensorizers.CharacterVocabTokenTensorizerScriptImpl method*), 466
tensorize() (*pytext.data.tensorizers.Float1DListTensorizer method*), 466
tensorize() (*pytext.data.tensorizers.FloatListSeqTensorizer method*), 467
tensorize() (*pytext.data.tensorizers.FloatListTensorizer method*), 467
tensorize() (*pytext.data.tensorizers.FloatTensorizer method*), 468
tensorize() (*pytext.data.tensorizers.GazetteerTensorizer method*), 469
tensorize() (*pytext.data.tensorizers.Integer1DListTensorizer method*), 469
tensorize() (*pytext.data.tensorizers.LabelListRankTensorizer method*), 470
tensorize() (*pytext.data.tensorizers.LabelListTensorizer method*), 470
tensorize() (*pytext.data.tensorizers.LabelTensorizer method*), 470
tensorize() (*pytext.data.tensorizers.MetricTensorizer method*), 471
tensorize() (*pytext.data.tensorizers.NtokensTensorizer method*), 471
tensorize() (*pytext.data.tensorizers.NumericLabelTensorizer method*), 471
tensorize() (*pytext.data.tensorizers.SeqTokenTensorizer method*), 472
tensorize() (*pytext.data.tensorizers.SlotLabelTensorizer method*), 473
tensorize() (*pytext.data.tensorizers.SoftLabelTensorizer*

*method*), 473
tensorize() (*pytext.data.tensorizers.String2DListTensorizer method*), 474
tensorize() (*pytext.data.tensorizers.String2DListTensorizerScriptImpl method*), 474
tensorize() (*pytext.data.tensorizers.Tensorizer method*), 475
tensorize() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 476
tensorize() (*pytext.data.tensorizers.TokenTensorizer method*), 477
tensorize() (*pytext.data.tensorizers.UidTensorizer method*), 477
tensorize() (*pytext.data.token_tensorizer.ScriptBasedTokenTensorizer method*), 479
tensorize() (*pytext.data.token_tensorizer.TokenTensorizerScriptImpl method*), 479
tensorize_wrapper() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 476
Tensorizer (*class in pytext.data*), 491
Tensorizer (*class in pytext.data.tensorizers*), 474
TENSORIZER (*pytext.config.component.ComponentType attribute*), 404
tensorizer_script_impl (*pytext.data.bert_tensorizer.BERTTensorizerBase attribute*), 443
tensorizer_script_impl (*pytext.data.Tensorizer attribute*), 492
tensorizer_script_impl (*pytext.data.tensorizers.CharacterVocabTokenTensorizer attribute*), 465
tensorizer_script_impl (*pytext.data.tensorizers.Float1DListTensorizer attribute*), 466
tensorizer_script_impl (*pytext.data.tensorizers.FloatListSeqTensorizer attribute*), 467
tensorizer_script_impl (*pytext.data.tensorizers.Integer1DListTensorizer attribute*), 469
tensorizer_script_impl (*pytext.data.tensorizers.String2DListTensorizer attribute*), 474
tensorizer_script_impl (*pytext.data.tensorizers.Tensorizer attribute*), 475
tensorizer_script_impl (*pytext.data.token_tensorizer.ScriptBasedTokenTensorizer attribute*), 479
tensorizer_script_impl (*pytext.data.xlm_tensorizer.XLMTensorizer attribute*), 482
tensorizers (*pytext.trainers.training_state.TrainingState attribute*), 757
tensorizers (*pytext.trainers.TrainingState attribute*), 759
TensorizerScriptImpl (*class in pytext.data.tensorizers*), 475
TensorizersTest (*class in pytext.data.test.tensorizers_test*), 434
TEST (*pytext.common.constants.Stage attribute*), 402
test (*pytext.data.sources.data_source.DataSource attribute*), 422
test (*pytext.data.sources.data_source.RootDataSource attribute*), 423
test (*pytext.data.sources.DataSource attribute*), 428
test (*pytext.data.sources.dense_retrieval.DenseRetrievalDataSource attribute*), 425
test (*pytext.data.sources.DenseRetrievalDataSource attribute*), 430
test (*pytext.data.sources.squad.SquadDataSource attribute*), 427
test (*pytext.data.sources.SquadDataSource attribute*), 429
test (*pytext.data.sources.tsv.MultilingualTSVDataSource attribute*), 427
test() (*pytext.task.task.TaskBase method*), 731
test() (*pytext.task.TaskBase method*), 736
test() (*pytext.trainers.Trainer method*), 758
test() (*pytext.trainers.trainer.Trainer method*), 756
test_align_target_label() (*pytext.data.test.utils_test.TargetTest method*), 436
test_alternate_prob_batch_sampler() (*pytext.data.test.batch_sampler_test.BatchSamplerTest method*), 431
test_annotation_num() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 434
test_bad_quoting() (*pytext.data.test.tsv_data_source_test.BlockShardedTSVDataSourceT method*), 435
test_bad_quoting() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 436
test_basic_tree_masking() (*pytext.data.test.mask_tensorizers_test.MaskTensorizersTest method*), 432
test_batch_iterator() (*pytext.data.test.round_robin_batchiterator_test.RoundRobinBatchIt method*), 432
test_batch_size (*pytext.data.data_handler.DataHandler attribute*), 448
test_batch_size (*pytext.data.DataHandler attribute*), 486
test_batch_size_greater_than_data() (*py-*

*text.data.test.dynamic_pooling_batcher_test.DynamicPoolingBatcherTest* *method*), 432
test_batcher() (*pytext.data.test.data_test.BatcherTest* *method*), 431
test_bert_pair_tensorizer() (*pytext.data.test.tensorizers_test.BERTTensorizerTest* *method*), 433
test_bert_tensorizer() (*pytext.data.test.tensorizers_test.BERTTensorizerTest* *method*), 433
test_byte_tensors_error_code() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_character_vocab_token_tensorizer() (*pytext.data.test.tensorizers_test.CharacterVocabTokenTensorizerTest* *method*), 433
test_convert_to_bytes() (*pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest* *method*), 433
test_create_batches() (*pytext.data.test.data_test.DataTest* *method*), 431
test_create_batches_different_tensorizers() (*pytext.data.test.data_test.DataTest* *method*), 431
test_create_batches_with_cache() (*pytext.data.test.data_test.DataTest* *method*), 431
test_create_byte_tensors() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_create_byte_token_tensors() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_create_data_no_batcher_provided() (*pytext.data.test.data_test.DataTest* *method*), 431
test_create_data_source() (*pytext.data.test.pandas_data_source_test.PandasDataSourceTest* *method*), 432
test_create_float_list_seq_tensor() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_create_float_list_tensor() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_create_from_config() (*pytext.data.test.pandas_data_source_test.PandasDataSourceTest* *method*), 432
test_create_label_list_tensors() (*pytext.data.test.tensorizers_test.ListTensorizersTest* *method*), 433
test_create_label_tensors() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_create_label_tensors_add_labels() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_create_label_tensors_label_vocab() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_create_normalized_float_list_tensor() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_csv() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest* *method*), 436
test_data_initializes_tensorsizers() (*pytext.data.test.data_test.DataTest* *method*), 431
test_data_iterate_multiple_times() (*pytext.data.test.data_test.DataTest* *method*), 431
test_empty_data() (*pytext.data.test.pandas_data_source_test.PandasDataSourceTest* *method*), 432
test_eval_batch_sampler() (*pytext.data.test.batch_sampler_test.BatchSamplerTest* *method*), 431
test_exponential_scheduler() (*pytext.data.test.dynamic_pooling_batcher_test.DynamicPoolingBatc* *method*), 432
test_float_1D_list_tensorizer() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_float_list_seq_tensor_prepare_input() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_float_list_seq_torchscriptify() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_float_list_tensor_prepare_input() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_fp16_padding() (*pytext.data.test.data_test.DataTest* *method*), 431
test_gazetteer_tensor() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_gazetteer_tensor_bad_json() (*pytext.data.test.tensorizers_test.TensorizersTest* *method*), 434
test_gpt2_bpe_tokenizer() (*pytext.data.test.tokenizers_test.GPT2BPETest* *method*), 435
test_initialize() (*pytext.data.test.tensorizers_test.SquadTensorizerTest*

method), 434
test_initialize_label_tensorizer() (pytext.data.test.tensorizers_test.TensorizersTest method), 434
test_initialize_list_tensorizers() (pytext.data.test.tensorizers_test.ListTensorizersTest method), 433
test_initialize_tensorizers() (pytext.data.test.tensorizers_test.TensorizersTest method), 434
test_initialize_token_tensorizer() (pytext.data.test.tensorizers_test.TensorizersTest method), 434
test_input_text_truncation() (pytext.data.test.tokenizers_test.SentencePieceTokenizerTest method), 435
test_integer_1D_list_tensorizer() (pytext.data.test.tensorizers_test.TensorizersTest method), 434
test_iterate_training_data_multiple_times() (pytext.data.test.tsv_data_source_test.TSVDataSourceTest method), 436
test_label_list_tensors_no_pad_in_vocab() (pytext.data.test.tensorizers_test.ListTensorizersTest method), 433
test_label_list_tensors_pad_missing() (pytext.data.test.tensorizers_test.ListTensorizersTest method), 433
test_linear_scheduler() (pytext.data.test.dynamic_pooling_batcher_test.DynamicPoolingBatcherTest method), 432
test_lookup_tokens() (pytext.data.test.tensorizers_test.LookupTokensTest method), 433
test_mask_all() (pytext.data.test.mask_tensorizers_test.MaskTensorizersTest method), 432
test_mask_at_depth_k() (pytext.data.test.mask_tensorizers_test.MaskTensorizersTest method), 432
test_mask_no_op() (pytext.data.test.mask_tensorizers_test.MaskTensorizersTest method), 432
test_mask_random() (pytext.data.test.mask_tensorizers_test.MaskTensorizersTest method), 432
test_model() (in module pytext.workflow), 772
test_model_from_snapshot_path() (in module pytext.workflow), 772
test_numberize_with_alphanumeric() (pytext.data.test.tensorizers_test.SquadTensorizerTest method), 434
test_numberize_with_script_token_tensorizer() (pytext.data.test.tensorizers_test.TensorizersTest method), 435
test_numberize_with_token_tensorizer() (pytext.data.test.tensorizers_test.TensorizersTest method), 435
test_numberize_with_wordpiece() (pytext.data.test.tensorizers_test.SquadTensorizerTest method), 434
test_original() (pytext.data.test.tensorizers_test.String2DListTensorizerTest method), 434
test_out_path (pytext.config.pytext_config.PyTextConfig attribute), 413
test_out_path (pytext.config.pytext_config.TestConfig attribute), 413
test_path (pytext.config.pytext_config.TestConfig attribute), 413
test_path (pytext.data.data_handler.DataHandler attribute), 448
test_path (pytext.data.DataHandler attribute), 485
test_pooling_batcher() (pytext.data.test.data_test.BatcherTest method), 431
test_prob_batch_sampler() (pytext.data.test.batch_sampler_test.BatchSamplerTest method), 431
test_quoting() (pytext.data.test.tsv_data_source_test.BlockShardedTSVDataSourceTest method), 435
test_quoting() (pytext.data.test.tsv_data_source_test.TSVDataSourceTest method), 436
test_raw_example_hashable() (pytext.data.test.data_test.RawExampleTest method), 431
test_read_data_source() (pytext.data.test.tsv_data_source_test.TSVDataSourceTest method), 436
test_read_data_source_with_column_remapping() (pytext.data.test.tsv_data_source_test.TSVDataSourceTest method), 436
test_read_data_source_with_utf8_issues() (pytext.data.test.tsv_data_source_test.TSVDataSourceTest method), 436
test_read_eval_data_source() (pytext.data.test.tsv_data_source_test.TSVDataSourceTest method), 436
test_read_session_data() (pytext.data.test.tsv_data_source_test.SessionTSVDataSourceTest method), 436
test_read_test_data_source() (pytext.data.test.tsv_data_source_test.TSVDataSourceTest method), 436

test_roberta_tensorizer() (pytext.data.test.tensorizers_test.RobertaTensorizerTest method), 433
test_round_robin_batch_sampler() (pytext.data.test.batch_sampler_test.BatchSamplerTest method), 431
test_rows (pytext.data.test.tensorizers_test.String2DListTensorizerTest attribute), 434
test_seq_tensor() (pytext.data.test.tensorizers_test.TensorizersTest method), 435
test_seq_tensor_max_turn() (pytext.data.test.tensorizers_test.TensorizersTest method), 435
test_seq_tensor_pad_batch() (pytext.data.test.tensorizers_test.TensorizersTest method), 435
test_seq_tensor_with_bos_eos_eol_bol() (pytext.data.test.tensorizers_test.TensorizersTest method), 435
test_sort() (pytext.data.test.data_test.DataTest method), 431
test_split_with_regex() (pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method), 433
test_split_with_regex() (pytext.data.test.tokenizers_test.TokenizeTest method), 435
test_squad_roberta_tensorizer() (pytext.data.test.tensorizers_test.SquadForRobertaTensorizerTest method), 433
test_squad_tensorizer() (pytext.data.test.tensorizers_test.SquadForBERTTensorizerTest method), 433
test_step_size() (pytext.data.test.dynamic_pooling_batcher_test.DynamicPoolingBatcherTest method), 432
test_tensorize_with_script_token_tensorizer() (pytext.data.test.tensorizers_test.TensorizersTest method), 435
test_tokenize() (pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method), 433
test_tokenize() (pytext.data.test.tokenizers_test.SentencePieceTokenizerTest method), 435
test_tokenize() (pytext.data.test.tokenizers_test.TokenizeTest method), 435
test_tokenize_add_sentence_markers() (pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method), 433
test_tokenize_dont_lowercase() (pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method), 433
test_tokenize_dont_lowercase() (pytext.data.test.tokenizers_test.TokenizeTest method), 435
test_tokenize_no_byte_offsets() (pytext.data.test.tokenizers_test.TokenizeTest method), 435
test_tokenize_use_byte_offsets() (pytext.data.test.tokenizers_test.TokenizeTest method), 435
test_torchscriptified() (pytext.data.test.tensorizers_test.String2DListTensorizerTest method), 434
test_tree_mask_with_bos_eos() (pytext.data.test.mask_tensorizers_test.MaskTensorizersTest method), 432
test_tsv_numberize_with_alphanumeric() (pytext.data.test.tensorizers_test.SquadTensorizerTest method), 434
test_wordpiece_tokenizer() (pytext.data.test.tokenizers_test.WordpieceTokenizerTest method), 435
testBuildVocabulary() (pytext.data.test.utils_test.VocabularyTest method), 436
TestConfig (class in pytext.config.pytext_config), 413
testPadding() (pytext.data.test.utils_test.PaddingTest method), 436
testPaddingProvideShape() (pytext.data.test.utils_test.PaddingTest method), 436
TEXT (pytext.config.contextual_intent_slot.ModelInput attribute), 407
text (pytext.data.data_structures.node.Node attribute), 497
text (pytext.metrics.intent_slot_metrics.Node attribute), 548
TEXT1 (pytext.config.pair_classification.ModelInput attribute), 411
text1 (pytext.config.pair_classification.ModelInputConfig attribute), 411
TEXT2 (pytext.config.pair_classification.ModelInput attribute), 411
text2 (pytext.config.pair_classification.ModelInputConfig attribute), 411
text_feature_name (pytext.data.data_handler.DataHandler attribute), 448
text_feature_name (pytext.data.DataHandler attribute), 485
TEXT_FIELD (pytext.common.constants.DatasetFieldName attribute), 401
TextFeatureField (class in pytext.fields), 506

TextFeatureField (*class in pytext.fields.field*), 502
TextFeatureFieldWithSpecialUnk (*class in pytext.fields*), 507
TextFeatureFieldWithSpecialUnk (*class in pytext.fields.text_field_with_special_unk*), 503
texts (*pytext.torchscript.utils.ScriptBatchInput attribute*), 753
tied_representation (*pytext.models.pair_classification_model.PairwiseModel.Config attribute*), 210
tile_encoder_out() (*pytext.models.seq_models.conv_encoder.LightConvEncoder method*), 672
tile_encoder_out() (*pytext.models.seq_models.rnn_encoder.LSTMSequenceEncoder method*), 683
time() (*pytext.utils.timing.HierarchicalTimer method*), 769
TimeMeter (*class in pytext.utils.meter*), 768
Timings (*class in pytext.utils.timing*), 770
to_actions() (*pytext.data.data_structures.annotation.Tree method*), 416
to_device() (*in module pytext.data.tensorizers*), 478
to_onehot() (*in module pytext.utils.model*), 768
Token (*class in pytext.common.constants*), 402
Token (*class in pytext.data.data_structures.annotation*), 416
Token (*class in pytext.data.tokenizers*), 438
Token (*class in pytext.data.tokenizers.tokenizer*), 437
TOKEN_INDICES (*pytext.common.constants.DatasetFieldName attribute*), 401
Token_Info (*class in pytext.data.data_structures.annotation*), 416
token_label() (*pytext.utils.data.Slot method*), 762
TOKEN_LPROB (*pytext.models.seq_models.mask_generator.BeamRankingAlgorithm attribute*), 674
token_overlap() (*pytext.utils.data.Slot method*), 762
token_prob() (*in module pytext.models.seq_models.mask_generator*), 676
TOKEN_RANGE (*pytext.common.constants.DatasetFieldName attribute*), 402
TOKEN_RANGE (*pytext.common.constants.DFColumn attribute*), 401
TOKEN_RANGE (*pytext.config.contextual_intent_slot.ExtraField attribute*), 407
token_ranges (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 419
token_ranges (*pytext.data.featurizer.OutputRecord attribute*), 420
tokenize() (*in module pytext.data.tensorizers*), 478
tokenize() (*pytext.data.bert_tensorizer.BERTTensorizerBaseScriptImpl method*), 443
tokenize() (*pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method*), 419
tokenize() (*pytext.data.featurizer.SimpleFeaturizer method*), 421
tokenize() (*pytext.data.tensorizers.CharacterVocabTokenTensorizerScr method*), 466
tokenize() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 476
tokenize() (*pytext.data.token_tensorizer.TokenTensorizerScriptImpl method*), 479
tokenize() (*pytext.data.tokenizers.DoNothingTokenizer method*), 439
tokenize() (*pytext.data.tokenizers.GPT2BPETokenizer method*), 438
tokenize() (*pytext.data.tokenizers.SentencePieceTokenizer method*), 439
tokenize() (*pytext.data.tokenizers.Tokenizer method*), 439
tokenize() (*pytext.data.tokenizers.tokenizer.BERTInitialTokenizer method*), 437
tokenize() (*pytext.data.tokenizers.tokenizer.DoNothingTokenizer method*), 437
tokenize() (*pytext.data.tokenizers.tokenizer.GPT2BPETokenizer method*), 437
tokenize() (*pytext.data.tokenizers.tokenizer.SentencePieceTokenizer method*), 437
tokenize() (*pytext.data.tokenizers.tokenizer.Tokenizer method*), 438
tokenize() (*pytext.data.tokenizers.tokenizer.WordPieceTokenizer method*), 438
tokenize() (*pytext.data.tokenizers.WordPieceTokenizer method*), 439
tokenize_batch() (*pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method*), 419
tokenize_batch() (*pytext.data.featurizer.SimpleFeaturizer method*), 421
Tokenizer (*class in pytext.data.tokenizers*), 438
Tokenizer (*class in pytext.data.tokenizers.tokenizer*), 438
TOKENIZER (*pytext.config.component.ComponentType attribute*), 404
TokenizeTest (*class in pytext.data.test.tokenizers_test*), 435
TOKENS (*pytext.common.constants.DatasetFieldName attribute*), 401
tokens (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 419
tokens (*pytext.data.featurizer.OutputRecord attribute*), 420
tokens (*pytext.torchscript.utils.ScriptBatchInput attribute*), 753

TOKENS_COLUMN (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter attribute*), 524
TOKENS_COLUMN (*pytext.metric_reporters.LanguageModelMetricReporter attribute*), 540
TokenTensorizer (*class in pytext.data.tensorizers*), 476
TokenTensorizerScriptImpl (*class in pytext.data.token_tensorizer*), 479
top_intent_accuracy (*pytext.metrics.intent_slot_metrics.AllMetrics attribute*), 546
torchscript_export() (*pytext.task.disjoint_multitask.NewDisjointMultitask method*), 727
torchscript_export() (*pytext.task.tasks.PairwiseClassificationTask method*), 734
torchscript_export() (*pytext.task.tasks.SequenceLabelingTask method*), 735
torchscript_predictions() (*pytext.models.output_layers.CRFOutputLayer method*), 615
torchscript_predictions() (*pytext.models.output_layers.doc_classification_output_layer.BinaryClassificationOutputLayer method*), 599
torchscript_predictions() (*pytext.models.output_layers.doc_classification_output_layer.MultiClassOutputLayer method*), 601
torchscript_predictions() (*pytext.models.output_layers.doc_classification_output_layer.MultiLabelOutputLayer method*), 601
torchscript_predictions() (*pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer method*), 603
torchscript_predictions() (*pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer method*), 604
torchscript_predictions() (*pytext.models.output_layers.multi_label_classification_layer.MultiLabelClassificationLayer method*), 606
torchscript_predictions() (*pytext.models.output_layers.RegressionOutputLayer method*), 617
torchscript_predictions() (*pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer method*), 612
torchscript_predictions() (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method*), 613
torchscript_predictions() (*pytext.models.output_layers.WordTaggingOutputLayer method*), 618
torchscript_predictions() (*pytext.config.pytext_config.ExportConfig attribute*), 411
torchscript_quantize (*pytext.config.pytext_config.PyTextConfig attribute*), 413
torchscriptify() (*pytext.data.bert_tensorizer.BERTTensorizerBaseScriptImpl method*), 443
torchscriptify() (*pytext.data.roberta_tensorizer.RoBERTaTokenLevelTensorizer method*), 460
torchscriptify() (*pytext.data.squad_for_bert_tensorizer.SquadForRoBERTaTensorizer method*), 461
torchscriptify() (*pytext.data.Tensorizer method*), 492
torchscriptify() (*pytext.data.tensorizers.Tensorizer method*), 475
torchscriptify() (*pytext.data.tensorizers.TensorizerScriptImpl method*), 476
torchscriptify() (*pytext.data.tokenizers.DoNothingTokenizer method*), 438
torchscriptify() (*pytext.data.tokenizers.SentencePieceTokenizer method*), 439
torchscriptify() (*pytext.data.tokenizers.Tokenizer method*), 439
torchscriptify() (*pytext.data.tokenizers.tokenizer.DoNothingTokenizer method*), 437
torchscriptify() (*pytext.data.tokenizers.tokenizer.SentencePieceTokenizer method*), 437
torchscriptify() (*pytext.data.tokenizers.tokenizer.Tokenizer method*), 438
torchscriptify() (*pytext.models.doc_model.ByteTokensDocumentModel method*), 690
torchscriptify() (*pytext.models.doc_model.DocModel method*), 690
torchscriptify() (*pytext.models.doc_model.PersonalizedDocModel method*), 691
torchscriptify() (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsemble method*), 590
torchscriptify() (*py-*

*text.models.ensembles.BaggingIntentSlotEnsembleModel* *method*), 593
torchscriptify() (*pytext.models.ensembles.ensemble.EnsembleModel* *method*), 592
torchscriptify() (*pytext.models.ensembles.EnsembleModel* *method*), 594
torchscriptify() (*pytext.models.roberta.RoBERTa* *method*), 700
torchscriptify() (*pytext.models.roberta.RoBERTaRegression* *method*), 701
torchscriptify() (*pytext.models.seq_models.seq2seq_model.Seq2SeqModel* *method*), 684
torchscriptify() (*pytext.models.two_tower_classification_model.TwoTowerClassificationModel* *method*), 703
torchscriptify() (*pytext.models.TwoTowerClassificationModel* *method*), 707
torchscriptify() (*pytext.models.word_model.WordTaggingLiteModel* *method*), 703
torchscriptify() (*pytext.models.word_model.WordTaggingModel* *method*), 703
torchscriptify() (*pytext.torchscript.tensorizer.ScriptFloat1DListTensorizer* *method*), 742
torchscriptify() (*pytext.torchscript.tensorizer.ScriptFloatListSeqTensorizer* *method*), 742
torchscriptify() (*pytext.torchscript.tensorizer.ScriptInteger1DListTensorizer* *method*), 743
torchscriptify() (*pytext.torchscript.tensorizer.tensorizer.ScriptFloat1DListTensorizer* *method*), 741
torchscriptify() (*pytext.torchscript.tensorizer.tensorizer.ScriptFloatListSeqTensorizer* *method*), 741
torchscriptify() (*pytext.torchscript.tensorizer.tensorizer.ScriptInteger1DListTensorizer* *method*), 742
total_error (*pytext.metrics.calibration_metrics.CalibrationMetrics* *attribute*), 545
TP (*pytext.metrics.Confusions attribute*), 554
tps (*pytext.metrics.RealtimeMetrics attribute*), 558
trace() (*pytext.models.BaseModel method*), 706
trace() (*pytext.models.model.BaseModel method*), 694
trace() (*pytext.models.roberta.RoBERTa method*), 700
trace() (*pytext.models.two_tower_classification_model.TwoTowerClassi* *method*), 703
trace() (*pytext.models.TwoTowerClassificationModel* *method*), 707
TraceableTransformerWrapper (*class in pytext.models.representations.traced_transformer_encoder*), 657
TracedTransformerEncoder (*class in pytext.models.representations.traced_transformer_encoder*), 657
TRAIN (*pytext.common.constants.Stage attribute*), 402
train (*pytext.data.sources.data_source.DataSource attribute*), 422
train (*pytext.data.sources.data_source.RootDataSource attribute*), 423
train (*pytext.data.sources.data_source.RowShardedDataSource attribute*), 423
train (*pytext.data.sources.DataSource attribute*), 428
train (*pytext.data.sources.dense_retrieval.DenseRetrievalDataSource attribute*), 425
train (*pytext.data.sources.DenseRetrievalDataSource attribute*), 430
train (*pytext.data.sources.squad.SquadDataSource attribute*), 427
train (*pytext.data.sources.SquadDataSource attribute*), 429
train (*pytext.data.sources.tsv.MultilingualTSVDataSource attribute*), 427
train() (*pytext.models.BaseModel method*), 706
train() (*pytext.models.distributed_model.DistributedModel method*), 689
train() (*pytext.models.model.BaseModel method*), 694
train() (*pytext.task.task.TaskBase method*), 731
train() (*pytext.task.TaskBase method*), 736
train() (*pytext.trainers.ensemble_trainer.EnsembleTrainer method*), 754
train() (*pytext.trainers.EnsembleTrainer method*), 760
train() (*pytext.trainers.Trainer method*), 758
train() (*pytext.trainers.trainer.Trainer method*), 756
train() (*pytext.models.BaseModel class method*), 706
train_batch() (*pytext.models.model.BaseModel class method*), 694
train_batch() (*pytext.models.r3f_models.R3FPyTextMixin class method*), 700
train_batch() (*pytext.models.roberta.RoBERTaR3F class method*), 701
train_batch_size (*pytext.data.data_handler.DataHandler attribute*), 448
train_batch_size (*pytext.data.DataHandler*

attribute), 485
train_from_state() (pytext.trainers.Trainer method), 759
train_from_state() (pytext.trainers.trainer.Trainer method), 756
train_model() (in module pytext.workflow), 772
train_model_distributed() (in module pytext.main), 771
train_path (pytext.data.data_handler.DataHandler attribute), 448
train_path (pytext.data.DataHandler attribute), 485
train_single_model() (pytext.task.tasks.EnsembleTask method), 732
train_step (pytext.metric_reporters.channel.TensorBoardChannel attribute), 515
train_unsharded (pytext.data.sources.data_source.RowShardedDataSource attribute), 423
train_unsharded (pytext.data.sources.tsv.BlockShardedTSVDataSource attribute), 427
Trainer (class in pytext.trainers), 757
Trainer (class in pytext.trainers.trainer), 755
TRAINER (pytext.config.component.ComponentType attribute), 404
TrainerBase (class in pytext.trainers.trainer), 757
TrainingState (class in pytext.trainers), 759
TrainingState (class in pytext.trainers.training_state), 757
Transformer (class in pytext.models.representations.transformer), 635
Transformer (class in pytext.models.representations.transformer.transformer), 629
TransformerLayer (class in pytext.models.representations.transformer), 636
TransformerLayer (class in pytext.models.representations.transformer.transformer), 630
TransformerRepresentation (class in pytext.models.representations.transformer), 636
TransformerRepresentation (class in pytext.models.representations.transformer.representation), 626
TransformerSentenceEncoder (class in pytext.models.representations.transformer_sentence_encoder), 658
TransformerSentenceEncoderBase (class in pytext.models.representations.transformer_sentence_encoder_base), 659
translate_roberta_state_dict() (in module pytext.models.representations.transformer.sentence_encoder), 628
Tree (class in pytext.data.data_structures.annotation), 416
tree_from_tokens_and_indx_actions() (pytext.metric_reporters.compositional_metric_reporter.Composition static method), 520
tree_from_tokens_and_indx_actions() (pytext.metric_reporters.CompositionalMetricReporter static method), 542
tree_metrics (pytext.metrics.intent_slot_metrics.AllMetrics attribute), 546, 547
tree_to_metric_node() (pytext.metric_reporters.compositional_metric_reporter.Composition static method), 521
tree_to_metric_node() (pytext.metric_reporters.CompositionalMetricReporter static method), 542
TreeBuilder (class in pytext.data.data_structures.annotation), 417
TreeMask (class in pytext.data.masked_util), 458
Trim1d (class in pytext.models.representations.deepcnn), 648
true_positives (pytext.metrics.PRF1Scores attribute), 556, 557
true_positives_lower_bound() (in module pytext.utils.loss), 767
truncate_to_vocab_size() (pytext.data.utils.VocabBuilder method), 480
TSV (class in pytext.data.sources.tsv), 428
TSVDataSource (class in pytext.data.sources), 429
TSVDataSource (class in pytext.data.sources.tsv), 428
TSVDataSourceTest (class in pytext.data.test.tsv_data_source_test), 436
TwoTowerClassificationModel (class in pytext.models), 706
TwoTowerClassificationModel (class in pytext.models.two_tower_classification_model), 702

## U

UidTensorizer (class in pytext.data.tensorizers), 477
unfold1d() (in module pytext.models.seq_models.utils), 685
UNIFORM (pytext.models.r3f_models.R3FNoiseType attribute), 699
UniformRegularizer (class in pytext.loss), 512
UniformRegularizer (class in pytext.loss.regularizer), 510
UninitializedLazyModuleError, 765
UnknownTypeError, 414
UNK (pytext.common.constants.SpecialTokens attribute), 402

UNK_BYTE (*pytext.data.tensorizers.ByteTensorizer attribute*), 463
UNK_NUM_TOKEN (*pytext.common.constants.VocabMeta attribute*), 403
UNK_TOKEN (*pytext.common.constants.VocabMeta attribute*), 403
unkify() (*in module pytext.utils.data*), 762
unscale() (*pytext.optimizer.fp16_optimizer.DynamicLossScaler method*), 714
unscale_grads() (*pytext.optimizer.fp16_optimizer.DynamicLossScaler method*), 714
update() (*pytext.metrics.PerLabelConfusions method*), 558
update() (*pytext.utils.meter.Meter method*), 768
update() (*pytext.utils.meter.TimeMeter method*), 768
update_best_model() (*pytext.trainers.Trainer method*), 759
update_best_model() (*pytext.trainers.trainer.Trainer method*), 757
update_meta_data() (*pytext.torchscript.tensorizer.normalizer.VectorNormalizer method*), 741
update_meta_data() (*pytext.torchscript.tensorizer.VectorNormalizer method*), 743
update_scale() (*pytext.optimizer.fp16_optimizer.DynamicLossScaler method*), 714
update_swa() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 723
update_swa_group() (*pytext.optimizer.swa.StochasticWeightAveraging method*), 724
update_to_version() (*in module pytext.config.config_adapter*), 406
update_tree() (*pytext.data.data_structures.annotation.TreeBuilder method*), 417
upgrade_export_config() (*in module pytext.config.config_adapter*), 406
upgrade_if_xlm() (*in module pytext.config.config_adapter*), 406
upgrade_one_version() (*in module pytext.config.config_adapter*), 406
upgrade_padding() (*in module pytext.config.config_adapter*), 406
upgrade_state_dict_named() (*pytext.models.representations.transformer_sentence_encoder.TransformerSentenceEncoder method*), 659
upgrade_to_latest() (*in module pytext.config.config_adapter*), 406
ups (*pytext.metrics.RealtimeMetrics attribute*), 558
upsample (*pytext.data.disjoint_multitask_data_handler.DisjointMultitask attribute*), 454
upsample (*pytext.data.disjoint_multitask_data_handler.DisjointMultitask attribute*), 73
upsample (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatch attribute*), 456
upsample (*pytext.data.DisjointMultitaskDataHandler attribute*), 488
upscale() (*pytext.optimizer.fp16_optimizer.DynamicLossScaler method*), 714
use_action (*pytext.models.semantic_parsers.rnng.rnng_parser.Ablation attribute*), 268
use_bias (*pytext.models.representations.augmented_lstm.AugmentedLST attribute*), 227
use_bio_labels (*pytext.config.field_config.WordLabelConfig attribute*), 409
use_buffer (*pytext.models.semantic_parsers.rnng.rnng_parser.Ablation attribute*), 268
use_config_from_snapshot (*pytext.config.pytext_config.PyTextConfig attribute*), 413
use_crf (*pytext.models.ensembles.bagging_intent_slot_ensemble.Bagging attribute*), 590
use_crf (*pytext.models.ensembles.BaggingIntentSlotEnsembleModel attribute*), 592
use_cuda_for_testing (*pytext.config.pytext_config.PyTextConfig attribute*), 413
use_cuda_if_available (*pytext.config.pytext_config.PyTextConfig attribute*), 413
use_cuda_if_available (*pytext.config.pytext_config.TestConfig attribute*), 413
use_deterministic_cudnn (*pytext.config.pytext_config.PyTextConfig attribute*), 413
use_doc_attention (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDo attribute*), 643
use_doc_probs_in_word (*pytext.models.decoders.intent_slot_model_decoder.IntentSlotModel attribute*), 566
use_doc_probs_in_word (*pytext.models.decoders.intent_slot_model_decoder.IntentSlotModel attribute*), 155
use_doc_probs_in_word (*pytext.models.decoders.IntentSlotModelDecoder attribute*), 571
use_fp16 (*pytext.config.pytext_config.PyTextConfig attribute*), 413
use_fp16 (*pytext.config.pytext_config.TestConfig attribute*), 413

use_gzip (*pytext.config.pytext_config.LogitsConfig attribute*), 412
use_highway (*pytext.models.representations.augmented_lstm.AugmentedLSTM.Config attribute*), 227
use_last_open_NT_feature (*pytext.models.semantic_parsers.rnng.rnng_parser.AblationParams attribute*), 268
use_stack (*pytext.models.semantic_parsers.rnng.rnng_parser.AblationParams attribute*), 268
use_tensorboard (*pytext.config.pytext_config.PyTextConfig attribute*), 413
use_tensorboard (*pytext.config.pytext_config.TestConfig attribute*), 413
use_word_attention (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 644
UTTERANCE (*pytext.common.constants.DFColumn attribute*), 401
UTTERANCE (*pytext.config.contextual_intent_slot.ExtraField attribute*), 407
UTTERANCE_COLUMN (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter attribute*), 524
UTTERANCE_COLUMN (*pytext.metric_reporters.LanguageModelMetricReporter attribute*), 540
UTTERANCE_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 402
UTTERANCE_PAIR (*pytext.config.pair_classification.ExtraField attribute*), 411

## V

v0_to_v1() (*in module pytext.config.config_adapter*), 406
v12_to_v13() (*in module pytext.config.config_adapter*), 406
v1_to_v2() (*in module pytext.config.config_adapter*), 406
v22_to_v23() (*in module pytext.config.config_adapter*), 406
v23_to_v22() (*in module pytext.config.config_adapter*), 406
v23_to_v24() (*in module pytext.config.config_adapter*), 406
v24_to_v23() (*in module pytext.config.config_adapter*), 406
v24_to_v25() (*in module pytext.config.config_adapter*), 407
v25_to_v24() (*in module pytext.config.config_adapter*), 407
v25_to_v26() (*in module pytext.config.config_adapter*), 407
v26_to_v25() (*in module pytext.config.config_adapter*), 407
v2_to_v3() (*in module pytext.config.config_adapter*), 407
v3_to_v4() (*in module pytext.config.config_adapter*), 407
v6_to_v7() (*in module pytext.config.config_adapter*), 407
valid_actions() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParserBase method*), 664
validate() (*pytext.torchscript.module.ScriptPyTextEmbeddingModule method*), 751
validate() (*pytext.torchscript.module.ScriptTwoTowerModule method*), 753
validate_batch_element() (*in module pytext.torchscript.batchutils*), 747
validate_config() (*pytext.models.seq_models.conv_model.CNNModel class method*), 673
validate_dense_feat() (*in module pytext.torchscript.batchutils*), 747
validate_make_prediction_batch_element() (*in module pytext.torchscript.batchutils*), 747
validate_node() (*pytext.data.data_structures.annotation.Intent method*), 415
validate_node() (*pytext.data.data_structures.annotation.Node method*), 416
validate_node() (*pytext.data.data_structures.annotation.Root method*), 416
validate_node() (*pytext.data.data_structures.annotation.Slot method*), 416
validate_node() (*pytext.data.data_structures.annotation.Token method*), 416
validate_onnx_export() (*in module pytext.utils.onnx*), 769
validate_tree() (*pytext.data.data_structures.annotation.Tree method*), 416
validated_annotation() (*pytext.metric_reporters.seq2seq_compositional.CompositionalSeq2S method*), 529
value (*pytext.data.tokenizers.Token attribute*), 438
value (*pytext.data.tokenizers.tokenizer.Token attribute*), 438
values() (*pytext.config.component.Registry class method*), 404

ValueSerializationError, 414
var_to_numpy() (*in module pytext.utils.cuda*), 761
Variable() (*in module pytext.utils.cuda*), 761
VectorNormalizer (*class in pytext.torchscript.tensorizer*), 743
VectorNormalizer (*class in pytext.torchscript.tensorizer.normalizer*), 740
verify_encoder_out() (*in module pytext.models.seq_models.utils*), 685
verify_encoder_out() (*pytext.models.seq_models.projection_layers.DecoupledDecoderHead method*), 681
visualize() (*pytext.models.embeddings.embedding_base.EmbeddingBase method*), 576
visualize() (*pytext.models.embeddings.embedding_list.EmbeddingList method*), 577
visualize() (*pytext.models.embeddings.EmbeddingBase method*), 582
visualize() (*pytext.models.embeddings.EmbeddingList method*), 583
visualize() (*pytext.models.embeddings.mlp_embedding.MLPEmbedding method*), 578
visualize() (*pytext.models.embeddings.MLPEmbedding method*), 589
visualize() (*pytext.models.embeddings.scriptable_embedding_list.ScriptableEmbeddingList method*), 579
visualize() (*pytext.models.embeddings.word_embedding.WordEmbedding method*), 580
visualize() (*pytext.models.embeddings.word_seq_embedding.WordSeqEmbedding method*), 582
visualize() (*pytext.models.embeddings.WordEmbedding method*), 584
visualize() (*pytext.models.embeddings.WordSeqEmbedding method*), 588
vocab_files (*pytext.data.tensorizers.VocabConfig attribute*), 477
vocab_map (*pytext.exporters.exporter.ModelExporter attribute*), 494
vocab_map (*pytext.exporters.ModelExporter attribute*), 496
vocab_to_export() (*pytext.models.doc_model.DocModel method*), 690
vocab_to_export() (*pytext.models.doc_model.PersonalizedDocModel method*), 691
vocab_to_export() (*pytext.models.ensembles.ensemble.EnsembleModel method*), 592
vocab_to_export() (*pytext.models.ensembles.EnsembleModel method*), 594
vocab_to_export() (*pytext.models.joint_model.IntentSlotModel method*), 692
vocab_to_export() (*pytext.models.language_models.lmlstm.LMLSTM method*), 596
vocab_to_export() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 662
vocab_to_export() (*pytext.models.seq_models.contextual_intent_slot.ContextualIntentSl method*), 668
vocab_to_export() (*pytext.models.word_model.WordTaggingLiteModel method*), 703
vocab_to_export() (*pytext.models.word_model.WordTaggingModel method*), 703
VocabBuilder (*class in pytext.data.utils*), 480
VocabConfig (*class in pytext.data.tensorizers*), 477
VocabFileConfig (*class in pytext.data.tensorizers*), 477
VocabLookup (*class in pytext.torchscript.tensorizer*), 744
VocabLookup (*class in pytext.torchscript.tensorizer.tensorizer*), 742
Vocabulary (*class in pytext.data.utils*), 480
VocabUsingField (*class in pytext.fields*), 506
VocabUsingField (*class in pytext.fields.field*), 502
VocabUsingNestedField (*class in pytext.fields*), 506
VocabUsingNestedField (*class in pytext.fields.field*), 502

## W

WarmupScheduler (*class in pytext.optimizer.scheduler*), 722
weight_norm (*pytext.config.module_config.CNNParams attribute*), 410
weighted_hinge_loss() (*in module pytext.utils.loss*), 767
word_attention (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDo attribute*), 644
word_decoder (*pytext.models.decoders.intent_slot_model_decoder.Inten attribute*), 566
word_decoder (*pytext.models.decoders.IntentSlotModelDecoder attribute*), 571
word_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 408
WORD_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 408

word_feat (*pytext.config.doc_classification.ModelInputConfig attribute*), 408
word_feat (*pytext.config.field_config.FeatureConfig attribute*), 409
WORD_LABEL (*pytext.common.constants.DFColumn attribute*), 401
WORD_LABEL_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 402
WORD_LABEL_PAD (*pytext.common.constants.Padding attribute*), 402
WORD_LABEL_PAD_IDX (*pytext.common.constants.Padding attribute*), 402
word_output (*pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer attribute*), 603
WORD_WEIGHT (*pytext.common.constants.DFColumn attribute*), 401
WORD_WEIGHT (*pytext.config.contextual_intent_slot.ExtraField attribute*), 407
WORD_WEIGHT_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 402
WordEmbedding (*class in pytext.models.embeddings*), 583
WordEmbedding (*class in pytext.models.embeddings.word_embedding*), 579
WordFeatConfig (*in module pytext.config.field_config*), 409
WordLabelConfig (*class in pytext.config.field_config*), 409
WordLabelField (*class in pytext.fields*), 506
WordLabelField (*class in pytext.fields.field*), 503
WordPieceTokenizer (*class in pytext.data.tokenizers*), 439
WordPieceTokenizer (*class in pytext.data.tokenizers.tokenizer*), 438
WordpieceTokenizerTest (*class in pytext.data.test.tokenizers_test*), 435
WordSeqEmbedding (*class in pytext.models.embeddings*), 587
WordSeqEmbedding (*class in pytext.models.embeddings.word_seq_embedding*), 581
WordTaggingLiteModel (*class in pytext.models.word_model*), 703
WordTaggingMetricReporter (*class in pytext.metric_reporters*), 541
WordTaggingMetricReporter (*class in pytext.metric_reporters.word_tagging_metric_reporter*), 533
WordTaggingModel (*class in pytext.models.word_model*), 703
WordTaggingOutputLayer (*class in pytext.models.output_layers*), 617
WordTaggingOutputLayer (*class in pytext.models.output_layers.word_tagging_output_layer*), 612
WordTaggingScores (*class in pytext.models.output_layers.word_tagging_output_layer*), 613
WordTaggingTask (*class in pytext.task.tasks*), 735


## X


XLMTensorizer (*class in pytext.data.xlm_tensorizer*), 481
XLMTensorizerScriptImpl (*class in pytext.data.xlm_tensorizer*), 482


## Z


ZERO (*pytext.config.field_config.EmbedInitStrategy attribute*), 409
zero_grad() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_FP16Optimiz method*), 713
zero_grad() (*pytext.optimizer.fairseq_fp16_utils.Fairseq_MemoryEfficie method*), 713
zero_grad() (*pytext.optimizer.fp16_optimizer.FP16Optimizer method*), 714
zero_grad() (*pytext.optimizer.fp16_optimizer.FP16OptimizerApex method*), 714
zero_grad() (*pytext.optimizer.fp16_optimizer.FP16OptimizerDeprecate method*), 715
zero_grads() (*pytext.trainers.Trainer method*), 759
zero_grads() (*pytext.trainers.trainer.Trainer method*), 757
zerovar() (*in module pytext.utils.cuda*), 761
zip_batch_any_list_list() (*in module pytext.torchscript.batchutils*), 748
zip_batch_tensor_list() (*in module pytext.torchscript.batchutils*), 748
zip_dicts() (*in module pytext.data.data*), 446