PyText Documentation

PyText Contributors

May 07, 2019

Getting Started

1 How To Use 3

2 Indices and tables 413

Python Module Index 415

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 < demos/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
{
  "task": {
    "DocClassificationTask": {
      "data_handler": {
        "train_path": "tests/data/train_data_tiny.tsv",
        "eval_path": "tests/data/test_data_tiny.tsv",
        "test_path": "tests/data/test_data_tiny.tsv"
      }
    }
  }
}
```

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 (TSV) format. 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
loss: 2.072155
Accuracy: 20.00

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

        reminder/set_reminder   20.00           100.00          33.33           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    5.00            25.00           8.33

Soft Metrics:
        Label           Average precision
        alarm/set_alarm 40.00
        alarm/time_left_on_alarm        100.00
        reminder/set_reminder   25.00
        alarm/show_alarms       25.00
        weather/find    nan
        alarm/modify_alarm      nan
        alarm/snooze_alarm      nan
        reminder/show_reminders nan
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
    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
```

(continues on next page)

(continued from previous page)

```
... [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:
  --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
```

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({"raw_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:'
    )

    # 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?"
}
```

(continues on next page)

(continued from previous page)

```
$ 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, thanks to the TensorBoardX library.

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:

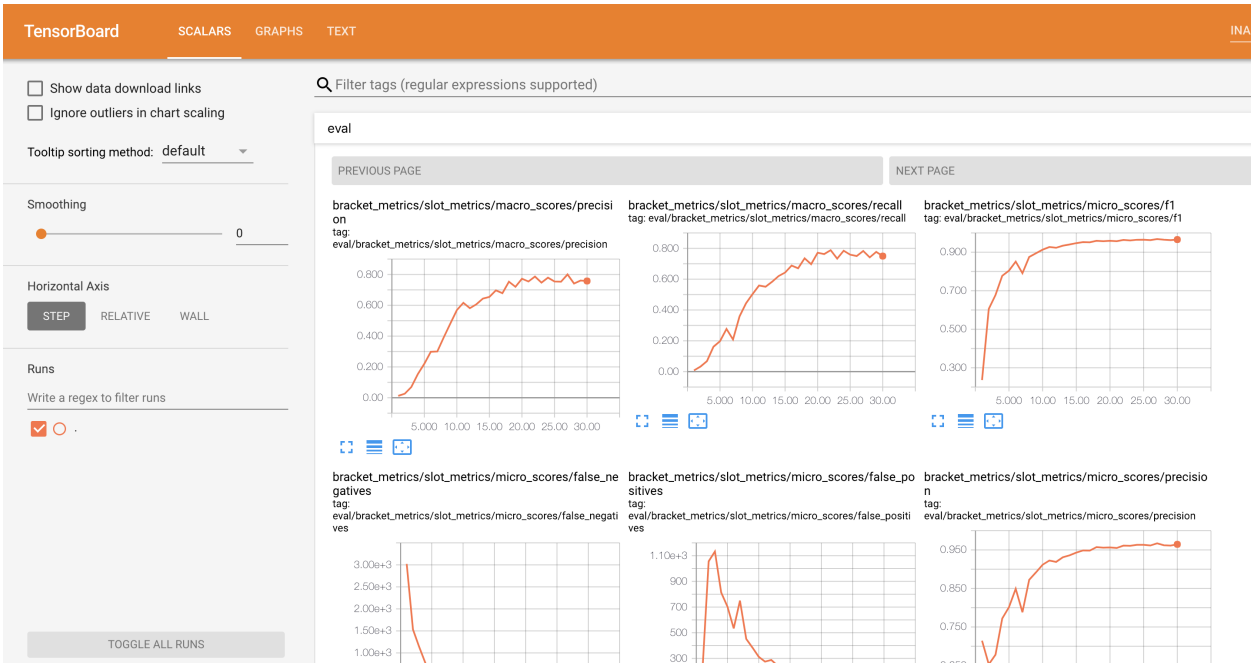
```
$ pip install tensorflow
```

1.4.4 4. View your visualizations

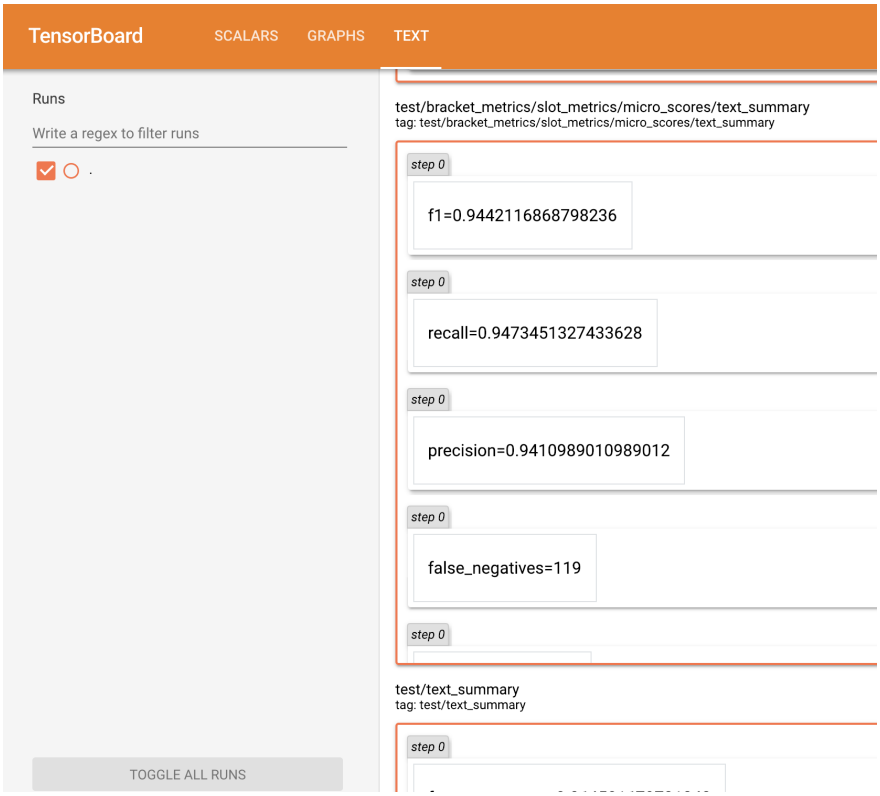
After launching the visualization server, you can view your visualizations in a web browser at *http://localhost:6006*.

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:

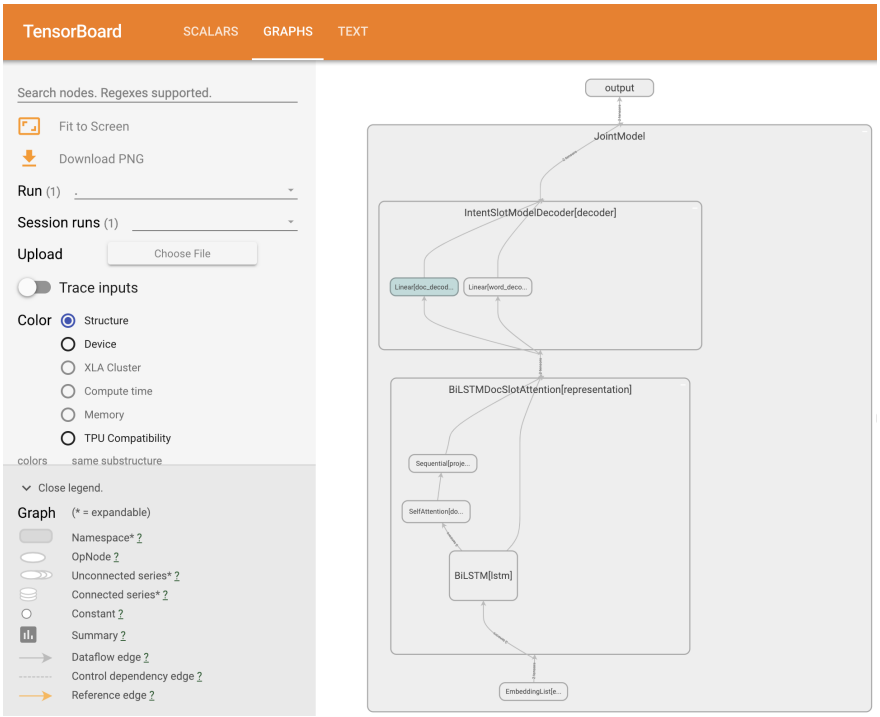
Training Metrics:



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.

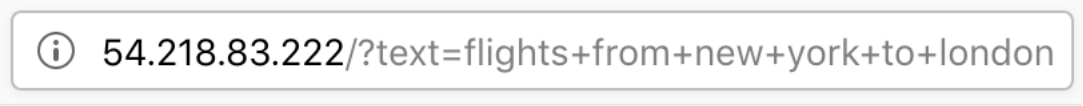
```
@app.route('/')
def predict():
    return json.dumps(atis.predict(request.args.get('text', '')))

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.



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

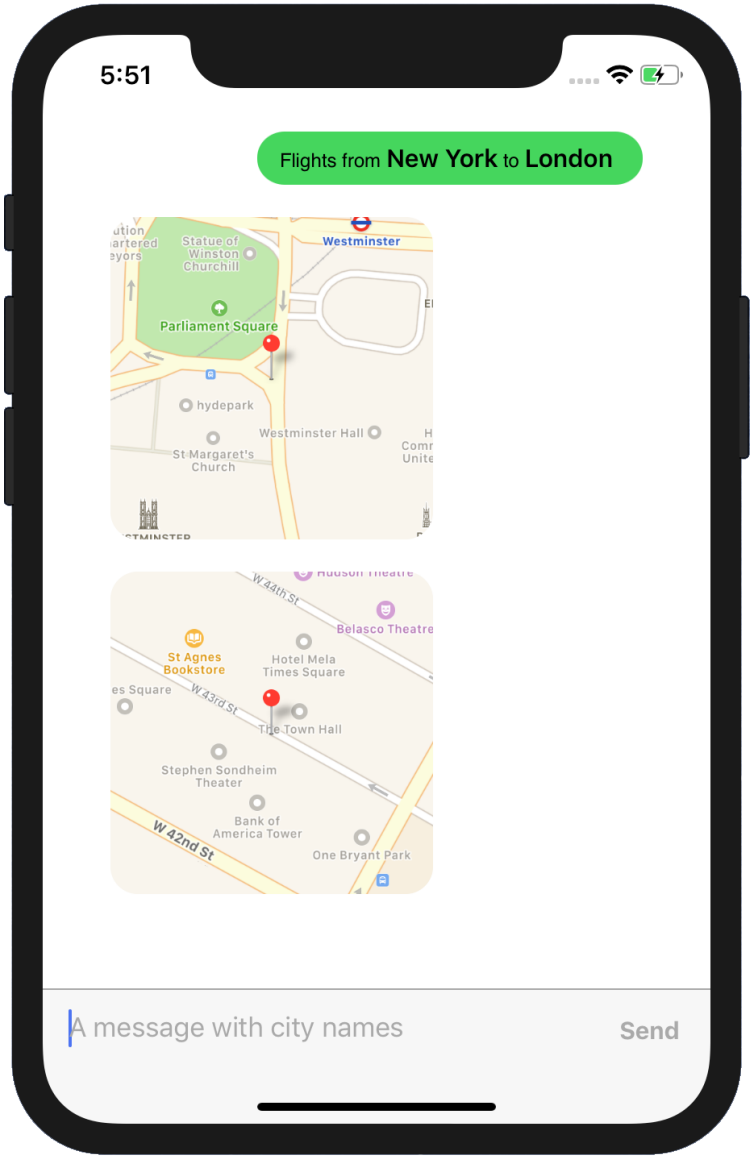
1.5.3 3. Implement the iOS app

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. Caffe2 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 Caffe2 intent-slot model trained on the ATIS 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. In an empty, folder, create a *Dockerfile* with the following contents:

Dockerfile

```
FROM ubuntu:16.04

# Install Caffe2 + dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
  build-essential \
  git \
  libgoogle-glog-dev \
  libgtest-dev \
  libiomp-dev \
  libleveldb-dev \
  liblmdb-dev \
  libopencv-dev \
  libopenmpi-dev \
  libsnappy-dev \
  openmpi-bin \
  openmpi-doc \
  python-dev \
  python-pip
RUN pip install --upgrade pip
RUN pip install setuptools wheel
RUN pip install future numpy protobuf typing hypothesis pyyaml
RUN apt-get install -y --no-install-recommends \
      libgflags-dev \
      cmake
RUN git clone https://github.com/pytorch/pytorch.git
WORKDIR pytorch
RUN git submodule update --init --recursive
RUN python setup.py install

# 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 \
  curl \
  flex \
  bison \
  pkg-config \
  libssl-dev
RUN curl https://www-us.apache.org/dist/thrift/0.11.0/thrift-0.11.0.tar.gz --output␣
↪thrift-0.11.0.tar.gz
RUN tar -xvf thrift-0.11.0.tar.gz
WORKDIR thrift-0.11.0
RUN ./bootstrap.sh
RUN ./configure
```

(continues on next page)

(continued from previous page)

```
RUN make
RUN make install
```

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.

Our C++ server will expose a very simple API that receives an sentence/utterance as a string, and return a map of label names(*string*) -> scores(*list<double>*). For document scores, the list will only contain one score, and for word scores, the list will contain one score per word. The corresponding thrift spec fo the API is below:

predictor.thrift

```
namespace cpp predictor_service

service Predictor {
   // Returns list of scores for each label
   map<string,list<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:
    NetDef mPredictNet;
    Workspace mWorkspace;

    NetDef loadAndInitModel(Workspace& workspace, string& modelFile) {
      auto db = unique_ptr<DBReader>(new DBReader("minidb", modelFile));
      auto metaNetDef = runGlobalInitialization(move(db), &workspace);
      const auto predictInitNet = getNet(
        *metaNetDef.get(),
        PredictorConsts::default_instance().predict_init_net_type()
      );
      CAFFE_ENFORCE(workspace.RunNetOnce(predictInitNet));

      auto predictNet = NetDef(getNet(
        *metaNetDef.get(),
        PredictorConsts::default_instance().predict_net_type()
      ));
      CAFFE_ENFORCE(workspace.CreateNet(predictNet));

      return predictNet;
    }
...
  public:
```

(continues on next page)

(continued from previous page)

```
    PredictorHandler(string &modelFile): mWorkspace("workspace") {
      mPredictNet = loadAndInitModel(mWorkspace, 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. Feed the input as tensors to the model
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, vector<double>>& _return, const string& doc) {
      // Pre-process: tokenize input doc
      vector<string> tokens;
      string docCopy = doc;
      tokenize(tokens, docCopy);

      // Feed input to model as tensors
      Tensor valTensor = TensorCPUFromValues<string>(
        {static_cast<int64_t>(1), static_cast<int64_t>(tokens.size())}, {tokens}
      );
      BlobGetMutableTensor(mWorkspace.CreateBlob("tokens_vals_str:value"), CPU)
        ->CopyFrom(valTensor);
      Tensor lensTensor = TensorCPUFromValues<int>(
        {static_cast<int64_t>(1)}, {static_cast<int>(tokens.size())}
      );
      BlobGetMutableTensor(mWorkspace.CreateBlob("tokens_lens"), CPU)
        ->CopyFrom(lensTensor);

      // Run the model
      CAFFE_ENFORCE(mWorkspace.RunNet(mPredictNet.name()));

      // Extract and populate results into the response
      for (int i = 0; i < mPredictNet.external_output().size(); i++) {
        string label = mPredictNet.external_output()[i];
        _return[label] = vector<double>();
        Tensor scoresTensor = mWorkspace.GetBlob(label)->Get<Tensor>();
        for (int j = 0; j < scoresTensor.numel(); j++) {
          float score = scoresTensor.data<float>()[j];
          _return[label].push_back(score);
        }
      }
    }
...
}
```

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, however it is not covered in the scope of this tutorial since the Thrift protocol is what we'll use in production.

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: *libthrift.so*, *libcaffe2.so*, *libprotobuf.so* and *libc10.so*. The *Makefile* below does this:

Makefile

```
CPPFLAGS += -g -std=c++11 -std=c++14 \
  -I./gen-cpp \
  -I/pytorch -I/pytorch/build \
      -I/pytorch/aten/src/ \
      -I/pytorch/third_party/protobuf/src/
CLIENT_LDFLAGS += -lthrift
SERVER_LDFLAGS += -L/pytorch/build/lib -lthrift -lcaffe2 -lprotobuf -lc10

# ...

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

# Add library search paths
RUN echo '/pytorch/build/lib/' >> /etc/ld.so.conf.d/local.conf
RUN echo '/usr/local/lib/' >> /etc/ld.so.conf.d/local.conf
RUN ldconfig
```

1.6.5 5. Test/Run the server

This section assumes that your local files match the one found here.

Now that you have implemented your server, we will run the following commands to take it for a test run. In your server folder:

1. Build the image:

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

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

2. Run the server. We use *models/atis_joint_model.c2* as the local path to our model file (add your trained model there):

```
$ docker run -it -p 8080:8080 predictor_service:latest ./server models/atis_joint_
↪model.c2
```

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

3. Test our server by sending a test utterance "Flight from Seattle to San Francisco":

```
$ curl -G "http://localhost:8080" --data-urlencode "doc=Flights from Seattle to San␣
↪Francisco"
```

If successful, you should see the scores printed out on the console. On further inspection, the doc score for "flight", the 3rd word score for "B-fromloc.city_name" corresponding to "Seattle", the 5th word score for "B-toloc.city_name" corresponding to "San", and the 6th word score for "I-toloc.city_name" corresponding to "Francisco" should be close to 0.

```
doc_scores:flight:-2.07426e-05
word_scores:B-fromloc.city_name:-14.5363 -12.8977 -0.000172928 -12.9868 -9.94603 -16.
↪0366
word_scores:B-toloc.city_name:-15.2309 -15.9051 -9.89932 -12.077 -0.000134 -8.52712
word_scores:I-toloc.city_name:-13.1989 -16.8094 -15.9375 -12.5332 -10.7318 -0.
↪000501401
```

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 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.

1.7.1 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
```

1.7.2 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.

1.7.3 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.

1.7.4 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):
    """
```

(continues on next page)

(continued from previous page)

```
    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",
        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]
```

(continues on next page)

(continued from previous page)

```
        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,
            }
```

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.

1.7.5 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
```

1.7.6 6. Training the Model

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

```
$ pytext --include my_classifier gen-default-config AtisIntentDataSource␣
↪NewDocumentClassification > 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
```

Now we edit the config to remove the parameters that we don't care about and can use the default values, 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": {
    "NewDocumentClassification": {
      "data": {
        "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": 3
}
```

And, at last, we can train the model

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

1.7.7 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())
```

1.8 Train Intent-Slot model on ATIS Dataset

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.

1.8.1 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.

1.8.2 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.

1.8.3 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": {
      "JointTextTask": {
        "model": {
          "representation": {
            "BiLSTMDocSlotAttention": {
              "pooling": {
                "SelfAttention": {}
              }
            }
          },
          "output_layer": {
            "doc_output": {
              "loss": {
                "CrossEntropyLoss": {}
              }
            },
            "word_output": {
              "CRFOutputLayer": {}
            }
          }
        },
        "features": {
          "word_feat": {
            "embed_dim": 100,
            "pretrained_embeddings_path": "demo/atis_joint_model/glove.6B.100d.txt"
          }
        },
        "optimizer": {
          "type": "adam",
          "lr": "0.001"
        },
        "trainer": {
```

(continues on next page)

(continued from previous page)

```
          "epochs": 20
        },
        "featurizer": {
          "SimpleFeaturizer": {}
        },
        "labels": [
          {
            "DocLabelConfig": {}
          },
          {
            "WordLabelConfig": {}
          }
        ],
        "data_handler": {
          "train_path": "demo/atis_joint_model/atis.processed.train.csv",
          "eval_path": "demo/atis_joint_model/atis.processed.val.csv",
          "test_path": "demo/atis_joint_model/atis.processed.test.csv"
        }
      }
    }
  }
}
```

We explain some of the parameters involved:

- `JointTextTask` 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_feat* attribute inside *features*.
- The *featurizer* (`SimpleFeaturizer`) splits the utterance into tokens on whitespace.

To train the PyText model,

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

1.8.4 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
```

1.8.5 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 <<< '{"raw_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),
 ...
}
```

1.9 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.

1.9.1 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
```

1.9.2 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*.

1.9.3 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.
```

1.9.4 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

1.10 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.

1.10.1 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
```

1.10.2 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
```

1.10.3 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.

1.10.4 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"
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

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.

1.10.5 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
    }
  }
```

1.10.6 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!

1.11 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.

1.11.1 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.

1.11.2 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
```

1.11.3 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*.

1.11.4 4. Train model with the downloaded dataset

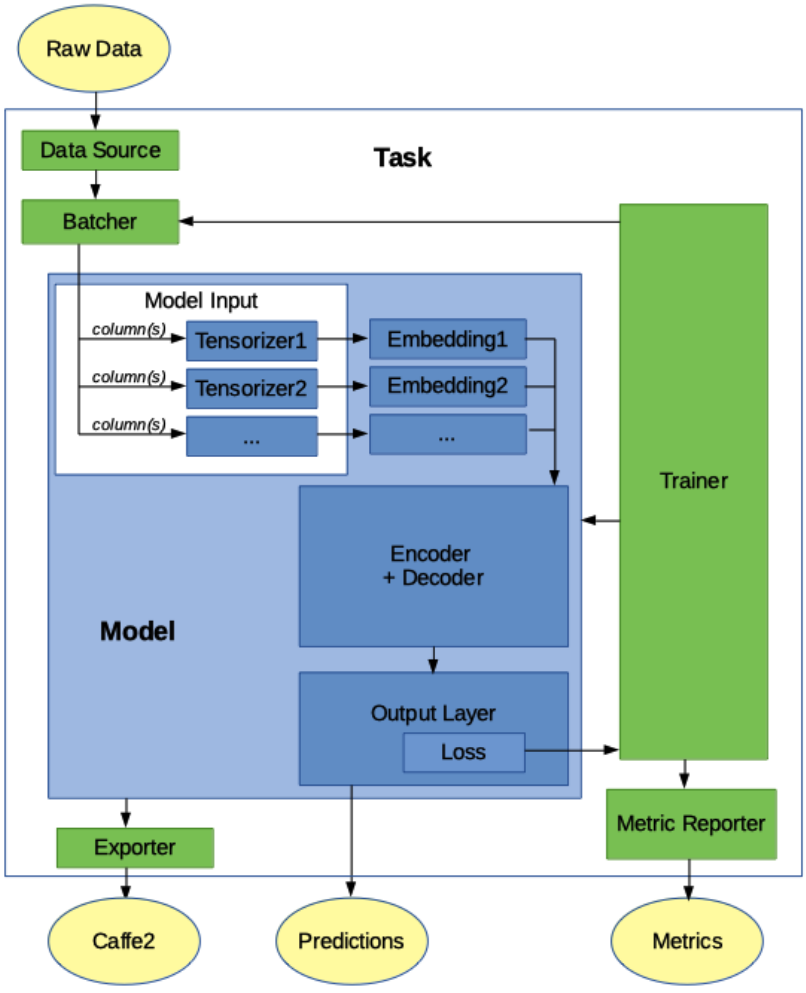
Train the model using the command below

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

1.12 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.

1.12.1 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
    - 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/*)

1.12.2 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)

1.12.3 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.

1.12.4 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"
          }
        }
```

(continues on next page)

(continued from previous page)

```
        }
      }
    }
}
```

1.12.5 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: WordLabelTensorizer.Config = WordLabelTensorizer.Config()
            # for metric reporter
            raw_text: RawString.Config = RawString.Config(column="text")

        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"]
```

(continues on next page)

(continued from previous page)

```
    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)
```

1.13 Creating A New Task

PyText uses a `Task` class as a central place to define components for data processing, model training, metric reporting etc. and wire up those components. One can easily inherit from an existing task and replace some (or all) components.

In this tutorial, we'll write a `WordTaggingTask`, and its associated components.

1.13.1 1. Define the `Task`

Usually features, targets, data_handler, model and metric_reporter are the components subject to change, and we can reuse the other more general ones e.g trainer, optimizer and exporter

```
from word_tagging import ModelInputConfig, TargetConfig

class WordTaggingTask(Task):
  class Config(Task.Config):
    features: ModelInputConfig = ModelInputConfig()
    targets: TargetConfig = TargetConfig()
    data_handler: WordTaggingDataHandler.Config = WordTaggingDataHandler.Config()
    model: WordTaggingModel.Config = WordTaggingModel.Config()
    trainer: Trainer.Config = Trainer.Config()
    optimizer: OptimizerParams = OptimizerParams()
    scheduler: Optional[SchedulerParams] = SchedulerParams()
    metric_reporter: WordTaggingMetricReporter.Config = WordTaggingMetricReporter.
↪Config()
    exporter: Optional[TextModelExporter.Config] = TextModelExporter.Config()
```

Every `Task` has an embedded `Config`, which defines the config of it's components in a nested way. The base `Task` has a *from_config* method that creates every component and wires them up.

1.13.2 2. Define `ModelInput` and `Target`

The first two configs in the `Config` are model inputs (features) and targets (expected outputs), which define the interface between data processing and model training.

```
# word_tagging.py

class ModelInputConfig(ModuleConfig):
  word_feat: WordFeatConfig = WordFeatConfig()
```

(continues on next page)

(continued from previous page)

```
  dict_feat: Optional[DictFeatConfig] = None
  char_feat: Optional[CharFeatConfig] = None

class TargetConfig(ConfigBase):
  # Transform sequence labels to BIO format
  use_bio_labels: bool = False
```

`ModelInputConfig` defines all the possible input to our model, and will be used in `DataHandler` to create TorchText `Field` to process raw data and also in `Model` to create the first model layer: the **Embedding**.

1.13.3 3. Implement `DataHandler`

PyText uses the open source library TorchText for part of data preprocessing, including padding, numericalization and batching. On top of TorchText, PyText incorporates a `Featurizer`, which provides data processing steps that are shared in both training and inference time. Tokenization is a typical step in Featurizer.

The general pipeline of a data handler is:

1. Read data from a file into a list of raw data examples.
2. Convert each row of row data to a TorchText `Example`.
3. Generate a TorchText `Dataset` from the examples and a list of predefined TorchText `Field`
4. Return a `BatchIterator` which will generate a tuple of (input, target, context) tensors for each iteration.

The base `DataHandler` already implements most of these steps, all we need to do is:

1. Define the fields in *from_config* classmethod, a factory method to create a component from a config:

```
@classmethod
def from_config(cls, config: Config, model_input_config, target_config, **kwargs):
    model_input_fields: Dict[str, Field] = create_fields(
      model_input_config,
        {
            ModelInput.WORD_FEAT: TextFeatureField,
            ModelInput.DICT_FEAT: DictFeatureField,
            ModelInput.CHAR_FEAT: CharFeatureField,
        },
    )
    target_fields: Dict[str, Field] = {WordLabelConfig._name: WordLabelField.from_
↪config(target_config)}
    extra_fields: Dict[str, Field] = {ExtraField.TOKEN_RANGE: RawField()}
    kwargs.update(config.items())
    return cls(
        raw_columns=config.columns_to_read,
        targets=target_fields,
        features=model_input_fields,
        extra_fields=extra_fields,
        **kwargs,
    )
```

We create input `Field` by using the *create_fields* method which combines the input config (first argument) with the provided map of name to Class (second argument). Each `Field` is constructed using its *from_config* method with the matching config from the *input_config*. Since this is a word labeling task, we need a `Field` for the expected labels, so we pass a single `WordLabelField` into *target_fields* along with its column name. Finally, we specify an extra field *token_range* which will be used later to merge predicted word labels into the slots. Extra fields are processed but

not used directly by the model. They are passed along as batch context, which, as mentioned above, will be used later in the process.

2. Override the *preprocess_row* method to convert a row of raw data into a TorchText `Example`:

```
def preprocess_row(self, row_data: Dict[str, Any]) -> Dict[str, Any]:
      features = self.featurizer.featurize(
          InputRecord(
              raw_text=row_data.get(RawData.TEXT, ""),
              raw_gazetteer_feats=row_data.get(RawData.DICT_FEAT, ""),
          )
      )
      res = {
          # features
          ModelInput.WORD_FEAT: self._get_tokens(features),
          ModelInput.DICT_FEAT: (
              features.gazetteer_feats,
              features.gazetteer_feat_weights,
              features.gazetteer_feat_lengths,
          ),
          ModelInput.CHAR_FEAT: self._get_chars(features),
          # target
          [Target.WORD_LABEL_FIELD] = data_utils.align_slot_labels(
              features.token_ranges,
              row_data[RawData.WORD_LABEL],
              self.targets[WordLabelConfig._name].use_bio_labels,
          )
          # extra data
          BatchContext.TOKEN_RANGE: features.token_ranges,
      }
      return res
```

Here we invoke the `Featurizer` and map the data to TorchText `Field` names to create a TorchText `Dataset` later. Note the *data_utils.align_slot_labels* method here, which breaks the slot labels that span multiple words into labels for each word (with word labels and token ranges as inputs). We do the processing here because TorchText assumes a 1:1 mapping between raw input and `Field`.

1.13.4 4. Implement `Model`

A typical model in PyText is organized in four layers: **Embedding**, **Representation**, **Decode** and **Output**. For any new model that conforms to this architecture, writing the model is no more than just defining the config of each layer, since the constructor and forward methods are already well defined in base `Model`.:

```
class WordTaggingModel(Model):
  class Config(ConfigBase):
    representation: Union[BiLSTMSlotAttention.Config, BSeqCNNRepresentation.Config] =␣
↪BiLSTMSlotAttention.Config()
    decoder: MLPDecoder.Config = MLPDecoder.Config()
    output_layer: Union[WordTaggingOutputLayer.Config, CRFOutputLayer.Config] =␣
↪WordTaggingOutputLayer.Config()
```

You may notice that there's no config for the embedding layer here, because it directly uses `ModelInputConfig`, already defined in the Task's `Config`. By default, the embedding layer use `EmbeddingList` which creates a list of sub embedding modules according to the `ModelInputConfig`, and concatenates their vectors in the forward method. We don't need to override anything in this example since the default behavior in base `Model` already does this:

```
@classmethod
def compose_embedding(cls, sub_embs, metadata):
  return EmbeddingList(sub_embs.values(), concat=True)
```

the *sub_embs* parameter contains the embeddings we previously defined in the `ModelInputConfig` (word_feat, dict_feat, char_feat).

If you're creating more complicated models, e.g PairNN, you can override this function to reflect the embedding structure:

```
@classmethod
def compose_embedding(cls, sub_embs, metadata):
  return EmbeddingList(
    EmbeddingList(sub_embs["word_feat_1"], sub_embs["dict_feat_1"], concat=True),
    EmbeddingList(sub_embs["word_feat_2"], sub_embs["dict_feat_2"], concat=True),
    concat=False
  )
```

Each layer can be either a single `Module` or a `Union` of multiple. In this example, we give the user the choosing between two different types of representation layers, which can be configured in config JSON file, with the default set to `BiLSTMSlotAttention`.

An example config of changing it to `BSeqCNNRepresentation` looks like:

```
{
  "model": {
    "representation": {
      "BSeqCNNRepresentation": {}
    }
  }
}
```

The Decoder layer is a simple `MLPDecoder`.

The Output layer does three things -

1) Computes loss
2) Gets the prediction
3) Exports to a Caffe2 net

Here we provide two options in this model: `WordTaggingOutputLayer` and `CRFOutputLayer`. The former calculates a cross entropy loss and applies log softmax to get the prediction, while the latter uses CRF (Conditional Random Fields) algorithm to get both. The source code of both classes can be found in the PyText codebase. We'll explain 3) in more detail in a following section.

What if I have a completely different model structure? Then you can completely override both the *from_config* and *forward* methods in your model class. However please inherit your model class from the base `Model` and use the *create_module* method to construct modules. Doing so will give you the features of freezing / saving / loading any part of the model for free. It's as easy as setting the value in the corresponding config:

```
{
  "model": {
    "representation": {
      "BSeqCNNRepresentation": {
        "freeze": true,
        "save_path": "representation_layer.pt"
        "load_path": "pretrained_representation_layer.pt"
```

(continues on next page)

(continued from previous page)

```
      }
    }
  }
}
```

1.13.5 5. Implement `MetricReporter`

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

```
class WordTaggingMetricReporter(MetricReporter):
    def __init__(self, channels, label_names, pad_index):
        super().__init__(channels)
        self.label_names = label_names
        self.pad_index = pad_index

    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)
                        if e != self.pad_index
                    )
                    for scores, pred, expect in zip(
                        self.all_scores, self.all_preds, self.all_targets
                    )
                )
            ),
            self.label_names,
                                                self.calculate_loss(),
        )

    def get_model_select_metric(self, metrics):
        return metrics.accuracy
```

The `MetricReporter` base class already 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.

Note that we also have to override the *get_model_select_metric* method to tell the `Trainer`, how to select best model.

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 WordTaggingTask(Task):
    # ... rest of the code
    def create_metric_reporter(self):
        return WordTaggingMetricReporter(
            channels=[ConsoleChannel(), TensorBoardChannel()],
            label_names=self.metadata.target.vocab.itos, # metadata is processed in␣
↪DataHandler
```

(continues on next page)

(continued from previous page)

```
            pad_index=self.metadata.target.pad_index,
        )
```

1.13.6 6. Implement the predict method

With the code above, we can train and test the model. Next, we need to add one more method in our `Trainer` to format the prediction results. The base `Task` comes with a generic batch predict function that gets predictions and scores from model and restores the order of input examples. By default it only returns the raw numeric predictions, so we will override the *format_prediction* method and make it more human readable:

```
@classmethod
def format_prediction(cls, predictions, scores, context, target_meta):
    label_names = target_meta.vocab.itos
    for prediction, score, token_ranges in zip(
        predictions, scores, context[BatchContext.TOKEN_RANGE]
    ):
        yield [
            {
                "prediction": label_names[word_pred.data],
                "score": {n: s for n, s in zip(label_names, word_score.tolist())},
                "token_range": token_range,
            }
            for word_pred, word_score, token_range in zip(
                prediction, score, token_ranges
            )
        ]
```

Note that we had created the *context[BatchContext.TOKEN_RANGE]* earlier as an extra field.

1.13.7 7. Implement `Exporter`

The predict method is only used when experimenting with the model in PyTorch. If we wish to run our model in the production-optimized Caffe2 environment, we'll have to create an `Exporter`.

An `Exporter` uses ONNX to translate a PyTorch model to a Caffe2 net. After that, we prepend/append any additional Caffe2 operators to the exported net. The default behavior in the base `Exporter` class is to prepend a string-to-vector operator for vocabulary lookup and appending a operator from model's output layer to format prediction results. In this exercise, that is all we need, so we don't have to create a new `Exporter` here.

All that we need to do is implement the *export_to_caffe2* method in the output layer:

```
class WordTaggingOutputLayer(OutputLayerBase):
  def export_to_caffe2(
      self, workspace, init_net, predict_net, model_out, output_name
  ) -> List[core.BlobReference]:
      scores = predict_net.Log(predict_net.Softmax(output_name, axis=2))
      label_scores = predict_net.Split(scores, self.target_names, axis=2)
      return [
          predict_net.Copy(label_score, "{}:{}".format(output_name, name))
          for name, label_score in zip(self.target_names, label_scores)
      ]
```

1.13.8 8. Generate sample config and run the task

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 gen_default_config WordTaggingTask > task_config.json
```

Tweak the config as you like, and then train the model via

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

Run predictions using the trained PyTorch model

```
(pytext) $ pytext predict_py --model-file="YOUR_PY_MODEL_FILE" < test.json
```

Run predictions using the exported Caffe2 model

```
(pytext) $ pytext --config-file="task_config.json" predict --exported-model="YOUR_C2_
↪MODEL_FILE" < test.json
```

Please refer to other tutorials in *PyText Documentation* for end to end working examples of training/predicting. The full code of this example is also available in `pytext.task`

1.14 pytext

1.14.1 config

contextual_intent_slot

ModelInputConfig

Component: *Module*

class pytext.config.contextual_intent_slot.**ModelInputConfig**

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: Optional[*WordEmbedding.Config*] = *WordEmbedding.Config*()

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

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

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

seq_word_feat: Optional[*WordEmbedding.Config*] = *WordEmbedding.Config*()

dense_feat: Optional[*FloatVectorConfig*] = 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,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "dict_feat": null,
    "char_feat": null,
    "contextual_token_embedding": null,
    "seq_word_feat": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "dense_feat": null
}
```


doc_classification

ModelInputConfig

Component: *Module*

class pytext.config.doc_classification.**ModelInputConfig**
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*()

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

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

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

dense_feat: Optional[*FloatVectorConfig***] = 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,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "dict_feat": null,
    "char_feat": null,
    "contextual_token_embedding": null,
    "dense_feat": null
}
```

field_config

DocLabelConfig

class pytext.config.field_config.**DocLabelConfig**
Bases: *ConfigBase*

All Attributes (including base classes)

export_output_names: list[str] = ['doc_scores']

label_weights: dict[str, float] = {}

target_prob: bool = False

Default JSON

```
{
    "export_output_names": [
        "doc_scores"
    ],
    "label_weights": {},
    "target_prob": false
}
```

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": {
```

(continues on next page)

(continued from previous page)


```
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "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
}
```


WordLabelConfig

class pytext.config.field_config.**WordLabelConfig**

Bases: *ConfigBase*

All Attributes (including base classes)

use_bio_labels: bool = `False`

export_output_names: list[str] = `['word_scores']`

Default JSON

```
{
    "use_bio_labels": false,
    "export_output_names": [
        "word_scores"
    ]
}
```

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]`

Default JSON

```
{
    "kernel_num": 100,
    "kernel_sizes": [
        3,
        4
    ]
}
```

pair_classification

ModelInputConfig

Component: *Module*

class pytext.config.pair_classification.**ModelInputConfig**
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`

text1: *WordEmbedding.Config* = *WordEmbedding.Config*()

text2: *WordEmbedding.Config* = *WordEmbedding.Config*()

Default JSON

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "text1": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "text2": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    }
}
```


pytext_config

PyTextConfig

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

Bases: *ConfigBase*

All Attributes (including base classes)

task: Union[*TaskBase.Config*, *Task.Config*, *_NewTask.Config*, *NewTask.Config*, *NewDocumentClassification.Config*, *Pairwis*

use_cuda_if_available: bool = `True`

use_fp16: bool = `False`

distributed_world_size: int = `1`

load_snapshot_path: str = `''`

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

export_caffe2_path: Optional[str] = `'/tmp/model.caffe2.predictor'`

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

export_torchscript_path: Optional[str] = `None`

modules_save_dir: str = `''`

save_module_checkpoints: bool = `False`

save_all_checkpoints: bool = `False`

use_tensorboard: bool = `True`

random_seed: Optional[int] = `None` Seed value to seed torch, python, and numpy random generators.

report_eval_results: bool = `False`

version: int

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.

query_document_pairwise_ranking

ModelInputConfig

Component: *Module*

class pytext.config.query_document_pairwise_ranking.**ModelInputConfig**

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`

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

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

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

Default JSON


```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "pos_response": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "neg_response": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "query": {
        "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,
        "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": []
    }
}
```

1.14.2 data

batch_sampler

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

```
{}
```

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": ""
}
```

bptt_lm_data_handler

BPTTLanguageModelDataHandler.Config

Component: *BPTTLanguageModelDataHandler*

class BPTTLanguageModelDataHandler.**Config**
Bases: *DataHandler.Config*

Configuration class for *BPTTLanguageModelDataHandler*.

columns_to_read
List containing the names of the columns to read from the data files.

Type List[str]

bptt_len
Input sequence length to backpropagate to.

Type int

All Attributes (including base classes)

columns_to_read: list[str] = ['text']

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

bptt_len: int = 35

Default JSON

```
{
    "columns_to_read": [
        "text"
    ],
    "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,
    "bptt_len": 35
}
```

compositional_data_handler

CompositionalDataHandler.Config

Component: *CompositionalDataHandler*

class CompositionalDataHandler.**Config**
Bases: *DataHandler.Config*

All Attributes (including base classes)

columns_to_read: list[str] = ['doc_label', 'word_label', 'text', 'dict_feat', 'seqlogical']

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 = 1

eval_batch_size: int = 1

test_batch_size: int = 1

Default JSON

```
{
    "columns_to_read": [
        "doc_label",
```

(continues on next page)

(continued from previous page)

```
        "word_label",
        "text",
        "dict_feat",
        "seqlogical"
    ],
    "shuffle": true,
    "sort_within_batch": true,
    "train_path": "train.tsv",
    "eval_path": "eval.tsv",
    "test_path": "test.tsv",
    "train_batch_size": 1,
    "eval_batch_size": 1,
    "test_batch_size": 1
}
```

contextual_intent_slot_data_handler

ContextualIntentSlotModelDataHandler.Config

Component: *ContextualIntentSlotModelDataHandler*

class ContextualIntentSlotModelDataHandler.**Config**
Bases: *JointModelDataHandler.Config*

All Attributes (including base classes)

columns_to_read: list[str] = ['doc_label', 'word_label', 'text', 'dict_feat', 'doc_weight', 'wo

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

Default JSON

```
{
    "columns_to_read": [
        "doc_label",
        "word_label",
        "text",
        "dict_feat",
        "doc_weight",
        "word_weight"
    ],
    "shuffle": true,
```

(continues on next page)

(continued from previous page)

```
    "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,
    "max_seq_len": -1
}
```

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*

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

epoch_size: Optional[int] = None define epoch to be a fixed number of batches. If not set, use the entire dataset

Default JSON


```
{
    "source": {
        "TSVDataSource": {
            "column_mapping": {},
            "train_filename": null,
            "test_filename": null,
            "eval_filename": null,
            "field_names": null,
            "delimiter": "\t"
        }
    },
    "batcher": {
        "PoolingBatcher": {
            "train_batch_size": 16,
            "eval_batch_size": 16,
            "test_batch_size": 16,
            "pool_num_batches": 10000
        }
    },
    "sort_key": null,
    "epoch_size": null
}
```


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 = 10000 Number of batches in a pool, to load at one time.

Default JSON


```
{
    "train_batch_size": 16,
    "eval_batch_size": 16,
    "test_batch_size": 16,
    "pool_num_batches": 10000
}
```


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

Subclasses

- *BPTTLanguageModelDataHandler.Config*
- *CompositionalDataHandler.Config*
- *ContextualIntentSlotModelDataHandler.Config*
- *DisjointMultitaskDataHandler.Config*
- *DocClassificationDataHandler.Config*
- *JointModelDataHandler.Config*
- *LanguageModelDataHandler.Config*
- *PairClassificationDataHandler.Config*
- *QueryDocumentPairwiseRankingDataHandler.Config*
- *SeqModelDataHandler.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
}
```

disjoint_multitask_data

DisjointMultitaskData.Config

Component: *DisjointMultitaskData*

class DisjointMultitaskData.**Config**
Bases: *Component.Config*

All Attributes (including base classes)

epoch_size: Optional[int] = None

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

test_key: Optional[str] = None

Default JSON

```
{
    "epoch_size": null,
    "sampler": {
        "EvalBatchSampler": {}
    },
    "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

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,
    "upsample": true
}
```

doc_classification_data_handler

DocClassificationDataHandler.Config

Component: *DocClassificationDataHandler*

class DocClassificationDataHandler.**Config**

Bases: *DataHandler.Config*

Configuration class for *DocClassificationDataHandler*.

columns_to_read
List containing the names of the columns to read from the data files.

Type List[str]

max_seq_len
Maximum sequence length for the input. The input is trimmed after the maximum sequence length.

Type int

All Attributes (including base classes)

columns_to_read: list[str] = ['doc_label', 'text', 'dict_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

Default JSON

```
{
    "columns_to_read": [
        "doc_label",
        "text",
        "dict_feat"
    ],
    "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,
    "max_seq_len": -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
}
```

joint_data_handler

JointModelDataHandler.Config

Component: *JointModelDataHandler*

class JointModelDataHandler.**Config**
Bases: *DataHandler.Config*

All Attributes (including base classes)

columns_to_read: list[str] = ['doc_label', 'word_label', 'text', 'dict_feat', 'doc_weight', 'wo

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

Subclasses

- *ContextualIntentSlotModelDataHandler.Config*
- *SeqModelDataHandler.Config*

Default JSON

```
{
    "columns_to_read": [
        "doc_label",
        "word_label",
        "text",
        "dict_feat",
        "doc_weight",
        "word_weight"
    ],
    "shuffle": true,
    "sort_within_batch": true,
    "train_path": "train.tsv",
    "eval_path": "eval.tsv",
    "test_path": "test.tsv",
```

(continues on next page)

(continued from previous page)

```
    "train_batch_size": 128,
    "eval_batch_size": 128,
    "test_batch_size": 128,
    "max_seq_len": -1
}
```

language_model_data_handler

LanguageModelDataHandler.Config

Component: *LanguageModelDataHandler*

class LanguageModelDataHandler.**Config**

Bases: *DataHandler.Config*

Configuration class for *LanguageModelDataHandler*.

columns_to_read
List containing the names of the columns to read from the data files.

Type List[str]

append_bos
If *True* appends beginning of sentence marker to sentences. Defaults to *True*.

Type bool

append_eos
If *True* appends end of sentence marker to sentences. Defaults to *True*.

Type bool

All Attributes (including base classes)

columns_to_read: list[str] = ['text']

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

append_bos: bool = True

append_eos: bool = True

Default JSON

```
{
    "columns_to_read": [
        "text"
    ],
```

(continues on next page)

(continued from previous page)

```
    "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,
    "append_bos": true,
    "append_eos": true
}
```

pair_classification_data_handler

PairClassificationDataHandler.Config

Component: *PairClassificationDataHandler*

class PairClassificationDataHandler.**Config**

Bases: *DataHandler.Config*

All Attributes (including base classes)

columns_to_read: list[str] = ['doc_label', 'text1', 'text2']

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

Default JSON

```
{
    "columns_to_read": [
        "doc_label",
        "text1",
        "text2"
    ],
    "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
}
```

query_document_pairwise_ranking_data_handler

QueryDocumentPairwiseRankingDataHandler.Config

Component: *QueryDocumentPairwiseRankingDataHandler*

class QueryDocumentPairwiseRankingDataHandler.**Config**
Bases: *DataHandler.Config*

All Attributes (including base classes)

columns_to_read: list[str] = `['query', 'pos_response', 'neg_response']`

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`

Default JSON

```
{
    "columns_to_read": [
        "query",
        "pos_response",
        "neg_response"
    ],
    "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
}
```

seq_data_handler

SeqModelDataHandler.Config

Component: *SeqModelDataHandler*

class SeqModelDataHandler.**Config**
Bases: *JointModelDataHandler.Config*

All Attributes (including base classes)

columns_to_read: list[str] = `['doc_label', 'text']`

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`

pretrained_embeds_file: str = `''`

Default JSON

```
{
    "columns_to_read": [
        "doc_label",
        "text"
    ],
    "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,
    "max_seq_len": -1,
    "pretrained_embeds_file": ""
}
```

sources

data_source

DataSource.Config

Component: *DataSource*

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

All Attributes (including base classes)

Subclasses

- *RowShardedDataSource.Config*
- *ShardedDataSource.Config*
- *SquadDataSource.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

- *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

```
{}
```

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

Default JSON

```
{
    "train_filename": "train-v2.0.json",
    "test_filename": "dev-v2.0.json",
    "eval_filename": "dev-v2.0.json",
    "ignore_impossible": true
}
```

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'

Default JSON

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

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'

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

Default JSON

```
{
    "column_mapping": {},
    "train_filename": null,
    "test_filename": null,
    "eval_filename": null,
    "field_names": null,
    "delimiter": "\t",
    "data_source_languages": {
        "train": "en",
        "eval": "en",
        "test": "en"
    }
}
```

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'

Default JSON

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


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.

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"
}
```


tensorizers

ByteTensorizer.Config

Component: *ByteTensorizer*

class ByteTensorizer.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

column: str = `'text'` The name of the text column to parse from the data source.

lower: bool = `True`

max_seq_len: Optional[int] = `None`

Default JSON

```
{
    "column": "text",
    "lower": true,
    "max_seq_len": null
}
```

CharacterTokenTensorizer.Config

Component: *CharacterTokenTensorizer*

class CharacterTokenTensorizer.**Config**

Bases: *TokenTensorizer.Config*

All Attributes (including base classes)

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`

max_char_length: int = `20` The max character length for a token.

Default JSON

```
{
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true
        }
    },
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false,
    "max_seq_len": null,
    "max_char_length": 20
}
```

FloatListTensorizer.Config

Component: *FloatListTensorizer*

class FloatListTensorizer.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

column: str The name of the label column to parse from the data source.

error_check: bool = False

dim: Optional[int] = None

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

JoinStringTensorizer.Config

Component: *JoinStringTensorizer*

class JoinStringTensorizer.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

columns: list[str] The name of the pass-through column to parse from the data source.

delimiter: str = ' | '

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

LabelTensorizer.Config

Component: *LabelTensorizer*

class LabelTensorizer.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

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

Default JSON

```
{
    "column": "label",
    "allow_unknown": false
}
```

MetricTensorizer.Config

Component: *MetricTensorizer*

class MetricTensorizer.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

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)

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)

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

```
{
    "column": "label",
    "rescale_range": null
}
```

RawJson.Config

Component: *RawJson*

class RawJson.**Config**
Bases: *RawString.Config*

All Attributes (including base classes)

column: str

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

RawString.Config

Component: *RawString*

class RawString.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

column: str The name of the pass-through column to parse from the data source.

Subclasses

- *RawJson.Config*

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

Tensorizer.Config

Component: *Tensorizer*

class Tensorizer.**Config**
Bases: *Component.Config*

All Attributes (including base classes)

Subclasses

- *ByteTensorizer.Config*
- *CharacterTokenTensorizer.Config*
- *FloatListTensorizer.Config*
- *JoinStringTensorizer.Config*
- *LabelTensorizer.Config*
- *MetricTensorizer.Config*
- *NtokensTensorizer.Config*
- *NumericLabelTensorizer.Config*

- *RawJson.Config*
- *RawString.Config*
- *TokenTensorizer.Config*
- *WordLabelTensorizer.Config*

Default JSON

```
{}
```

TokenTensorizer.Config

Component: *TokenTensorizer*

class TokenTensorizer.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

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

Subclasses

- *CharacterTokenTensorizer.Config*

Default JSON

```
{
    "column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true
        }
    },
    "add_bos_token": false,
    "add_eos_token": false,
    "use_eos_token_for_bos": false,
    "max_seq_len": null
}
```

WordLabelTensorizer.Config

Component: *WordLabelTensorizer*

class WordLabelTensorizer.**Config**
Bases: *Tensorizer.Config*

All Attributes (including base classes)

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

Default JSON

```
{
    "slot_column": "slots",
    "text_column": "text",
    "tokenizer": {
        "Tokenizer": {
            "split_regex": "\\s+",
            "lowercase": true
        }
    },
    "allow_unknown": false
}
```

tokenizers

tokenizer

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.

Default JSON

```
{
    "split_regex": "\\s+",
    "lowercase": true
}
```

1.14.3 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
}
```

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*

Default JSON

```
{
    "export_logits": false,
    "export_raw_to_metrics": false
}
```

1.14.4 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
}
```

CrossEntropyLoss.Config

Component: *CrossEntropyLoss*

class CrossEntropyLoss.**Config**
Bases: *Loss.Config*

All Attributes (including base classes)

Default JSON

```
{}
```

KLDivergenceBCELoss.Config

Component: *KLDivergenceBCELoss*

class KLDivergenceBCELoss.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

temperature: float = 1.0

Default JSON

```
{
    "temperature": 1.0
}
```

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

Default JSON

```
{
    "beta": 0.1
}
```

Loss.Config

Component: *Loss*

class Loss.**Config**
Bases: *Component.Config*

All Attributes (including base classes)

Subclasses

- *CrossEntropyLoss.Config*

Default JSON

```
{}
```

MSELoss.Config

Component: *MSELoss*

class MSELoss.**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
}
```

SoftHardBCELoss.Config

Component: *SoftHardBCELoss*

class SoftHardBCELoss.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

temperature: float = 1.0

Default JSON

```
{
    "temperature": 1.0
}
```

1.14.5 metric_reporters

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'

model_select_metric: ComparableClassificationMetric = <ComparableClassificationMetric.ACCURACY: 'acc

target_label: Optional[str] = None

Default JSON

```
{
    "output_path": "/tmp/test_out.txt",
    "model_select_metric": "accuracy",
    "target_label": null
}
```

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'

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

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'

use_subtask_select_metric: bool = False

Default JSON

```
{
    "output_path": "/tmp/test_out.txt",
    "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'

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

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'

Subclasses

- *MaskedLMMetricReporter.Config*

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

MaskedLMMetricReporter.Config

Component: *MaskedLMMetricReporter*

class MaskedLMMetricReporter.**Config**
Bases: *LanguageModelMetricReporter.Config*

All Attributes (including base classes)

output_path: str = '/tmp/test_out.txt'

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

metric_reporter

MetricReporter.Config

Component: *MetricReporter*

class MetricReporter.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

output_path: str = '/tmp/test_out.txt'

Subclasses

- *ClassificationMetricReporter.Config*
- *CompositionalMetricReporter.Config*
- *DisjointMultitaskMetricReporter.Config*
- *IntentSlotMetricReporter.Config*
- *LanguageModelMetricReporter.Config*
- *MaskedLMMetricReporter.Config*
- *PairwiseRankingMetricReporter.Config*
- *RegressionMetricReporter.Config*

- *SimpleWordTaggingMetricReporter.Config*
- *WordTaggingMetricReporter.Config*

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

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'

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

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'

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

word_tagging_metric_reporter

SimpleWordTaggingMetricReporter.Config

Component: *SimpleWordTaggingMetricReporter*

class SimpleWordTaggingMetricReporter.**Config**
Bases: *MetricReporter.Config*

All Attributes (including base classes)

output_path: str = `'/tmp/test_out.txt'`

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

WordTaggingMetricReporter.Config

Component: *WordTaggingMetricReporter*

class WordTaggingMetricReporter.**Config**
Bases: *MetricReporter.Config*

All Attributes (including base classes)

output_path: str = `'/tmp/test_out.txt'`

Default JSON

```
{
    "output_path": "/tmp/test_out.txt"
}
```

1.14.6 models

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*

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

Default JSON


```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "use_doc_probs_in_word": 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]

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

Default JSON

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "hidden_dims": [],
    "out_dim": null
}
```

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": []
}
```

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

DocModel_Deprecated.Config

Component: *DocModel_Deprecated*

class DocModel_Deprecated.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

representation: Union[***PureDocAttention.Config***, ***BiLSTMDocAttention.Config***, ***DocNNRepresentation.Config***] = ***BiLSTMD***

decoder: ***MLPDecoder.Config*** = ***MLPDecoder.Config***()

output_layer: ***ClassificationOutputLayer.Config*** = ***ClassificationOutputLayer.Config***()

Subclasses

- *NewDocModel.Config*
- *NewDocRegressionModel.Config*

Default JSON


```
{
    "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
            },
            "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
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        }
    }
}
```


ModelInput

class pytext.models.doc_model.**ModelInput**
Bases: ModelInput

All Attributes (including base classes)

tokens: *TokenTensorizer.Config* = *TokenTensorizer.Config*()

labels: *LabelTensorizer.Config* = *LabelTensorizer.Config*(allow_unknown=`True`)

raw_text: *RawString.Config* = *RawString.Config*(column=`'text'`)

Default JSON

```
{
    "tokens": {
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null
    },
    "labels": {
        "column": "label",
        "allow_unknown": true
    },
    "raw_text": {
        "column": "text"
    }
}
```

NewDocModel.Config

Component: *NewDocModel*

class NewDocModel.**Config**

Bases: *DocModel_Deprecated.Config*

All Attributes (including base classes)

representation: Union[*PureDocAttention.Config*, *BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*] = *BiLSTMD*

decoder: *MLPDecoder.Config* = *MLPDecoder.Config*()

output_layer: *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

inputs: *ModelInput* = *ModelInput*()

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

Subclasses

- *NewDocRegressionModel.Config*

Default JSON

```
{
    "representation": {
        "BiLSTMDocAttention": {
            "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
            },
            "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
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        }
    },
    "inputs": {
        "tokens": {
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null
        },
        "labels": {
            "column": "label",
            "allow_unknown": true
```

(continues on next page)

(continued from previous page)


```
        },
        "raw_text": {
            "column": "text"
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    }
}
```


NewDocRegressionModel.Config

Component: *NewDocRegressionModel*

class NewDocRegressionModel.**Config**

Bases: *NewDocModel.Config*

All Attributes (including base classes)

representation: Union[*PureDocAttention.Config*, *BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*] = *BiLSTMDo*

decoder: *MLPDecoder.Config* = *MLPDecoder.Config*()

output_layer: *RegressionOutputLayer.Config* = *RegressionOutputLayer.Config*()

inputs: *RegressionModelInput* = *RegressionModelInput*()

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

Default JSON


```
{
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
```


(continues on next page)

(continued from previous page)

```
            "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
            },
            "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
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {},
        "squash_to_unit_range": false
    },
    "inputs": {
        "tokens": {
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null
        },
        "labels": {
            "column": "label",
            "rescale_range": null
        }
    },
    "embedding": {
        "load_path": null,
```

(continues on next page)

(continued from previous page)

```
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    }
}
```

RegressionModelInput

class pytext.models.doc_model.**RegressionModelInput**

Bases: ModelInput

All Attributes (including base classes)

tokens: *TokenTensorizer.Config* = *TokenTensorizer.Config*()

labels: *NumericLabelTensorizer.Config* = *NumericLabelTensorizer.Config*()

Default JSON

```
{
    "tokens": {
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null
    },
    "labels": {
        "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
        ]
    },
    "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']

Default JSON

```
{
    "embed_dim": 0,
    "model_paths": null,
    "export_input_names": [
        "contextual_token_embedding"
    ]
}
```

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

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
}
```


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*

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
}
```

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

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]] = []

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,
    "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": []
}
```


ensembles

bagging_doc_ensemble

BaggingDocEnsemble.Config

Component: *BaggingDocEnsemble*

class BaggingDocEnsemble.**Config**

Bases: *Ensemble.Config*

Configuration class for *BaggingDocEnsemble*. These attributes are used by *Ensemble.from_config()* to construct instance of *BaggingDocEnsemble*.

models

List of document classification model configurations.

Type List[DocModel.Config]

All Attributes (including base classes)

models: list[*DocModel_Deprecated.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

BaggingIntentSlotEnsemble.Config

Component: *BaggingIntentSlotEnsemble*

class BaggingIntentSlotEnsemble.**Config**
Bases: *Ensemble.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[JointModel.Config]

output_layer
Output layer of intent-slot model responsible for computing loss and predictions.

Type IntentSlotOutputLayer

All Attributes (including base classes)

models: list[*JointModel.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

Ensemble.Config

Component: *Ensemble*

class Ensemble.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

models: list[Any]

sample_rate: float = 1.0

Subclasses

- *BaggingDocEnsemble.Config*
- *BaggingIntentSlotEnsemble.Config*

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

joint_model

JointModel.Config

Component: *JointModel*

class JointModel.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

representation: Union[*BiLSTMDocSlotAttention.Config*, *JointCNNRepresentation.Config*] = *BiLSTMDocSlotAttention.Conf*

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


```
{
    "representation": {
        "BiLSTMDocSlotAttention": {
            "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
            },
            "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,
```


(continues on next page)

(continued from previous page)

```
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            }
        },
        "word_output": {
            "WordTaggingOutputLayer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {}
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "use_doc_probs_in_word": false
    },
    "default_doc_loss_weight": 0.2,
    "default_word_loss_weight": 0.5
}
```

language_models

lmlstm

LMLSTM.Config

Component: *LMLSTM*

class LMLSTM.**Config**

Bases: *ConfigBase*

Configuration class for *LMLSTM*.

representation
Config for the BiLSTM representation.

Type BiLSTM.Config

decoder
Config for the MLP Decoder.

Type MLPDecoder.Config

output_layer
Config for the language model output layer.

Type LMOutputLayer.Config

tied_weights
If *True* use a common weights matrix between the word embeddings and the decoder. Defaults to *False*.

Type bool

stateful
If *True*, do not reset hidden state of LSTM across batches.

Type bool

All Attributes (including base classes)

representation: *BiLSTM.Config* = *BiLSTM.Config*(**bidirectional=False**)

decoder: *MLPDecoder.Config* = *MLPDecoder.Config*()

output_layer: *LMOutputLayer.Config* = *LMOutputLayer.Config*()

tied_weights: bool = False

stateful: bool = False

Default JSON


```
{
    "representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": false
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {}
    },
    "tied_weights": false,
    "stateful": false
}
```


model

BaseModel.Config

Component: *BaseModel*

class BaseModel.**Config**
Bases: *Component.Config*

All Attributes (including base classes)

inputs: *ModelInput* = *ModelInput*()

Subclasses

- *DisjointMultitaskModel.Config*
- *NewDisjointMultitaskModel.Config*
- *Model.Config*
- *BasePairwiseClassificationModel.Config*
- *PairwiseClassificationModel.Config*
- *NewWordTaggingModel.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*
- *NewWordTaggingModel.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

- *ModelInputConfig*
- *ModelInputConfig*
- *FeatureConfig*
- *ModelInputConfig*
- *ModelInputConfig*
- *DecoderBase.Config*
- *IntentSlotModelDecoder.Config*
- *MLPDecoder.Config*
- *MLPDecoderQueryResponse.Config*
- *CharacterEmbedding.Config*
- *DictEmbedding.Config*
- *EmbeddingBase.Config*
- *EmbeddingList.Config*
- *WordEmbedding.Config*
- *BinaryClassificationOutputLayer.Config*
- *ClassificationOutputLayer.Config*
- *MulticlassOutputLayer.Config*
- *RegressionOutputLayer.Config*
- *IntentSlotOutputLayer.Config*
- *LMOutputLayer.Config*
- *OutputLayerBase.Config*
- *PairwiseRankingOutputLayer.Config*
- *CRFOutputLayer.Config*
- *WordTaggingOutputLayer.Config*

- *BiLSTM.Config*
- *BiLSTMDocAttention.Config*
- *BiLSTMDocSlotAttention.Config*
- *BiLSTMSlotAttention.Config*
- *BSeqCNNRepresentation.Config*
- *ContextualIntentSlotRepresentation.Config*
- *DocNNRepresentation.Config*
- *JointCNNRepresentation.Config*
- *PairRepresentation.Config*
- *PassThroughRepresentation.Config*
- *LastTimestepPool.Config*
- *MaxPool.Config*
- *MeanPool.Config*
- *NoPool.Config*
- *PureDocAttention.Config*
- *QueryDocumentPairwiseRankingRep.Config*
- *RepresentationBase.Config*
- *SeqRepresentation.Config*

Default JSON

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

output_layers

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*, *AUCPRHingeLoss.Config*, *KLDivergenceBCELoss.C*

Default JSON

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    }
}
```

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*, *AUCPRHingeLoss.Config*, *KLDivergenceBCELoss.C*

Subclasses

- *BinaryClassificationOutputLayer.Config*
- *MulticlassOutputLayer.Config*

Default JSON

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    }
}
```

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*, *AUCPRHingeLoss.Config*, *KLDivergenceBCELoss.C*

Default JSON

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {
        "CrossEntropyLoss": {}
    }
}
```

doc_regression_output_layer

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": {}
        }
    },
    "word_output": {
        "WordTaggingOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {}
        }
    }
}
```


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

- *BinaryClassificationOutputLayer.Config*
- *ClassificationOutputLayer.Config*
- *MulticlassOutputLayer.Config*
- *RegressionOutputLayer.Config*
- *IntentSlotOutputLayer.Config*
- *LMOutputLayer.Config*
- *PairwiseRankingOutputLayer.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
    }
}
```


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,
    "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: ***CrossEntropyLoss.Config*** **=** ***CrossEntropyLoss.Config*****()**

Default JSON


```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "loss": {}
}
```


pair_classification_model

BasePairwiseClassificationModel.Config

Component: *BasePairwiseClassificationModel*

class BasePairwiseClassificationModel.**Config**
Bases: *BaseModel.Config*

All Attributes (including base classes)

inputs: ***ModelInput*** **=** ***ModelInput*****()**

Subclasses

- *PairwiseClassificationModel.Config*

Default JSON


```
{
    "inputs": {}
}
```

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*()

raw_text: *JoinStringTensorizer.Config* = *JoinStringTensorizer.Config*(columns=['text1', 'text2'])

Default JSON


```
{
    "tokens1": {
        "column": "text1",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null
    },
    "tokens2": {
        "column": "text2",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null
    },
    "labels": {
        "column": "label",
        "allow_unknown": false
    },
    "raw_text": {
        "columns": [
            "text1",
            "text2"
        ],
        "delimiter": " | "
    }
}
```

PairClassificationModel.Config

Component: *PairClassificationModel*

class PairClassificationModel.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

representation: *PairRepresentation.Config* = *PairRepresentation.Config*()

decoder: *MLPDecoder.Config* = *MLPDecoder.Config*()

output_layer: *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

Default JSON


```
{
    "representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "subrepresentation": {
            "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
                },
                "pooling": {
                    "SelfAttention": {
                        "attn_dimension": 64,
                        "dropout": 0.4
                    }
                },
                "mlp_decoder": null
            }
        },
        "subrepresentation_right": null,
        "encode_relations": true
    },
    "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)

```
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        }
    }
}
```

PairwiseClassificationModel.Config

Component: *PairwiseClassificationModel*

class PairwiseClassificationModel.**Config**

Bases: *BasePairwiseClassificationModel.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*()

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

representation: Union[*BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*] = *BiLSTMDocAttention.Config*()

shared_representations: bool = **True**

decoder: *MLPDecoder.Config* = *MLPDecoder.Config*()

output_layer: *ClassificationOutputLayer.Config* = *ClassificationOutputLayer.Config*()

encode_relations: bool = **True**

Default JSON

```
{
    "inputs": {
        "tokens1": {
            "column": "text1",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true
                }
            },
            "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
        },
        "tokens2": {
            "column": "text2",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null
        },
        "labels": {
            "column": "label",
            "allow_unknown": false
        },
        "raw_text": {
            "columns": [
                "text1",
                "text2"
            ],
            "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,
        "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": []
    },
    "representation": {
        "BiLSTMDocAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm": {
```

(continues on next page)

(continued from previous page)

```
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    },
    "shared_representations": true,
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null
    },
    "output_layer": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "loss": {
            "CrossEntropyLoss": {}
        }
    },
    "encode_relations": true
}
```

query_document_pairwise_ranking_model

QueryDocumentPairwiseRankingModel.Config

Component: *QueryDocumentPairwiseRankingModel*

class QueryDocumentPairwiseRankingModel.**Config**

Bases: *ConfigBase*

All Attributes (including base classes)

representation: *QueryDocumentPairwiseRankingRep.Config* = *QueryDocumentPairwiseRankingRep.Config*()

decoder: *MLPDecoderQueryResponse.Config* = *MLPDecoderQueryResponse.Config*()

output_layer: *PairwiseRankingOutputLayer.Config* = *PairwiseRankingOutputLayer.Config*()

decoder_output_dim: int = 64

Default JSON


```
{
    "representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "subrepresentation": {
            "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
                },
                "pooling": {
                    "SelfAttention": {
                        "attn_dimension": 64,
                        "dropout": 0.4
                    }
                },
                "mlp_decoder": null
            }
        },
        "shared_representations": true
    },
    "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
        }
    },
    "decoder_output_dim": 64
}
```

representations

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

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

Default JSON

```
{
    "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
}
```

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,
```

(continues on next page)

(continued from previous page)

```
    "lstm": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true
    },
    "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: *BiLSTM.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": {
```

(continues on next page)

(continued from previous page)


```
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true
    },
    "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,
    "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
    },
    "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": [
            3,
            4
        ]
    },
    "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
            ]
        }
    },
    "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
            ]
        }
    },
    "joint_representation": {
        "BiLSTMDocSlotAttention": {
```


(continues on next page)

(continued from previous page)

```
            "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
            },
            "pooling": null,
            "slot_attention": null,
            "doc_mlp_layers": 0,
            "word_mlp_layers": 0
        }
    }
}
```

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*()

Default JSON

```
{
    "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
        ]
    }
}
```

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: *BSeqCNNRepresentation.Config* **=** *BSeqCNNRepresentation.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": [
                3,
                4
            ]
        }
    },
    "word_representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "cnn": {
            "kernel_num": 100,
```

(continues on next page)

(continued from previous page)

```
            "kernel_sizes": [
                3,
                4
            ]
        },
        "fwd_bwd_context_len": 5,
        "surrounding_context_len": 2
    }
}
```

pair_rep

PairRepresentation.Config

Component: *PairRepresentation*

class PairRepresentation.**Config**

Bases: *RepresentationBase.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

subrepresentation

the sub-representation used for the inputs. If *subrepresentation_right* is not given, then this representation is used for both inputs with tied weights.

Type SubRepresentation

subrepresentation_right

the sub-representation used for the right input. Optional. If missing, *subrepresentation* is used with tied weights. Default: *None*.

Type Optional[SubRepresentation]

All Attributes (including base classes)

load_path: Optional[str] = None

save_path: Optional[str] = None

freeze: bool = False

shared_module_key: Optional[str] = None

subrepresentation: Union[*BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*] = *BiLSTMDocAttention.Config*()

subrepresentation_right: Union[*BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*, NoneType] = None

encode_relations: bool = True

Default JSON

```
{
    "load_path": null,
    "save_path": null,
    "freeze": false,
    "shared_module_key": null,
    "subrepresentation": {
        "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
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    },
    "subrepresentation_right": null,
    "encode_relations": true
}
```


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,
    "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,
    "pooling": {
        "SelfAttention": {
            "attn_dimension": 64,
            "dropout": 0.4
        }
    },
    "mlp_decoder": null
}
```


query_document_pairwise_ranking_rep

QueryDocumentPairwiseRankingRep.Config

Component: *QueryDocumentPairwiseRankingRep*

class QueryDocumentPairwiseRankingRep.**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

subrepresentation: Union[*BiLSTMDocAttention.Config*, *DocNNRepresentation.Config*] = *BiLSTMDocAttention.Config*()

shared_representations: bool = True

Default JSON


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


(continues on next page)

(continued from previous page)

```
                "dropout": 0.4,
                "lstm_dim": 32,
                "num_layers": 1,
                "bidirectional": true
            },
            "pooling": {
                "SelfAttention": {
                    "attn_dimension": 64,
                    "dropout": 0.4
                }
            },
            "mlp_decoder": null
        }
    },
    "shared_representations": true
}
```

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

- *BiLSTM.Config*
- *BiLSTMDocAttention.Config*
- *BiLSTMDocSlotAttention.Config*
- *BiLSTMSlotAttention.Config*
- *BSeqCNNRepresentation.Config*
- *ContextualIntentSlotRepresentation.Config*
- *DocNNRepresentation.Config*
- *JointCNNRepresentation.Config*
- *PairRepresentation.Config*
- *PassThroughRepresentation.Config*
- *PureDocAttention.Config*
- *QueryDocumentPairwiseRankingRep.Config*

- *SeqRepresentation.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": [
                3,
                4
            ]
        }
    },
    "seq_representation": {
        "BiLSTMDocAttention": {
            "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
            },
            "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"
}
```

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
}
```

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: *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'>`

Default JSON

```
{
    "version": 2,
    "lstm": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
```

(continues on next page)

(continued from previous page)


```
        "shared_module_key": null,
        "dropout": 0.4,
        "lstm_dim": 32,
        "num_layers": 1,
        "bidirectional": true
    },
    "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

contextual_intent_slot

ContextualIntentSlotModel.Config

Component: *ContextualIntentSlotModel*

class ContextualIntentSlotModel.**Config**

Bases: *JointModel.Config*

All Attributes (including base classes)

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

Default JSON


```
{
    "representation": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "sen_representation": {
```


(continues on next page)

(continued from previous page)

```
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "cnn": {
                "kernel_num": 100,
                "kernel_sizes": [
                    3,
                    4
                ]
            }
        },
        "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
                ]
            }
        },
        "joint_representation": {
            "BiLSTMDocSlotAttention": {
                "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
                },
                "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": {
```

(continues on next page)

(continued from previous page)


```
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            }
        },
        "word_output": {
            "WordTaggingOutputLayer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {}
            }
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "use_doc_probs_in_word": false
    },
    "default_doc_loss_weight": 0.2,
    "default_word_loss_weight": 0.5
}
```


seqnn

SeqNNModel.Config

Component: *SeqNNModel*

class SeqNNModel.**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
                ]
            }
        },
        "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
                },
                "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": {}
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null
    }
}
```

word_model

ModelInput

class pytext.models.word_model.**ModelInput**
Bases: ModelInput

All Attributes (including base classes)

tokens: *TokenTensorizer.Config* = *TokenTensorizer.Config*()

labels: *WordLabelTensorizer.Config* = *WordLabelTensorizer.Config*()

raw_text: *RawString.Config* = *RawString.Config*(column='text')

Default JSON

```
{
    "tokens": {
        "column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true
            }
        },
        "add_bos_token": false,
        "add_eos_token": false,
        "use_eos_token_for_bos": false,
        "max_seq_len": null
    },
    "labels": {
        "slot_column": "slots",
        "text_column": "text",
        "tokenizer": {
            "Tokenizer": {
                "split_regex": "\\s+",
                "lowercase": true
            }
        },
        "allow_unknown": false
    },
    "raw_text": {
        "column": "text"
    }
}
```

NewWordTaggingModel.Config

Component: *NewWordTaggingModel*

class NewWordTaggingModel.**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*()

Default JSON


```
{
    "inputs": {
        "tokens": {
            "column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true
                }
            },
            "add_bos_token": false,
            "add_eos_token": false,
            "use_eos_token_for_bos": false,
            "max_seq_len": null
        },
        "labels": {
            "slot_column": "slots",
            "text_column": "text",
            "tokenizer": {
                "Tokenizer": {
                    "split_regex": "\\s+",
                    "lowercase": true
                }
            },
            "allow_unknown": false
        },
        "raw_text": {
            "column": "text"
        }
    },
    "embedding": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "embed_dim": 100,
        "embedding_init_strategy": "random",
        "embedding_init_range": null,
        "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": []
    },
    "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": {}
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null
    }
}
```

WordTaggingModel.Config

Component: *WordTaggingModel*

class WordTaggingModel.**Config**

Bases: *Model.Config*

All Attributes (including base classes)

inputs: *ModelInput* = *ModelInput*()

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": {},
    "representation": {
        "BiLSTMSlotAttention": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
```

(continues on next page)

(continued from previous page)

```
            "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
            },
            "slot_attention": {
                "attn_dimension": 64,
                "attention_type": "no_attention"
            },
            "mlp_decoder": null
        }
    },
    "output_layer": {
        "WordTaggingOutputLayer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {}
        }
    },
    "decoder": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "hidden_dims": [],
        "out_dim": null
    }
}
```

1.14.7 optimizer

scheduler

BatchScheduler.Config

Component: *BatchScheduler*

class BatchScheduler.**Config**
Bases: *Scheduler.Config*

All Attributes (including base classes)

Subclasses

- *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
}
```

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
}
```

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*
- *ExponentialLR.Config*
- *LmFineTuning.Config*
- *ReduceLROnPlateau.Config*
- *StepLR.Config*
- *WarmupScheduler.Config*

Default JSON

```
{}
```

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

Default JSON

```
{
    "warmup_steps": 10000
}
```

Adam.Config

Component: *Adam*

class Adam.**Config**
Bases: *Optimizer.Config*

All Attributes (including base classes)

lr: float = 0.001

weight_decay: float = 1e-05

Default JSON

```
{
    "lr": 0.001,
    "weight_decay": 1e-05
}
```

Optimizer.Config

Component: *Optimizer*

class Optimizer.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

Subclasses

- *Adam.Config*
- *SGD.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
}
```

1.14.8 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.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: *NewTaskTrainer.Config* = *NewTaskTrainer.Config*()

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": {
        "epoch_size": null,
        "sampler": {
            "EvalBatchSampler": {}
        },
        "test_key": null
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "tasks": {},
    "task_weights": {},
    "target_task_name": null,
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "use_subtask_select_metric": false
    }
}
```


new_task

NewDocumentClassification.Config

Component: *NewDocumentClassification*

class NewDocumentClassification.**Config**

Bases: *NewTask.Config*

All Attributes (including base classes)

data: *Data.Config* = *Data.Config*()

trainer: *NewTaskTrainer.Config* = *NewTaskTrainer.Config*()

model: *BaseModel.Config* = *NewDocModel.Config*()

metric_reporter: *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*()

Subclasses

- *PairwiseClassificationTask.Config*

Default JSON


```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t"
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 10000
                }
            },
            "sort_key": null,
            "epoch_size": null
        }
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "model": {
        "NewDocModel": {
            "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
                },
                "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
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            }
        },
        "inputs": {
            "tokens": {
                "column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null
            },
            "labels": {
                "column": "label",
                "allow_unknown": true
            },
            "raw_text": {
                "column": "text"
            }
        },
        "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,
                "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": []
            }
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "model_select_metric": "accuracy",
        "target_label": null
    }
}
```


NewDocumentRegression.Config

Component: *NewDocumentRegression*

class NewDocumentRegression.**Config**

Bases: *NewTask.Config*

All Attributes (including base classes)

data: *Data.Config* = *Data.Config*()

trainer: *NewTaskTrainer.Config* = *NewTaskTrainer.Config*()

model: *BaseModel.Config* = *NewDocRegressionModel.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"
                }
            },
            "batcher": {
```


(continues on next page)

(continued from previous page)

```
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 10000
                }
            },
            "sort_key": null,
            "epoch_size": null
        }
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "model": {
        "NewDocRegressionModel": {
            "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
                    },
                    "pooling": {
                        "SelfAttention": {
                            "attn_dimension": 64,
                            "dropout": 0.4
                        }
                    },
                    "mlp_decoder": null
                }
            },
            "decoder": {
                "load_path": null,
                "save_path": null,
```

(continues on next page)

(continued from previous page)

```
                "freeze": false,
                "shared_module_key": null,
                "hidden_dims": [],
                "out_dim": null
            },
            "output_layer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {},
                "squash_to_unit_range": false
            },
            "inputs": {
                "tokens": {
                    "column": "text",
                    "tokenizer": {
                        "Tokenizer": {
                            "split_regex": "\\s+",
                            "lowercase": true
                        }
                    },
                    "add_bos_token": false,
                    "add_eos_token": false,
                    "use_eos_token_for_bos": false,
                    "max_seq_len": null
                },
                "labels": {
                    "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,
                "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": []
            }
        }
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

NewTask.Config

Component: *NewTask*

class NewTask.**Config**
Bases: _NewTask.Config

All Attributes (including base classes)

data: *Data.Config* = *Data.Config*()

trainer: *NewTaskTrainer.Config* = *NewTaskTrainer.Config*()

model: *BaseModel.Config*

Subclasses

- *NewDocumentClassification.Config*
- *NewDocumentRegression.Config*
- *PairwiseClassificationTask.Config*
- *NewWordTaggingTask.Config*

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

NewTaskTrainer.Config

Component: *NewTaskTrainer*

class NewTaskTrainer.**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`

optimizer: *Optimizer.Config* = *Adam.Config*()

scheduler: Optional[*Scheduler.Config***]** = `None`

Default JSON

```
{
    "epochs": 10,
    "early_stop_after": 0,
    "max_clip_norm": null,
    "report_train_metrics": true,
    "target_time_limit_seconds": null,
    "do_eval": true,
    "optimizer": {
        "Adam": {
            "lr": 0.001,
            "weight_decay": 1e-05
        }
    },
    "scheduler": null
}
```


PairwiseClassificationTask.Config

Component: *PairwiseClassificationTask*

class PairwiseClassificationTask.**Config**

Bases: *NewDocumentClassification.Config*

All Attributes (including base classes)

data: *Data.Config* = *Data.Config*()

trainer: *NewTaskTrainer.Config* = *NewTaskTrainer.Config*()

model: *PairwiseClassificationModel.Config* = *PairwiseClassificationModel.Config*()

metric_reporter: *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*()

Default JSON


```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t"
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 10000
                }
            },
            "sort_key": null,
            "epoch_size": null
```


(continues on next page)

(continued from previous page)

```
        }
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "model": {
        "inputs": {
            "tokens1": {
                "column": "text1",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null
            },
            "tokens2": {
                "column": "text2",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null
            },
            "labels": {
                "column": "label",
                "allow_unknown": false
            },
            "raw_text": {
                "columns": [
                    "text1",
                    "text2"
                ],
                "delimiter": " | "
            }
        },
```

(continues on next page)

(continued from previous page)


```
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "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
                },
                "pooling": {
                    "SelfAttention": {
                        "attn_dimension": 64,
                        "dropout": 0.4
                    }
                },
                "mlp_decoder": null
            }
        },
        "shared_representations": true,
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null
        },
        "output_layer": {
```


(continues on next page)

(continued from previous page)

```
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            }
        },
        "encode_relations": true
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "model_select_metric": "accuracy",
        "target_label": null
    }
}
```

_NewTask.Config

Component: _NewTask

class _NewTask.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

data: *Data.Config* = *Data.Config*()

trainer: *NewTaskTrainer.Config* = *NewTaskTrainer.Config*()

Subclasses

- *NewDisjointMultitask.Config*
- *NewDocumentClassification.Config*
- *NewDocumentRegression.Config*
- *NewTask.Config*
- *PairwiseClassificationTask.Config*
- *NewWordTaggingTask.Config*

Default JSON

```
{
    "data": {
        "Data": {
            "source": {
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t"
                }
            },
```

(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": 10000
                }
            },
            "sort_key": null,
            "epoch_size": null
        }
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    }
}
```


task

Task.Config

Component: *Task*

class Task.**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**

Subclasses

- *ContextualIntentSlotTask.Config*
- *DocClassificationTask.Config*
- *EnsembleTask.Config*
- *JointTextTask.Config*

- *LMTask.Config*
- *PairClassificationTask.Config*
- *QueryDocumentPairwiseRankingTask.Config*
- *SemanticParsingTask.Config*
- *SeqNNTask.Config*
- *WordTaggingTask.Config*

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

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.Config*
- *ContextualIntentSlotTask.Config*
- *DocClassificationTask.Config*
- *EnsembleTask.Config*
- *JointTextTask.Config*
- *LMTask.Config*
- *PairClassificationTask.Config*
- *QueryDocumentPairwiseRankingTask.Config*
- *SemanticParsingTask.Config*
- *SeqNNTask.Config*
- *WordTaggingTask.Config*

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

tasks

ContextualIntentSlotTask.Config

Component: *ContextualIntentSlotTask*

class ContextualIntentSlotTask.**Config**
Bases: *Task.Config*

All Attributes (including base classes)

features: *ModelInputConfig* = *ModelInputConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: *ContextualIntentSlotModelDataHandler.Config* = *ContextualIntentSlotModelDataHandler.Config*()

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*DenseFeatureExporter.Config*] = **None**

labels: list[Union[*DocLabelConfig*, *WordLabelConfig*]]

model: *ContextualIntentSlotModel.Config* = *ContextualIntentSlotModel.Config*()

metric_reporter: *IntentSlotMetricReporter.Config* = *IntentSlotMetricReporter.Config*()

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

DocClassificationTask.Config

Component: *DocClassificationTask*

class DocClassificationTask.**Config**
Bases: *Task.Config*

All Attributes (including base classes)

features: *ModelInputConfig* = *ModelInputConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: *DocClassificationDataHandler.Config* = *DocClassificationDataHandler.Config*()

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*DenseFeatureExporter.Config*] = **None**

model: *DocModel_Deprecated.Config* = *DocModel_Deprecated.Config*()

labels: *DocLabelConfig* = *DocLabelConfig*()

metric_reporter: *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*()

Default JSON

```
{
    "features": {
        "load_path": null,
```

(continues on next page)

(continued from previous page)

```
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "word_feat": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "dict_feat": null,
        "char_feat": null,
        "contextual_token_embedding": null,
        "dense_feat": null
    },
    "featurizer": {
        "SimpleFeaturizer": {
            "sentence_markers": null,
            "lowercase_tokens": true,
            "split_regex": "\\s+",
            "convert_to_bytes": false
        }
    },
    "data_handler": {
        "columns_to_read": [
            "doc_label",
            "text",
            "dict_feat"
        ],
        "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,
        "max_seq_len": -1
    },
    "trainer": {
        "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,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "exporter": null,
    "model": {
        "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
                },
                "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
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            }
        }
    },
```

(continues on next page)

(continued from previous page)

```
    "labels": {
        "export_output_names": [
            "doc_scores"
        ],
        "label_weights": {},
        "target_prob": false
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "model_select_metric": "accuracy",
        "target_label": null
    }
}
```

EnsembleTask.Config

Component: *EnsembleTask*

class EnsembleTask.**Config**
Bases: *Task.Config*

All Attributes (including base classes)

features: ***FeatureConfig*** = ***FeatureConfig***()

featurizer: ***Featurizer.Config*** = ***SimpleFeaturizer.Config***()

data_handler: ***JointModelDataHandler.Config*** = ***JointModelDataHandler.Config***()

trainer: ***EnsembleTrainer.Config*** = ***EnsembleTrainer.Config***()

exporter: Optional[***ModelExporter.Config***] = **None**

model: Union[***BaggingDocEnsemble.Config***, ***BaggingIntentSlotEnsemble.Config***]

labels: list[Union[***DocLabelConfig***, ***WordLabelConfig***]]

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.

JointTextTask.Config

Component: *JointTextTask*

class JointTextTask.**Config**
Bases: *Task.Config*

All Attributes (including base classes)

features: ***FeatureConfig*** = ***FeatureConfig***()

featurizer: ***Featurizer.Config*** = ***SimpleFeaturizer.Config***()

data_handler: ***JointModelDataHandler.Config*** = ***JointModelDataHandler.Config***()

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*ModelExporter.Config*] = `None`

labels: list[Union[*DocLabelConfig*, *WordLabelConfig*]]

model: *JointModel.Config* = *JointModel.Config*()

metric_reporter: *IntentSlotMetricReporter.Config* = *IntentSlotMetricReporter.Config*()

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

LMTask.Config

Component: *LMTask*

class LMTask.**Config**
Bases: *Task.Config*

All Attributes (including base classes)

features: *FeatureConfig* = *FeatureConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: Union[*LanguageModelDataHandler.Config*, *BPTTLanguageModelDataHandler.Config*] = *LanguageModelD*

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*ModelExporter.Config*] = `None`

model: *LMLSTM.Config* = *LMLSTM.Config*()

labels: Optional[*WordLabelConfig*] = `None`

metric_reporter: *LanguageModelMetricReporter.Config* = *LanguageModelMetricReporter.Config*()

Default JSON


```
{
    "features": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "word_feat": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "export_input_names": [
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "",
            "vocab_file": "",
```


(continues on next page)

(continued from previous page)

```
            "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": []
        },
        "seq_word_feat": null,
        "dict_feat": null,
        "char_feat": null,
        "dense_feat": null,
        "contextual_token_embedding": null
    },
    "featurizer": {
        "SimpleFeaturizer": {
            "sentence_markers": null,
            "lowercase_tokens": true,
            "split_regex": "\\s+",
            "convert_to_bytes": false
        }
    },
    "data_handler": {
        "LanguageModelDataHandler": {
            "columns_to_read": [
                "text"
            ],
            "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,
            "append_bos": true,
            "append_eos": true
        }
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "exporter": null,
    "model": {
        "representation": {
```

(continues on next page)

(continued from previous page)


```
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "dropout": 0.4,
            "lstm_dim": 32,
            "num_layers": 1,
            "bidirectional": false
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {}
        },
        "tied_weights": false,
        "stateful": false
    },
    "labels": null,
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt"
    }
}
```


NewWordTaggingTask.Config

Component: *NewWordTaggingTask*

class NewWordTaggingTask.**Config**

Bases: *NewTask.Config*

All Attributes (including base classes)

data: *Data.Config* = *Data.Config*()

trainer: *NewTaskTrainer.Config* = *NewTaskTrainer.Config*()

model: *NewWordTaggingModel.Config* = *NewWordTaggingModel.Config*()

metric_reporter: *SimpleWordTaggingMetricReporter.Config* = *SimpleWordTaggingMetricReporter.Config*()

Default JSON


```
{
    "data": {
        "Data": {
            "source": {
```


(continues on next page)

(continued from previous page)

```
                "TSVDataSource": {
                    "column_mapping": {},
                    "train_filename": null,
                    "test_filename": null,
                    "eval_filename": null,
                    "field_names": null,
                    "delimiter": "\t"
                }
            },
            "batcher": {
                "PoolingBatcher": {
                    "train_batch_size": 16,
                    "eval_batch_size": 16,
                    "test_batch_size": 16,
                    "pool_num_batches": 10000
                }
            },
            "sort_key": null,
            "epoch_size": null
        }
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "model": {
        "inputs": {
            "tokens": {
                "column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
                        "lowercase": true
                    }
                },
                "add_bos_token": false,
                "add_eos_token": false,
                "use_eos_token_for_bos": false,
                "max_seq_len": null
            },
            "labels": {
                "slot_column": "slots",
                "text_column": "text",
                "tokenizer": {
                    "Tokenizer": {
                        "split_regex": "\\s+",
```

(continues on next page)

(continued from previous page)


```
                        "lowercase": true
                    }
                },
                "allow_unknown": false
            },
            "raw_text": {
                "column": "text"
            }
        },
        "embedding": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "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": {}
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null
        }
    },
    "metric_reporter": {
```


(continues on next page)

(continued from previous page)

```
        "output_path": "/tmp/test_out.txt"
    }
}
```

PairClassificationTask.Config

Component: *PairClassificationTask*

class PairClassificationTask.**Config**

Bases: *Task.Config*

All Attributes (including base classes)

features: *ModelInputConfig* = *ModelInputConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: *PairClassificationDataHandler.Config* = *PairClassificationDataHandler.Config*()

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*ModelExporter.Config*] = **None**

model: *PairClassificationModel.Config* = *PairClassificationModel.Config*()

labels: *DocLabelConfig* = *DocLabelConfig*()

metric_reporter: *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*()

Default JSON

```
{
    "features": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "text1": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "text2": {
```

(continues on next page)

(continued from previous page)

```
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        }
    },
    "featurizer": {
        "SimpleFeaturizer": {
            "sentence_markers": null,
            "lowercase_tokens": true,
            "split_regex": "\\s+",
            "convert_to_bytes": false
        }
    },
    "data_handler": {
        "columns_to_read": [
            "doc_label",
            "text1",
            "text2"
        ],
        "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
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
```

(continues on next page)

(continued from previous page)


```
        "scheduler": null
    },
    "exporter": null,
    "model": {
        "representation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "subrepresentation": {
                "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
                    },
                    "pooling": {
                        "SelfAttention": {
                            "attn_dimension": 64,
                            "dropout": 0.4
                        }
                    },
                    "mlp_decoder": null
                }
            },
            "subrepresentation_right": null,
            "encode_relations": true
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null
        },
        "output_layer": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "loss": {
                "CrossEntropyLoss": {}
            }
        }
    },
    "labels": {
```


(continues on next page)

(continued from previous page)

```
        "export_output_names": [
            "doc_scores"
        ],
        "label_weights": {},
        "target_prob": false
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "model_select_metric": "accuracy",
        "target_label": null
    }
}
```

QueryDocumentPairwiseRankingTask.Config

Component: *QueryDocumentPairwiseRankingTask*

class QueryDocumentPairwiseRankingTask.**Config**

Bases: *Task.Config*

All Attributes (including base classes)

features: *ModelInputConfig* = *ModelInputConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: *QueryDocumentPairwiseRankingDataHandler.Config* = *QueryDocumentPairwiseRankingDataHandler.Config*

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*ModelExporter.Config*] = **None**

model: *QueryDocumentPairwiseRankingModel.Config* = *QueryDocumentPairwiseRankingModel.Config*()

labels: Optional[*DocLabelConfig*] = **None**

metric_reporter: *PairwiseRankingMetricReporter.Config* = *PairwiseRankingMetricReporter.Config*()

Default JSON

```
{
    "features": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "pos_response": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "neg_response": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "query": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        }
    },
    "featurizer": {
        "SimpleFeaturizer": {
```


(continues on next page)

(continued from previous page)

```
            "sentence_markers": null,
            "lowercase_tokens": true,
            "split_regex": "\\s+",
            "convert_to_bytes": false
        }
    },
    "data_handler": {
        "columns_to_read": [
            "query",
            "pos_response",
            "neg_response"
        ],
        "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
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "exporter": null,
    "model": {
        "representation": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "subrepresentation": {
                "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
                    },
                    "pooling": {
                        "SelfAttention": {
                            "attn_dimension": 64,
                            "dropout": 0.4
                        }
                    },
                    "mlp_decoder": null
                }
            },
            "shared_representations": true
        },
        "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
            }
        },
        "decoder_output_dim": 64
    },
    "labels": null,
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt"
    }
}
```

SemanticParsingTask.Config

Component: *SemanticParsingTask*

class SemanticParsingTask.**Config**

Bases: *Task.Config*

All Attributes (including base classes)

features: *FeatureConfig* = *FeatureConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: *CompositionalDataHandler.Config* = *CompositionalDataHandler.Config*()

trainer: *HogwildTrainer.Config* = *HogwildTrainer.Config*()

exporter: Optional[*ModelExporter.Config*] = **None**

model: *RNNGParser.Config* = *RNNGParser.Config*()

labels: Optional[*WordLabelConfig*] = None

metric_reporter: *CompositionalMetricReporter.Config* = *CompositionalMetricReporter.Config*()

Default JSON


```
{
    "features": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "word_feat": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "seq_word_feat": null,
        "dict_feat": null,
        "char_feat": null,
        "dense_feat": null,
        "contextual_token_embedding": null
    },
    "featurizer": {
        "SimpleFeaturizer": {
            "sentence_markers": null,
            "lowercase_tokens": true,
            "split_regex": "\\s+",
            "convert_to_bytes": false
        }
    },
    "data_handler": {
        "columns_to_read": [
            "doc_label",
            "word_label",
            "text",
            "dict_feat",
            "seqlogical"
        ],
        "shuffle": true,
        "sort_within_batch": true,
        "train_path": "train.tsv",
        "eval_path": "eval.tsv",
```


(continues on next page)

(continued from previous page)

```
        "test_path": "test.tsv",
        "train_batch_size": 1,
        "eval_batch_size": 1,
        "test_batch_size": 1
    },
    "trainer": {
        "real_trainer": {
            "epochs": 10,
            "early_stop_after": 0,
            "max_clip_norm": null,
            "report_train_metrics": true,
            "target_time_limit_seconds": null,
            "do_eval": true,
            "optimizer": {
                "Adam": {
                    "lr": 0.001,
                    "weight_decay": 1e-05
                }
            },
            "scheduler": null
        },
        "num_workers": 1
    },
    "exporter": 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
        },
        "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"
    },
    "labels": null,
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt"
    }
```

(continues on next page)

(continued from previous page)

```
}
```

SeqNNTask.Config

Component: *SeqNNTask*

class SeqNNTask.**Config**

Bases: *Task.Config*

All Attributes (including base classes)

features: *FeatureConfig* = *FeatureConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: *SeqModelDataHandler.Config* = *SeqModelDataHandler.Config*()

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*ModelExporter.Config*] = **None**

model: *SeqNNModel.Config* = *SeqNNModel.Config*()

labels: *DocLabelConfig* = *DocLabelConfig*()

metric_reporter: *ClassificationMetricReporter.Config* = *ClassificationMetricReporter.Config*()

Default JSON

```
{
    "features": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "word_feat": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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": []
        },
        "seq_word_feat": null,
        "dict_feat": null,
        "char_feat": null,
```

(continues on next page)

(continued from previous page)

```
        "dense_feat": null,
        "contextual_token_embedding": null
    },
    "featurizer": {
        "SimpleFeaturizer": {
            "sentence_markers": null,
            "lowercase_tokens": true,
            "split_regex": "\\s+",
            "convert_to_bytes": false
        }
    },
    "data_handler": {
        "columns_to_read": [
            "doc_label",
            "text"
        ],
        "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,
        "max_seq_len": -1,
        "pretrained_embeds_file": ""
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "exporter": null,
    "model": {
        "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,
```

(continues on next page)

(continued from previous page)

```
                    "kernel_sizes": [
                        3,
                        4
                    ]
                }
            },
            "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
                    },
                    "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": {}
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null
        }
    },
    "labels": {
        "export_output_names": [
            "doc_scores"
        ],
        "label_weights": {},
        "target_prob": false
```

(continues on next page)

(continued from previous page)

```
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt",
        "model_select_metric": "accuracy",
        "target_label": null
    }
}
```

WordTaggingTask.Config

Component: *WordTaggingTask*

class WordTaggingTask.**Config**

Bases: *Task.Config*

All Attributes (including base classes)

features: *FeatureConfig* = *FeatureConfig*()

featurizer: *Featurizer.Config* = *SimpleFeaturizer.Config*()

data_handler: *JointModelDataHandler.Config* = *JointModelDataHandler.Config*()

trainer: *Trainer.Config* = *Trainer.Config*()

exporter: Optional[*ModelExporter.Config*] = **None**

model: *WordTaggingModel.Config* = *WordTaggingModel.Config*()

labels: *WordLabelConfig* = *WordLabelConfig*()

metric_reporter: *WordTaggingMetricReporter.Config* = *WordTaggingMetricReporter.Config*()

Default JSON

```
{
    "features": {
        "load_path": null,
        "save_path": null,
        "freeze": false,
        "shared_module_key": null,
        "word_feat": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "embed_dim": 100,
            "embedding_init_strategy": "random",
            "embedding_init_range": null,
            "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,
```

(continues on next page)

(continued from previous page)

```
            "min_freq": 1,
            "mlp_layer_dims": []
        },
        "seq_word_feat": null,
        "dict_feat": null,
        "char_feat": null,
        "dense_feat": null,
        "contextual_token_embedding": null
    },
    "featurizer": {
        "SimpleFeaturizer": {
            "sentence_markers": null,
            "lowercase_tokens": true,
            "split_regex": "\\s+",
            "convert_to_bytes": false
        }
    },
    "data_handler": {
        "columns_to_read": [
            "doc_label",
            "word_label",
            "text",
            "dict_feat",
            "doc_weight",
            "word_weight"
        ],
        "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,
        "max_seq_len": -1
    },
    "trainer": {
        "epochs": 10,
        "early_stop_after": 0,
        "max_clip_norm": null,
        "report_train_metrics": true,
        "target_time_limit_seconds": null,
        "do_eval": true,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "exporter": null,
    "model": {
        "inputs": {},
        "representation": {
            "BiLSTMSlotAttention": {
                "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
                },
                "slot_attention": {
                    "attn_dimension": 64,
                    "attention_type": "no_attention"
                },
                "mlp_decoder": null
            }
        },
        "output_layer": {
            "WordTaggingOutputLayer": {
                "load_path": null,
                "save_path": null,
                "freeze": false,
                "shared_module_key": null,
                "loss": {}
            }
        },
        "decoder": {
            "load_path": null,
            "save_path": null,
            "freeze": false,
            "shared_module_key": null,
            "hidden_dims": [],
            "out_dim": null
        }
    },
    "labels": {
        "use_bio_labels": false,
        "export_output_names": [
            "word_scores"
        ]
    },
    "metric_reporter": {
        "output_path": "/tmp/test_out.txt"
    }
}
```

1.14.9 trainers

ensemble_trainer

EnsembleTrainer.Config

Component: *EnsembleTrainer*

class EnsembleTrainer.**Config**
Bases: *ConfigBase*

All Attributes (including base classes)

real_trainer: *Trainer.Config* = *Trainer.Config*()

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,
        "optimizer": {
            "Adam": {
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    }
}
```

hogwild_trainer

HogwildTrainer.Config

Component: *HogwildTrainer*

class HogwildTrainer.**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,
        "optimizer": {
            "Adam": {
```

(continues on next page)

(continued from previous page)

```
                "lr": 0.001,
                "weight_decay": 1e-05
            }
        },
        "scheduler": null
    },
    "num_workers": 1
}
```

trainer

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.

optimizer: *Optimizer.Config* = *Adam.Config*()

scheduler: Optional[*Scheduler.Config***] = None**

Subclasses

- *NewTaskTrainer.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,
    "optimizer": {
        "Adam": {
            "lr": 0.001,
            "weight_decay": 1e-05
        }
    },
    "scheduler": null
}
```

TrainerBase.Config

Component: *TrainerBase*

class TrainerBase.**Config**
Bases: *Component.Config*

All Attributes (including base classes)

Default JSON

```
{}
```

1.15 pytext package

1.15.1 Subpackages

pytext.common package

Submodules

pytext.common.constants module

class pytext.common.constants.**BatchContext**
Bases: object

IGNORE_LOSS = 'ignore_loss'

INDEX = 'index'

TASK_NAME = 'task_name'

class pytext.common.constants.**DFColumn**
Bases: object

ALIGNMENT = 'alignment'

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_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

WORD_LABEL_PAD = 'PAD_LABEL'

WORD_LABEL_PAD_IDX = 0

class pytext.common.constants.**Stage**
Bases: enum.Enum

An enumeration.

EVAL = 'Evaluation'

TEST = 'Test'

TRAIN = 'Training'

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>'

Module contents

pytext.config package

Submodules

pytext.config.component module

class pytext.config.component.**Component**(*config=None*, **args*, ***kwargs*)
Bases: object

Config
alias of *Component.Config*

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'

METRIC_REPORTER = 'metric_reporter'

MODEL = 'model'

MODEL2 = 'model2'

MODULE = 'module'

OPTIMIZER = 'optimizer'

PREDICTOR = 'predictor'

SCHEDULER = 'scheduler'

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_scheduler**(*scheduler_config*, *optimizer*, **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.**doc_model_deprecated**(*json_config*)
Rename DocModel to DocModel_Deprecated

pytext.config.config_adapter.**find_dicts_containing_key**(*json_config*, *key*)

pytext.config.config_adapter.**register_adapter**(*from_version*)

pytext.config.config_adapter.**upgrade_one_version**(*json_config*)

pytext.config.config_adapter.**upgrade_to_latest**(*json_config*)

pytext.config.config_adapter.**v0_to_v1**(*json_config*)

pytext.config.config_adapter.**v1_to_v2**(*json_config*)

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.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 = 'utterance'

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']

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.**CNNParams**(***kwargs*)
Bases: *pytext.config.pytext_config.ConfigBase*

kernel_num = 100

kernel_sizes = [3, 4]

pytext.config.module_config.**ModuleConfig**
alias of *pytext.config.module_config.ModuleConfig*

class pytext.config.module_config.**PoolingType**
Bases: enum.Enum

An enumeration.

MAX = 'max'

MEAN = 'mean'

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.**PlaceHolder**
Bases: object

class pytext.config.pytext_config.**PyTextConfig**(***kwargs*)
Bases: *pytext.config.pytext_config.ConfigBase*

debug_path = '/tmp/model.debug'

distributed_world_size = 1

export_caffe2_path = '/tmp/model.caffe2.predictor'

export_onnx_path = '/tmp/model.onnx'

export_torchscript_path = None

load_snapshot_path = ''

modules_save_dir = ''

random_seed = None
Seed value to seed torch, python, and numpy random generators.

report_eval_results = False

save_all_checkpoints = False

save_module_checkpoints = False

save_snapshot_path = '/tmp/model.pt'

test_out_path = '/tmp/test_out.txt'

use_cuda_if_available = True

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_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*

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*)

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()

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()

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()

class pytext.data.data_structures.annotation.**Slot**(*label*)

Bases: *pytext.data.data_structures.annotation.Node*

validate_node()

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()

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 = ''*)
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*)

to_actions()

validate_tree()
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()

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*)

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

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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.

Config
alias of *SimpleFeaturizer.Config*

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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.

Config
alias of *SimpleFeaturizer.Config*

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.data_source module

class pytext.data.sources.data_source.**DataSource**(*schema*)
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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*, *column_mapping=()*)
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.

Config
alias of *RootDataSource.Config*

DATA_SOURCE_TYPES = {<class 'str'>: <function load_text>, typing.List[pytext.utils.da

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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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 "Resource-Warning: 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*)

Bases: *pytext.data.sources.data_source.DataSource*

Base class for sharded data sources.

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

pytext.data.sources.data_source.**generator_property**

alias of *pytext.data.sources.data_source.GeneratorMethodProperty*

pytext.data.sources.data_source.**load_slots**(*s*)

pytext.data.sources.data_source.**load_text**(*s*)

pytext.data.sources.squad module

class pytext.data.sources.squad.**SquadDataSource**(*train_filename=None*, *test_filename=None*, *eval_filename=None*, *ignore_impossible=True*)

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)

Config

alias of *SquadDataSource.Config*

eval = <pytext.data.sources.data_source.GeneratorIterator object>

classmethod from_config(*config: pytext.data.sources.squad.SquadDataSource.Config*, *schema=None*)

test = <pytext.data.sources.data_source.GeneratorIterator object>

train = <pytext.data.sources.data_source.GeneratorIterator object>

pytext.data.sources.squad.**unflatten**(*fname*, *ignore_impossible*)

pytext.data.sources.tsv module

class pytext.data.sources.tsv.**BlockShardedTSV**(*file*, *field_names=None*, *delimiter='t'*, *block_id=0*, *num_blocks=1*)

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*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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'}*, ***kwargs*)

Bases: *pytext.data.sources.tsv.TSVDataSource*

Config

alias of *MultilingualTSVDataSource.Config*

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*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

merge_session(*session*)

class pytext.data.sources.tsv.**TSV**(*file*, *field_names=None*, *delimiter='t'*)

Bases: object

class pytext.data.sources.tsv.**TSVDataSource**(*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 TSV sources. Uses python's csv library.

Config
alias of *TSVDataSource.Config*

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*)
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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*)
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)

Config
alias of *SquadDataSource.Config*

eval = <pytext.data.sources.data_source.GeneratorIterator object>

classmethod from_config(*config: pytext.data.sources.squad.SquadDataSource.Config, schema=None*)

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'*, ***kwargs*)

Bases: *pytext.data.sources.data_source.RootDataSource*

DataSource which loads data from TSV sources. Uses python's csv library.

Config
alias of *TSVDataSource.Config*

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.

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_eval_batch_sampler()

test_prob_batch_sampler()

test_round_robin_batch_sampler()

pytext.data.test.bptt_lm_data_handler_test module

class pytext.data.test.bptt_lm_data_handler_test.**BPTTLanguageModelDataHandlerTest**(*methodName*

Bases: unittest.case.TestCase

test_data_handler()

pytext.data.test.compositional_datahandler_test module

class pytext.data.test.compositional_datahandler_test.**CompositionalDataHandlerTest**(*methodNa*

Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_intermediate_result()

test_min_freq()
Test that UNKification is triggered when min_freq is 2.

test_test_tensors()

test_train_tensors()

test_uppercase_tokens()
Test that the text is not lower-cased when lowercase_tokens is False.

pytext.data.test.contextual_intent_slot_data_handler_test module

class pytext.data.test.contextual_intent_slot_data_handler_test.**ContextualIntentSlotModelDa**
Bases: unittest.case.TestCase

test_read_file_with_dense_features()

class pytext.data.test.contextual_intent_slot_data_handler_test.**ContextualIntentSlotModelDa**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_create_from_config()

test_intermediate_result()

test_read_from_file()

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_data_no_batcher_provided()

test_data_initializes_tensorsizers()

test_data_iterate_multiple_times()

test_sort()

pytext.data.test.datahandler_test module

class pytext.data.test.datahandler_test.**DataHandlerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_init_feature_metadata()

test_read_from_csv()

test_read_partially_from_csv()

pytext.data.test.doc_classification_data_handler_test module

class pytext.data.test.doc_classification_data_handler_test.**DocClassificationDataHandlerTes**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_create_from_config()

test_read_from_file()

test_tokenization()

pytext.data.test.joint_data_handler_test module

class pytext.data.test.joint_data_handler_test.**JointDataHandlerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_create_from_config()

test_read_from_file()

test_tokenization()

pytext.data.test.kd_doc_classification_data_handler_test module

class pytext.data.test.kd_doc_classification_data_handler_test.**KDDocClassificationDataHandl**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_create_from_config()

test_read_from_file()

test_tokenization()

pytext.data.test.language_model_data_handler_test module

class pytext.data.test.language_model_data_handler_test.**LanguageModelDataHandlerTest**(*methodl*
Bases: unittest.case.TestCase

classmethod create_language_model_data_handler() → py-text.data.language_model_data_handler.LanguageModelDa

test_data_handler()

test_sharding()

pytext.data.test.query_document_pairwise_ranking_data_handler_test module

class pytext.data.test.query_document_pairwise_ranking_data_handler_test.**QueryDocumentPairw**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_create_from_config()

test_read_from_file()

test_tokenization()

pytext.data.test.round_robin_batchiterator_test module

class pytext.data.test.round_robin_batchiterator_test.**RoundRobinBatchIteratorTest**(*methodName*
Bases: unittest.case.TestCase

test_batch_iterator()

pytext.data.test.seq_data_handler_test module

class pytext.data.test.seq_data_handler_test.**SeqModelDataHandlerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_intermediate_result()

test_process_data()

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.**TensorizersTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_create_byte_tensors()

test_create_float_list_tensor()

test_create_label_tensors()

test_create_word_character_tensors()

test_create_word_tensors()

test_initialize_label_tensorizer()

test_initialize_tensorizers()

test_initialize_word_tensorizer()

pytext.data.test.tokenizers_test module

class pytext.data.test.tokenizers_test.**TokenizeTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_split_with_regex()

test_tokenize()

test_tokenize_dont_lowercase()

pytext.data.test.tsv_data_source_test module

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_iterate_training_data_multiple_times()

test_quoting()

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.**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*)
Bases: *pytext.config.component.Component*

A simple regex-splitting tokenizer.

Config
alias of *Tokenizer.Config*

classmethod from_config(*config: pytext.data.tokenizers.tokenizer.Tokenizer.Config*)

tokenize(*input: str*) → List[pytext.data.tokenizers.tokenizer.Token]

Module contents

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*)
Bases: *pytext.config.component.Component*

A simple regex-splitting tokenizer.

Config
alias of *Tokenizer.Config*

classmethod from_config(*config: pytext.data.tokenizers.tokenizer.Tokenizer.Config*)

tokenize(*input: str*) → List[pytext.data.tokenizers.tokenizer.Token]

Submodules

pytext.data.batch_sampler module

class pytext.data.batch_sampler.**BaseBatchSampler**
Bases: *pytext.config.component.Component*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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]

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.**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 epoch_size in Data.Config.

Example

Iterator A: [A, B, C, D], Iterator B: [a, b]

epoch_size = 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.

Config
alias of *RandomizedBatchSampler.Config*

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.

Config
alias of *RoundRobinBatchSampler.Config*

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.bptt_lm_data_handler module

class pytext.data.bptt_lm_data_handler.**BPTTLanguageModelDataHandler**(*bptt_len: int*, ***kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

BPTTLanguageModelDataHandler treats data as a single document, concatenating all tokens together. BPTTIterator arranges the dataset into columns of batch size and subdivides the source data into chunks of length bptt_len. It enables hidden state of ith batch carried over to (i+1)th batch.

Parameters **bptt_len** (*int*) – Input sequence length to backpropagate to.

Config
alias of *BPTTLanguageModelDataHandler.Config*

classmethod from_config(*config: pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler.Config*, *feature_config: pytext.config.field_config.FeatureConfig*, *label_config: pytext.config.field_config.WordLabelConfig*, ***kwargs*)
Factory method to construct an instance of *BPTTLanguageModelDataHandler* from the module's config object and feature config object.

Parameters

- **config** (*LanguageModelDataHandler.Config*) – Configuration object specifying all the parameters of *BPTTLanguageModelDataHandler*.
- **feature_config** (*FeatureConfig*) – Configuration object specifying all the parameters of all input features.

Returns An instance of *BPTTLanguageModelDataHandler*.

Return type type

get_test_iter(*file_path: str*, *batch_size: int*) → pytext.data.data_handler.BatchIterator
Get test data iterator from test data file.

Parameters

- **file_path** (*str*) – Path to test data file.
- **batch_size** (*int*) – Batch size

Returns

An instance of BatchIterator to iterate over the supplied test data file.

Return type BatchIterator

init_feature_metadata(*train_data: torchtext.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.data.dataset.Dataset*)
Prepares the metadata for the language model features.

init_target_metadata(*train_data: torchtext.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.data.dataset.Dataset*)
Prepares the metadata for the language model target.

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]*) → List[str]
Preprocess steps for a single input row.

Parameters **row_data** (*Dict[str, Any]*) – Dict representing the input row and columns.

Returns List of tokens.

Return type List[str]

pytext.data.compositional_data_handler module

class pytext.data.compositional_data_handler.**CompositionalDataHandler**(*raw_columns: List[str], labels: Dict[str, pytext.fields.field.Field], features: Dict[str, pytext.fields.field.Field], featurizer: pytext.data.featurizer.featurizer.F 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:*

Config
alias of *CompositionalDataHandler.Config*

FULL_FEATURES = ['word_feat', 'dict_feat', 'action_idx_feature', 'contextual_token_emb

classmethod from_config(*config: pytext.data.compositional_data_handler.CompositionalDataHandler.Config*, *feature_config: pytext.config.field_config.FeatureConfig*, **args*, ***kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

pytext.data.contextual_intent_slot_data_handler module

class pytext.data.contextual_intent_slot_data_handler.**ContextualIntentSlotModelDataHandler**

Data Handler to build pipeline to process data and generate tensors to be consumed by ContextualIntentSlotModel. Columns of Input data includes:

1. doc label for intent classification
2. word label for slot tagging of the last utterance
3. a sequence of utterances (e.g., a dialog)
4. Optional dictionary feature contained in the last utterance
5. Optional doc weight that stands for the weight of intent task in joint loss.
6. Optional word weight that stands for the weight of slot task in joint loss.

raw_columns
columns to read from data source. In case of files, the order should match the data stored in that file. Raw columns include

```
[
    RawData.DOC_LABEL,
    RawData.WORD_LABEL,
    RawData.TEXT,
    RawData.DICT_FEAT (Optional),
    RawData.DOC_WEIGHT (Optional),
    RawData.WORD_WEIGHT (Optional),
]
```

labels
doc labels and word labels

features
embeddings generated from sequences of utterances and dictionary features of the last utterance

extra_fields
doc weights, word weights, and etc.

Config
alias of *ContextualIntentSlotModelDataHandler.Config*

classmethod from_config(*config: pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotModelDataHand*
feature_config: pytext.config.contextual_intent_slot.ModelInputConfig,
target_config: List[Union[pytext.config.field_config.DocLabelConfig,
*pytext.config.field_config.WordLabelConfig]], **kwargs*)
Factory method to construct an instance of ContextualIntentSlotModelDataHandler object from the module's config, model input config and target config.

Parameters

- **config** (*Config*) – Configuration object specifying all the parameters of ContextualIntentSlotModelDataHandler.
- **feature_config** (*ModelInputConfig*) – Configuration object specifying model input.
- **target_config** (*TargetConfig*) – Configuration object specifying target.

Returns An instance of ContextualIntentSlotModelDataHandler.

Return type type

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
Preprocess steps for a single input row: 1. apply tokenization to a sequence of utterances; 2. process dictionary features to align with the last utterance. 3. align word labels with the last utterance.

Parameters **row_data** (*Dict[str, Any]*) – Dict of one row data with column names as keys. Keys includes "doc_label", "word_label", "text", "dict_feat", "word weight" and "doc weight".

Returns

Preprocessed dict of one row data includes:

"seq_word_feat" (list of list of string) tokenized words of sequence of utterances

"word_feat" (list of string) tokenized words of last utterance

"raw_word_label" (string) raw word label

"token_range" (list of tuple) token ranges of word labels, each tuple contains the start position index and the end position index

"utterance" (list of string) raw utterances

"word_label" (list of string) list of labels of words in last utterance

"doc_label" (string) doc label for intent classification

"word_weight" (float) weight of word label

"doc_weight" (float) weight of document label

"dict_feat" (tuple, optional) tuple of three lists, the first is the label of each words, the second is the weight of the feature, the third is the length of the feature.

Return type Dict[str, Any]

class pytext.data.contextual_intent_slot_data_handler.**RawData**

Bases: object

DENSE_FEAT = 'dense_feat'

DICT_FEAT = 'dict_feat'

DOC_LABEL = 'doc_label'

DOC_WEIGHT = 'doc_weight'

TEXT = 'text'

WORD_LABEL = 'word_label'

WORD_WEIGHT = 'word_weight'

pytext.data.data module

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.

Config

alias of *Batcher.Config*

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, epoch_size: Optional[int] = None*)

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.

Config
alias of *Data.Config*

batches(*stage: pytext.common.constants.Stage*, *data_source=None*)
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

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, **kwargs*)

numberize_rows(*rows*)

class pytext.data.data.**PoolingBatcher**(*train_batch_size=16, eval_batch_size=16, test_batch_size=16, pool_num_batches=10000*)

Bases: *pytext.data.data.Batcher*

Batcher that looks at pools of data, and sorts, batches, and shuffles them, before padding.

Config
alias of *PoolingBatcher.Config*

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, by

1. Load pool of batch_size * pool_num_batches examples.
2. Sort rows, if necessary.
3. Form batches with batch_size examples each.
4. Shuffle batches and yield all batches.

classmethod from_config(*config: pytext.data.data.PoolingBatcher.Config*)

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, **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

Config
alias of *DataHandler.Config*

gen_dataset(*data: Iterable[Dict[str, Any]], include_label_fields: bool = True, shard_range: Tuple[int, int] = None*) → torchtext.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.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.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.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.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.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.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.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 epoch_size: Optional[int] = None, 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

Config
alias of *DisjointMultitaskData.Config*

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*)

pytext.data.disjoint_multitask_data_handler module

class pytext.data.disjoint_multitask_data_handler.**DisjointMultitaskDataHandler**(*config: pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler.Config*, *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

Config
alias of *DisjointMultitaskDataHandler.Config*

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.BatchIterator]*, *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.doc_classification_data_handler module

class pytext.data.doc_classification_data_handler.**DocClassificationDataHandler**(*raw_columns: List[str], la- bels: Dict[str, py- text.fields.field.Fi fea- tures: Dict[str, py- text.fields.field.Fi fea- tur- izer: py- text.data.featuriz ex- tra_fields: Dict[str, py- text.fields.field.Fi = None, text_feature_nam str = 'word_feat', shuf- fle: bool = True, sort_within_batc 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:*

The *DocClassificationDataHandler* prepares the data for document classification. Each sentence is read line by line with its label as the target.

Config
alias of *DocClassificationDataHandler.Config*

classmethod from_config(*config: pytext.data.doc_classification_data_handler.DocClassificationDataHandler.Config*, *model_input_config: pytext.config.doc_classification.ModelInputConfig*, *target_config: pytext.config.field_config.DocLabelConfig*, ***kwargs*)
Factory method to construct an instance of *DocClassificationDataHandler* from the module's config object and feature config object.

Parameters

- **config** (*DocClassificationDataHandler.Config*) – Configuration object specifying all the parameters of *DocClassificationDataHandler*.
- **model_input_config** (*ModelInputConfig*) – Configuration object specifying all the parameters of the model config.
- **target_config** (*TargetConfig*) – Configuration object specifying all the parameters of the target.

Returns An instance of *DocClassificationDataHandler*.

Return type type

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

class pytext.data.doc_classification_data_handler.**RawData**
Bases: object

DICT_FEAT = 'dict_feat'

DOC_LABEL = 'doc_label'

TEXT = 'text'

pytext.data.joint_data_handler module

class pytext.data.joint_data_handler.**JointModelDataHandler**(*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, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

Config
alias of *JointModelDataHandler.Config*

featurize(*row_data: Dict[str, Any]*)

classmethod from_config(*config: pytext.data.joint_data_handler.JointModelDataHandler.Config, feature_config: pytext.config.field_config.FeatureConfig, label_configs: Union[pytext.config.field_config.DocLabelConfig, pytext.config.field_config.WordLabelConfig, List[Union[pytext.config.field_config.DocLabelConfig, pytext.config.field_config.WordLabelConfig]]], **kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

pytext.data.language_model_data_handler module

class pytext.data.language_model_data_handler.**LanguageModelDataHandler**(*raw_columns: List[str], labels: Dict[str, pytext.fields.field.Field], features: Dict[str, pytext.fields.field.Field], featurizer: pytext.data.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:

The *LanguageModelDataHandler* reads input sentences one line at a time and prepares the input and the target for language modeling. Each sentence is assumed to be independent of any other sentence.

Config
alias of *LanguageModelDataHandler.Config*

classmethod from_config(*config: pytext.data.language_model_data_handler.LanguageModelDataHandler.Config*, *feature_config: pytext.config.field_config.FeatureConfig*, **args*, ***kwargs*)
Factory method to construct an instance of *LanguageModelDataHandler* from the module's config object and feature config object.

Parameters

- **config** (*LanguageModelDataHandler.Config*) – Configuration object specifying all the parameters of *LanguageModelDataHandler*.
- **feature_config** (*FeatureConfig*) – Configuration object specifying all the parameters of all input features.

Returns An instance of *LanguageModelDataHandler*.

Return type type

init_target_metadata(*train_data: torchtext.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.data.dataset.Dataset*)
Prepares the metadata for the language model target.

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
Preprocess steps for a single input row.

Parameters **row_data** (*Dict[str, Any]*) – Dict representing the input row and columns.

Returns

Dictionary with feature names as keys and feature values.

Return type Dict[str, Any]

pytext.data.pair_classification_data_handler module

class pytext.data.pair_classification_data_handler.**PairClassificationDataHandler**(*raw_columns: List[str], labels: Dict[str, pytext.fields.field features: Dict[str, pytext.fields.field featurizer: pytext.data.featu extra_fields: Dict[str, pytext.fields.field = None, text_feature_n str = 'word_feat', shuffle: bool = True, sort_within_b bool = True, train_path: str = 'train.tsv', eval_path: str = 'eval.tsv', test_path: str = 'test.tsv', train_batch_s int = 128, eval_batch_si int = 128, test_batch_siz*

Config
alias of *PairClassificationDataHandler.Config*

classmethod from_config(*config: pytext.data.pair_classification_data_handler.PairClassificationDataHandler.Config*, *feature_config: pytext.config.pair_classification.ModelInputConfig*, *target_config: pytext.config.field_config.DocLabelConfig*, ***kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

sort_key(*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.pair_classification_data_handler.**RawData**
Bases: object

DOC_LABEL = 'doc_label'

TEXT1 = 'text1'

TEXT2 = 'text2'

pytext.data.query_document_pairwise_ranking_data_handler module

class pytext.data.query_document_pairwise_ranking_data_handler.**QueryDocumentPairwiseRanking**

Config
alias of *QueryDocumentPairwiseRankingDataHandler.Config*

classmethod from_config(*config: pytext.data.query_document_pairwise_ranking_data_handler.QueryDocumentPairwi*
feature_config: pytext.config.query_document_pairwise_ranking.ModelInputConfig,
target_config: None, ***kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

sort_key(*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.seq_data_handler module

class pytext.data.seq_data_handler.**SeqModelDataHandler**(*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, **kwargs*)
Bases: *pytext.data.joint_data_handler.JointModelDataHandler*

Config
alias of *SeqModelDataHandler.Config*

FULL_FEATURES = ['word_feat']

classmethod from_config(*config: pytext.data.seq_data_handler.SeqModelDataHandler.Config*, *feature_config: pytext.config.field_config.FeatureConfig*, *label_config: pytext.config.field_config.DocLabelConfig*, ***kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

pytext.data.tensorizers module

class pytext.data.tensorizers.**ByteTensorizer**(*text_column*, *lower=True*, *max_seq_len=None*)

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

Config
alias of *ByteTensorizer.Config*

NUM = 256

PAD_BYTE = 0

UNK_BYTE = 0

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.**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.

Config
alias of *CharacterTokenTensorizer.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*)
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.**FloatListTensorizer**(*column: str, error_check: bool, dim: Optional[int]*)
Bases: *pytext.data.tensorizers.Tensorizer*

Numberize numeric labels.

Config
alias of *FloatListTensorizer.Config*

classmethod from_config(*config: pytext.data.tensorizers.FloatListTensorizer.Config*)

numberize(*row*)

tensorize(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

class pytext.data.tensorizers.**JoinStringTensorizer**(*columns: List[str], delimiter: str*)
Bases: *pytext.data.tensorizers.Tensorizer*

A pass-through tensorizer to include raw fields from datasource in the batch. Used mostly for metric reporting.

Config
alias of *JoinStringTensorizer.Config*

classmethod from_config(*config: pytext.data.tensorizers.JoinStringTensorizer.Config*)

numberize(*row*)

class pytext.data.tensorizers.**LabelTensorizer**(*label_column: str = 'label', allow_unknown: bool = False*)
Bases: *pytext.data.tensorizers.Tensorizer*

Numberize labels.

Config
alias of *LabelTensorizer.Config*

classmethod from_config(*config: pytext.data.tensorizers.LabelTensorizer.Config*)

initialize()
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]*)
Bases: *pytext.data.tensorizers.Tensorizer*

A tensorizer which use other tensorizers' numerized data. Used mostly for metric reporting.

Config
alias of *MetricTensorizer.Config*

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]*)
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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*)
Bases: *pytext.data.tensorizers.Tensorizer*

Numberize numeric labels.

Config
alias of *NumericLabelTensorizer.Config*

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.**RawJson**(*column: str*)
Bases: *pytext.data.tensorizers.RawString*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

numberize(*row*)

class pytext.data.tensorizers.**RawString**(*column: str*)
Bases: *pytext.data.tensorizers.Tensorizer*

A pass-through tensorizer to include raw fields from datasource in the batch. Used mostly for metric reporting.

Config
alias of *RawString.Config*

classmethod from_config(*config: pytext.data.tensorizers.RawString.Config*)

numberize(*row*)

class pytext.data.tensorizers.**Tensorizer**(*column_schema: List[Tuple[str, Type[CT_co]]]*)
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.

Config
alias of *Tensorizer.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*)

sort_key(*row*)

tensorize(*batch*)
Tensorizer knows how to pad and tensorize a batch of it's own output.

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=None*)
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.

Config
alias of *TokenTensorizer.Config*

classmethod from_config(*config: pytext.data.tensorizers.TokenTensorizer.Config*)

initialize(*vocab_builder=None*)
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.tensorizers.**WordLabelTensorizer**(*slot_column: str = 'slots'*, *text_column: str = 'text'*, *tokenizer: pytext.data.tokenizers.tokenizer.Tokenizer = None*, *allow_unknown: bool = False*)
Bases: *pytext.data.tensorizers.Tensorizer*

Numberize word/slot labels.

Config
alias of *WordLabelTensorizer.Config*

classmethod from_config(*config: pytext.config.component.Component.Config*)

initialize()
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.

pytext.data.tensorizers.**initialize_tensorizers**(*tensorizers*, *data_source*)
A utility function to stream a data source to the initialize functions of a dict of tensorizers.

pytext.data.utils module

class pytext.data.utils.**SpecialToken**
Bases: str

class pytext.data.utils.**VocabBuilder**
Bases: object

Helper class for aggregating and building *Vocabulary* objects.

add(*value*) → None
Count a single value in the vocabulary.

add_all(*nested_values*) → None
Count a value or nested container of values in the vocabulary.

make_vocab() → pytext.data.utils.Vocabulary
Build a Vocabulary object from the values seen by the builder.

class pytext.data.utils.**Vocabulary**(*vocab_list*, *counts=None*, *replacements=None*)
Bases: object

A mapping from indices to vocab elements.

lookup_all(*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**(*target: List[float], label_list: List[str], batch_label_list: List[List[str]]*)
align the target in the order of label_list, batch_label_list stores the original target order.

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).

Module contents

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.

Config
alias of *Batcher.Config*

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*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.**BPTTLanguageModelDataHandler**(*bptt_len: int*, ***kwargs*)
Bases: *pytext.data.data_handler.DataHandler*

BPTTLanguageModelDataHandler treats data as a single document, concatenating all tokens together. BPTTIterator arranges the dataset into columns of batch size and subdivides the source data into chunks of length bptt_len. It enables hidden state of ith batch carried over to (i+1)th batch.

Parameters **bptt_len** (*int*) – Input sequence length to backpropagate to.

Config
alias of *BPTTLanguageModelDataHandler.Config*

classmethod from_config(*config: pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler.Config*, *feature_config: pytext.config.field_config.FeatureConfig*, *label_config: pytext.config.field_config.WordLabelConfig*, ***kwargs*)
Factory method to construct an instance of *BPTTLanguageModelDataHandler* from the module's config object and feature config object.

Parameters

- **config** (*LanguageModelDataHandler.Config*) – Configuration object specifying all the parameters of *BPTTLanguageModelDataHandler*.
- **feature_config** (*FeatureConfig*) – Configuration object specifying all the parameters of all input features.

Returns An instance of *BPTTLanguageModelDataHandler*.

Return type type

get_test_iter(*file_path: str*, *batch_size: int*) → pytext.data.data_handler.BatchIterator
Get test data iterator from test data file.

Parameters

- **file_path** (*str*) – Path to test data file.
- **batch_size** (*int*) – Batch size

Returns

An instance of BatchIterator to iterate over the supplied test data file.

Return type BatchIterator

init_feature_metadata(*train_data: torchtext.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.data.dataset.Dataset*)
Prepares the metadata for the language model features.

init_target_metadata(*train_data: torchtext.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.data.dataset.Dataset*)
Prepares the metadata for the language model target.

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]*) → List[str]
Preprocess steps for a single input row.

Parameters **row_data** (*Dict[str, Any]*) – Dict representing the input row and columns.

Returns List of tokens.

Return type List[str]

class pytext.data.**CommonMetadata**
Bases: object

class pytext.data.**CompositionalDataHandler**(*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, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

Config
alias of *CompositionalDataHandler.Config*

FULL_FEATURES = ['word_feat', 'dict_feat', 'action_idx_feature', 'contextual_token_emb

classmethod from_config(*config: pytext.data.compositional_data_handler.CompositionalDataHandler.Config, feature_config: pytext.config.field_config.FeatureConfig, *args, **kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

class pytext.data.**ContextualIntentSlotModelDataHandler**(*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, **kwargs*)

Bases: *pytext.data.joint_data_handler.JointModelDataHandler*

Data Handler to build pipeline to process data and generate tensors to be consumed by ContextualIntentSlotModel. Columns of Input data includes:

1. doc label for intent classification
2. word label for slot tagging of the last utterance
3. a sequence of utterances (e.g., a dialog)

4. Optional dictionary feature contained in the last utterance
5. Optional doc weight that stands for the weight of intent task in joint loss.
6. Optional word weight that stands for the weight of slot task in joint loss.

raw_columns
columns to read from data source. In case of files, the order should match the data stored in that file. Raw columns include

```
[
    RawData.DOC_LABEL,
    RawData.WORD_LABEL,
    RawData.TEXT,
    RawData.DICT_FEAT (Optional),
    RawData.DOC_WEIGHT (Optional),
    RawData.WORD_WEIGHT (Optional),
]
```

labels
doc labels and word labels

features
embeddings generated from sequences of utterances and dictionary features of the last utterance

extra_fields
doc weights, word weights, and etc.

Config
alias of *ContextualIntentSlotModelDataHandler.Config*

classmethod from_config(*config: pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotModelDataHand*
feature_config: pytext.config.contextual_intent_slot.ModelInputConfig,
target_config: List[Union[pytext.config.field_config.DocLabelConfig,
*pytext.config.field_config.WordLabelConfig]], **kwargs*)
Factory method to construct an instance of ContextualIntentSlotModelDataHandler object from the module's config, model input config and target config.

Parameters

- **config** (*Config*) – Configuration object specifying all the parameters of ContextualIntentSlotModelDataHandler.
- **feature_config** (*ModelInputConfig*) – Configuration object specifying model input.
- **target_config** (*TargetConfig*) – Configuration object specifying target.

Returns An instance of ContextualIntentSlotModelDataHandler.

Return type type

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
Preprocess steps for a single input row: 1. apply tokenization to a sequence of utterances; 2. process dictionary features to align with the last utterance. 3. align word labels with the last utterance.

Parameters **row_data** (*Dict[str, Any]*) – Dict of one row data with column names as keys. Keys includes "doc_label", "word_label", "text", "dict_feat", "word weight" and "doc weight".

Returns

Preprocessed dict of one row data includes:

"seq_word_feat" (list of list of string) tokenized words of sequence of utterances

"word_feat" (list of string) tokenized words of last utterance

"raw_word_label" (string) raw word label

"token_range" (list of tuple) token ranges of word labels, each tuple contains the start position index and the end position index

"utterance" (list of string) raw utterances

"word_label" (list of string) list of labels of words in last utterance

"doc_label" (string) doc label for intent classification

"word_weight" (float) weight of word label

"doc_weight" (float) weight of document label

"dict_feat" (tuple, optional) tuple of three lists, the first is the label of each words, the second is the weight of the feature, the third is the length of the feature.

Return type Dict[str, Any]

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, epoch_size: Optional[int] = None*)

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.

Config
alias of *Data.Config*

batches(*stage: pytext.common.constants.Stage*, *data_source=None*)
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

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, **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, **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

Config
alias of *DataHandler.Config*

gen_dataset(*data: Iterable[Dict[str, Any]], include_label_fields: bool = True, shard_range: Tuple[int, int] = None*) → torchtext.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.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.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.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.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.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.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], epoch_size: Optional[int] = None, 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

Config
alias of *DisjointMultitaskData.Config*

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*)

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

Config
alias of *DisjointMultitaskDataHandler.Config*

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.**DocClassificationDataHandler**(*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, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

The *DocClassificationDataHandler* prepares the data for document classification. Each sentence is read line by line with its label as the target.

Config
alias of *DocClassificationDataHandler.Config*

classmethod from_config(*config: pytext.data.doc_classification_data_handler.DocClassificationDataHandler.Config*, *model_input_config: pytext.config.doc_classification.ModelInputConfig*, *target_config: pytext.config.field_config.DocLabelConfig*, ***kwargs*)
Factory method to construct an instance of *DocClassificationDataHandler* from the module's config object and feature config object.

Parameters

- **config** (*DocClassificationDataHandler.Config*) – Configuration object specifying all the parameters of *DocClassificationDataHandler*.
- **model_input_config** (*ModelInputConfig*) – Configuration object specifying all the parameters of the model config.
- **target_config** (*TargetConfig*) – Configuration object specifying all the parameters of the target.

Returns An instance of *DocClassificationDataHandler*.

Return type type

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

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]

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.**JointModelDataHandler**(*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, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

Config
alias of *JointModelDataHandler.Config*

featurize(*row_data: Dict[str, Any]*)

classmethod from_config(*config: pytext.data.joint_data_handler.JointModelDataHandler.Config, feature_config: pytext.config.field_config.FeatureConfig, label_configs: Union[pytext.config.field_config.DocLabelConfig, pytext.config.field_config.WordLabelConfig, List[Union[pytext.config.field_config.DocLabelConfig, pytext.config.field_config.WordLabelConfig]]], **kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

class pytext.data.**LanguageModelDataHandler**(*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, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

The *LanguageModelDataHandler* reads input sentences one line at a time and prepares the input and the target for language modeling. Each sentence is assumed to be independent of any other sentence.

Config
alias of *LanguageModelDataHandler.Config*

classmethod from_config(*config: pytext.data.language_model_data_handler.LanguageModelDataHandler.Config, feature_config: pytext.config.field_config.FeatureConfig, *args, **kwargs*)
Factory method to construct an instance of *LanguageModelDataHandler* from the module's config object and feature config object.

Parameters

- **config** (*LanguageModelDataHandler.Config*) – Configuration object specifying all the parameters of *LanguageModelDataHandler*.

- **feature_config** (*FeatureConfig*) – Configuration object specifying all the parameters of all input features.

Returns An instance of *LanguageModelDataHandler*.

Return type type

init_target_metadata(*train_data: torchtext.data.dataset.Dataset*, *eval_data: torchtext.data.dataset.Dataset*, *test_data: torchtext.data.dataset.Dataset*)
Prepares the metadata for the language model target.

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
Preprocess steps for a single input row.

Parameters **row_data** (*Dict[str, Any]*) – Dict representing the input row and columns.

Returns

Dictionary with feature names as keys and feature values.

Return type Dict[str, Any]

class pytext.data.**PairClassificationDataHandler**(*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, **kwargs*)
Bases: *pytext.data.data_handler.DataHandler*

Config
alias of *PairClassificationDataHandler.Config*

classmethod from_config(*config: pytext.data.pair_classification_data_handler.PairClassificationDataHandler.Config*, *feature_config: pytext.config.pair_classification.ModelInputConfig*, *target_config: pytext.config.field_config.DocLabelConfig*, ***kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

sort_key(*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.**PoolingBatcher**(*train_batch_size=16*, *eval_batch_size=16*, *test_batch_size=16*, *pool_num_batches=10000*)
Bases: *pytext.data.data.Batcher*

Batcher that looks at pools of data, and sorts, batches, and shuffles them, before padding.

Config
alias of *PoolingBatcher.Config*

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, by

1. Load pool of batch_size * pool_num_batches examples.
2. Sort rows, if necessary.
3. Form batches with batch_size examples each.
4. Shuffle batches and yield all batches.

classmethod from_config(*config: pytext.data.data.PoolingBatcher.Config*)

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 epoch_size in Data.Config.

Example

Iterator A: [A, B, C, D], Iterator B: [a, b]

epoch_size = 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.

Config
alias of *RandomizedBatchSampler.Config*

batchify(*iterators: Dict[str, collections.abc.Iterator]*)

classmethod from_config(*config: pytext.data.batch_sampler.RandomizedBatchSampler.Config*)

class pytext.data.**QueryDocumentPairwiseRankingDataHandler**(*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, **kwargs*)

Bases: *pytext.data.data_handler.DataHandler*

Config
alias of *QueryDocumentPairwiseRankingDataHandler.Config*

classmethod from_config(*config: pytext.data.query_document_pairwise_ranking_data_handler.QueryDocumentPairwi feature_config: pytext.config.query_document_pairwise_ranking.ModelInputConfig, target_config: None, **kwargs*)

preprocess_row(*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

sort_key(*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.**RawData**
Bases: object

DICT_FEAT = 'dict_feat'

DOC_LABEL = 'doc_label'

TEXT = 'text'

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.

Config
alias of *RoundRobinBatchSampler.Config*

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.**SeqModelDataHandler** (*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, **kwargs*)
Bases: *pytext.data.joint_data_handler.JointModelDataHandler*

Config
alias of *SeqModelDataHandler.Config*

FULL_FEATURES = ['word_feat']

classmethod from_config (*config: pytext.data.seq_data_handler.SeqModelDataHandler.Config, feature_config: pytext.config.field_config.FeatureConfig, label_config: pytext.config.field_config.DocLabelConfig, **kwargs*)

preprocess_row (*row_data: Dict[str, Any]*) → Dict[str, Any]
preprocess steps for a single input row, sub class should override it

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

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

classmethod get_feature_metadata(*feature_config: pytext.config.field_config.FeatureConfig*, *feature_meta: Dict[str, pytext.fields.field.FieldMeta]*)

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]

Config
alias of *ModelExporter.Config*

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-pytext/envs/latest/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]

Config
alias of *ModelExporter.Config*

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-pytext/envs/latest/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

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

classmethod get_feature_metadata(*feature_config: pytext.config.field_config.FeatureConfig*, *feature_meta: Dict[str, pytext.fields.field.FieldMeta]*)

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*, ***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**(*label_weights: Mapping[str, float] = None*, ***kwargs*)
Bases: *pytext.fields.field.Field*

get_meta()

class pytext.fields.field.**Field**(**args*, ***kwargs*)
Bases: torchtext.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.data.field.NestedField

get_meta()

load_meta(*metadata: pytext.fields.field.FieldMeta*)

class pytext.fields.field.**RawField**(**args*, *is_target=False*, ***kwargs*)
Bases: torchtext.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=<function unkify>*, ***kwargs*)

Bases: *pytext.fields.field.TextFeatureField*

build_vocab(**args*, *min_freq=1*, ***kwargs*)

Code is exactly same as as torchtext.data.Field.build_vocab() before the UNKification logic. The reason super().build_vocab() cannot be called is because the Counter object computed in torchtext.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.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],
 ],
 [[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*, ***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**(*label_weights: Mapping[str, float] = None*, ***kwargs*)

Bases: *pytext.fields.field.Field*

get_meta()

class pytext.fields.**Field**(**args*, ***kwargs*)

Bases: torchtext.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.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.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.data.Field.build_vocab() before the UNKification logic. The reason super().build_vocab() cannot be called is because the Counter object computed in torchtext.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.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

Config

alias of *AUCPRHingeLoss.Config*

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*

Config

alias of *BinaryCrossEntropyLoss.Config*

class pytext.loss.loss.**CrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

class pytext.loss.loss.**KLDivergenceBCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config

alias of *KLDivergenceBCELoss.Config*

class pytext.loss.loss.**KLDivergenceCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config
alias of *KLDivergenceCELoss.Config*

class pytext.loss.loss.**LabelSmoothedCrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config
alias of *LabelSmoothedCrossEntropyLoss.Config*

class pytext.loss.loss.**Loss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.config.component.Component*

Base class for loss functions

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

class pytext.loss.loss.**MSELoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Mean squared error loss, for regression tasks.

Config
alias of *MSELoss.Config*

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

Config
alias of *PairwiseRankingLoss.Config*

static get_similarities(*embeddings*)

class pytext.loss.loss.**SoftHardBCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Reference implementation from Distilling the knowledge in a Neural Network: https://arxiv.org/pdf/1503.02531.pdf

Config
alias of *SoftHardBCELoss.Config*

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

Config
alias of *AUCPRHingeLoss.Config*

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

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

class pytext.loss.**CrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

class pytext.loss.**BinaryCrossEntropyLoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config

alias of *BinaryCrossEntropyLoss.Config*

class pytext.loss.**KLDivergenceBCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config

alias of *KLDivergenceBCELoss.Config*

class pytext.loss.**KLDivergenceCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Config

alias of *KLDivergenceCELoss.Config*

class pytext.loss.**MSELoss**(*config=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Mean squared error loss, for regression tasks.

Config

alias of *MSELoss.Config*

class pytext.loss.**SoftHardBCELoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)

Bases: *pytext.loss.loss.Loss*

Reference implementation from Distilling the knowledge in a Neural Network: https://arxiv.org/pdf/1503.02531.pdf

Config
alias of *SoftHardBCELoss.Config*

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

Config
alias of *PairwiseRankingLoss.Config*

static get_similarities(*embeddings*)

class pytext.loss.**LabelSmoothedCrossEntropyLoss**(*config*, *ignore_index=-100*, *weight=None*, **args*, ***kwargs*)
Bases: *pytext.loss.loss.Loss*

Config
alias of *LabelSmoothedCrossEntropyLoss.Config*

pytext.metric_reporters package

Submodules

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'>)*)
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*, *meta*, **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
- **meta** (*Dict[str, Any]*) – global metadata, such as target names

class pytext.metric_reporters.channel.**ConsoleChannel**(*stages: Tuple[pytext.common.constants.Stage, ...] = (<Stage.TRAIN: 'Training'>, <Stage.EVAL: 'Evaluation'>, <Stage.TEST: 'Test'>)*)

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*, *meta*, **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
- **meta** (*Dict[str, Any]*) – global metadata, such as target names

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*, *contexts*)

get_title()

report(*stage*, *epoch*, *metrics*, *model_select_metric*, *loss*, *preds*, *targets*, *scores*, *context*, *meta*, **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
- **meta** (*Dict[str, Any]*) – global metadata, such as target names

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 TensorBoardX SummaryWriter class, or an object that implements the same interface. https://tensorboardx.readthedocs.io/en/latest/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.

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).

report(*stage*, *epoch*, *metrics*, *model_select_metric*, *loss*, *preds*, *targets*, *scores*, *context*, *meta*, *model*, **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*

Config
alias of *ClassificationMetricReporter.Config*

UTTERANCE_COLUMN = 'raw_text'

batch_context(*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

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.**IntentModelChannel**(*stages*, *file_path*)

Bases: *pytext.metric_reporters.channel.FileChannel*

gen_content(*metrics*, *loss*, *preds*, *targets*, *scores*, *contexts*)

get_title()

pytext.metric_reporters.compositional_metric_reporter module

class pytext.metric_reporters.compositional_metric_reporter.**CompositionalFileChannel**(*stages*, *file_path*

Bases: *pytext.metric_reporters.channel.FileChannel*

gen_content(*metrics*, *loss*, *preds*, *targets*, *scores*, *context*)

get_title()

class pytext.metric_reporters.compositional_metric_reporter.**CompositionalMetricReporter**(*acti*
cha
nels
List

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

calculate_metric()
Calculate metrics, each sub class should implement it

create_frame_prediction_pairs()

classmethod from_config(*config*, *metadata: pytext.data.data_handler.CommonMetadata*)

gen_extra_context()
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

static tree_from_tokens_and_indx_actions(*token_str_list: List[str]*, *actions_vocab: List[str]*, *actions_indices: List[int]*)

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.disjoint_multitask_metric_reporter module

class pytext.metric_reporters.disjoint_multitask_metric_reporter.**DisjointMultitaskMetricRep**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of *DisjointMultitaskMetricReporter.Config*

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(*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*)
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

pytext.metric_reporters.intent_slot_detection_metric_reporter module

class pytext.metric_reporters.intent_slot_detection_metric_reporter.**IntentSlotChannel**(*stages*, *file_pa*

Bases: *pytext.metric_reporters.channel.FileChannel*

static create_annotation(*utterance: str*, *intent_label: str*, *slots_label: str*) → str

gen_content(*metrics*, *loss*, *preds*, *targets*, *scores*, *context*)

get_title()

class pytext.metric_reporters.intent_slot_detection_metric_reporter.**IntentSlotMetricReporte**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

aggregate_preds(*new_batch*)

aggregate_scores(*new_batch*)

aggregate_targets(*new_batch*)

calculate_metric()
Calculate metrics, each sub class should implement it

classmethod from_config(*config*, *meta: pytext.data.data_handler.CommonMetadata*)

gen_extra_context()
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

process_pred(*pred: List[int]*) → List[str]
pred is a list of token label index

pytext.metric_reporters.intent_slot_detection_metric_reporter.**create_frame**(*intent_label*, *slot_names_str*, *utterance*)

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()

class pytext.metric_reporters.language_model_metric_reporter.**LanguageModelMetricReporter**(*ch*

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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

classmethod from_config(*config*, *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*)

Bases: *pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*)

pytext.metric_reporters.metric_reporter module

class pytext.metric_reporters.metric_reporter.**MetricReporter**(*channels*)
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.

Config
alias of *MetricReporter.Config*

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*)

classmethod aggregate_data(*all_data*, *new_batch*)
Aggregate a batch of data, basically just convert tensors to list of native python data

aggregate_preds(*new_batch*)

aggregate_scores(*new_batch*)

aggregate_targets(*new_batch*)

batch_context(*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()
Generate any extra intermediate context data for metric calculation

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

lower_is_better = False

realtime_report_freq = 500

report_metric(*model*, *stage*, *epoch*, *reset=True*, *print_to_channels=True*)
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()

pytext.metric_reporters.pairwise_ranking_metric_reporter module

class pytext.metric_reporters.pairwise_ranking_metric_reporter.**PairwiseRankingMetricReporte**
Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*)

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*)
Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of *RegressionMetricReporter.Config*

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.word_tagging_metric_reporter module

class pytext.metric_reporters.word_tagging_metric_reporter.**SimpleWordTaggingMetricReporter**

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

batch_context(*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.**WordTaggingMetricReporter**(*label_na*
List[str]
use_bio
bool,
chan-
nels:
List[pyt

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.**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'>)*)

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*, *meta*, **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
- **meta** (*Dict[str, Any]*) – global metadata, such as target names

class pytext.metric_reporters.**MetricReporter**(*channels*)
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.

Config
alias of *MetricReporter.Config*

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*)

classmethod aggregate_data(*all_data*, *new_batch*)
Aggregate a batch of data, basically just convert tensors to list of native python data

aggregate_preds(*new_batch*)

aggregate_scores(*new_batch*)

aggregate_targets(*new_batch*)

batch_context(*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()
Generate any extra intermediate context data for metric calculation

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

lower_is_better = False

realtime_report_freq = 500

report_metric(*model*, *stage*, *epoch*, *reset=True*, *print_to_channels=True*)
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()

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*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of *ClassificationMetricReporter.Config*

UTTERANCE_COLUMN = 'raw_text'

batch_context(*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

class pytext.metric_reporters.**RegressionMetricReporter**(*channels*)
Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of *RegressionMetricReporter.Config*

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]*)
Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

aggregate_preds(*new_batch*)

aggregate_scores(*new_batch*)

aggregate_targets(*new_batch*)

calculate_metric()
Calculate metrics, each sub class should implement it

classmethod from_config(*config*, *meta: pytext.data.data_handler.CommonMetadata*)

gen_extra_context()
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

process_pred(*pred: List[int]*) → List[str]
pred is a list of token label index

class pytext.metric_reporters.**LanguageModelMetricReporter**(*channels*)
Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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

classmethod from_config(*config*, *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.**WordTaggingMetricReporter**(*label_names: List[str]*, *use_bio_labels: bool*, *channels: List[pytext.metric_reporters.channel.Channel]*)
Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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]*)
Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

calculate_metric()
Calculate metrics, each sub class should implement it

create_frame_prediction_pairs()

classmethod from_config(*config*, *metadata: pytext.data.data_handler.CommonMetadata*)

gen_extra_context()
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

static tree_from_tokens_and_indx_actions(*token_str_list: List[str], actions_vocab: List[str], actions_indices: List[int]*)

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*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*)

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.**SimpleWordTaggingMetricReporter**(*label_names*, *channels*)

Bases: *pytext.metric_reporters.metric_reporter.MetricReporter*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

batch_context(*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

pytext.metrics package

Submodules

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*)
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*

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.FramePred 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*) → 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* → Tuple[float, int]

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_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

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() → None

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.**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.**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

pytext.metrics.**RECALL_AT_PRECISION_THREHOLDS = [0.2, 0.4, 0.6, 0.8, 0.9]**
Basic metric classes and functions for single-label prediction problems.

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

recall_at_precision
Alias for field number 1

roc_auc
Alias for field number 2

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_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]*) → 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

Returns ClassificationMetrics which contains various classification metrics.

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_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_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]*) → Dict[str, pytext.metrics.SoftClassificationMetrics]

Computes soft classification metrics (for now, average precision) 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

Returns Dict from label strings to their corresponding soft metrics.

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 condifence 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

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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

Config
alias of *IntentSlotModelDecoder.Config*

forward(*x_d: torch.Tensor, x_w: torch.Tensor, dense: Optional[torch.Tensor] = None*) → 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 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]

Config
alias of *MLPDecoder.Config*

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

Config
alias of *MLPDecoderQueryResponse.Config*

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]*)

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

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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]

Config
alias of *MLPDecoder.Config*

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

Config
alias of *IntentSlotModelDecoder.Config*

forward(*x_d: torch.Tensor*, *x_w: torch.Tensor*, *dense: Optional[torch.Tensor] = None*) → 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 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

Config
alias of *pytext.config.field_config.CharFeatConfig*

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*, *labels: Optional[pytext.data.utils.Vocabulary] = 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**(*embedding_dim: int*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for providing token embeddings from a pretrained model.

Config
alias of *pytext.config.field_config.ContextualTokenEmbeddingConfig*

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*, **args*, ***kwargs*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*, `torch.nn.modules.sparse.Embedding`

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

Config
alias of *pytext.config.field_config.DictFeatConfig*

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.
- **weights** (*torch.Tensor*) – Batch of sentences with dictionary feature
- **for the dictionary features.** (*weights*) –
- **lengths** (*torch.Tensor*) – Batch of sentences with the number of
- **features per token.** (*dictionary*) –

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*)
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

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

get_param_groups_for_optimizer() → List[Dict[str, torch.nn.parameter.Parameter]]
Organize module parameters into param_groups (or layers), so the optimizer and / or schedulers can have custom behavior per layer.

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

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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]]

get_param_groups_for_optimizer() → List[Dict[str, torch.nn.parameter.Parameter]]
Organize child embedding parameters into param_groups (or layers), so the optimizer and / or schedulers can have custom behavior per layer. The param_groups from each child embedding are aligned at the first (lowest) param_group.

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*, *unk_token_idx: int = 0*, *mlp_layer_dims: List[int] = ()*)

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.

Config
alias of *pytext.config.field_config.WordFeatConfig*

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*)

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

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

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

get_param_groups_for_optimizer() → List[Dict[str, torch.nn.parameter.Parameter]]

Organize module parameters into param_groups (or layers), so the optimizer and / or schedulers can have custom behavior per layer.

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

Config
alias of `pytext.config.component.ComponentMeta.__new__.<locals>.Config`

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]]

get_param_groups_for_optimizer() → List[Dict[str, torch.nn.parameter.Parameter]]
Organize child embedding parameters into param_groups (or layers), so the optimizer and / or schedulers can have custom behavior per layer. The param_groups from each child embedding are aligned at the first (lowest) param_group.

class pytext.models.embeddings.**WordEmbedding**(*num_embeddings: int*, *embedding_dim: int = 300*, *embeddings_weight: Optional[torch.Tensor] = None*, *init_range: Optional[List[int]] = None*, *unk_token_idx: int = 0*, *mlp_layer_dims: List[int] = ()*)
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.

Config
alias of *pytext.config.field_config.WordFeatConfig*

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*)
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

class pytext.models.embeddings.**DictEmbedding**(*num_embeddings: int*, *embed_dim: int*, *pooling_type: pytext.config.module_config.PoolingType*, **args*, ***kwargs*)
Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*, `torch.nn.modules.sparse.Embedding`

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

Config
alias of *pytext.config.field_config.DictFeatConfig*

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.
- **weights** (*torch.Tensor*) – Batch of sentences with dictionary feature
- **for the dictionary features.** (*weights*) –
- **lengths** (*torch.Tensor*) – Batch of sentences with the number of
- **features per token.** (*dictionary*) –

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*)
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

Config
alias of *pytext.config.field_config.CharFeatConfig*

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*, *labels: Optional[pytext.data.utils.Vocabulary] = 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**(*embedding_dim: int*)

Bases: *pytext.models.embeddings.embedding_base.EmbeddingBase*

Module for providing token embeddings from a pretrained model.

Config

alias of *pytext.config.field_config.ContextualTokenEmbeddingConfig*

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.ensembles package

Submodules

pytext.models.ensembles.bagging_doc_ensemble module

class pytext.models.ensembles.bagging_doc_ensemble.**BaggingDocEnsemble**(*config: py-text.models.ensembles.ensemb mod-els: List[pytext.models.model.Mod *args, **kwargs*)

Bases: *pytext.models.ensembles.ensemble.Ensemble*

Ensemble class that uses bagging for ensembling document classification models.

Config

alias of *BaggingDocEnsemble.Config*

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.**BaggingIntentSlotEnsemble**(*config: py-text.mod mod-els: List[pyt *args, **kwarg

Bases: *pytext.models.ensembles.ensemble.Ensemble*

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

Config
alias of *BaggingIntentSlotEnsemble.Config*

forward(**args*, ***kwargs*) → 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

merge_sub_models() → None
Merges all sub-models' transition matrices when using CRF. Otherwise does nothing.

pytext.models.ensembles.ensemble module

class pytext.models.ensembles.ensemble.**Ensemble**(*config: pytext.models.ensembles.ensemble.Ensemble.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]

Config
alias of *Ensemble.Config*

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.Ensemble.Config*, *feat_config: pytext.config.field_config.FeatureConfig*, **args*, ***kwargs*)
Factory method to construct an instance of Ensemble or one its derived classes from the module's config object and the field's metadata object. It creates sub-models in the ensemble using the sub-model's configuration.

Parameters

- **config** (*Config*) – Configuration object specifying all the parameters of Ensemble.
- **feat_config** (*FeatureConfig*) – Configuration object specifying all the parameters of the input features to the model.

Returns An instance of Ensemble.

Return type type

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 "".

Module contents

class pytext.models.ensembles.**BaggingDocEnsemble**(*config: pytext.models.ensembles.ensemble.Ensemble.Config*, *models: List[pytext.models.model.Model]*, **args*, ***kwargs*)
Bases: *pytext.models.ensembles.ensemble.Ensemble*

Ensemble class that uses bagging for ensembling document classification models.

Config
alias of *BaggingDocEnsemble.Config*

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.**BaggingIntentSlotEnsemble**(*config: pytext.models.ensembles.bagging_intent_slot_ensemb models: List[pytext.models.model.Model], *args, **kwargs*)
Bases: *pytext.models.ensembles.ensemble.Ensemble*

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

Config
alias of *BaggingIntentSlotEnsemble.Config*

forward(**args*, ***kwargs*) → 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

merge_sub_models() → None
Merges all sub-models' transition matrices when using CRF. Otherwise does nothing.

pytext.models.language_models package

Submodules

pytext.models.language_models.lmlstm module

class pytext.models.language_models.lmlstm.**LMLSTM**(**inputs*)
Bases: *pytext.models.model.Model*

LMLSTM implements a word-level language model that uses LSTMs to represent the document.

Config
alias of *LMLSTM.Config*

forward(*tokens*, **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(*model_config*, *feat_config*, *metadata: pytext.data.data_handler.CommonMetadata*)
Factory method to construct an instance of LMLSTM from the module's config object and the field's metadata object.

Parameters

- **config** (*LMLSTM.Config*) – Configuration object specifying all the parameters of LMLSTM.
- **feat_config** (*FeatureConfig*) – Configuration object specifying all the parameters of all input features.
- **metadata** (*FieldMeta*) – Object containing this field's metadata.

Returns An instance of *LMLSTM*.

Return type type

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]

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.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*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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.

Config
alias of *ClassificationOutputLayer.Config*

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

Bases: torch.jit.ScriptModule

class pytext.models.output_layers.doc_classification_output_layer.**MulticlassOutputLayer**(*targ*
Op-
tion
=
Non
loss
Op-
tion
=
Non
**ar*
***k*

Bases: *pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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.**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.

Config
alias of *RegressionOutputLayer.Config*

classmethod from_config(*config: pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer.Co*

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]

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

Config
alias of *IntentSlotOutputLayer.Config*

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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_meta: pytext.fields.field.FieldMeta, word_meta: pytext.fields.field.FieldMeta*)

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 by the *JointModelDataHandler*. 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 by the *JointModelDataHandler*. Defaults to None.

Returns Model prediction and scores.

Return type Tuple[torch.Tensor, torch.Tensor]

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.

Config

alias of *LMOutputLayer.Config*

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(*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 *LMLSTM*.
- **targets** (*torch.Tensor*) – True words.
- **context** (*Dict[str, Any]*) – Context is a dictionary of items that's passed as additional metadata by the *LanguageModelDataHandler* or *BPTTLanguageModelDataHandler*.

Returns Model prediction and scores.

Return type Tuple[torch.Tensor, torch.Tensor]

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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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 pre-

dict_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*

Config
alias of *PairwiseRankingOutputLayer.Config*

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.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.word_tagging_output_layer module

class pytext.models.output_layers.word_tagging_output_layer.**CRFOutputLayer**(*num_tags*, **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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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, 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*)
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 by the *JointModelDataHandler*. 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 by the *JointModelDataHandler*. Defaults to None.

Returns Model prediction and scores.

Return type Tuple[torch.Tensor, torch.Tensor]

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.

Config
alias of *WordTaggingOutputLayer.Config*

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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*
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: 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 by the *JointModelDataHandler*. 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]

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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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*, **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.

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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*, *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*)
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 by the *JointModelDataHandler*. 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 by the *JointModelDataHandler*. Defaults to None.

Returns Model prediction and scores.

Return type Tuple[torch.Tensor, torch.Tensor]

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.

Config
alias of *ClassificationOutputLayer.Config*

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.

Config
alias of *RegressionOutputLayer.Config*

classmethod from_config(*config: pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer.Co*

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]

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.

Config
alias of *WordTaggingOutputLayer.Config*

export_to_caffe2(*workspace: <module 'caffe2.python.workspace' from '/home/docs/checkouts/readthedocs.org/user_builds/pytext-pytext/envs/latest/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 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: 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 by the *JointModelDataHandler*. 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]

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*

Config
alias of *PairwiseRankingOutputLayer.Config*

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.**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.representations package

Submodules

pytext.models.representations.augmented_lstm module

class pytext.models.representations.augmented_lstm.**AugmentedLSTM**(*config: pytext.models.representations.augmented*
input_size: 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_rate controlls the level of variational dropout done.

Parameters

- **config** (*Config*) – Configuration object of type BiLSTM.Config.
- **input_size** (*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

Config
alias of *AugmentedLSTM.Config*

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**(*input_size: int*, *hidden_size: int*, *go_forward: bool = True*, *recurrent_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_rate controlls the level of variational dropout done.

Parameters

- **input_size** (*int*) – The number of expected features in the input.
- **hidden_size** (*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

Config
alias of *BiLSTM.Config*

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.represent*
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

Config
alias of *BiLSTMDocAttention.Config*

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: py-text.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

Config
alias of *BiLSTMDocSlotAttention.Config*

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*, *embed_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

Config
alias of *BiLSTMSlotAttention.Config*

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.

Config
alias of *BSeqCNNRepresentation.Config*

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.

---

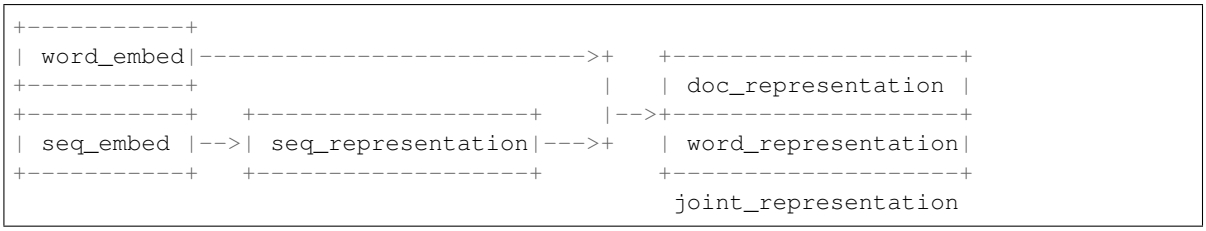
pytext.models.representations.contextual_intent_slot_rep module

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.



Config
alias of *ContextualIntentSlotRepresentation.Config*

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.docnn module

class pytext.models.representations.docnn.**DocNNRepresentation**(*config: pytext.models.representations.docnn.DocNN*
embed_dim: int)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

CNN based representation of a document.

Config
alias of *DocNNRepresentation.Config*

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.jointcnn_rep module

class pytext.models.representations.jointcnn_rep.**JointCNNRepresentation**(*config: pytext.models.representations.jointcnn_rep.JointCNNRepresentation.Config*, *embed_dim: int*)

Bases: *pytext.models.representations.representation_base.RepresentationBase*

Config
alias of *JointCNNRepresentation.Config*

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.pair_rep module

class pytext.models.representations.pair_rep.**PairRepresentation**(*config: pytext.models.representations.pair_rep.PairRepresentation.Config*, *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.

Config
alias of *PairRepresentation.Config*

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*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.BoundaryPool*, *n_input: int*)

Bases: *pytext.models.module.Module*

Config
alias of *BoundaryPool.Config*

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*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*

Config
alias of `pytext.config.component.ComponentMeta.__new__.<locals>.Config`

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*

Config
alias of `pytext.config.component.ComponentMeta.__new__.<locals>.Config`

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*

Config
alias of `pytext.config.component.ComponentMeta.__new__.<locals>.Config`

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*

Config
alias of *SelfAttention.Config*

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

Config
alias of *PureDocAttention.Config*

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.query_document_pairwise_ranking_rep module

class pytext.models.representations.query_document_pairwise_ranking_rep.**QueryDocumentPairw**

Bases: *pytext.models.representations.representation_base.RepresentationBase*

Wrapper representation for a query with 2 reponses. where model is trained on pairwise ranking loss first input: pos_response (higher ranked) second input: neg_response (lower ranked) third input: query both responses and the query use the same embeddings

Config
alias of *QueryDocumentPairwiseRankingRep.Config*

forward(*embeddings: Tuple[torch.Tensor, ...], *lengths*) → 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.representation_base module

class pytext.models.representations.representation_base.**RepresentationBase**(*config*)
Bases: *pytext.models.module.Module*

Config
alias of `pytext.config.component.ComponentMeta.__new__.<locals>.Config`

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

Config
alias of *SeqRepresentation.Config*

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*

Config
alias of *SlotAttention.Config*

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.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

Config
alias of *StackedBidirectionalRNN.Config*

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

Module contents

pytext.models.semantic_parsers package

Subpackages

pytext.models.semantic_parsers.rnng package

Submodules

pytext.models.semantic_parsers.rnng.rnng_data_structures module

class pytext.models.semantic_parsers.rnng.rnng_data_structures.**CompositionFunction**
Bases: torch.nn.modules.module.Module

Combines a list / sequence of embeddings into one

class pytext.models.semantic_parsers.rnng.rnng_data_structures.**CompositionalNN**(*lstm_dim: int*)

Bases: *pytext.models.semantic_parsers.rnng.rnng_data_structures.CompositionFunction*

Combines a list / sequence of embeddings into one using a biLSTM

forward(*x: List[torch.Tensor]*) → torch.Tensor

Embed the sequence. If the input corresponds to [IN:GL where am I at]: - x will contain the embeddings of [at I am where IN:GL] in that order. - Forward LSTM will embed the sequence [IN:GL where am I at]. - Backward LSTM will embed the sequence [IN:GL at I am where]. The final hidden states are concatenated and then projected.

Parameters **x** – Embeddings of the input tokens in *reversed* order

Shapes: x: (1, lstm_dim) each return value: (1, lstm_dim)

class pytext.models.semantic_parsers.rnng.rnng_data_structures.**CompositionalSummationNN**(*lstm*

int)

Bases: *pytext.models.semantic_parsers.rnng.rnng_data_structures.CompositionFunction*

Simpler version of CompositionalNN

forward(*x: List[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.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: pytext.models.semantic_parsers.rnng.rn*)

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)

Config
alias of *RNNGParser.Config*

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*, *metadata: pytext.data.data_handler.CommonMetadata*)

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_param_groups_for_optimizer()

This is called by code that looks for an instance of pytext.models.model.Model.

get_pred(*logits: List[Tuple[torch.Tensor, torch.Tensor]], *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.contextual_intent_slot module

class pytext.models.seq_models.contextual_intent_slot.**ContextualIntentSlotModel**(**args*, ***kwargs*)

Bases: *pytext.models.joint_model.JointModel*

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}}]}

Config
alias of *ContextualIntentSlotModel.Config*

classmethod compose_embedding(*sub_embs*, *metadata*)
Compose embedding list for ContextualIntentSlot model training. The first is the word embedding of the last utterance concatenated with the word level dictionary feature. The second is the word embedding of a sequence of utterances (includes the last utterance). Two embeddings are not concatenated and passed to the model individually.

Parameters **sub_embs** (*type*) – sub-embeddings.

Returns

EmbeddingList object contains embedding of the last utterance with dictionary feature and embedding of the sequence of utterances.

Return type type

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.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.

Config
alias of *SeqNNModel.Config*

Module contents

Submodules

pytext.models.crf module

class pytext.models.crf.**CRF**(*num_tags: 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.FloatTensor*, *seq_lens: torch.LongTensor*) → 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.FloatTensor*, *tags: torch.LongTensor*, *ignore_index=0*, *reduce: bool = True*) → torch.autograd.variable.Variable
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

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

arrange_model_inputs(*tensor_dict*)

arrange_targets(*tensor_dict*)

caffe2_export(*tensorizers*, *tensor_dict*, *path*, *export_onnx_path=None*)

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.**DocModel_Deprecated**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase*, *representation: pytext.models.representations.representation_base.Representat*, *decoder: pytext.models.decoders.decoder_base.DecoderBase*, *output_layer: pytext.models.output_layers.output_layer_base.OutputLayerBa*)
Bases: *pytext.models.model.Model*

An n-ary document classification model. It can be used for all text classification scenarios. It supports `PureDocAttention`, `BiLSTMDocAttention` and `DocNNRepresentation` as the ways to represent the document followed by multi-layer perceptron (`MLPDecoder`) for projecting the document representation into label/target space.

It can be instantiated just like any other `Model`.

DEPRECATED: Use NewDocModel instead

Config
alias of *DocModel_Deprecated.Config*

class pytext.models.doc_model.**NewDocModel**(*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.doc_model.DocModel_Deprecated*

DocModel that's compatible with the new Model abstraction, which is responsible for describing which inputs it expects and arranging its input tensors.

Config
alias of *NewDocModel.Config*

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.NewDocModel.Config*, *representation_dim: int*, *num_labels: int*)

classmethod create_embedding(*config: pytext.models.doc_model.NewDocModel.Config*, *tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

classmethod from_config(*config: pytext.models.doc_model.NewDocModel.Config*, *tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

get_export_input_names(*tensorizers*)

get_export_output_names(*tensorizers*)

torchscriptify(*tensorizers*, *traced_model*)

vocab_to_export(*tensorizers*)

class pytext.models.doc_model.**NewDocRegressionModel**(*embedding: pytext.models.embeddings.embedding_base.EmbeddingBase*, *representation: pytext.models.representations.representation_base.Represen*, *decoder: pytext.models.decoders.decoder_base.DecoderBase*, *output_layer: pytext.models.output_layers.output_layer_base.OutputLaye*)

Bases: *pytext.models.doc_model.NewDocModel*

Model that's compatible with the new Model abstraction, and is configured for regression tasks (specifically for labels, predictions, and loss).

Config
alias of *NewDocRegressionModel.Config*

classmethod from_config(*config: pytext.models.doc_model.NewDocRegressionModel.Config*, *tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

pytext.models.joint_model module

class pytext.models.joint_model.**JointModel**(**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`.

Config
alias of *JointModel.Config*

classmethod from_config(*model_config*, *feat_config*, *metadata: pytext.data.data_handler.CommonMetadata*)

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.

Config
alias of *BaseModel.Config*

SUPPORT_FP16_OPTIMIZER = False

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_param_groups_for_optimizer() → List[Dict[str, List[torch.nn.parameter.Parameter]]]
Returns a list of parameter groups of the format {"params": param_list}. The parameter groups loosely correspond to layers and are ordered from low to high. Currently, only the embedding layer can provide multiple param groups, and other layers are put into one param group. The output of this method is passed to the optimizer so that schedulers can change learning rates by layer.

get_pred(*logit*, *target=None*, *context=None*, **args*)

prepare_for_onnx_export_(***kwargs*)
Make model exportable via ONNX trace.

save_modules(*base_path: str = ''*, *suffix: str = ''*)
Save each sub-module in separate files for reusing later.

train(*mode=True*)
Override to explicitly maintain the stage (train, eval, test).

classmethod train_batch(*model*, *batch*)

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

Config
alias of *Model.Config*

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.

Config
alias of *pytext.config.module_config.ModuleConfig*

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.**BasePairwiseClassificationModel**(*representations: torch.nn.modules.c*
de-
coder:
py-
text.models.decode
out-
put_layer:
py-
text.models.output
Bases: *pytext.models.model.BaseModel*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

save_modules(*base_path: str = ''*, *suffix: str = ''*)
Save each sub-module in separate files for reusing later.

class pytext.models.pair_classification_model.**PairClassificationModel**(*embedding: pytext.models.embeddings.embed representation: pytext.models.representations.rep decoder: pytext.models.decoders.decoder_ output_layer: pytext.models.output_layers.outp*

Bases: *pytext.models.model.Model*

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`.

Config
alias of *PairClassificationModel.Config*

classmethod compose_embedding(*sub_embs*, *metadata*)
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

save_modules(*base_path: str = ''*, *suffix: str = ''*)
Save each sub-module in separate files for reusing later.

class pytext.models.pair_classification_model.**PairwiseClassificationModel**(*embeddings: torch.nn.modules.contai*
rep-
re-
sen-
ta-
tions:
torch.nn.modules.contai
de-
coder:
py-
text.models.decoders.ml
out-
put_layer:
py-
text.models.output_layer
en-
code_relations:
bool)

Bases: *pytext.models.pair_classification_model.BasePairwiseClassificationModel*

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`.

Config
alias of *PairwiseClassificationModel.Config*

arrange_model_inputs(*tensor_dict*)

arrange_targets(*tensor_dict*)

forward(*tokens: List[torch.Tensor], seq_lens: List[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.PairwiseClassificationModel.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

pytext.models.query_document_pairwise_ranking_model module

class pytext.models.query_document_pairwise_ranking_model.**QueryDocumentPairwiseRankingModel**

Bases: *pytext.models.model.Model*

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

Config
alias of *QueryDocumentPairwiseRankingModel.Config*

classmethod compose_embedding(*sub_embs*, *metadata*)
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_sub_embs(*emb_config: pytext.config.query_document_pairwise_ranking.ModelInputConfig*, *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*, *feat_config*, *metadata: pytext.data.data_handler.CommonMetadata*)

save_modules(*base_path: str = ''*, *suffix: str = ''*)
Save each sub-module in separate files for reusing later.

pytext.models.word_model module

class pytext.models.word_model.**NewWordTaggingModel**(**args*, ***kwargs*)
Bases: *pytext.models.model.Model*

Config
alias of *NewWordTaggingModel.Config*

arrange_model_inputs(*tensor_dict*)

arrange_targets(*tensor_dict*)

classmethod create_embedding(*config*, *tensorizers*)

classmethod from_config(*config*, *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`.

Config
alias of *WordTaggingModel.Config*

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

Config
alias of *Model.Config*

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.

Config

alias of *BaseModel.Config*

SUPPORT_FP16_OPTIMIZER = False

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_param_groups_for_optimizer () → List[Dict[str, List[torch.nn.parameter.Parameter]]]
Returns a list of parameter groups of the format {"params": param_list}. The parameter groups loosely correspond to layers and are ordered from low to high. Currently, only the embedding layer can provide multiple param groups, and other layers are put into one param group. The output of this method is passed to the optimizer so that schedulers can change learning rates by layer.

get_pred (*logit*, *target=None*, *context=None*, **args*)

prepare_for_onnx_export_ (***kwargs*)
Make model exportable via ONNX trace.

save_modules (*base_path: str = ''*, *suffix: str = ''*)
Save each sub-module in separate files for reusing later.

train (*mode=True*)
Override to explicitly maintain the stage (train, eval, test).

classmethod train_batch (*model*, *batch*)

pytext.optimizer package

Submodules

pytext.optimizer.scheduler module

class pytext.optimizer.scheduler.**BatchScheduler** (*config=None*, **args*, ***kwargs*)
Bases: *pytext.optimizer.scheduler.Scheduler*

Config
alias of `pytext.config.component.ComponentMeta.__new__.<locals>.Config`

prepare (*train_iter*, *total_epochs*)

class pytext.optimizer.scheduler.**CosineAnnealingLR** (*optimizer*, *T_max*, *eta_min=0*, *last_epoch=-1*)
Bases: `torch.optim.lr_scheduler.CosineAnnealingLR`, *pytext.optimizer.scheduler.BatchScheduler*

Wrapper around *torch.optim.lr_scheduler.CosineAnnealingLR* See the original documentation for more details.

Config
alias of *CosineAnnealingLR.Config*

classmethod from_config (*config: pytext.optimizer.scheduler.CosineAnnealingLR.Config*, *optimizer: pytext.optimizer.Optimizer*)

step_batch (*metrics=None*, *epoch=None*)

class pytext.optimizer.scheduler.**ExponentialLR** (*optimizer*, *gamma*, *last_epoch=-1*)
Bases: `torch.optim.lr_scheduler.ExponentialLR`, *pytext.optimizer.scheduler.Scheduler*

Wrapper around *torch.optim.lr_scheduler.ExponentialLR* See the original documentation for more details.

Config
alias of *ExponentialLR.Config*

classmethod from_config(*config: pytext.optimizer.scheduler.ExponentialLR.Config*, *optimizer: pytext.optimizer.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.

Config
alias of *LmFineTuning.Config*

classmethod from_config(*config: pytext.optimizer.scheduler.LmFineTuning.Config*, *optimizer*)

get_lr()

step_batch(*metrics=None*, *epoch=None*)

class pytext.optimizer.scheduler.**ReduceLROnPlateau**(*optimizer*, *mode='min'*, *factor=0.1*, *patience=10*, *verbose=False*, *threshold=0.0001*, *threshold_mode='rel'*, *cooldown=0*, *min_lr=0*, *eps=1e-08*)
Bases: torch.optim.lr_scheduler.ReduceLROnPlateau, *pytext.optimizer.scheduler.Scheduler*

Wrapper around *torch.optim.lr_scheduler.ReduceLROnPlateau* See the original documentation for more details.

Config
alias of *ReduceLROnPlateau.Config*

classmethod from_config(*config: pytext.optimizer.scheduler.ReduceLROnPlateau.Config*, *optimizer: pytext.optimizer.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.

Config
alias of *Scheduler.Config*

prepare(*train_iter*, *total_epochs*)

step_batch(***kwargs*) → None

step_epoch(***kwargs*) → None

class pytext.optimizer.scheduler.**StepLR**(*optimizer*, *step_size*, *gamma=0.1*, *last_epoch=-1*)
Bases: torch.optim.lr_scheduler.StepLR, *pytext.optimizer.scheduler.Scheduler*

Wrapper around *torch.optim.lr_scheduler.StepLR* See the original documentation for more details.

Config
alias of *StepLR.Config*

classmethod from_config(*config: pytext.optimizer.scheduler.StepLR.Config*, *optimizer*)

step_epoch(*metrics=None*, *epoch=None*)

class pytext.optimizer.scheduler.**WarmupScheduler**(*optimizer*, *warmup_steps*)
Bases: torch.optim.lr_scheduler._LRScheduler, *pytext.optimizer.scheduler.BatchScheduler*

Scheduler to linearly increase learning rate from 0 to final value at the beginning of training.

Config
alias of *WarmupScheduler.Config*

classmethod from_config(*config: pytext.optimizer.scheduler.WarmupScheduler.Config*, *optimizer: pytext.optimizer.Optimizer*)

get_lr()

prepare(*train_iter*, *total_epochs*)

step_batch()

Module contents

class pytext.optimizer.**Adam**(*parameters*, *lr*, *weight_decay*)
Bases: torch.optim.adam.Adam, *pytext.optimizer.Optimizer*

Config
alias of *Adam.Config*

classmethod from_config(*config: pytext.optimizer.Adam.Config*, *model: torch.nn.modules.module.Module*, **args*, ***kwargs*)

class pytext.optimizer.**Optimizer**(*config=None*, **args*, ***kwargs*)
Bases: *pytext.config.component.Component*

Config
alias of *Optimizer.Config*

class pytext.optimizer.**SGD**(*parameters*, *lr*, *momentum*)
Bases: torch.optim.sgd.SGD, *pytext.optimizer.Optimizer*

Config
alias of *SGD.Config*

classmethod from_config(*config: pytext.optimizer.SGD.Config*, *model: torch.nn.modules.module.Module*, **args*, ***kwargs*)

pytext.optimizer.**learning_rates**(*optimizer*)

pytext.task package

Submodules

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.

Config
alias of *DisjointMultitask.Config*

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*, *metadata=None*, *model_state=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.task.new_task.NewTaskTrainer] = 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.

Config
alias of *NewDisjointMultitask.Config*

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*, *metadata=None*, *model_state=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*)

pytext.task.new_task module

class pytext.task.new_task.**NewDocumentClassification**(*data: pytext.data.data.Data*, *model: pytext.models.model.BaseModel*, *metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricRe = None*, *trainer: Optional[pytext.task.new_task.NewTaskTrainer] = None*)
Bases: *pytext.task.new_task.NewTask*

Config
alias of *NewDocumentClassification.Config*

class pytext.task.new_task.**NewDocumentRegression**(*data: pytext.data.data.Data*, *model: pytext.models.model.BaseModel*, *metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporte = None*, *trainer: Optional[pytext.task.new_task.NewTaskTrainer] = None*)
Bases: *pytext.task.new_task.NewTask*

Config
alias of *NewDocumentRegression.Config*

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.task.new_task.NewTaskTrainer] = None*)

Bases: pytext.task.new_task._NewTask

Config
alias of *NewTask.Config*

class pytext.task.new_task.**NewTaskTrainer**(*config: pytext.trainers.trainer.Trainer.Config*, *model: torch.nn.modules.module.Module*)

Bases: *pytext.trainers.trainer.Trainer*

Config
alias of *NewTaskTrainer.Config*

run_epoch(*state: pytext.trainers.trainer.TrainingState*, *data*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)
Our run_epoch is a bit different, because we're wrapping the model forward call with model.train_batch, which arranges tensors and gets loss, etc.

class pytext.task.new_task.**PairwiseClassificationTask**(*data: pytext.data.data.Data*, *model: pytext.models.model.BaseModel*, *metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricR = None*, *trainer: Optional[pytext.task.new_task.NewTaskTrainer] = None*)

Bases: *pytext.task.new_task.NewDocumentClassification*

Config
alias of *PairwiseClassificationTask.Config*

pytext.task.serialize module

pytext.task.serialize.**load**(*load_path: str*)
Load task, will construct the task using the saved config then load metadata and model state.

pytext.task.serialize.**save**(*config: pytext.config.pytext_config.PyTextConfig*, *model: pytext.models.model.Model*, *meta: pytext.data.data_handler.CommonMetadata*) → None
Save a task, will save the original config, model state and metadata

pytext.task.task module

class pytext.task.task.**Task**(*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*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.

Config
alias of *TaskBase.Config*

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*, *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*, *dist_init_url=''*)
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.task.**create_task**(*task_config*, *metadata=None*, *model_state=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.**ContextualIntentSlotTask**(*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.Task*

Config

alias of *ContextualIntentSlotTask.Config*

classmethod example_config()

class pytext.task.tasks.**DocClassificationTask**(*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.Task*

Config

alias of *DocClassificationTask.Config*

classmethod format_prediction(*predictions*, *scores*, *context*, *target_meta*)

Format the prediction and score from model output, by default just return them in a dict

class pytext.task.tasks.**EnsembleTask**(*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.Task*

Config

alias of *EnsembleTask.Config*

classmethod example_config()

train_single_model(*train_config*, *model_id*, *rank=0*, *world_size=1*, *dist_init_url=''*)

class pytext.task.tasks.**JointTextTask**(*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.Task*

Config
alias of *JointTextTask.Config*

classmethod example_config()

classmethod format_prediction(*predictions*, *scores*, *context*, *target_meta*)
Format the prediction and score from model output, by default just return them in a dict

class pytext.task.tasks.**LMTask**(*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.Task*

Config
alias of *LMTask.Config*

class pytext.task.tasks.**NewWordTaggingTask**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.task.new_task.NewTaskTrainer] = None*)
Bases: *pytext.task.new_task.NewTask*

Config
alias of *NewWordTaggingTask.Config*

classmethod create_metric_reporter(*config: pytext.task.tasks.NewWordTaggingTask.Config, tensorizers: Dict[str, pytext.data.tensorizers.Tensorizer]*)

class pytext.task.tasks.**PairClassificationTask**(*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.Task*

Config
alias of *PairClassificationTask.Config*

class pytext.task.tasks.**QueryDocumentPairwiseRankingTask**(*trainer: pytext.trainers.trainer.Trainer, data_handler: pytext.data.data_handler.DataHandler, model: pytext.models.model.Model, metric_reporter: pytext.metric_reporters.metric_reporter.MetricRepo exporter: Optional[pytext.exporters.exporter.ModelExporter]*)
Bases: *pytext.task.task.Task*

Config
alias of *QueryDocumentPairwiseRankingTask.Config*

class pytext.task.tasks.**SemanticParsingTask**(*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.Task*

Config
alias of *SemanticParsingTask.Config*

class pytext.task.tasks.**SeqNNTask**(*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.Task*

Config
alias of *SeqNNTask.Config*

class pytext.task.tasks.**WordTaggingTask**(*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.Task*

Config
alias of *WordTaggingTask.Config*

classmethod format_prediction(*predictions*, *scores*, *context*, *target_meta*)
Format the prediction and score from model output, by default just return them in a dict

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.task.new_task.NewTaskTrainer] = None*)
Bases: pytext.task.new_task._NewTask

Config
alias of *NewTask.Config*

class pytext.task.**NewDocumentClassification**(*data: pytext.data.data.Data, model: pytext.models.model.BaseModel, metric_reporter: Optional[pytext.metric_reporters.metric_reporter.MetricReporter] = None, trainer: Optional[pytext.task.new_task.NewTaskTrainer] = None*)

Bases: *pytext.task.new_task.NewTask*

Config
alias of *NewDocumentClassification.Config*

class pytext.task.**Task**(*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*

Config
alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

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.

Config
alias of *TaskBase.Config*

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*, *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*, *dist_init_url=''*)
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: pytext.data.data_handler.CommonMetadata*) → None
Save a task, will save the original config, model state and metadata

pytext.task.**load**(*load_path: str*)
Load task, will construct the task using the saved config then load metadata and model state.

pytext.task.**create_task**(*task_config*, *metadata=None*, *model_state=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.trainers package

Submodules

pytext.trainers.ensemble_trainer module

class pytext.trainers.ensemble_trainer.**EnsembleTrainer**(*real_trainer*)
Bases: *pytext.trainers.trainer.TrainerBase*

Trainer for ensemble models

real_trainer
the actual trainer to run

Type Trainer

Config
alias of *EnsembleTrainer.Config*

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*

Config
alias of *HogwildTrainer.Config*

classmethod from_config(*config: pytext.trainers.hogwild_trainer.HogwildTrainer.Config*, *model: torch.nn.modules.module.Module*, **args*, ***kwargs*)

run_epoch(*state: pytext.trainers.trainer.TrainingState*, *data_iter: torchtext.data.iterator.Iterator*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

set_up_training(*state: pytext.trainers.trainer.TrainingState*, *training_data*)

pytext.trainers.trainer module

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

Config
alias of *Trainer.Config*

backprop(*state*, *loss*)

continue_training(*state: pytext.trainers.trainer.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.trainer.TrainingState*)

run_epoch(*state: pytext.trainers.trainer.TrainingState*, *data: pytext.data.data_handler.BatchIterator*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

save_checkpoint(*state: pytext.trainers.trainer.TrainingState*, *train_config: pytext.config.pytext_config.PyTextConfig*)

set_up_training(*state: pytext.trainers.trainer.TrainingState*, *training_data: pytext.data.data_handler.BatchIterator*)

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. 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

- **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
- **training_result** (*Optional*) – only meaningful for Hogwild training. default is None
- **rank** (*int*) – only used in distributed training, the rank of the current training thread, evaluation will only be done in rank 0

Returns the trained model together with the best metric

Return type model, best_metric

update_best_model(*state: pytext.trainers.trainer.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*

Config

alias of pytext.config.component.ComponentMeta.__new__.<locals>.Config

class pytext.trainers.trainer.**TrainingState**(***kwargs*)

Bases: object

best_model_metric = None

best_model_state = None

epoch = 0

epochs_since_last_improvement = 0

rank = 0

stage = 'Training'

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

Config
alias of *Trainer.Config*

backprop(*state*, *loss*)

continue_training(*state: pytext.trainers.trainer.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.trainer.TrainingState*)

run_epoch(*state: pytext.trainers.trainer.TrainingState*, *data: pytext.data.data_handler.BatchIterator*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

save_checkpoint(*state: pytext.trainers.trainer.TrainingState*, *train_config: pytext.config.pytext_config.PyTextConfig*)

set_up_training(*state: pytext.trainers.trainer.TrainingState*, *training_data: pytext.data.data_handler.BatchIterator*)

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. 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

- **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
- **training_result** (*Optional*) – only meaningful for Hogwild training. default is None
- **rank** (*int*) – only used in distributed training, the rank of the current training thread, evaluation will only be done in rank 0

Returns the trained model together with the best metric

Return type model, best_metric

update_best_model(*state: pytext.trainers.trainer.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'

class pytext.trainers.**EnsembleTrainer**(*real_trainer*)

Bases: *pytext.trainers.trainer.TrainerBase*

Trainer for ensemble models

real_trainer
the actual trainer to run

Type Trainer

Config
alias of *EnsembleTrainer.Config*

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*

Config

alias of *HogwildTrainer.Config*

classmethod from_config(*config: pytext.trainers.hogwild_trainer.HogwildTrainer.Config*, *model: torch.nn.modules.module.Module*, **args*, ***kwargs*)

run_epoch(*state: pytext.trainers.trainer.TrainingState*, *data_iter: torchtext.data.iterator.Iterator*, *metric_reporter: pytext.metric_reporters.metric_reporter.MetricReporter*)

set_up_training(*state: pytext.trainers.trainer.TrainingState*, *training_data*)

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.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.**tensor**(*data*, *dtype*)

pytext.utils.cuda.**var_to_numpy**(*v*)

pytext.utils.cuda.**xaviervar**(**size*)

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'

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.**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_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*)

pytext.utils.distributed module

pytext.utils.distributed.**dist_init**(*distributed_rank: int*, *world_size: int*, *init_method: str*, *backend: str = 'nccl'*)

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.**eprint**(**args*, ***kwargs*)

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_klass=<class 'pytext.config.component.Component'>*)

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*)
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

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*) → 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).

pytext.utils.embeddings.**append_dialect**(*word: str*, *dialect: str*) → str

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.model module

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.precision module

pytext.utils.precision.**activate**(*model*)

pytext.utils.precision.**backward**(*optimizer*, *loss*)

pytext.utils.precision.**clip_grad_norm**(*model*, *optimizer*, *max_clip_norm*)

pytext.utils.precision.**deactivate**(*model*)

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.precision.**unwrap_optimizer**(*wrapped_optimizer*)

pytext.utils.precision.**wrap_optimizer**(*optimizer*)

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()

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*)
Bases: object

add(*time*)

average

pytext.utils.timing.**format_time**(*seconds*)

pytext.utils.timing.**report_snapshot**(*fn*)

pytext.utils.torch module

class pytext.utils.torch.**Vocabulary**(*vocab_list*, *unk_idx: int = 0*)
Bases: torch.jit.ScriptModule

Module contents

pytext.utils.**cls_vars**(*cls*)

pytext.utils.**set_random_seeds**(*seed*)

1.15.2 Submodules

1.15.3 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**()

1.15.4 pytext.main module

class pytext.main.**Attrs**
Bases: object

pytext.main.**gen_config_impl**(*task_name*, *options*)

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]]*)

1.15.5 pytext.workflow module

pytext.workflow.**batch_predict**(*model_file: str, examples: List[Dict[str, Any]]*)

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*) → 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*)

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*) → pytext.task.task.Task

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.**save_and_export**(*config: pytext.config.pytext_config.PyTextConfig, task: pytext.task.task.Task, metric_channels: Optional[List[pytext.metric_reporters.channel.Channel]] = None*) → 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

1.15.6 Module contents

pytext.**create_predictor**(*config: pytext.config.pytext_config.PyTextConfig*, *model_file: Optional[str] = None*) → 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, 412
pytext.builtin_task, 410
pytext.common, 195
pytext.common.constants, 193
pytext.config, 202
pytext.config.component, 195
pytext.config.config_adapter, 197
pytext.config.contextual_intent_slot, 197
pytext.config.doc_classification, 198
pytext.config.field_config, 198
pytext.config.module_config, 200
pytext.config.pair_classification, 200
pytext.config.pytext_config, 200
pytext.config.query_document_pairwise_ranking, 201
pytext.config.serialize, 202
pytext.data, 252
pytext.data.batch_sampler, 220
pytext.data.bptt_lm_data_handler, 222
pytext.data.compositional_data_handler, 224
pytext.data.contextual_intent_slot_data_handler, 226
pytext.data.data, 228
pytext.data.data_handler, 230
pytext.data.data_structures, 205
pytext.data.data_structures.annotation, 203
pytext.data.data_structures.node, 205
pytext.data.disjoint_multitask_data, 234
pytext.data.disjoint_multitask_data_handler, 235
pytext.data.doc_classification_data_handler, 238
pytext.data.featurizer, 207
pytext.data.featurizer.featurizer, 206
pytext.data.featurizer.simple_featurizer, 207
pytext.data.joint_data_handler, 240
pytext.data.language_model_data_handler, 241
pytext.data.pair_classification_data_handler, 243
pytext.data.query_document_pairwise_ranking_data_ha
245
pytext.data.seq_data_handler, 246
pytext.data.sources, 213
pytext.data.sources.data_source, 209
pytext.data.sources.squad, 211
pytext.data.sources.tsv, 212
pytext.data.tensorizers, 247
pytext.data.test, 219
pytext.data.test.batch_sampler_test, 214
pytext.data.test.bptt_lm_data_handler_test, 214
pytext.data.test.compositional_datahandler_test, 214
pytext.data.test.contextual_intent_slot_data_handle
215
pytext.data.test.data_test, 215
pytext.data.test.datahandler_test, 216
pytext.data.test.doc_classification_data_handler_te
216
pytext.data.test.joint_data_handler_test, 216
pytext.data.test.kd_doc_classification_data_handler
216
pytext.data.test.language_model_data_handler_test, 217
pytext.data.test.query_document_pairwise_ranking_da
217
pytext.data.test.round_robin_batchiterator_test, 217
pytext.data.test.seq_data_handler_test, 217
pytext.data.test.simple_featurizer_test, 217

pytext.data.test.tensorizers_test, 218
pytext.data.test.tokenizers_test, 218
pytext.data.test.tsv_data_source_test, 218
pytext.data.test.utils_test, 219
pytext.data.tokenizers, 220
pytext.data.tokenizers.tokenizer, 219
pytext.data.utils, 251
pytext.exporters, 270
pytext.exporters.custom_exporters, 268
pytext.exporters.exporter, 268
pytext.fields, 277
pytext.fields.char_field, 272
pytext.fields.contextual_token_embedding_field, 273
pytext.fields.dict_field, 273
pytext.fields.field, 274
pytext.fields.text_field_with_special_unk, 276
pytext.loss, 282
pytext.loss.loss, 281
pytext.main, 411
pytext.metric_reporters, 296
pytext.metric_reporters.channel, 284
pytext.metric_reporters.classification_metric_reporter, 288
pytext.metric_reporters.compositional_metric_reporter, 289
pytext.metric_reporters.disjoint_multitask_metric_reporter, 290
pytext.metric_reporters.intent_slot_detection_metric_reporter, 291
pytext.metric_reporters.language_model_metric_reporter, 292
pytext.metric_reporters.metric_reporter, 293
pytext.metric_reporters.pairwise_ranking_metric_reporter, 294
pytext.metric_reporters.regression_metric_reporter, 295
pytext.metric_reporters.word_tagging_metric_reporter, 295
pytext.metrics, 307
pytext.metrics.intent_slot_metrics, 302
pytext.metrics.language_model_metrics, 307
pytext.models, 385
pytext.models.crf, 373
pytext.models.decoders, 318
pytext.models.decoders.decoder_base, 315
pytext.models.decoders.intent_slot_model_decoder, 316
pytext.models.decoders.mlp_decoder, 317
pytext.models.decoders.mlp_decoder_query_response, 318
pytext.models.disjoint_multitask_model, 374
pytext.models.distributed_model, 375
pytext.models.doc_model, 376
pytext.models.embeddings, 327
pytext.models.embeddings.char_embedding, 321
pytext.models.embeddings.contextual_token_embedding, 323
pytext.models.embeddings.dict_embedding, 323
pytext.models.embeddings.embedding_base, 324
pytext.models.embeddings.embedding_list, 325
pytext.models.embeddings.word_embedding, 326
pytext.models.ensembles, 334
pytext.models.ensembles.bagging_doc_ensemble, 332
pytext.models.ensembles.bagging_intent_slot_ensemble, 333
pytext.models.ensembles.ensemble, 333
pytext.models.joint_model, 377
pytext.models.language_models, 336
pytext.models.language_models.lmlstm, 335
pytext.models.model, 378
pytext.models.model_base, 380
pytext.models.output_layers, 347
pytext.models.output_layers.doc_classification_output, 337
pytext.models.output_layers.doc_regression_output_layer, 339
pytext.models.output_layers.intent_slot_output_layer, 340
pytext.models.output_layers.lm_output_layer, 341
pytext.models.output_layers.output_layer_base, 342
pytext.models.output_layers.pairwise_ranking_output, 344
pytext.models.output_layers.utils, 344
pytext.models.output_layers.word_tagging_output_layer, 344
pytext.models.pair_classification_model, 381
pytext.models.query_document_pairwise_ranking_model, 384
pytext.models.representations, 368
pytext.models.representations.augmented_lstm, 352

pytext.models.representations.bilstm, 355
pytext.models.representations.bilstm_doc_attention, 356
pytext.models.representations.bilstm_doc_slot_attention, 357
pytext.models.representations.bilstm_slot_attn, 359
pytext.models.representations.biseqcnn, 360
pytext.models.representations.contextual_intent_slot_rep, 360
pytext.models.representations.docnn, 361
pytext.models.representations.jointcnn_rep, 362
pytext.models.representations.pair_rep, 362
pytext.models.representations.pass_through, 363
pytext.models.representations.pooling, 363
pytext.models.representations.pure_doc_attention, 365
pytext.models.representations.query_document_pairwise_ranking_rep, 365
pytext.models.representations.representation_base, 366
pytext.models.representations.seq_rep, 366
pytext.models.representations.slot_attention, 367
pytext.models.representations.stacked_bidirectional_rnn, 367
pytext.models.semantic_parsers, 372
pytext.models.semantic_parsers.rnng, 372
pytext.models.semantic_parsers.rnng.rnng_data_structures, 368
pytext.models.semantic_parsers.rnng.rnng_parser, 370
pytext.models.seq_models, 373
pytext.models.seq_models.contextual_intent_slot, 372
pytext.models.seq_models.seqnn, 373
pytext.models.word_model, 385
pytext.optimizer, 390
pytext.optimizer.scheduler, 388
pytext.task, 397
pytext.task.disjoint_multitask, 391
pytext.task.new_task, 392
pytext.task.serialize, 393
pytext.task.task, 393
pytext.task.tasks, 395
pytext.trainers, 402
pytext.trainers.ensemble_trainer, 399
pytext.trainers.hogwild_trainer, 399
pytext.trainers.trainer, 400
pytext.utils, 410
pytext.utils.ascii_table, 404
pytext.utils.cuda, 404
pytext.utils.data, 404
pytext.utils.distributed, 405
pytext.utils.documentation, 405
pytext.utils.embeddings, 406
pytext.utils.loss, 406
pytext.utils.meter, 408
pytext.utils.model, 409
pytext.utils.onnx, 409
pytext.utils.precision, 409
pytext.utils.test, 409
pytext.utils.timing, 410
pytext.utils.torch, 410
pytext.workflow, 411

Index

A

ACCURACY (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 288
accuracy (*pytext.metrics.ClassificationMetrics attribute*), 307, 308
accuracy (*pytext.metrics.PairwiseRankingMetrics attribute*), 311
ActionField (*class in pytext.fields*), 277
ActionField (*class in pytext.fields.field*), 274
activate() (*in module pytext.utils.precision*), 409
Adam (*class in pytext.optimizer*), 390
add() (*pytext.config.component.Registry class method*), 196
add() (*pytext.data.utils.VocabBuilder method*), 251
add() (*pytext.utils.timing.Timings method*), 410
add_all() (*pytext.data.utils.VocabBuilder method*), 251
add_batch_stats() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 290
add_batch_stats() (*pytext.metric_reporters.language_model_metric_reporter.MaskedLMMetricReporter method*), 292
add_batch_stats() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 293
add_batch_stats() (*pytext.metric_reporters.MetricReporter method*), 297
add_batch_stats() (*pytext.metric_reporters.pairwise_ranking_metric_reporter.PairwiseRankingMetricReporter method*), 294
add_batch_stats() (*pytext.metric_reporters.PairwiseRankingMetricReporter method*), 301
add_channel() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 290
add_channel() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 293
add_channel() (*pytext.metric_reporters.MetricReporter method*), 297
add_feats_numericalize_ops() (*in module pytext.utils.onnx*), 409
add_include() (*in module pytext.builtin_task*), 410
add_scalars() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 286
add_texts() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 286
aggregate_data() (*pytext.metric_reporters.metric_reporter.MetricReporter class method*), 293
aggregate_data() (*pytext.metric_reporters.MetricReporter class method*), 297
aggregate_preds() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 291
aggregate_preds() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 299
aggregate_preds() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 293
aggregate_preds() (*pytext.metric_reporters.MetricReporter method*), 297
aggregate_scores() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 291
aggregate_scores() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 299
aggregate_scores() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 293

aggregate_scores() (*pytext.metric_reporters.MetricReporter method*), 297
aggregate_targets() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 291
aggregate_targets() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 299
aggregate_targets() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 293
aggregate_targets() (*pytext.metric_reporters.MetricReporter method*), 297
align_slot_labels() (*in module pytext.utils.data*), 405
align_target_label() (*in module pytext.data.utils*), 251
ALIGNMENT (*pytext.common.constants.DFColumn attribute*), 193
AllConfusions (*class in pytext.metrics*), 307
AllMetrics (*class in pytext.metrics.intent_slot_metrics*), 302
Annotation (*class in pytext.data.data_structures.annotation*), 203
annotations_and_defaults() (*pytext.config.pytext_config.ConfigBaseMeta method*), 201
append_bos (*pytext.data.language_model_data_handler.LanguageModelDataHandler.Config attribute*), 66
append_dialect() (*in module pytext.utils.embeddings*), 406
append_eos (*pytext.data.language_model_data_handler.LanguageModelDataHandler.Config attribute*), 66
arrange_model_inputs() (*pytext.models.BaseModel method*), 387
arrange_model_inputs() (*pytext.models.disjoint_multitask_model.NewDisjointMultitaskModel method*), 375
arrange_model_inputs() (*pytext.models.doc_model.NewDocModel method*), 376
arrange_model_inputs() (*pytext.models.model.BaseModel method*), 378
arrange_model_inputs() (*pytext.models.pair_classification_model.PairwiseClassificationModel method*), 383
arrange_model_inputs() (*pytext.models.word_model.NewWordTaggingModel method*), 385
arrange_targets() (*pytext.models.BaseModel method*), 387
arrange_targets() (*pytext.models.disjoint_multitask_model.NewDisjointMultitaskModel method*), 375
arrange_targets() (*pytext.models.doc_model.NewDocModel method*), 376
arrange_targets() (*pytext.models.model.BaseModel method*), 378
arrange_targets() (*pytext.models.pair_classification_model.PairwiseClassificationModel method*), 383
arrange_targets() (*pytext.models.word_model.NewWordTaggingModel method*), 385
ascii_table() (*in module pytext.utils.ascii_table*), 404
ascii_table_from_dict() (*in module pytext.utils.ascii_table*), 404
attention (*pytext.models.representations.bilstm_doc_attention.BiLSTM attribute*), 356
attention (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotA attribute*), 359
Attrs (*class in pytext.main*), 411
AUCPRHingeLoss (*class in pytext.loss*), 282
AUCPRHingeLoss (*class in pytext.loss.loss*), 281
AugmentedLSTM (*class in pytext.models.representations.augmented_lstm*), 352
AugmentedLSTMCell (*class in pytext.models.representations.augmented_lstm*), 353
AugmentedLSTMUnidirectional (*class in pytext.models.representations.augmented_lstm*), 354
average_precision (*pytext.metrics.SoftClassificationMetrics attribute*), 312
average_precision_score() (*in module pytext.metrics*), 312
average_score_difference (*pytext.metrics.PairwiseRankingMetrics attribute*), 311
avg (*pytext.utils.meter.Meter attribute*), 408
avg (*pytext.utils.meter.TimeMeter attribute*), 408

B

B_LABEL_PREFIX (*pytext.utils.data.Slot attribute*), 404
backprop() (*pytext.trainers.Trainer method*), 402
backprop() (*pytext.trainers.trainer.Trainer method*), 400
backward() (*in module pytext.utils.precision*), 409
backward() (*pytext.utils.loss.LagrangeMultiplier static method*), 406

backward_layers (*pytext.models.representations.augmented_lstm.AugmentedLSTM attribute*), 352
BaggingDocEnsemble (*class in pytext.models.ensembles*), 334
BaggingDocEnsemble (*class in pytext.models.ensembles.bagging_doc_ensemble*), 332
BaggingIntentSlotEnsemble (*class in pytext.models.ensembles*), 335
BaggingIntentSlotEnsemble (*class in pytext.models.ensembles.bagging_intent_slot_ensemble*), 333
BaseBatchSampler (*class in pytext.data*), 252
BaseBatchSampler (*class in pytext.data.batch_sampler*), 220
BaseModel (*class in pytext.models*), 387
BaseModel (*class in pytext.models.model*), 378
BasePairwiseClassificationModel (*class in pytext.models.pair_classification_model*), 381
batch_context() (*pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method*), 288
batch_context() (*pytext.metric_reporters.ClassificationMetricReporter method*), 298
batch_context() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 290
batch_context() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 293
batch_context() (*pytext.metric_reporters.MetricReporter method*), 297
batch_context() (*pytext.metric_reporters.SimpleWordTaggingMetricReporter method*), 301
batch_context() (*pytext.metric_reporters.word_tagging_metric_reporter.SimpleWordTaggingMetricReporter method*), 295
batch_predict() (*in module pytext.workflow*), 411
BATCH_SAMPLER (*pytext.config.component.ComponentType attribute*), 195
BatchContext (*class in pytext.common.constants*), 193
Batcher (*class in pytext.data*), 252
Batcher (*class in pytext.data.data*), 228
BATCHER (*pytext.config.component.ComponentType attribute*), 195
BatcherTest (*class in pytext.data.test.data_test*), 215
batches() (*pytext.data.Data method*), 256
batches() (*pytext.data.data.Data method*), 229
batchify() (*pytext.data.BaseBatchSampler method*), 252
batchify() (*pytext.data.batch_sampler.BaseBatchSampler method*), 220
batchify() (*pytext.data.batch_sampler.EvalBatchSampler method*), 220
batchify() (*pytext.data.batch_sampler.RandomizedBatchSampler method*), 221
batchify() (*pytext.data.batch_sampler.RoundRobinBatchSampler method*), 221
batchify() (*pytext.data.Batcher method*), 252
batchify() (*pytext.data.data.Batcher method*), 228
batchify() (*pytext.data.data.PoolingBatcher method*), 229
batchify() (*pytext.data.EvalBatchSampler method*), 262
batchify() (*pytext.data.PoolingBatcher method*), 264
batchify() (*pytext.data.RandomizedBatchSampler method*), 265
batchify() (*pytext.data.RoundRobinBatchSampler method*), 267
BatchIterator (*class in pytext.data*), 252
BatchIterator (*class in pytext.data.data_handler*), 230
BatchSamplerTest (*class in pytext.data.test.batch_sampler_test*), 214
BatchScheduler (*class in pytext.trainers.training_state*), 388
best_model_metric (*pytext.trainers.trainer.TrainingState attribute*), 401
best_model_metric (*pytext.trainers.TrainingState attribute*), 403
best_model_state (*pytext.trainers.trainer.TrainingState attribute*), 401
best_model_state (*pytext.trainers.TrainingState attribute*), 403
bidirectional (*pytext.models.representations.bilstm.BiLSTM.Config attribute*), 122
BiLSTM (*class in pytext.models.representations.bilstm*), 355
BiLSTMDocAttention (*class in pytext.models.representations.bilstm_doc_attention*), 356
BiLSTMDocSlotAttention (*class in pytext.models.representations.bilstm_doc_slot_attention*), 357
BiLSTMSlotAttention (*class in pytext.models.representations.bilstm_slot_attn*), 359
BinaryClassificationOutputLayer (*class in py-*

text.models.output_layers.doc_classification_output_layer), 337
BinaryCrossEntropyLoss (*class in pytext.loss*), 283
BinaryCrossEntropyLoss (*class in pytext.loss.loss*), 281
BlockShardedTSV (*class in pytext.data.sources.tsv*), 212
BlockShardedTSVDataSource (*class in pytext.data.sources.tsv*), 212
BoundaryPool (*class in pytext.models.representations.pooling*), 363
bptt_len (*pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler.Config attribute*), 56
BPTTLanguageModelDataHandler (*class in pytext.data*), 252
BPTTLanguageModelDataHandler (*class in pytext.data.bptt_lm_data_handler*), 222
BPTTLanguageModelDataHandlerTest (*class in pytext.data.test.bptt_lm_data_handler_test*), 214
bracket_metrics (*pytext.metrics.intent_slot_metrics.AllMetrics attribute*), 302
BSeqCNNRepresentation (*class in pytext.models.representations.biseqcnn*), 360
build_class_priors() (*in module pytext.utils.loss*), 407
build_tree() (*pytext.data.data_structures.annotation.Annotation method*), 203
build_vocab() (*pytext.fields.char_field.CharFeatureField method*), 272
build_vocab() (*pytext.fields.CharFeatureField method*), 277
build_vocab() (*pytext.fields.dict_field.DictFeatureField method*), 273
build_vocab() (*pytext.fields.DictFeatureField method*), 278
build_vocab() (*pytext.fields.text_field_with_special_unk.TextFeatureFieldWithSpecialUnk method*), 276
build_vocab() (*pytext.fields.TextFeatureFieldWithSpecialUnk method*), 280
ByteTensorizer (*class in pytext.data.tensorizers*), 247

C

cache_pretrained_embeddings() (*pytext.utils.embeddings.PretrainedEmbedding method*), 406
caffe2_export() (*pytext.models.BaseModel method*), 387
caffe2_export() (*pytext.models.disjoint_multitask_model.NewDisjointMultitaskModel method*), 375
caffe2_export() (*pytext.models.doc_model.NewDocModel method*), 377
caffe2_export() (*pytext.models.model.BaseModel method*), 378
calculate_loss() (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter.Config method*), 292
calculate_loss() (*pytext.metric_reporters.language_model_metric_reporter.MaskedLMMetricReporter method*), 292
calculate_loss() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 299
calculate_loss() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 293
calculate_loss() (*pytext.metric_reporters.MetricReporter method*), 297
calculate_loss() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTaggingMetricReporter method*), 296
calculate_loss() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 300
calculate_metric() (*pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method*), 288
calculate_metric() (*pytext.metric_reporters.ClassificationMetricReporter method*), 298
calculate_metric() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method*), 289
calculate_metric() (*pytext.metric_reporters.CompositionalMetricReporter method*), 300
calculate_metric() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 291
calculate_metric() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 299
calculate_metric() (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method*), 292
calculate_metric() (*pytext.metric_reporters.LanguageModelMetricReporter*

method), 300
calculate_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 294
calculate_metric() (*pytext.metric_reporters.MetricReporter method*), 297
calculate_metric() (*pytext.metric_reporters.pairwise_ranking_metric_reporter.PairwiseRankingMetricReporter method*), 295
calculate_metric() (*pytext.metric_reporters.PairwiseRankingMetricReporter method*), 301
calculate_metric() (*pytext.metric_reporters.regression_metric_reporter.RegressionMetricReporter method*), 295
calculate_metric() (*pytext.metric_reporters.RegressionMetricReporter method*), 299
calculate_metric() (*pytext.metric_reporters.SimpleWordTaggingMetricReporter method*), 301
calculate_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.SimpleWordTaggingMetricReporter method*), 295
calculate_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTaggingMetricReporter method*), 296
calculate_metric() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 300
calculate_perplexity() (*pytext.models.output_layers.lm_output_layer.LMOutputLayer static method*), 341
cell (*pytext.models.representations.augmented_lstm.AugmentedLSTMUnidirectional attribute*), 354
Channel (*class in pytext.metric_reporters*), 296
Channel (*class in pytext.metric_reporters.channel*), 284
channels (*pytext.metric_reporters.metric_reporter.MetricReporter attribute*), 293
channels (*pytext.metric_reporters.MetricReporter attribute*), 297
CHAR (*pytext.config.contextual_intent_slot.ModelInput attribute*), 197
char_embed (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 321
char_embed (*pytext.models.embeddings.CharacterEmbedding attribute*), 331
char_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 198
CHAR_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 198
char_feat (*pytext.config.doc_classification.ModelInputConfig attribute*), 198
char_feat (*pytext.config.field_config.FeatureConfig attribute*), 199
CHAR_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
CharacterEmbedding (*class in pytext.models.embeddings*), 330
CharacterEmbedding (*class in pytext.models.embeddings.char_embedding*), 321
characters (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 206
characters (*pytext.data.featurizer.OutputRecord attribute*), 208
CharacterTokenTensorizer (*class in pytext.data.tensorizers*), 247
CharFeatConfig (*in module pytext.config.field_config*), 198
CharFeatureField (*class in pytext.fields*), 277
CharFeatureField (*class in pytext.fields.char_field*), 272
children (*pytext.data.data_structures.node.Node attribute*), 205
children (*pytext.metrics.intent_slot_metrics.Node attribute*), 304
children_flat_str_spans() (*pytext.data.data_structures.annotation.Node method*), 203
ClassificationMetricReporter (*class in pytext.metric_reporters*), 298
ClassificationMetricReporter (*class in pytext.metric_reporters.classification_metric_reporter*), 288
ClassificationMetrics (*class in pytext.metrics*), 307
ClassificationOutputLayer (*class in pytext.models.output_layers*), 349
ClassificationOutputLayer (*class in pytext.models.output_layers.doc_classification_output_layer*), 337
ClassificationScores (*class in pytext.models.output_layers.doc_classification_output_layer*), 338
clip_grad_norm() (*in module pytext.utils.precision*), 409
close() (*pytext.metric_reporters.channel.Channel method*), 296
close() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 284
cls_vars() (*in module pytext.utils*), 286
CNNParams (*class in pytext.config.module_config*), 410
COLUMN (*pytext.config.component.ComponentType attribute*), 200

columns_to_read (*pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler attribute*), 56
columns_to_read (*pytext.data.doc_classification_data_handler.DocClassificationDataHandler attribute*), 63
columns_to_read (*pytext.data.language_model_data_handler.LanguageModelDataHandler attribute*), 66
CommonMetadata (*class in pytext.data*), 253
CommonMetadata (*class in pytext.data.data_handler*), 230
ComparableClassificationMetric (*class in pytext.metric_reporters.classification_metric_reporter*), 288
compare_frames() (*in module pytext.metrics.intent_slot_metrics*), 304
compare_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 294
compare_metric() (*pytext.metric_reporters.MetricReporter method*), 298
Component (*class in pytext.config.component*), 195
ComponentMeta (*class in pytext.config.component*), 195
ComponentType (*class in pytext.config.component*), 195
compose_embedding() (*pytext.models.Model class method*), 386
compose_embedding() (*pytext.models.model.Model class method*), 379
compose_embedding() (*pytext.models.pair_classification_model.PairClassificationModel class method*), 382
compose_embedding() (*pytext.models.query_document_pairwise_ranking_model.QueryDocumentPairwiseRankingModel class method*), 384
compose_embedding() (*pytext.models.seq_models.contextual_intent_slot.ContextualIntentSlotModel class method*), 372
CompositionalDataHandler (*class in pytext.data*), 253
CompositionalDataHandler (*class in pytext.data.compositional_data_handler*), 224
CompositionalDataHandlerTest (*class in pytext.data.test.compositional_datahandler_test*), 214
CompositionalFileChannel (*class in pytext.metric_reporters.compositional_metric_reporter*), 289
CompositionalMetricReporter (*class in pytext.metric_reporters*), 300
CompositionalMetricReporter (*class in pytext.metric_reporters.compositional_metric_reporter*), 289
CompositionalMetricReporter.Config (*class in pytext.metric_reporters.compositional_metric_reporter*), 289
CompositionalNN (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 368
CompositionalNN.Config (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 368
CompositionalSummationNN (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 368
CompositionalSummationNN.Config (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 368
CompositionFunction (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 368
compute_all_metrics() (*in module pytext.metrics.intent_slot_metrics*), 305
compute_classification_metrics() (*in module pytext.metrics*), 313
compute_frame_accuracies_by_depth() (*in module pytext.metrics.intent_slot_metrics*), 305
compute_frame_accuracy() (*in module pytext.metrics.intent_slot_metrics*), 305
compute_frame_accuracy_top_k() (*in module pytext.metrics.intent_slot_metrics*), 306
compute_intent_slot_metrics() (*in module pytext.metrics.intent_slot_metrics*), 306
compute_language_model_metric() (*in module pytext.metrics.language_model_metrics*), 307
compute_matthews_correlation_coefficients() (*in module pytext.metrics*), 313
compute_metrics() (*pytext.metrics.AllConfusions method*), 307
compute_metrics() (*pytext.metrics.Confusions method*), 308
compute_metrics() (*pytext.metrics.PerLabelConfusions method*), 311
compute_pairwise_ranking_metrics() (*in module pytext.metrics*), 313
compute_prf1() (*in module pytext.metrics*), 314
compute_prf1_metrics() (*in module pytext.metrics.intent_slot_metrics*), 306
compute_regression_metrics() (*in module pytext.metrics*), 314
compute_roc_auc() (*in module pytext.metrics*), 314
compute_soft_metrics() (*in module pytext.metrics*), 314
compute_top_intent_accuracy() (*in module pytext.metrics.intent_slot_metrics*), 307
CONCAT (*pytext.config.module_config.SlotAttentionType attribute*), 200
concat (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 325
concat (*pytext.models.embeddings.EmbeddingList attribute*), 328
Config (*pytext.config.component.Component attribute*),

195
Config (pytext.data.BaseBatchSampler attribute), 252
Config (pytext.data.batch_sampler.BaseBatchSampler attribute), 220
Config (pytext.data.batch_sampler.EvalBatchSampler attribute), 220
Config (pytext.data.batch_sampler.RandomizedBatchSampler attribute), 221
Config (pytext.data.batch_sampler.RoundRobinBatchSampler attribute), 221
Config (pytext.data.Batcher attribute), 252
Config (pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler attribute), 222
Config (pytext.data.BPTTLanguageModelDataHandler attribute), 253
Config (pytext.data.compositional_data_handler.CompositionalDataHandler attribute), 224
Config (pytext.data.CompositionalDataHandler attribute), 254
Config (pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotModelDataHandler attribute), 227
Config (pytext.data.ContextualIntentSlotModelDataHandler attribute), 255
Config (pytext.data.Data attribute), 256
Config (pytext.data.data.Batcher attribute), 228
Config (pytext.data.data.Data attribute), 229
Config (pytext.data.data.PoolingBatcher attribute), 229
Config (pytext.data.data_handler.DataHandler attribute), 232
Config (pytext.data.DataHandler attribute), 258
Config (pytext.data.disjoint_multitask_data.DisjointMultitaskData attribute), 234
Config (pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler attribute), 235
Config (pytext.data.DisjointMultitaskData attribute), 260
Config (pytext.data.DisjointMultitaskDataHandler attribute), 261
Config (pytext.data.doc_classification_data_handler.DocClassificationDataHandler attribute), 239
Config (pytext.data.DocClassificationDataHandler attribute), 262
Config (pytext.data.EvalBatchSampler attribute), 262
Config (pytext.data.featurizer.Featurizer attribute), 207
Config (pytext.data.featurizer.featurizer.Featurizer attribute), 206
Config (pytext.data.featurizer.simple_featurizer.SimpleFeaturizer attribute), 207
Config (pytext.data.featurizer.SimpleFeaturizer attribute), 208
Config (pytext.data.joint_data_handler.JointModelDataHandler attribute), 240
Config (pytext.data.JointModelDataHandler attribute), 263
Config (pytext.data.language_model_data_handler.LanguageModelDataHandler attribute), 242
Config (pytext.data.LanguageModelDataHandler attribute), 263
Config (pytext.data.pair_classification_data_handler.PairClassificationDataHandler attribute), 243
Config (pytext.data.PairClassificationDataHandler attribute), 264
Config (pytext.data.PoolingBatcher attribute), 264
Config (pytext.data.query_document_pairwise_ranking_data_handler.QueryDocumentPairwiseRankingDataHandler attribute), 245
Config (pytext.data.QueryDocumentPairwiseRankingDataHandler attribute), 266
Config (pytext.data.RandomizedBatchSampler attribute), 265
Config (pytext.data.RoundRobinBatchSampler attribute), 267
Config (pytext.data.seq_data_handler.SeqModelDataHandler attribute), 246
Config (pytext.data.SeqModelDataHandler attribute), 267
Config (pytext.data.sources.data_source.DataSource attribute), 209
Config (pytext.data.sources.data_source.RootDataSource attribute), 210
Config (pytext.data.sources.data_source.RowShardedDataSource attribute), 210
Config (pytext.data.sources.data_source.ShardedDataSource attribute), 211
Config (pytext.data.sources.DataSource attribute), 213
Config (pytext.data.sources.squad.SquadDataSource attribute), 211
Config (pytext.data.sources.SquadDataSource attribute), 213
Config (pytext.data.sources.tsv.BlockShardedTSVDataSource attribute), 212
Config (pytext.data.sources.tsv.MultilingualTSVDataSource attribute), 212
Config (pytext.data.sources.tsv.SessionTSVDataSource attribute), 212
Config (pytext.data.sources.tsv.TSVDataSource attribute), 212
Config (pytext.data.sources.TSVDataSource attribute), 214
Config (pytext.data.tensorizers.ByteTensorizer attribute), 247
Config (pytext.data.tensorizers.CharacterTokenTensorizer attribute), 247
Config (pytext.data.tensorizers.FloatListTensorizer attribute), 248
Config (pytext.data.tensorizers.JoinStringTensorizer attribute), 248
Config (pytext.data.tensorizers.LabelTensorizer attribute), 248

Config (*pytext.data.tensorizers.MetricTensorizer attribute*), 248
Config (*pytext.data.tensorizers.NtokensTensorizer attribute*), 249
Config (*pytext.data.tensorizers.NumericLabelTensorizer attribute*), 249
Config (*pytext.data.tensorizers.RawJson attribute*), 249
Config (*pytext.data.tensorizers.RawString attribute*), 249
Config (*pytext.data.tensorizers.Tensorizer attribute*), 249
Config (*pytext.data.tensorizers.TokenTensorizer attribute*), 250
Config (*pytext.data.tensorizers.WordLabelTensorizer attribute*), 250
Config (*pytext.data.tokenizers.Tokenizer attribute*), 220
Config (*pytext.data.tokenizers.tokenizer.Tokenizer attribute*), 219
Config (*pytext.exporters.custom_exporters.DenseFeatureExporter attribute*), 268
Config (*pytext.exporters.DenseFeatureExporter attribute*), 272
Config (*pytext.exporters.exporter.ModelExporter attribute*), 268
Config (*pytext.exporters.ModelExporter attribute*), 270
Config (*pytext.loss.AUCPRHingeLoss attribute*), 282
Config (*pytext.loss.BinaryCrossEntropyLoss attribute*), 283
Config (*pytext.loss.CrossEntropyLoss attribute*), 283
Config (*pytext.loss.KLDivergenceBCELoss attribute*), 283
Config (*pytext.loss.KLDivergenceCELoss attribute*), 283
Config (*pytext.loss.LabelSmoothedCrossEntropyLoss attribute*), 284
Config (*pytext.loss.Loss attribute*), 283
Config (*pytext.loss.loss.AUCPRHingeLoss attribute*), 281
Config (*pytext.loss.loss.BinaryCrossEntropyLoss attribute*), 281
Config (*pytext.loss.loss.CrossEntropyLoss attribute*), 281
Config (*pytext.loss.loss.KLDivergenceBCELoss attribute*), 281
Config (*pytext.loss.loss.KLDivergenceCELoss attribute*), 282
Config (*pytext.loss.loss.LabelSmoothedCrossEntropyLoss attribute*), 282
Config (*pytext.loss.loss.Loss attribute*), 282
Config (*pytext.loss.loss.MSELoss attribute*), 282
Config (*pytext.loss.loss.PairwiseRankingLoss attribute*), 282
Config (*pytext.loss.loss.SoftHardBCELoss attribute*), 282
Config (*pytext.loss.MSELoss attribute*), 283
Config (*pytext.loss.PairwiseRankingLoss attribute*), 284
Config (*pytext.loss.SoftHardBCELoss attribute*), 284
Config (*pytext.metric_reporters.classification_metric_reporter.Classificat attribute*), 288
Config (*pytext.metric_reporters.ClassificationMetricReporter attribute*), 298
Config (*pytext.metric_reporters.compositional_metric_reporter.Composit attribute*), 289
Config (*pytext.metric_reporters.CompositionalMetricReporter attribute*), 300
Config (*pytext.metric_reporters.disjoint_multitask_metric_reporter.Disjoi attribute*), 290
Config (*pytext.metric_reporters.intent_slot_detection_metric_reporter.Int attribute*), 291
Config (*pytext.metric_reporters.IntentSlotMetricReporter attribute*), 299
Config (*pytext.metric_reporters.language_model_metric_reporter.Langua attribute*), 292
Config (*pytext.metric_reporters.language_model_metric_reporter.Masked attribute*), 292
Config (*pytext.metric_reporters.LanguageModelMetricReporter attribute*), 299
Config (*pytext.metric_reporters.metric_reporter.MetricReporter attribute*), 293
Config (*pytext.metric_reporters.MetricReporter attribute*), 297
Config (*pytext.metric_reporters.pairwise_ranking_metric_reporter.Pairwi attribute*), 294
Config (*pytext.metric_reporters.PairwiseRankingMetricReporter attribute*), 301
Config (*pytext.metric_reporters.regression_metric_reporter.RegressionMe attribute*), 295
Config (*pytext.metric_reporters.RegressionMetricReporter attribute*), 299
Config (*pytext.metric_reporters.SimpleWordTaggingMetricReporter attribute*), 301
Config (*pytext.metric_reporters.word_tagging_metric_reporter.SimpleWo attribute*), 295
Config (*pytext.metric_reporters.word_tagging_metric_reporter.WordTagg attribute*), 295
Config (*pytext.metric_reporters.WordTaggingMetricReporter attribute*), 300
Config (*pytext.models.BaseModel attribute*), 387
Config (*pytext.models.decoders.decoder_base.DecoderBase attribute*), 315
Config (*pytext.models.decoders.DecoderBase attribute*), 318
Config (*pytext.models.decoders.intent_slot_model_decoder.IntentSlotMod attribute*), 316
Config (*pytext.models.decoders.IntentSlotModelDecoder attribute*), 320
Config (*pytext.models.decoders.mlp_decoder.MLPDecoder*

attribute), 317
Config (*pytext.models.decoders.mlp_decoder_query_response.MLPDecoderQueryResponse attribute*), 318
Config (*pytext.models.decoders.MLPDecoder attribute*), 319
Config (*pytext.models.disjoint_multitask_model.DisjointMultitaskModel attribute*), 374
Config (*pytext.models.disjoint_multitask_model.NewDisjointMultitaskModel attribute*), 375
Config (*pytext.models.doc_model.DocModel_Deprecated attribute*), 376
Config (*pytext.models.doc_model.NewDocModel attribute*), 376
Config (*pytext.models.doc_model.NewDocRegressionModel attribute*), 377
Config (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 322
Config (*pytext.models.embeddings.CharacterEmbedding attribute*), 331
Config (*pytext.models.embeddings.contextual_token_embedding.ContextualTokenEmbedding attribute*), 323
Config (*pytext.models.embeddings.ContextualTokenEmbedding attribute*), 332
Config (*pytext.models.embeddings.dict_embedding.DictEmbedding attribute*), 324
Config (*pytext.models.embeddings.DictEmbedding attribute*), 330
Config (*pytext.models.embeddings.embedding_base.EmbeddingBase attribute*), 325
Config (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 325
Config (*pytext.models.embeddings.EmbeddingBase attribute*), 327
Config (*pytext.models.embeddings.EmbeddingList attribute*), 328
Config (*pytext.models.embeddings.word_embedding.WordEmbedding attribute*), 326
Config (*pytext.models.embeddings.WordEmbedding attribute*), 329
Config (*pytext.models.ensembles.bagging_doc_ensemble.BaggingDocEnsemble attribute*), 332
Config (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsemble attribute*), 333
Config (*pytext.models.ensembles.BaggingDocEnsemble attribute*), 334
Config (*pytext.models.ensembles.BaggingIntentSlotEnsemble attribute*), 335
Config (*pytext.models.ensembles.ensemble.Ensemble attribute*), 334
Config (*pytext.models.joint_model.JointModel attribute*), 377
Config (*pytext.models.language_models.lmlstm.LMLSTM attribute*), 335
Config (*pytext.models.Model attribute*), 386
Config (*pytext.models.model.BaseModel attribute*), 378
Config (*pytext.models.model.NewBaseModel attribute*), 379
Config (*pytext.models.module.Module attribute*), 381
Config (*pytext.models.output_layers.ClassificationOutputLayer attribute*), 349
Config (*pytext.models.output_layers.CRFOutputLayer attribute*), 348
Config (*pytext.models.output_layers.doc_classification_output_layer.BinaryDocClassificationOutputLayer attribute*), 337
Config (*pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer attribute*), 337
Config (*pytext.models.output_layers.doc_classification_output_layer.MultiLabelDocClassificationOutputLayer attribute*), 338
Config (*pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer attribute*), 339
Config (*pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer attribute*), 340
Config (*pytext.models.output_layers.lm_output_layer.LMOutputLayer attribute*), 341
Config (*pytext.models.output_layers.output_layer_base.OutputLayerBase attribute*), 342
Config (*pytext.models.output_layers.OutputLayerBase attribute*), 347
Config (*pytext.models.output_layers.pairwise_ranking_output_layer.PairwiseRankingOutputLayer attribute*), 344
Config (*pytext.models.output_layers.PairwiseRankingOutputLayer attribute*), 351
Config (*pytext.models.output_layers.RegressionOutputLayer attribute*), 350
Config (*pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer attribute*), 345
Config (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer attribute*), 346
Config (*pytext.models.output_layers.WordTaggingOutputLayer attribute*), 351
Config (*pytext.models.pair_classification_model.BasePairwiseClassificationModel attribute*), 381
Config (*pytext.models.pair_classification_model.PairClassificationModel attribute*), 382
Config (*pytext.models.pair_classification_model.PairwiseClassificationModel attribute*), 383
Config (*pytext.models.pairwise_query_document_pairwise_ranking_model.QueryDocumentPairwiseRankingModel attribute*), 384
Config (*pytext.models.representations.augmented_lstm.AugmentedLSTM attribute*), 353
Config (*pytext.models.representations.bilstm.BiLSTM attribute*), 356
Config (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute*), 357
Config (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 358
Config (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttn attribute*), 359
Config (*pytext.models.representations.biseqcnn.BSeqCNNRepresentation*

attribute), 360
Config (pytext.models.representations.contextual_intent_slot_rep.ContextualIntentSlotRepresentation attribute), 361
Config (pytext.models.representations.docnn.DocNNRepresentation attribute), 361
Config (pytext.models.representations.jointcnn_rep.JointCNNRepresentation attribute), 362
Config (pytext.models.representations.pair_rep.PairRepresentation attribute), 362
Config (pytext.models.representations.pass_through.PassThroughRepresentation attribute), 363
Config (pytext.models.representations.pooling.BoundaryPool attribute), 363
Config (pytext.models.representations.pooling.LastTimestepPool attribute), 363
Config (pytext.models.representations.pooling.MaxPool attribute), 364
Config (pytext.models.representations.pooling.MeanPool attribute), 364
Config (pytext.models.representations.pooling.NoPool attribute), 364
Config (pytext.models.representations.pooling.SelfAttention attribute), 365
Config (pytext.models.representations.pure_doc_attention.PureDocAttention attribute), 365
Config (pytext.models.representations.query_document_pairwise_ranking_rep.QueryDocumentPairwiseRankingRep attribute), 366
Config (pytext.models.representations.representation_base.RepresentationBase attribute), 366
Config (pytext.models.representations.seq_rep.SeqRepresentation attribute), 366
Config (pytext.models.representations.slot_attention.SlotAttention attribute), 367
Config (pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN attribute), 368
Config (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser attribute), 370
Config (pytext.models.seq_models.contextual_intent_slot.ContextualIntentSlotModel attribute), 372
Config (pytext.models.seq_models.seqnn.SeqNNModel attribute), 373
Config (pytext.models.word_model.NewWordTaggingModel attribute), 385
Config (pytext.models.word_model.WordTaggingModel attribute), 385
Config (pytext.optimizer.Adam attribute), 390
Config (pytext.optimizer.Optimizer attribute), 390
Config (pytext.optimizer.scheduler.BatchScheduler attribute), 388
Config (pytext.optimizer.scheduler.CosineAnnealingLR attribute), 388
Config (pytext.optimizer.scheduler.ExponentialLR attribute), 389
Config (pytext.optimizer.scheduler.LmFineTuning attribute), 389
Config (pytext.optimizer.scheduler.ReduceLROnPlateau attribute), 389
Config (pytext.optimizer.scheduler.Scheduler attribute), 390
Config (pytext.optimizer.scheduler.StepLR attribute), 390
Config (pytext.optimizer.scheduler.WarmupScheduler attribute), 390
Config (pytext.optimizer.SGD attribute), 390
Config (pytext.task.disjoint_multitask.DisjointMultitask attribute), 391
Config (pytext.task.disjoint_multitask.NewDisjointMultitask attribute), 392
Config (pytext.task.new_task.NewDocumentClassification attribute), 392
Config (pytext.task.new_task.NewDocumentRegression attribute), 392
Config (pytext.task.new_task.NewTask attribute), 393
Config (pytext.task.new_task.NewTaskTrainer attribute), 393
Config (pytext.task.new_task.PairwiseClassificationTask attribute), 393
Config (pytext.task.NewDocumentClassification attribute), 398
Config (pytext.task.NewQueryDocumentPairwiseRankingTask attribute), 397
Config (pytext.task.Task attribute), 398
Config (pytext.task.task.Task attribute), 393
Config (pytext.task.task.TaskBase attribute), 394
Config (pytext.task.TaskBase attribute), 398
Config (pytext.task.tasks.ContextualIntentSlotTask attribute), 395
Config (pytext.task.tasks.DocClassificationTask attribute), 395
Config (pytext.task.tasks.EnsembleTask attribute), 395
Config (pytext.task.tasks.JointTextTask attribute), 395
Config (pytext.task.tasks.LMTask attribute), 396
Config (pytext.task.tasks.NewWordTaggingTask attribute), 396
Config (pytext.task.tasks.PairClassificationTask attribute), 396
Config (pytext.task.tasks.QueryDocumentPairwiseRankingTask attribute), 396
Config (pytext.task.tasks.SemanticParsingTask attribute), 397
Config (pytext.task.tasks.SeqNNTask attribute), 397
Config (pytext.task.tasks.WordTaggingTask attribute), 397
Config (pytext.trainers.ensemble_trainer.EnsembleTrainer attribute), 399
Config (pytext.trainers.EnsembleTrainer attribute), 403
Config (pytext.trainers.hogwild_trainer.HogwildTrainer attribute), 399
Config (pytext.trainers.HogwildTrainer attribute), 404

Config (*pytext.trainers.Trainer attribute*), 402
Config (*pytext.trainers.trainer.Trainer attribute*), 400
Config (*pytext.trainers.trainer.TrainerBase attribute*), 401
config_from_json() (*in module pytext.config.serialize*), 202
config_to_json() (*in module pytext.config.serialize*), 202
ConfigBase (*class in pytext.config.pytext_config*), 200
ConfigBaseMeta (*class in pytext.config.pytext_config*), 200
ConfigParseError, 202
configs() (*pytext.config.component.Registry class method*), 196
Confusions (*class in pytext.metrics*), 308
confusions (*pytext.metrics.AllConfusions attribute*), 307
ConsoleChannel (*class in pytext.metric_reporters.channel*), 285
CONTEXTUAL_TOKEN_EMBEDDING (*pytext.common.constants.DatasetFieldName attribute*), 194
CONTEXTUAL_TOKEN_EMBEDDING (*pytext.config.contextual_intent_slot.ModelInput attribute*), 197
contextual_token_embedding (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 198
CONTEXTUAL_TOKEN_EMBEDDING (*pytext.config.doc_classification.ModelInput attribute*), 198
contextual_token_embedding (*pytext.config.doc_classification.ModelInputConfig attribute*), 198
contextual_token_embedding (*pytext.config.field_config.FeatureConfig attribute*), 199
contextual_token_embedding (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 206
contextual_token_embedding (*pytext.data.featurizer.OutputRecord attribute*), 208
ContextualIntentSlotModel (*class in pytext.models.seq_models.contextual_intent_slot*), 372
ContextualIntentSlotModelDataHandler (*class in pytext.data*), 254
ContextualIntentSlotModelDataHandler (*class in pytext.data.contextual_intent_slot_data_handler*), 226
ContextualIntentSlotModelDataHandlerDenseTest (*class in pytext.data.test.contextual_intent_slot_data_handler_test*), 215
ContextualIntentSlotModelDataHandlerTest (*class in pytext.data.test.contextual_intent_slot_data_handler_test*), 215
ContextualIntentSlotRepresentation (*class in pytext.models.representations.contextual_intent_slot_rep*), 360
ContextualIntentSlotTask (*class in pytext.task.tasks*), 395
contextualize() (*pytext.models.BaseModel method*), 387
contextualize() (*pytext.models.disjoint_multitask_model.DisjointMultitaskModel method*), 374
contextualize() (*pytext.models.model.BaseModel method*), 378
contextualize() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 370
ContextualTokenEmbedding (*class in pytext.models.embeddings*), 332
ContextualTokenEmbedding (*class in pytext.models.embeddings.contextual_token_embedding*), 323
ContextualTokenEmbeddingConfig (*in module pytext.config.field_config*), 198
ContextualTokenEmbeddingField (*class in pytext.fields*), 277
ContextualTokenEmbeddingField (*class in pytext.fields.contextual_token_embedding_field*), 273
continue_training() (*pytext.trainers.Trainer method*), 402
continue_training() (*pytext.trainers.trainer.Trainer method*), 400
conv_and_pool() (*pytext.models.representations.docnn.DocNNRepresentation method*), 361
convert_caffe2_blob_name() (*in module pytext.utils.onnx*), 409
convs (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 321
convs (*pytext.models.embeddings.CharacterEmbedding attribute*), 331
copy() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.Parse method*), 369
copy() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.Stack method*), 369
CosineAnnealingLR (*class in pytext.optimizer.scheduler*), 388
cpu() (*pytext.models.distributed_model.DistributedModel*

method), 375
create_annotation() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotChannel static method*), 291
create_component() (*in module pytext.config.component*), 196
create_data_handler() (*in module pytext.config.component*), 196
create_decoder() (*pytext.models.doc_model.NewDocModel class method*), 377
create_embedding() (*pytext.models.doc_model.NewDocModel class method*), 377
create_embedding() (*pytext.models.Model class method*), 387
create_embedding() (*pytext.models.model.Model class method*), 380
create_embedding() (*pytext.models.word_model.NewWordTaggingModel class method*), 385
create_exporter() (*in module pytext.config.component*), 196
create_featurizer() (*in module pytext.config.component*), 196
create_fields() (*in module pytext.fields*), 277
create_fields() (*in module pytext.fields.field*), 276
create_frame() (*in module pytext.metric_reporters.intent_slot_detection_metric_reporter*), 291
create_frame_prediction_pairs() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method*), 289
create_frame_prediction_pairs() (*pytext.metric_reporters.CompositionalMetricReporter method*), 300
create_label_fields() (*in module pytext.fields*), 277
create_label_fields() (*in module pytext.fields.field*), 276
create_language_model_data_handler() (*pytext.data.test.language_model_data_handler_test.LanguageModelDataHandlerTest class method*), 217
create_loss() (*in module pytext.config.component*), 196
create_metric_reporter() (*in module pytext.config.component*), 196
create_metric_reporter() (*pytext.task.tasks.NewWordTaggingTask class method*), 396
create_model() (*in module pytext.config.component*), 196
create_module() (*in module pytext.models.module*), 381
create_optimizer() (*in module pytext.config.component*), 196
create_parser() (*in module pytext*), 412
create_predictor() (*in module pytext.config.component*), 197
create_scheduler() (*in module pytext.config.component*), 197
create_sub_embs() (*pytext.models.Model class method*), 387
create_sub_embs() (*pytext.models.model.Model class method*), 380
create_sub_embs() (*pytext.models.query_document_pairwise_ranking_model.QueryDoc class method*), 384
create_task() (*in module pytext.task*), 399
create_task() (*in module pytext.task.task*), 394
create_trainer() (*in module pytext.config.component*), 197
create_vocab_index() (*in module pytext.utils.onnx*), 409
create_vocab_indices_map() (*in module pytext.utils.onnx*), 409
CRF (*class in pytext.models.crf*), 373
CRFOutputLayer (*class in pytext.models.output_layers*), 348
CRFOutputLayer (*class in pytext.models.output_layers.word_tagging_output_layer*), 344
CrossEntropyLoss (*class in pytext.loss*), 283
CrossEntropyLoss (*class in pytext.loss.loss*), 281
current_model (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler attribute*), 374
cycle() (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatchI class method*), 237

D

Data (*class in pytext.data*), 256
Data (*class in pytext.data.data*), 228
data_dict (*pytext.data.disjoint_multitask_data.DisjointMultitaskData attribute*), 234
data_size (*pytext.data.DisjointMultitaskData attribute*), 260
DATA_HANDLER (*pytext.config.component.ComponentType attribute*), 195
data_handlers (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH attribute*), 235
data_handlers (*pytext.data.DisjointMultitaskDataHandler attribute*), 261
DATA_SOURCE (*pytext.config.component.ComponentType attribute*), 196

DATA_SOURCE_TYPES (*pytext.data.sources.data_source.RootDataSource attribute*), 210
DATA_TYPE (*pytext.config.component.ComponentType attribute*), 196
DataHandler (*class in pytext.data*), 256
DataHandler (*class in pytext.data.data_handler*), 230
DataHandlerTest (*class in pytext.data.test.datahandler_test*), 216
DatasetFieldName (*class in pytext.common.constants*), 194
DataSource (*class in pytext.data.sources*), 213
DataSource (*class in pytext.data.sources.data_source*), 209
DataTest (*class in pytext.data.test.data_test*), 215
deactivate() (*in module pytext.utils.precision*), 409
debug_path (*pytext.config.pytext_config.PyTextConfig attribute*), 201
decode() (*pytext.models.crf.CRF method*), 373
decoder (*pytext.models.language_models.lmlstm.LMLSTM.Config attribute*), 105
decoder (*pytext.models.Model attribute*), 386
decoder (*pytext.models.model.Model attribute*), 379
DecoderBase (*class in pytext.models.decoders*), 318
DecoderBase (*class in pytext.models.decoders.decoder_base*), 315
DENSE (*pytext.config.contextual_intent_slot.ModelInput attribute*), 198
dense (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention.Config attribute*), 357
dense (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttention.Config attribute*), 359
DENSE_FEAT (*pytext.common.constants.DFColumn attribute*), 193
dense_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 198
DENSE_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 198
dense_feat (*pytext.config.doc_classification.ModelInputConfig attribute*), 198
dense_feat (*pytext.config.field_config.FeatureConfig attribute*), 199
DENSE_FEAT (*pytext.data.contextual_intent_slot_data_handler.RawData attribute*), 228
dense_feats (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 206
dense_feats (*pytext.data.featurizer.OutputRecord attribute*), 208
DENSE_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
DenseFeatureExporter (*class in pytext.exporters*), 271
DenseFeatureExporter (*class in pytext.exporters.custom_exporters*), 268
depth() (*pytext.data.data_structures.annotation.Tree method*), 204
DFColumn (*class in pytext.common.constants*), 193
DICT (*pytext.config.contextual_intent_slot.ModelInput attribute*), 198
DICT_FEAT (*pytext.common.constants.DFColumn attribute*), 193
dict_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 198
DICT_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 198
dict_feat (*pytext.config.doc_classification.ModelInputConfig attribute*), 198
dict_feat (*pytext.config.field_config.FeatureConfig attribute*), 199
DICT_FEAT (*pytext.data.contextual_intent_slot_data_handler.RawData attribute*), 228
DICT_FEAT (*pytext.data.doc_classification_data_handler.RawData attribute*), 239
DICT_FEAT (*pytext.data.RawData attribute*), 266
DICT_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
DictEmbedding (*class in pytext.models.embeddings*), 329
DictEmbedding (*class in pytext.models.embeddings.dict_embedding*), 323
DictFeatConfig (*in module pytext.config.field_config*), 199
DictFeatureField (*class in pytext.fields*), 278
DictFeatureField (*class in pytext.fields.dict_field*), 273
dim (*pytext.config.field_config.FloatVectorConfig attribute*), 199
dim_error_check (*pytext.config.field_config.FloatVectorConfig attribute*), 199
DisjointMultitask (*class in pytext.task.disjoint_multitask*), 391
DisjointMultitaskData (*class in pytext.data*), 260
DisjointMultitaskData (*class in pytext.data.disjoint_multitask_data*), 234
DisjointMultitaskDataHandler (*class in pytext.data*), 260
DisjointMultitaskDataHandler (*class in pytext.data.disjoint_multitask_data_handler*), 235
DisjointMultitaskMetricReporter (*class in pytext.metric_reporters.disjoint_multitask_metric_reporter*), 290
DisjointMultitaskModel (*class in pytext.models.disjoint_multitask_model*), 374
dist_init() (*in module pytext.utils.distributed*), 405

distributed_world_size (pytext.config.pytext_config.PyTextConfig attribute), 201
DistributedModel (class in pytext.models.distributed_model), 375
doc_attention (pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute), 358
doc_decoder (pytext.models.decoders.intent_slot_model_decoder.IntentSlotModelDecoder attribute), 316
doc_decoder (pytext.models.decoders.IntentSlotModelDecoder attribute), 320
DOC_LABEL (pytext.common.constants.DFColumn attribute), 193
DOC_LABEL (pytext.config.field_config.Target attribute), 199
DOC_LABEL (pytext.data.contextual_intent_slot_data_handler.RawData attribute), 228
DOC_LABEL (pytext.data.doc_classification_data_handler.RawData attribute), 239
DOC_LABEL (pytext.data.pair_classification_data_handler.RawData attribute), 244
DOC_LABEL (pytext.data.RawData attribute), 266
DOC_LABEL_FIELD (pytext.common.constants.DatasetFieldName attribute), 194
doc_model_deprecated() (in module pytext.config.config_adapter), 197
doc_output (pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer attribute), 340
DOC_WEIGHT (pytext.common.constants.DFColumn attribute), 193
DOC_WEIGHT (pytext.config.contextual_intent_slot.ExtraField attribute), 197
DOC_WEIGHT (pytext.data.contextual_intent_slot_data_handler.RawData attribute), 228
DOC_WEIGHT_FIELD (pytext.common.constants.DatasetFieldName attribute), 194
DocClassificationDataHandler (class in pytext.data), 261
DocClassificationDataHandler (class in pytext.data.doc_classification_data_handler), 238
DocClassificationDataHandlerTest (class in pytext.data.test.doc_classification_data_handler_test), 216
DocClassificationTask (class in pytext.task.tasks), 395
DocLabelConfig (class in pytext.config.field_config), 199
DocLabelField (class in pytext.fields), 279
DocLabelField (class in pytext.fields.field), 274
DocModel_Deprecated (class in pytext.models.doc_model), 376
DocNNRepresentation (class in pytext.models.representations.docnn), 361
DOT (pytext.config.module_config.SlotAttentionType attribute), 200
dropout (pytext.models.representations.bilstm.BiLSTM attribute), 355
dropout (pytext.models.representations.bilstm.BiLSTM.Config attribute), 122
dropout (pytext.models.representations.bilstm_doc_attention.BiLSTMDoc attribute), 356
dropout (pytext.models.representations.bilstm_doc_attention.BiLSTMDoc attribute), 123
dropout (pytext.models.representations.bilstm_doc_slot_attention.BiLST attribute), 357
dropout (pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAtten attribute), 359
dropout (pytext.models.representations.stacked_bidirectional_rnn.Stacke attribute), 368
dummy_model_input (pytext.exporters.exporter.ModelExporter attribute), 268
dummy_model_input (pytext.exporters.ModelExporter attribute), 270
dummy_model_input (pytext.fields.char_field.CharFeatureField attribute), 272
dummy_model_input (pytext.fields.CharFeatureField attribute), 277
dummy_model_input (pytext.fields.dict_field.DictFeatureField attribute), 274
dummy_model_input (pytext.fields.DictFeatureField attribute), 278
dummy_model_input (pytext.fields.field.SeqFeatureField attribute), 275
dummy_model_input (pytext.fields.field.TextFeatureField attribute), 275
dummy_model_input (pytext.fields.SeqFeatureField attribute), 280
dummy_model_input (pytext.fields.TextFeatureField attribute), 279

E

early_stop_after (pytext.trainers.Trainer attribute), 402
early_stop_after (pytext.trainers.trainer.Trainer attribute), 400
elapsed_time (pytext.utils.meter.TimeMeter attribute), 408

Element (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 369
element_from_top() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLSTM method*), 369
embedding (*pytext.models.Model attribute*), 386
embedding (*pytext.models.model.Model attribute*), 379
embedding() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLSTM method*), 369
embedding_dim (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 322
embedding_dim (*pytext.models.embeddings.CharacterEmbedding attribute*), 331
embedding_dim (*pytext.models.embeddings.embedding_base.EmbeddingBase attribute*), 324
embedding_dim (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 325
embedding_dim (*pytext.models.embeddings.EmbeddingBase attribute*), 327
embedding_dim (*pytext.models.embeddings.EmbeddingList attribute*), 328
EmbeddingBase (*class in pytext.models.embeddings*), 327
EmbeddingBase (*class in pytext.models.embeddings.embedding_base*), 324
EmbeddingList (*class in pytext.models.embeddings*), 327
EmbeddingList (*class in pytext.models.embeddings.embedding_list*), 325
EmbedInitStrategy (*class in pytext.config.field_config*), 199
encode_relations (*pytext.models.pair_classification_model.PairwiseClassificationModel.Config attribute*), 118
encode_relations (*pytext.models.representations.pair_rep.PairRepresentation.Config attribute*), 130
end (*pytext.data.data_structures.node.Span attribute*), 205
end (*pytext.data.tokenizers.Token attribute*), 220
end (*pytext.data.tokenizers.tokenizer.Token attribute*), 219
Ensemble (*class in pytext.models.ensembles.ensemble*), 333
EnsembleTask (*class in pytext.task.tasks*), 395
EnsembleTrainer (*class in pytext.trainers*), 403
EnsembleTrainer (*class in pytext.trainers.ensemble_trainer*), 399
EnumTypeError, 202
EOS (*pytext.common.constants.VocabMeta attribute*), 195
EOS_TOKEN (*pytext.common.constants.VocabMeta attribute*), 195
epoch (*pytext.trainers.trainer.TrainingState attribute*), 401
epoch (*pytext.trainers.TrainingState attribute*), 403
epoch (*pytext.trainers.Trainer attribute*), 402
epochs (*pytext.trainers.trainer.Trainer attribute*), 400
epochs_since_last_improvement (*pytext.trainers.trainer.TrainingState attribute*), 401
epochs_since_last_improvement (*pytext.trainers.TrainingState attribute*), 403
eprint() (*in module pytext.utils.documentation*), 405
escape_brackets() (*in module pytext.data.data_structures.annotation*), 204
EVAL (*pytext.common.constants.Stage attribute*), 195
eval (*pytext.data.sources.data_source.DataSource attribute*), 209
eval (*pytext.data.sources.data_source.RootDataSource attribute*), 210
eval (*pytext.data.sources.DataSource attribute*), 213
eval (*pytext.data.sources.squad.SquadDataSource attribute*), 211
eval (*pytext.data.sources.SquadDataSource attribute*), 213
eval (*pytext.data.sources.tsv.MultilingualTSVDataSource attribute*), 212
eval() (*pytext.models.BaseModel method*), 388
eval() (*pytext.models.distributed_model.DistributedModel method*), 375
eval() (*pytext.models.model.BaseModel method*), 378
eval_batch_size (*pytext.data.data_handler.DataHandler attribute*), 232
eval_batch_size (*pytext.data.DataHandler attribute*), 258
eval_path (*pytext.data.data_handler.DataHandler attribute*), 231
eval_path (*pytext.data.DataHandler attribute*), 258
EvalBatchSampler (*class in pytext.data*), 262
EvalBatchSampler (*class in pytext.data.batch_sampler*), 220
example_config() (*pytext.task.tasks.ContextualIntentSlotTask class method*), 395
example_config() (*pytext.task.tasks.EnsembleTask class method*), 395
example_config() (*pytext.task.tasks.JointTextTask*

class method), 396
expected_frame (*pytext.metrics.intent_slot_metrics.FramePredictionPair attribute*), 303
expected_label (*pytext.metrics.LabelPrediction attribute*), 309
expected_nodes (*pytext.metrics.intent_slot_metrics.NodesPredictionPair attribute*), 304
ExponentialLR (*class in pytext.optimizer.scheduler*), 388
export() (*pytext.metric_reporters.Channel method*), 296
export() (*pytext.metric_reporters.channel.Channel method*), 284
export() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 286
export() (*pytext.task.disjoint_multitask.DisjointMultitask method*), 391
export() (*pytext.task.disjoint_multitask.NewDisjointMultitask method*), 392
export() (*pytext.task.task.TaskBase method*), 394
export() (*pytext.task.TaskBase method*), 398
export_caffe2_path (*pytext.config.pytext_config.PyTextConfig attribute*), 201
export_input_names (*pytext.config.field_config.FloatVectorConfig attribute*), 199
export_nets_to_predictor_file() (*in module pytext.utils.onnx*), 409
export_onnx_path (*pytext.config.pytext_config.PyTextConfig attribute*), 201
export_output_names (*pytext.config.field_config.DocLabelConfig attribute*), 199
export_output_names (*pytext.config.field_config.WordLabelConfig attribute*), 199
export_saved_model_to_caffe2() (*in module pytext.workflow*), 411
export_saved_model_to_torchscript() (*in module pytext.workflow*), 411
export_to_caffe2() (*pytext.exporters.exporter.ModelExporter method*), 268
export_to_caffe2() (*pytext.exporters.ModelExporter method*), 270
export_to_caffe2() (*pytext.models.crf.CRF method*), 373
export_to_caffe2() (*pytext.models.output_layers.CRFOutputLayer method*), 348
export_to_caffe2() (*pytext.models.output_layers.doc_classification_output_layer.BinaryDocClassificationOutputLayer method*), 337
export_to_caffe2() (*pytext.models.output_layers.doc_classification_output_layer.MulticlassDocClassificationOutputLayer method*), 338
export_to_caffe2() (*pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer method*), 340
export_to_caffe2() (*pytext.models.output_layers.output_layer_base.OutputLayerBase method*), 342
export_to_caffe2() (*pytext.models.output_layers.OutputLayerBase method*), 347
export_to_caffe2() (*pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer method*), 345
export_to_caffe2() (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method*), 346
export_to_caffe2() (*pytext.models.output_layers.WordTaggingOutputLayer method*), 351
export_to_metrics() (*pytext.exporters.exporter.ModelExporter method*), 269
export_to_metrics() (*pytext.exporters.ModelExporter method*), 271
export_torchscript_path (*pytext.config.pytext_config.PyTextConfig attribute*), 201
EXPORTER (*pytext.config.component.ComponentType attribute*), 196
extra_fields (*pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotDataHandler attribute*), 227
extra_fields (*pytext.data.ContextualIntentSlotModelDataHandler attribute*), 255
extra_fields (*pytext.data.data_handler.DataHandler attribute*), 231
extra_fields (*pytext.data.DataHandler attribute*), 257
ExtraField (*class in pytext.config.contextual_intent_slot*), 197
ExtraField (*class in pytext.config.doc_classification*), 198
ExtraField (*class in pytext.config.pair_classification*), 200

F

f1 (*pytext.metrics.MacroPRF1Scores attribute*), 309
f1 (*pytext.metrics.PRF1Scores attribute*), 310
false_negatives (*pytext.metrics.PRF1Scores attribute*), 310

false_positives (*pytext.metrics.PRF1Scores attribute*), 310
false_postives_upper_bound() (*in module pytext.utils.loss*), 407
FeatureConfig (*class in pytext.config.field_config*), 199
features (*pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotModelDataHandler attribute*), 227
features (*pytext.data.ContextualIntentSlotModelDataHandler attribute*), 255
features (*pytext.data.data_handler.DataHandler attribute*), 231
features (*pytext.data.DataHandler attribute*), 257
featurize() (*pytext.data.featurizer.Featurizer method*), 207
featurize() (*pytext.data.featurizer.featurizer.Featurizer method*), 206
featurize() (*pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method*), 207
featurize() (*pytext.data.featurizer.SimpleFeaturizer method*), 208
featurize() (*pytext.data.joint_data_handler.JointModelDataHandler method*), 240
featurize() (*pytext.data.JointModelDataHandler method*), 263
featurize_batch() (*pytext.data.featurizer.Featurizer method*), 207
featurize_batch() (*pytext.data.featurizer.featurizer.Featurizer method*), 206
featurize_batch() (*pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method*), 207
featurize_batch() (*pytext.data.featurizer.SimpleFeaturizer method*), 208
Featurizer (*class in pytext.data.featurizer*), 207
Featurizer (*class in pytext.data.featurizer.featurizer*), 206
FEATURIZER (*pytext.config.component.ComponentType attribute*), 196
featurizer (*pytext.data.data_handler.DataHandler attribute*), 231
featurizer (*pytext.data.DataHandler attribute*), 257
Field (*class in pytext.fields*), 279
Field (*class in pytext.fields.field*), 274
field_names (*pytext.config.pytext_config.TestConfig attribute*), 201
FieldMeta (*class in pytext.fields*), 279
FieldMeta (*class in pytext.fields.field*), 274
FileChannel (*class in pytext.metric_reporters.channel*), 285
finalize_tree() (*pytext.data.data_structures.annotation.TreeBuilder method*), 204
find_config_class() (*in module pytext.utils.documentation*), 405
find_dicts_containing_key() (*in module pytext.config.config_adapter*), 197
finished() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLSTM method*), 369
flat_str() (*pytext.data.data_structures.annotation.Node method*), 203
flat_str() (*pytext.data.data_structures.annotation.Tree method*), 204
FloatField (*class in pytext.fields*), 279
FloatField (*class in pytext.fields.field*), 275
FloatListTensorizer (*class in pytext.data.tensorizers*), 248
FloatTensor() (*in module pytext.utils.cuda*), 404
FloatVectorConfig (*class in pytext.config.field_config*), 199
FloatVectorField (*class in pytext.fields*), 279
FloatVectorField (*class in pytext.fields.field*), 275
FN (*pytext.metrics.Confusions attribute*), 308
format_prediction() (*pytext.task.task.TaskBase class method*), 394
format_prediction() (*pytext.task.TaskBase class method*), 398
format_prediction() (*pytext.task.tasks.DocClassificationTask class method*), 395
format_prediction() (*pytext.task.tasks.JointTextTask class method*), 396
format_prediction() (*pytext.task.tasks.WordTaggingTask class method*), 397
format_time() (*in module pytext.utils.timing*), 410
forward() (*pytext.loss.AUCPRHingeLoss method*), 282
forward() (*pytext.loss.loss.AUCPRHingeLoss method*), 281
forward() (*pytext.models.crf.CRF method*), 373
forward() (*pytext.models.decoders.decoder_base.DecoderBase method*), 315
forward() (*pytext.models.decoders.DecoderBase method*), 318
forward() (*pytext.models.decoders.intent_slot_model_decoder.IntentSlo method*), 316
forward() (*pytext.models.decoders.IntentSlotModelDecoder method*), 320
forward() (*pytext.models.decoders.mlp_decoder.MLPDecoder method*), 317
forward() (*pytext.models.decoders.mlp_decoder_query_response.MLPD method*), 318
forward() (*pytext.models.decoders.MLPDecoder method*), 319

forward() (pytext.models.disjoint_multitask_model.DisjointMultitaskModel method), 374
forward() (pytext.models.embeddings.char_embedding.CharacterEmbedding method), 322
forward() (pytext.models.embeddings.char_embedding.Highway method), 322
forward() (pytext.models.embeddings.CharacterEmbedding method), 331
forward() (pytext.models.embeddings.contextual_token_embedding.ContextualTokenEmbedding method), 323
forward() (pytext.models.embeddings.ContextualTokenEmbedding method), 332
forward() (pytext.models.embeddings.dict_embedding.DictEmbedding method), 324
forward() (pytext.models.embeddings.DictEmbedding method), 330
forward() (pytext.models.embeddings.embedding_list.EmbeddingList method), 325
forward() (pytext.models.embeddings.EmbeddingList method), 328
forward() (pytext.models.embeddings.word_embedding.WordEmbedding method), 326
forward() (pytext.models.embeddings.WordEmbedding method), 329
forward() (pytext.models.ensembles.bagging_doc_ensemble.BaggingDocEnsemble method), 332
forward() (pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsemble method), 333
forward() (pytext.models.ensembles.BaggingDocEnsemble method), 335
forward() (pytext.models.ensembles.BaggingIntentSlotEnsemble method), 335
forward() (pytext.models.ensembles.ensemble.Ensemble method), 334
forward() (pytext.models.language_models.lmlstm.LMLSTM method), 336
forward() (pytext.models.Model method), 387
forward() (pytext.models.model.Model method), 380
forward() (pytext.models.pair_classification_model.PairwiseClassificationModel method), 383
forward() (pytext.models.query_document_pairwise_ranking_model.QueryDocumentPairwiseRankingModel method), 384
forward() (pytext.models.representations.augmented_lstm.AugmentedLSTM method), 353
forward() (pytext.models.representations.augmented_lstm.AugmentedLSTMCell method), 353
forward() (pytext.models.representations.augmented_lstm.AugmentedLSTMUnidirectional method), 355
forward() (pytext.models.representations.bilstm.BiLSTM method), 356
forward() (pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention method), 357
forward() (pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention method), 358
forward() (pytext.models.representations.bilstm_slot_attn.BiLSTMSlotA method), 359
forward() (pytext.models.representations.biseqcnn.BSeqCNNRepresenta method), 360
forward() (pytext.models.representations.contextual_intent_slot_rep.Co method), 361
forward() (pytext.models.representations.docnn.DocNNRepresentation method), 361
forward() (pytext.models.representations.jointcnn_rep.JointCNNRepres method), 362
forward() (pytext.models.representations.pair_rep.PairRepresentation method), 362
forward() (pytext.models.representations.pass_through.PassThroughRep method), 363
forward() (pytext.models.representations.pooling.BoundaryPool method), 363
forward() (pytext.models.representations.pooling.LastTimestepPool method), 363
forward() (pytext.models.representations.pooling.MaxPool method), 364
forward() (pytext.models.representations.pooling.MeanPool method), 364
forward() (pytext.models.representations.pooling.NoPool method), 364
forward() (pytext.models.representations.pooling.SelfAttention method), 365
forward() (pytext.models.representations.pure_doc_attention.PureDocA method), 365
forward() (pytext.models.representations.query_document_pairwise_ran method), 366
forward() (pytext.models.representations.representation_base.Represent method), 366
forward() (pytext.models.representations.seq_rep.SeqRepresentation method), 366
forward() (pytext.models.representations.slot_attention.SlotAttention method), 367
forward() (pytext.models.representations.stacked_bidirectional_rnn.Stac method), 368
forward() (pytext.models.semantic_parsers.rnng.rnng_data_structures.C method), 369
forward() (pytext.models.semantic_parsers.rnng.rnng_data_structures.C method), 369
forward() (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGPar method), 370
forward() (pytext.utils.loss.LagrangeMultiplier static method), 407
forward_lstm_Augmented_LSTM_Unidirectional attribute), 352
FP (pytext.metrics.Confusions attribute), 308
frame_accuracy (pytext.metrics.intent_slot_metrics.AllMetrics attribute), 302
frame_accuracy (py-

text.metrics.intent_slot_metrics.AllMetrics attribute), 302
frame_accuracy (*pytext.metrics.intent_slot_metrics.FrameAccuracy attribute*), 302
frame_accuracy_top_k (*pytext.metrics.intent_slot_metrics.AllMetrics attribute*), 302
FrameAccuraciesByDepth (*in module pytext.metrics.intent_slot_metrics*), 302
FrameAccuracy (*class in pytext.metrics.intent_slot_metrics*), 302
FramePredictionPair (*class in pytext.metrics.intent_slot_metrics*), 303
freeze() (*pytext.models.embeddings.word_embedding.WordEmbedding method*), 326
freeze() (*pytext.models.embeddings.WordEmbedding method*), 329
freeze() (*pytext.models.module.Module method*), 381
from_config() (*pytext.config.component.Component class method*), 195
from_config() (*pytext.data.BaseBatchSampler class method*), 252
from_config() (*pytext.data.batch_sampler.BaseBatchSampler class method*), 220
from_config() (*pytext.data.batch_sampler.RandomizedBatchSampler class method*), 221
from_config() (*pytext.data.batch_sampler.RoundRobinBatchSampler class method*), 221
from_config() (*pytext.data.Batcher class method*), 252
from_config() (*pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler class method*), 222
from_config() (*pytext.data.BPTTLanguageModelDataHandler class method*), 253
from_config() (*pytext.data.compositional_data_handler.CompositionalDataHandler class method*), 225
from_config() (*pytext.data.CompositionalDataHandler class method*), 254
from_config() (*pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotMockedDataHandler class method*), 227
from_config() (*pytext.data.ContextualIntentSlotModelDataHandler class method*), 255
from_config() (*pytext.data.Data class method*), 256
from_config() (*pytext.data.data.Batcher class method*), 228
from_config() (*pytext.data.data.Data class method*), 229
from_config() (*pytext.data.data.PoolingBatcher class method*), 229
from_config() (*pytext.data.disjoint_multitask_data.DisjointMultitaskData class method*), 234
from_config() (*pytext.data.DisjointMultitaskData class method*), 260
from_config() (*pytext.data.doc_classification_data_handler.DocClassificationData class method*), 239
from_config() (*pytext.data.DocClassificationDataHandler class method*), 262
from_config() (*pytext.data.featurizer.Featurizer class method*), 207
from_config() (*pytext.data.featurizer.featurizer.Featurizer class method*), 206
from_config() (*pytext.data.joint_data_handler.JointModelDataHandler class method*), 240
from_config() (*pytext.data.JointModelDataHandler class method*), 263
from_config() (*pytext.data.language_model_data_handler.LanguageModelDataHa class method*), 242
from_config() (*pytext.data.LanguageModelDataHandler class method*), 263
from_config() (*pytext.data.pair_classification_data_handler.PairClassificationData class method*), 244
from_config() (*pytext.data.PairClassificationDataHandler class method*), 264
from_config() (*pytext.data.PoolingBatcher class method*), 265
from_config() (*pytext.data.query_document_pairwise_ranking_data_handler.Query class method*), 246
from_config() (*pytext.data.QueryDocumentPairwiseRankingDataHandler class method*), 266
from_config() (*pytext.data.RoundRobinBatchSampler class method*), 265
from_config() (*pytext.data.RoundRobinBatchSampler class method*), 267
from_config() (*pytext.data.seq_data_handler.SeqModelDataHandler*

class method), 246
from_config() (*pytext.data.SeqModelDataHandler class method*), 267
from_config() (*pytext.data.sources.squad.SquadDataSource class method*), 211
from_config() (*pytext.data.sources.SquadDataSource class method*), 213
from_config() (*pytext.data.sources.tsv.TSVDataSource class method*), 213
from_config() (*pytext.data.sources.TSVDataSource class method*), 214
from_config() (*pytext.data.tensorizers.ByteTensorizer class method*), 247
from_config() (*pytext.data.tensorizers.FloatListTensorizer class method*), 248
from_config() (*pytext.data.tensorizers.JoinStringTensorizer class method*), 248
from_config() (*pytext.data.tensorizers.LabelTensorizer class method*), 248
from_config() (*pytext.data.tensorizers.MetricTensorizer class method*), 248
from_config() (*pytext.data.tensorizers.NumericLabelTensorizer class method*), 249
from_config() (*pytext.data.tensorizers.RawString class method*), 249
from_config() (*pytext.data.tensorizers.TokenTensorizer class method*), 250
from_config() (*pytext.data.tensorizers.WordLabelTensorizer class method*), 251
from_config() (*pytext.data.tokenizers.Tokenizer class method*), 220
from_config() (*pytext.data.tokenizers.tokenizer.Tokenizer class method*), 219
from_config() (*pytext.exporters.exporter.ModelExporter class method*), 269
from_config() (*pytext.exporters.ModelExporter class method*), 271
from_config() (*pytext.fields.Field class method*), 279
from_config() (*pytext.fields.field.Field class method*), 274
from_config() (*pytext.metric_reporters.classification_metric_reporter.Classificatio class method*), 288
from_config() (*pytext.metric_reporters.ClassificationMetricReporter class method*), 298
from_config() (*pytext.metric_reporters.compositional_metric_reporter.Compositio class method*), 289
from_config() (*pytext.metric_reporters.CompositionalMetricReporter class method*), 300
from_config() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.Inten class method*), 291
from_config() (*pytext.metric_reporters.IntentSlotMetricReporter class method*), 299
from_config() (*pytext.metric_reporters.language_model_metric_reporter.Language class method*), 292
from_config() (*pytext.metric_reporters.language_model_metric_reporter.MaskedL class method*), 292
from_config() (*pytext.metric_reporters.LanguageModelMetricReporter class method*), 300
from_config() (*pytext.metric_reporters.pairwise_ranking_metric_reporter.Pairwise class method*), 295
from_config() (*pytext.metric_reporters.PairwiseRankingMetricReporter class method*), 301
from_config() (*pytext.metric_reporters.regression_metric_reporter.RegressionMetr class method*), 295
from_config() (*pytext.metric_reporters.RegressionMetricReporter class method*), 299
from_config() (*pytext.metric_reporters.SimpleWordTaggingMetricReporter class method*), 301
from_config() (*pytext.metric_reporters.word_tagging_metric_reporter.SimpleWord class method*), 295
from_config() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTaggin class method*), 296
from_config() (*pytext.metric_reporters.WordTaggingMetricReporter class method*), 300
from_config() (*pytext.models.doc_model.NewDocModel class method*), 377

from_config() (pytext.models.doc_model.NewDocRegressionModel class method), 377
from_config() (pytext.models.embeddings.char_embedding.CharacterEmbedding class method), 322
from_config() (pytext.models.embeddings.CharacterEmbedding class method), 331
from_config() (pytext.models.embeddings.contextual_token_embedding.ContextualTokenEmbedding class method), 323
from_config() (pytext.models.embeddings.ContextualTokenEmbedding class method), 332
from_config() (pytext.models.embeddings.dict_embedding.DictEmbedding class method), 324
from_config() (pytext.models.embeddings.DictEmbedding class method), 330
from_config() (pytext.models.embeddings.word_embedding.WordEmbedding class method), 327
from_config() (pytext.models.embeddings.WordEmbedding class method), 329
from_config() (pytext.models.ensembles.ensemble.Ensemble class method), 334
from_config() (pytext.models.joint_model.JointModel class method), 377
from_config() (pytext.models.language_models.lmlstm.LMLSTM class method), 336
from_config() (pytext.models.Model class method), 387
from_config() (pytext.models.model.Model class method), 380
from_config() (pytext.models.output_layers.ClassificationOutputLayer class method), 349
from_config() (pytext.models.output_layers.CRFOutputLayer class method), 348
from_config() (pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer class method), 337
from_config() (pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer class method), 339
from_config() (pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer class method), 340
from_config() (pytext.models.output_layers.lm_output_layer.LMOutputLayer class method), 341
from_config() (pytext.models.output_layers.pairwise_ranking_output_layer.PairwiseRankingOutputLayer class method), 344
from_config() (pytext.models.output_layers.PairwiseRankingOutputLayer class method), 351
from_config() (pytext.models.output_layers.RegressionOutputLayer class method), 350
from_config() (pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer class method), 345
from_config() (pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer class method), 346
from_config() (pytext.models.output_layers.WordTaggingOutputLayer class method), 351
from_config() (pytext.models.pair_classification_model.PairwiseClassificationModel class method), 383
from_config() (pytext.models.query_document_pairwise_ranking_model.QueryDocumentPairwiseRankingModel class method), 385
from_config() (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser class method), 371
from_config() (pytext.models.word_model.NewWordTaggingModel class method), 385
from_config() (pytext.optimizer.Adam class method), 390
from_config() (pytext.optimizer.scheduler.CosineAnnealingLR class method), 388
from_config() (pytext.optimizer.scheduler.ExponentialLR class method), 389
from_config() (pytext.optimizer.scheduler.LmFineTuning class method), 389
from_config() (pytext.optimizer.scheduler.ReduceLROnPlateau class method), 389
from_config() (pytext.optimizer.scheduler.StepLR class method), 390
from_config() (pytext.optimizer.scheduler.WarmupScheduler class method), 390
from_config() (pytext.optimizer.SGD class method),

391
from_config() (pytext.task.disjoint_multitask.DisjointMultitask class method), 391
from_config() (pytext.task.disjoint_multitask.NewDisjointMultitask class method), 392
from_config() (pytext.task.task.TaskBase class method), 394
from_config() (pytext.task.TaskBase class method), 398
from_config() (pytext.trainers.ensemble_trainer.EnsembleTrainer class method), 399
from_config() (pytext.trainers.EnsembleTrainer class method), 403
from_config() (pytext.trainers.hogwild_trainer.HogwildTrainer class method), 399
from_config() (pytext.trainers.HogwildTrainer class method), 404
from_config() (pytext.trainers.Trainer class method), 402
from_config() (pytext.trainers.trainer.Trainer class method), 400
from_config_and_label_names() (pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter class method), 288
from_config_and_label_names() (pytext.metric_reporters.ClassificationMetricReporter class method), 298
FULL_FEATURES (pytext.data.compositional_data_handler.CompositionalDataHandler attribute), 225
FULL_FEATURES (pytext.data.CompositionalDataHandler attribute), 254
FULL_FEATURES (pytext.data.seq_data_handler.SeqModelDataHandler attribute), 246
FULL_FEATURES (pytext.data.SeqModelDataHandler attribute), 267

G

gazetteer_feat_lengths (pytext.data.featurizer.featurizer.OutputRecord attribute), 206
gazetteer_feat_lengths (pytext.data.featurizer.OutputRecord attribute), 208
gazetteer_feat_weights (pytext.data.featurizer.featurizer.OutputRecord attribute), 206
gazetteer_feat_weights (pytext.data.featurizer.OutputRecord attribute), 208
gazetteer_feats (pytext.data.featurizer.featurizer.OutputRecord attribute), 206
gazetteer_feats (pytext.data.featurizer.OutputRecord attribute), 208
gen_additional_blobs() (pytext.models.output_layers.OutputLayerUtils static method), 352
gen_additional_blobs() (pytext.models.output_layers.utils.OutputLayerUtils static method), 344
gen_config_impl() (in module pytext.main), 411
gen_content() (pytext.metric_reporters.channel.FileChannel method), 285
gen_content() (pytext.metric_reporters.classification_metric_reporter.IntentModelC method), 289
gen_content() (pytext.metric_reporters.compositional_metric_reporter.Composition method), 289
gen_content() (pytext.metric_reporters.intent_slot_detection_metric_reporter.Intent method), 291
gen_content() (pytext.metric_reporters.language_model_metric_reporter.Language method), 292
gen_dataset() (pytext.data.data_handler.DataHandler method), 232
gen_dataset() (pytext.data.DataHandler method), 258
gen_dataset_from_path() (pytext.data.data_handler.DataHandler method), 232
gen_dataset_from_path() (pytext.data.DataHandler method), 258
gen_extra_context() (pytext.metric_reporters.compositional_metric_reporter.Composition method), 289
gen_extra_context() (pytext.metric_reporters.CompositionalMetricReporter method), 300
gen_extra_context() (pytext.metric_reporters.intent_slot_detection_metric_reporter.Intent method), 291
gen_extra_context() (pytext.metric_reporters.IntentSlotMetricReporter method), 299
gen_extra_context() (py-

text.metric_reporters.metric_reporter.MetricReporter method), 294
gen_extra_context() (*pytext.metric_reporters.MetricReporter method*), 298
generator_iterator() (*in module pytext.data*), 262
generator_iterator() (*in module pytext.data.data*), 229
generator_property (*in module pytext.data.sources.data_source*), 211
GeneratorIterator (*class in pytext.data.sources.data_source*), 209
GeneratorMethodProperty (*class in pytext.data.sources.data_source*), 209
get() (*pytext.config.component.Registry class method*), 196
get_class_members_recursive() (*in module pytext.utils.documentation*), 405
get_component_name() (*in module pytext.config.component*), 197
get_config_fields() (*in module pytext.utils.documentation*), 405
get_decoder() (*pytext.models.decoders.decoder_base.DecoderBase method*), 315
get_decoder() (*pytext.models.decoders.DecoderBase method*), 319
get_decoder() (*pytext.models.decoders.intent_slot_model_decoder.IntentSlotModelDecoder method*), 317
get_decoder() (*pytext.models.decoders.IntentSlotModelDecoder method*), 320
get_decoder() (*pytext.models.decoders.mlp_decoder.MLPDecoder method*), 317
get_decoder() (*pytext.models.decoders.mlp_decoder_query_response.MLPDecoderQueryResponse method*), 318
get_decoder() (*pytext.models.decoders.MLPDecoder method*), 319
get_depth() (*pytext.data.data_structures.node.Node method*), 205
get_dropout_mask() (*pytext.models.representations.augmented_lstm.AugmentedLSTMUnidirectional method*), 355
get_eval_iter() (*pytext.data.data_handler.DataHandler method*), 232
get_eval_iter() (*pytext.data.DataHandler method*), 258
get_eval_iter() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH method*), 235
get_eval_iter() (*pytext.data.DisjointMultitaskDataHandler method*), 261
get_export_input_names() (*pytext.models.doc_model.NewDocModel method*), 377
get_export_output_names() (*pytext.models.doc_model.NewDocModel method*), 377
get_extra_params() (*pytext.exporters.exporter.ModelExporter method*), 269
get_extra_params() (*pytext.exporters.ModelExporter method*), 271
get_feature_metadata() (*pytext.exporters.custom_exporters.DenseFeatureExporter class method*), 268
get_feature_metadata() (*pytext.exporters.DenseFeatureExporter class method*), 272
get_feature_metadata() (*pytext.exporters.exporter.ModelExporter class method*), 269
get_feature_metadata() (*pytext.exporters.ModelExporter class method*), 271
get_in_dim() (*pytext.models.decoders.decoder_base.DecoderBase method*), 315
get_in_dim() (*pytext.models.decoders.DecoderBase method*), 319
get_info() (*pytext.data.data_structures.annotation.Node method*), 203
get_logits() (*in module pytext.workflow*), 411
get_loss() (*pytext.models.BaseModel method*), 388
get_loss() (*pytext.models.disjoint_multitask_model.DisjointMultitaskM method*), 374
get_loss() (*pytext.models.model.BaseModel method*), 378
get_loss() (*pytext.models.output_layers.CRFOutputLayer method*), 348
get_loss() (*pytext.models.output_layers.doc_regression_output_layer.R method*), 339
get_loss() (*pytext.models.output_layers.intent_slot_output_layer.Intent method*), 340
get_loss() (*pytext.models.output_layers.lm_output_layer.LMOutputLay method*), 341
get_loss() (*pytext.models.output_layers.output_layer_base.OutputLaye method*), 343
get_loss() (*pytext.models.output_layers.OutputLayerBase method*), 347
get_loss() (*pytext.models.output_layers.RegressionOutputLayer*

method), 350
get_loss() (*pytext.models.output_layers.word_tagging_output_layer.CRFOutputLayer method*), 345
get_loss() (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method*), 346
get_loss() (*pytext.models.output_layers.WordTaggingOutputLayer method*), 351
get_loss() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 371
get_lr() (*pytext.optimizer.scheduler.LmFineTuning method*), 389
get_lr() (*pytext.optimizer.scheduler.WarmupScheduler method*), 390
get_meta() (*pytext.fields.DocLabelField method*), 279
get_meta() (*pytext.fields.Field method*), 279
get_meta() (*pytext.fields.field.DocLabelField method*), 274
get_meta() (*pytext.fields.field.Field method*), 274
get_meta() (*pytext.fields.field.NestedField method*), 275
get_meta() (*pytext.fields.field.RawField method*), 275
get_meta() (*pytext.fields.field.WordLabelField method*), 276
get_meta() (*pytext.fields.NestedField method*), 280
get_meta() (*pytext.fields.RawField method*), 279
get_meta() (*pytext.fields.WordLabelField method*), 280
get_meta() (*pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method*), 288
get_meta() (*pytext.metric_reporters.ClassificationMetricReporter method*), 299
get_meta() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 294
get_meta() (*pytext.metric_reporters.MetricReporter method*), 298
get_mlp() (*pytext.models.decoders.mlp_decoder_query_response.MLPDecoderQueryResponse static method*), 318
get_model_select_metric() (*pytext.metric_reporters.classification_metric_reporter.ClassificationMetricReporter method*), 288
get_model_select_metric() (*pytext.metric_reporters.ClassificationMetricReporter method*), 299
get_model_select_metric() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter method*), 289
get_model_select_metric() (*pytext.metric_reporters.CompositionalMetricReporter method*), 300
get_model_select_metric() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter method*), 290
get_model_select_metric() (*pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method*), 291
get_model_select_metric() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 299
get_model_select_metric() (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter method*), 292
get_model_select_metric() (*pytext.metric_reporters.LanguageModelMetricReporter method*), 300
get_model_select_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 294
get_model_select_metric() (*pytext.metric_reporters.MetricReporter method*), 298
get_model_select_metric() (*pytext.metric_reporters.pairwise_ranking_metric_reporter.PairwiseRankingMetricReporter static method*), 295
get_model_select_metric() (*pytext.metric_reporters.PairwiseRankingMetricReporter static method*), 301
get_model_select_metric() (*pytext.metric_reporters.regression_metric_reporter.RegressionMetricReporter method*), 295
get_model_select_metric() (*pytext.metric_reporters.RegressionMetricReporter method*), 299
get_model_select_metric() (*pytext.metric_reporters.SimpleWordTaggingMetricReporter static method*), 301
get_model_select_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.SimpleWordTaggingMetricReporter static method*), 295
get_model_select_metric() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTaggingMetricReporter method*), 296
get_model_select_metric() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 300
get_numericalize_net() (*in module pytext.utils.onnx*), 409
get_out_dim() (*pytext.models.decoders.decoder_base.DecoderBase method*), 315
get_out_dim() (*pytext.models.decoders.DecoderBase method*), 319
get_param_groups_for_optimizer() (*pytext.models.BaseModel method*), 388
get_param_groups_for_optimizer() (*pytext.models.embeddings.embedding_base.EmbeddingBase method*), 325
get_param_groups_for_optimizer() (*py-*

text.models.embeddings.embedding_list.EmbeddingList method), 326
get_param_groups_for_optimizer() (*pytext.models.embeddings.EmbeddingBase method*), 327
get_param_groups_for_optimizer() (*pytext.models.embeddings.EmbeddingList method*), 328
get_param_groups_for_optimizer() (*pytext.models.model.BaseModel method*), 378
get_param_groups_for_optimizer() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 371
get_parent() (*pytext.data.data_structures.annotation.Node_Info method*), 203
get_parent() (*pytext.data.data_structures.annotation.Token_Info method*), 204
get_pred() (*pytext.models.BaseModel method*), 388
get_pred() (*pytext.models.disjoint_multitask_model.DisjointMultitaskModel method*), 374
get_pred() (*pytext.models.model.BaseModel method*), 378
get_pred() (*pytext.models.output_layers.ClassificationOutputLayer method*), 349
get_pred() (*pytext.models.output_layers.CRFOutputLayer method*), 349
get_pred() (*pytext.models.output_layers.doc_classification_output_layer.BinaryClassificationOutputLayer method*), 337
get_pred() (*pytext.models.output_layers.doc_classification_output_layer.ClassificationOutputLayer method*), 338
get_pred() (*pytext.models.output_layers.doc_classification_output_layer.MultiLabelClassificationOutputLayer method*), 338
get_pred() (*pytext.models.output_layers.doc_regression_output_layer.RegressionOutputLayer method*), 339
get_pred() (*pytext.models.output_layers.intent_slot_output_layer.IntentSlotOutputLayer method*), 341
get_pred() (*pytext.models.output_layers.lm_output_layer.LMOutputLayer method*), 342
get_pred() (*pytext.models.output_layers.output_layer_base.OutputLayerBase method*), 343
get_pred() (*pytext.models.output_layers.OutputLayerBase method*), 348
get_pred() (*pytext.models.output_layers.pairwise_ranking_output_layer.PairwiseRankingOutputLayer method*), 344
get_pred() (*pytext.models.output_layers.PairwiseRankingOutputLayer method*), 351
get_pred() (*pytext.models.output_layers.RegressionOutputLayer method*), 350
get_pred() (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method*), 345
get_pred() (*pytext.models.output_layers.word_tagging_output_layer.WordTaggingOutputLayer method*), 346
get_pred() (*pytext.models.output_layers.WordTaggingOutputLayer method*), 351
get_pred() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 371
get_predict_iter() (*pytext.data.data_handler.DataHandler method*), 232
get_predict_iter() (*pytext.data.DataHandler method*), 258
get_representation_dim() (*pytext.models.representations.representation_base.RepresentationBase method*), 366
get_same_span() (*pytext.data.data_structures.annotation.Node_Info method*), 203
get_sentence_markers() (*pytext.data.featurizer.Featurizer method*), 207
get_sentence_markers() (*pytext.data.featurizer.featurizer.Featurizer method*), 206
get_sentence_markers() (*pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method*), 207
get_sentence_markers() (*pytext.data.featurizer.SimpleFeaturizer method*), 208
get_shard_range() (*in module pytext.utils.distributed*), 405
get_similarities() (*pytext.loss.loss.PairwiseRankingLoss static method*), 283
get_similarities() (*pytext.loss.PairwiseRankingLoss static method*), 284
get_slot_labels() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 371
get_slots() (*in module pytext.metric_reporters.word_tagging_metric_reporter*), 296
get_summary() (*in module pytext.utils.documentation*), 406
get_test_iter() (*pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler method*), 232
get_test_iter() (*pytext.data.BPTTLanguageModelDataHandler method*), 253
get_test_iter() (*pytext.data.data_handler.DataHandler method*), 232
get_test_iter() (*pytext.data.DataHandler method*), 258
get_test_iter() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler method*), 235

get_test_iter() (pytext.data.DisjointMultitaskDataHandler method), 261
get_test_iter_from_path() (pytext.data.data_handler.DataHandler method), 232
get_test_iter_from_path() (pytext.data.DataHandler method), 259
get_test_iter_from_raw_data() (pytext.data.data_handler.DataHandler method), 232
get_test_iter_from_raw_data() (pytext.data.DataHandler method), 259
get_title() (pytext.metric_reporters.channel.FileChannel method), 285
get_title() (pytext.metric_reporters.classification_metric_reporter.IntentModelChannel method), 289
get_title() (pytext.metric_reporters.compositional_metric_reporter.CompositionalFileChannel method), 289
get_title() (pytext.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotChannel method), 291
get_title() (pytext.metric_reporters.language_model_metric_reporter.LanguageModelChannel method), 292
get_token_indices() (pytext.data.data_structures.annotation.Node method), 203
get_token_span() (pytext.data.data_structures.annotation.Node method), 203
get_train_iter() (pytext.data.data_handler.DataHandler method), 232
get_train_iter() (pytext.data.DataHandler method), 259
get_train_iter() (pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler method), 235
get_train_iter() (pytext.data.DisjointMultitaskDataHandler method), 261
get_train_iter_from_path() (pytext.data.data_handler.DataHandler method), 232
get_train_iter_from_path() (pytext.data.DataHandler method), 259
get_train_iter_from_raw_data() (pytext.data.data_handler.DataHandler method), 233
get_train_iter_from_raw_data() (pytext.data.DataHandler method), 259
get_transitions() (pytext.models.crf.CRF method), 374
GetTensor() (in module pytext.utils.cuda), 404
GRU (pytext.models.representations.stacked_bidirectional_rnn.RnnType attribute), 367

H

hidden_dims (pytext.models.decoders.mlp_decoder.MLPDecoder attribute), 317
hidden_dims (pytext.models.decoders.mlp_decoder.MLPDecoder.Config attribute), 89
hidden_dims (pytext.models.decoders.MLPDecoder attribute), 319
HierarchicalTimer (class in pytext.utils.timing), 410
Highway (class in pytext.models.embeddings.char_embedding), 322
highway_layers (pytext.models.embeddings.char_embedding.CharacterEmbedding attribute), 321
highway_layers (pytext.models.embeddings.CharacterEmbedding attribute), 334
HogwildTrainer (class in pytext.trainers), 403
hogwild_trainer (class in pytext.trainers.hogwild_trainer), 399

I

I_LABEL_PREFIX (pytext.utils.data.Slot attribute), 404
IGNORE_LOSS (pytext.common.constants.BatchContext attribute), 193
ignore_loss_for_unsupported (pytext.models.semantic_parsers.rnng.rnng_parser.RNNGConstraint attribute), 139
import_tests_module() (in module pytext.utils.test), 409
in_dim (pytext.models.decoders.decoder_base.DecoderBase attribute), 315
in_dim (pytext.models.decoders.DecoderBase attribute), 318
IncorrectTypeError, 202
INDEX (pytext.common.constants.BatchContext attribute), 193
init_feature_metadata() (pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler method), 222
init_feature_metadata() (pytext.data.BPTTLanguageModelDataHandler method), 253
init_feature_metadata() (pytext.data.data_handler.DataHandler method), 233
init_feature_metadata() (pytext.data.DataHandler method), 259
init_hidden() (pytext.models.language_models.lmlstm.LMLSTM

method), 336
init_metadata() (*pytext.data.data_handler.DataHandler method*), 233
init_metadata() (*pytext.data.DataHandler method*), 259
init_metadata() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler method*), 235
init_metadata() (*pytext.data.DisjointMultitaskDataHandler method*), 261
init_metadata_from_path() (*pytext.data.data_handler.DataHandler method*), 233
init_metadata_from_path() (*pytext.data.DataHandler method*), 259
init_metadata_from_raw_data() (*pytext.data.data_handler.DataHandler method*), 233
init_metadata_from_raw_data() (*pytext.data.DataHandler method*), 259
INIT_SEQ (*pytext.common.constants.VocabMeta attribute*), 195
init_target_metadata() (*pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler method*), 222
init_target_metadata() (*pytext.data.BPTTLanguageModelDataHandler method*), 253
init_target_metadata() (*pytext.data.data_handler.DataHandler method*), 233
init_target_metadata() (*pytext.data.DataHandler method*), 259
init_target_metadata() (*pytext.data.language_model_data_handler.LanguageModelDataHandler method*), 242
init_target_metadata() (*pytext.data.LanguageModelDataHandler method*), 264
INIT_TOKEN (*pytext.common.constants.VocabMeta attribute*), 195
init_weights() (*pytext.models.representations.pooling.SelfAttention method*), 365
initialize() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 247
initialize() (*pytext.data.tensorizers.LabelTensorizer method*), 248
initialize() (*pytext.data.tensorizers.Tensorizer method*), 249
initialize() (*pytext.data.tensorizers.TokenTensorizer method*), 250
initialize() (*pytext.data.tensorizers.WordLabelTensorizer method*), 251
initialize_embeddings_weights() (*pytext.utils.embeddings.PretrainedEmbedding method*), 406
initialize_tensorizers() (*in module pytext.data.tensorizers*), 251
input (*pytext.models.representations.augmented_lstm.AugmentedLSTMCell attribute*), 353
input_names (*pytext.exporters.exporter.ModelExporter attribute*), 268
input_names (*pytext.exporters.ModelExporter attribute*), 270
input_start_indices (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 325
input_start_indices (*pytext.models.embeddings.EmbeddingList attribute*), 328
InputRecord (*class in pytext.data.featurizer*), 207
InputRecord (*class in pytext.data.featurizer.featurizer*), 206
Intent (*class in pytext.data.data_structures.annotation*), 203
IntentConfusions (*pytext.metrics.intent_slot_metrics.IntentSlotConfusions attribute*), 303
intent_metrics (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics attribute*), 303
intent_slot_nesting (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGConstraint attribute*), 139
IntentModelChannel (*class in pytext.metric_reporters.classification_metric_reporter*), 288
IntentsAndSlots (*pytext.metrics.intent_slot_metrics.IntentsAndSlots attribute*), 303
IntentsAndSlots (*class in pytext.metrics.intent_slot_metrics*), 303
IntentSlotChannel (*class in pytext.metric_reporters.intent_slot_detection_metric_reporter*), 291
IntentSlotConfusions (*class in pytext.metrics.intent_slot_metrics*), 303
IntentSlotMetricReporter (*class in pytext.metric_reporters*), 299
IntentSlotMetricReporter (*class in pytext.metric_reporters.intent_slot_detection_metric_reporter*), 291
IntentSlotMetrics (*class in pytext.metrics.intent_slot_metrics*), 303
IntentSlotModelDecoder (*class in py-*

text.models.decoders), 319
IntentSlotModelDecoder (*class in pytext.models.decoders.intent_slot_model_decoder*), 316
IntentSlotOutputLayer (*class in pytext.models.output_layers.intent_slot_output_layer*), 340
is_intent_nonterminal() (*in module pytext.data.data_structures.annotation*), 204
is_number() (*in module pytext.utils.data*), 405
is_slot_nonterminal() (*in module pytext.data.data_structures.annotation*), 204
is_unsupported() (*in module pytext.data.data_structures.annotation*), 204
is_valid_nonterminal() (*in module pytext.data.data_structures.annotation*), 204
items() (*pytext.config.pytext_config.ConfigBase method*), 200
iter_to_set_epoch (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatchIterator attribute*), 237
iterators (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatchIterator attribute*), 236

J

JoinStringTensorizer (*class in pytext.data.tensorizers*), 248
JointCNNRepresentation (*class in pytext.models.representations.jointcnn_rep*), 362
JointDataHandlerTest (*class in pytext.data.test.joint_data_handler_test*), 216
JointModel (*class in pytext.models.joint_model*), 377
JointModelDataHandler (*class in pytext.data*), 262
JointModelDataHandler (*class in pytext.data.joint_data_handler*), 240
JointTextTask (*class in pytext.task.tasks*), 395

K

KDDocClassificationDataHandlerTest (*class in pytext.data.test.kd_doc_classification_data_handler_test*), 216
kernel_num (*pytext.config.module_config.CNNParams attribute*), 200
kernel_sizes (*pytext.config.module_config.CNNParams attribute*), 200
KLDivergenceBCELoss (*class in pytext.loss*), 283
KLDivergenceBCELoss (*class in pytext.loss.loss*), 281
KLDivergenceCELoss (*class in pytext.loss*), 283
KLDivergenceCELoss (*class in pytext.loss.loss*), 281

L

label (*pytext.data.data_structures.node.Node attribute*), 205
label (*pytext.metrics.intent_slot_metrics.Node attribute*), 304
LABEL_AVG_PRECISION (*pytext.metric_reporters.classification_metric_reporter.ComparableC attribute*), 289
label_confusions_map (*pytext.metrics.PerLabelConfusions attribute*), 311
LABEL_F1 (*pytext.metric_reporters.classification_metric_reporter.Compar attribute*), 289
LABEL_ROC_AUC (*pytext.metric_reporters.classification_metric_reporter.ComparableC attribute*), 289
label_scores (*pytext.metrics.LabelPrediction attribute*), 308, 309
label_weights (*pytext.config.field_config.DocLabelConfig attribute*), 199
LabelPrediction (*class in pytext.metrics*), 308
labels (*pytext.data.contextual_intent_slot_data_handler.ContextualIntent attribute*), 227
labels (*pytext.data.ContextualIntentSlotModelDataHandler attribute*), 255
labels (*pytext.data.data_handler.DataHandler attribute*), 231
labels (*pytext.data.DataHandler attribute*), 257
LabelSmoothedCrossEntropyLoss (*class in pytext.loss*), 284
LabelSmoothedCrossEntropyLoss (*class in pytext.loss.loss*), 282
LabelTensorizer (*class in pytext.data.tensorizers*), 248
lagrange_multiplier() (*in module pytext.utils.loss*), 407
LagrangeMultiplier (*class in pytext.utils.loss*), 406
LANGUAGE_ID (*pytext.common.constants.DFColumn attribute*), 193
LANGUAGE_ID_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
LanguageModelChannel (*class in pytext.metric_reporters.language_model_metric_reporter*), 292
LanguageModelDataHandler (*class in pytext.data*), 263
LanguageModelDataHandler (*class in pytext.data.language_model_data_handler*), 241
LanguageModelDataHandlerTest (*class in pytext.data.test.language_model_data_handler_test*), 217

LanguageModelMetric (*class in pytext.metrics.language_model_metrics*), 307
LanguageModelMetricReporter (*class in pytext.metric_reporters*), 299
LanguageModelMetricReporter (*class in pytext.metric_reporters.language_model_metric_reporter*), 292
LastTimestepPool (*class in pytext.models.representations.pooling*), 363
learning_rates() (*in module pytext.optimizer*), 391
list_ancestors() (*pytext.data.data_structures.annotation.Node method*), 203
list_from_actions() (*in module pytext.data.data_structures.annotation*), 204
list_nonTerminals() (*pytext.data.data_structures.annotation.Node method*), 203
list_terminals() (*pytext.data.data_structures.annotation.Node method*), 203
list_tokens() (*pytext.data.data_structures.annotation.Node method*), 203
list_tokens() (*pytext.data.data_structures.annotation.Tree method*), 204
LmFineTuning (*class in pytext.optimizer.scheduler*), 389
LMLSTM (*class in pytext.models.language_models.lmlstm*), 335
LMOutputLayer (*class in pytext.models.output_layers.lm_output_layer*), 341
LMTask (*class in pytext.task.tasks*), 396
load() (*in module pytext.task*), 399
load() (*in module pytext.task.serialize*), 393
load() (*pytext.data.sources.data_source.RootDataSource method*), 210
load_best_model() (*pytext.trainers.Trainer method*), 402
load_best_model() (*pytext.trainers.trainer.Trainer method*), 400
load_cached_embeddings() (*pytext.utils.embeddings.PretrainedEmbedding method*), 406
load_config() (*in module pytext*), 412
load_meta() (*pytext.fields.Field method*), 279
load_meta() (*pytext.fields.field.Field method*), 274
load_meta() (*pytext.fields.field.NestedField method*), 275
load_meta() (*pytext.fields.NestedField method*), 280
load_metadata() (*pytext.data.data_handler.DataHandler method*), 233
load_metadata() (*pytext.data.DataHandler method*), 259
load_metadata() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH method*), 236
load_metadata() (*pytext.data.DisjointMultitaskDataHandler method*), 261
load_pretrained_embeddings() (*pytext.utils.embeddings.PretrainedEmbedding method*), 406
load_slots() (*in module pytext.data.sources.data_source*), 211
load_snapshot_path (*pytext.config.pytext_config.PyTextConfig attribute*), 201
load_state_dict() (*pytext.models.distributed_model.DistributedModel method*), 375
load_text() (*in module pytext.data.sources.data_source*), 211
load_vocab() (*pytext.data.data_handler.DataHandler method*), 233
load_vocab() (*pytext.data.DataHandler method*), 259
locale (*pytext.data.featurizer.featurizer.InputRecord attribute*), 206
locale (*pytext.data.featurizer.InputRecord attribute*), 208
LongTensor() (*in module pytext.utils.cuda*), 404
lookup_all() (*pytext.data.utils.Vocabulary method*), 251
Loss (*class in pytext.loss*), 283
Loss (*class in pytext.loss.loss*), 282
LOSS (*pytext.config.component.ComponentType attribute*), 196
loss (*pytext.metrics.ClassificationMetrics attribute*), 308
loss (*pytext.metrics.intent_slot_metrics.AllMetrics attribute*), 302
loss_fn (*pytext.models.output_layers.ClassificationOutputLayer attribute*), 349
loss_fn (*pytext.models.output_layers.doc_classification_output_layer.Cl attribute*), 337
loss_fn (*pytext.models.output_layers.lm_output_layer.LMOutputLayer attribute*), 341
loss_fn (*pytext.models.output_layers.output_layer_base.OutputLayerBas attribute*), 342
loss_fn (*pytext.models.output_layers.OutputLayerBase attribute*), 347
loss_fn (*pytext.models.output_layers.word_tagging_output_layer.WordT attribute*), 346
loss_fn (*pytext.models.output_layers.WordTaggingOutputLayer*

attribute), 351
lotv_str() (*pytext.data.data_structures.annotation.Tree method*), 204
lower_is_better (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointMultitaskMetricReporter attribute*), 290
lower_is_better (*pytext.metric_reporters.language_model_metric_reporter.LanguageModelMetricReporter attribute*), 292
lower_is_better (*pytext.metric_reporters.LanguageModelMetricReporter attribute*), 300
lower_is_better (*pytext.metric_reporters.metric_reporter.MetricReporter attribute*), 293, 294
lower_is_better (*pytext.metric_reporters.MetricReporter attribute*), 297, 298
lower_is_better (*pytext.metric_reporters.regression_metric_reporter.RegressionMetricReporter attribute*), 295
lower_is_better (*pytext.metric_reporters.RegressionMetricReporter attribute*), 299
lstm (*pytext.models.representations.bilstm.BiLSTM attribute*), 355
lstm (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute*), 356
lstm (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention.Config attribute*), 123
lstm (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 358
lstm (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttention attribute*), 359
LSTM (*pytext.models.representations.stacked_bidirectional_rnn.RNNType attribute*), 367
lstm (*pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN attribute*), 368
lstm_dim (*pytext.models.representations.bilstm.BiLSTM.Config attribute*), 122

M

MACRO_F1 (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 289
macro_prf1_metrics (*pytext.metrics.ClassificationMetrics attribute*), 307, 308
macro_scores (*pytext.metrics.MacroPRF1Metrics attribute*), 309
macro_scores (*pytext.metrics.PRF1Metrics attribute*), 310
MacroPRF1Metrics (*class in pytext.metrics*), 309
MacroPRF1Scores (*class in pytext.metrics*), 309
make_vocab() (*pytext.data.utils.VocabBuilder method*), 251
MaskedLMMetricReporter (*class in pytext.metric_reporters.language_model_metric_reporter*), 292
MAX (*pytext.config.module_config.PoolingType attribute*), 200
max_clip_gradient_norm (*pytext.trainers.trainer.Trainer attribute*), 402
max_clip_norm (*pytext.trainers.trainer.Trainer attribute*), 400
max_seq_len (*pytext.data.data_handler.DataHandler attribute*), 232
max_seq_len (*pytext.data.DataHandler attribute*), 258
max_seq_len (*pytext.data.doc_classification_data_handler.DocClassific attribute*), 63
MaxPool (*class in pytext.models.representations.pooling*), 364
maybe_half() (*in module pytext.utils.precision*), 409
maybe_half() (*in module pytext.utils.precision*), 409
MCC (*pytext.metric_reporters.classification_metric_reporter.ComparableCl attribute*), 289
mcc (*pytext.metrics.ClassificationMetrics attribute*), 308
MEAN (*pytext.config.module_config.PoolingType attribute*), 200
MeanPool (*class in pytext.models.representations.pooling*), 364
MegaTextDataConfig (*in module pytext.data.sources.tsv*), 212
merge() (*pytext.data.sources.tsv.SessionTSVDataSource method*), 213
merge_sub_models() (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingInte method*), 333
merge_sub_models() (*pytext.models.ensembles.BaggingIntentSlotEnsemble method*), 335
merge_sub_models() (*pytext.models.ensembles.ensemble.Ensemble method*), 334
merge_token_labels_by_bio() (*in module pytext.utils.data*), 405
merge_token_labels_by_label() (*in module pytext.utils.data*), 405
merge_token_labels_to_slot() (*in module pytext.utils.data*), 405
metadata_to_save() (*pytext.data.data_handler.DataHandler method*), 233
metadata_to_save() (*pytext.data.DataHandler method*), 259
metadata_to_save() (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH method*), 236

metadata_to_save() (*pytext.data.DisjointMultitaskDataHandler method*), 261
Meter (*class in pytext.utils.meter*), 408
metric_name (*pytext.metric_reporters.channel.TensorBoardChannel attribute*), 286
METRIC_REPORTER (*pytext.config.component.ComponentType attribute*), 196
MetricReporter (*class in pytext.metric_reporters*), 296
MetricReporter (*class in pytext.metric_reporters.metric_reporter*), 293
MetricTensorizer (*class in pytext.data.tensorizers*), 248
micro_scores (*pytext.metrics.PRF1Metrics attribute*), 310
MissingValueError, 202
mlp (*pytext.models.decoders.mlp_decoder.MLPDecoder attribute*), 317
mlp (*pytext.models.decoders.MLPDecoder attribute*), 319
mlp_decoder (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute*), 123
MLPDecoder (*class in pytext.models.decoders*), 319
MLPDecoder (*class in pytext.models.decoders.mlp_decoder*), 317
MLPDecoderQueryResponse (*class in pytext.models.decoders.mlp_decoder_query_response*), 318
Model (*class in pytext.models*), 385
Model (*class in pytext.models.model*), 378
MODEL (*pytext.config.component.ComponentType attribute*), 196
MODEL2 (*pytext.config.component.ComponentType attribute*), 196
MODEL_FEATS (*pytext.common.constants.DFColumn attribute*), 193
ModelExporter (*class in pytext.exporters*), 270
ModelExporter (*class in pytext.exporters.exporter*), 268
ModelInput (*class in pytext.config.contextual_intent_slot*), 197
ModelInput (*class in pytext.config.doc_classification*), 198
ModelInput (*class in pytext.config.pair_classification*), 200
ModelInput (*class in pytext.config.query_document_pairwise_ranking*), 201
ModelInputBase (*class in pytext.models.model*), 380
ModelInputConfig (*class in pytext.config.contextual_intent_slot*), 198
ModelInputConfig (*class in pytext.config.doc_classification*), 198
ModelInputConfig (*class in pytext.config.pair_classification*), 200
ModelInputConfig (*class in pytext.config.query_document_pairwise_ranking*), 202
ModelInputMeta (*class in pytext.models.model*), 380
models (*pytext.models.ensembles.bagging_doc_ensemble.BaggingDocEns attribute*), 102
models (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingI attribute*), 103
models (*pytext.models.ensembles.ensemble.Ensemble attribute*), 334
Module (*class in pytext.models.module*), 380
MODULE (*pytext.config.component.ComponentType attribute*), 196
ModuleConfig (*in module pytext.config.module_config*), 200
modules_save_dir (*pytext.config.pytext_config.PyTextConfig attribute*), 201
mse (*pytext.metrics.RegressionMetrics attribute*), 312
MSELoss (*class in pytext.loss.loss*), 282
MulticlassOutputLayer (*class in pytext.models.output_layers.doc_classification_output_layer*), 338
MultilingualTSVDataSource (*class in pytext.data.sources.tsv*), 212
MULTIPLY (*pytext.config.module_config.SlotAttentionType attribute*), 200

N

NEG_RESPONSE (*pytext.config.query_document_pairwise_ranking.ModelI attribute*), 201
neg_response (*pytext.config.query_document_pairwise_ranking.ModelI attribute*), 202
NEGATIVE_LOSS (*pytext.metric_reporters.classification_metric_reporter.ComparableC attribute*), 289
NestedField (*class in pytext.fields*), 280
NestedField (*class in pytext.fields.field*), 275
NewDisjointMultitask (*class in pytext.task.disjoint_multitask*), 391
NewDisjointMultitaskModel (*class in pytext.models.disjoint_multitask_model*), 374
NewDocModel (*class in pytext.models.doc_model*), 376
NewDocRegressionModel (*class in pytext.models.doc_model*), 377
NewDocumentClassification (*class in pytext.task*), 397
NewDocumentClassification (*class in pytext.task.new_task*), 392

NewDocumentRegression (*class in pytext.task.new_task*), 392
NewTask (*class in pytext.task*), 397
NewTask (*class in pytext.task.new_task*), 392
NewTaskTrainer (*class in pytext.task.new_task*), 393
NewWordTaggingModel (*class in pytext.models.word_model*), 385
NewWordTaggingTask (*class in pytext.task.tasks*), 396
NO_ATTENTION (*pytext.config.module_config.SlotAttentionType attribute*), 200
NO_LABEL_SLOT (*pytext.utils.data.Slot attribute*), 405
no_slots_inside_unsupported (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGConstraints attribute*), 140
no_tokenize() (*in module pytext.utils.data*), 405
Node (*class in pytext.data.data_structures.annotation*), 203
Node (*class in pytext.data.data_structures.node*), 205
Node (*class in pytext.metrics.intent_slot_metrics*), 304
Node_Info (*class in pytext.data.data_structures.annotation*), 203
node_to_metrics_node() (*pytext.metric_reporters.compositional_metric_reporter.CompositionalMetricReporter static method*), 289
node_to_metrics_node() (*pytext.metric_reporters.CompositionalMetricReporter static method*), 300
NodesPredictionPair (*class in pytext.metrics.intent_slot_metrics*), 304
NoPool (*class in pytext.models.representations.pooling*), 364
NtokensTensorizer (*class in pytext.data.tensorizers*), 248
NUM (*pytext.data.tensorizers.ByteTensorizer attribute*), 247
num_anchors (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 81
num_classes (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 81
num_emb_modules (*pytext.models.embeddings.embedding_base.EmbeddingBase attribute*), 324
num_emb_modules (*pytext.models.embeddings.embedding_list.EmbeddingList attribute*), 325
num_emb_modules (*pytext.models.embeddings.EmbeddingBase attribute*), 327
num_emb_modules (*pytext.models.embeddings.EmbeddingList attribute*), 328
num_examples (*pytext.metrics.PairwiseRankingMetrics attribute*), 311
num_examples (*pytext.metrics.RegressionMetrics attribute*), 312
num_label (*pytext.metrics.MacroPRF1Scores attribute*), 309
num_labels (*pytext.metrics.MacroPRF1Scores attribute*), 309
num_layers (*pytext.models.representations.bilstm.BiLSTM.Config attribute*), 122
num_samples (*pytext.metrics.intent_slot_metrics.FrameAccuracy attribute*), 302
num_tags (*pytext.models.output_layers.CRFOutputLayer attribute*), 348
num_tags (*pytext.models.output_layers.word_tagging_output_layer.CRFO attribute*), 344
NUM_TOKENS (*pytext.common.constants.DatasetFieldName attribute*), 194
numberize() (*pytext.data.tensorizers.ByteTensorizer method*), 247
numberize() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 247
numberize() (*pytext.data.tensorizers.FloatListTensorizer method*), 248
numberize() (*pytext.data.tensorizers.JoinStringTensorizer method*), 248
numberize() (*pytext.data.tensorizers.LabelTensorizer method*), 248
numberize() (*pytext.data.tensorizers.MetricTensorizer method*), 248
numberize() (*pytext.data.tensorizers.NumericLabelTensorizer method*), 249
numberize() (*pytext.data.tensorizers.RawJson method*), 249
numberize() (*pytext.data.tensorizers.RawString method*), 249
numberize() (*pytext.data.tensorizers.Tensorizer method*), 250
numberize() (*pytext.data.tensorizers.TokenTensorizer method*), 250
numberize() (*pytext.data.tensorizers.WordLabelTensorizer method*), 251
numberize_rows() (*pytext.data.Data method*), 256
numberize_rows() (*pytext.data.data.Data method*), 229
numericalize() (*pytext.fields.char_field.CharFeatureField method*), 272
numericalize() (*pytext.fields.CharFeatureField method*), 277
numericalize() (*pytext.fields.contextual_token_embedding_field.ContextualTokenEm method*), 273
numericalize() (*pytext.fields.ContextualTokenEmbeddingField method*), 277

numericalize() (*pytext.fields.dict_field.DictFeatureField method*), 274
numericalize() (*pytext.fields.DictFeatureField method*), 278
numericalize() (*pytext.fields.text_field_with_special_unk.TextFeatureFieldWithSpecialUnk method*), 276
numericalize() (*pytext.fields.TextFeatureFieldWithSpecialUnk method*), 280
NumericLabelTensorizer (*class in pytext.data.tensorizers*), 249

O

Optimizer (*class in pytext.optimizer*), 390
OPTIMIZER (*pytext.config.component.ComponentType attribute*), 196
ordered_unique() (*in module pytext.utils.ascii_table*), 404
out_dim (*pytext.models.decoders.decoder_base.DecoderBase attribute*), 315
out_dim (*pytext.models.decoders.DecoderBase attribute*), 318
out_dim (*pytext.models.decoders.mlp_decoder.MLPDecoder attribute*), 317
out_dim (*pytext.models.decoders.MLPDecoder attribute*), 319
output_layer (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsemble attribute*), 333
output_layer (*pytext.models.ensembles.bagging_intent_slot_ensemble.BaggingIntentSlotEnsemble.Config attribute*), 103
output_layer (*pytext.models.ensembles.BaggingIntentSlotEnsemble attribute*), 335
output_layer (*pytext.models.ensembles.ensemble.Ensemble attribute*), 333
output_layer (*pytext.models.language_models.lmlstm.LMLSTM.Config attribute*), 105
output_layer (*pytext.models.Model attribute*), 386
output_layer (*pytext.models.model.Model attribute*), 379
output_names (*pytext.exporters.exporter.ModelExporter attribute*), 268
output_names (*pytext.exporters.ModelExporter attribute*), 270
OutputLayerBase (*class in pytext.models.output_layers*), 347
OutputLayerBase (*class in pytext.models.output_layers.output_layer_base*), 342
OutputLayerUtils (*class in pytext.models.output_layers*), 352
OutputLayerUtils (*class in pytext.models.output_layers.utils*), 344
OutputRecord (*class in pytext.data.featurizer*), 208
OutputRecord (*class in pytext.data.featurizer.featurizer*), 206
overall_metrics (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics attribute*), 303

P

PackageFileName (*class in pytext.common.constants*), 194
pad() (*in module pytext.data.utils*), 251
pad() (*pytext.fields.char_field.CharFeatureField method*), 272
pad() (*pytext.fields.CharFeatureField method*), 277
pad() (*pytext.fields.contextual_token_embedding_field.ContextualTokenEmbeddingField method*), 273
pad() (*pytext.fields.ContextualTokenEmbeddingField method*), 278
pad() (*pytext.fields.dict_field.DictFeatureField method*), 274
pad() (*pytext.fields.DictFeatureField method*), 279
pad_and_tensorize() (*in module pytext.data.utils*), 251
pad_and_tensorize_batches() (*in module pytext.data.data*), 230
PAD_BYTE (*pytext.data.tensorizers.ByteTensorizer attribute*), 247
pad_length() (*in module pytext.utils.precision*), 409
pad_length() (*pytext.fields.field.Field method*), 274
PAD_SEQ (*pytext.common.constants.VocabMeta attribute*), 195
PAD_TOKEN (*pytext.common.constants.VocabMeta attribute*), 195
Padding (*class in pytext.common.constants*), 194
padding_value (*pytext.models.representations.augmented_lstm.AugmentedLSTM attribute*), 352
padding_value (*pytext.models.representations.bilstm.BiLSTM attribute*), 355
padding_value (*pytext.models.representations.stacked_bidirectional_rnn.StackedBiLSTM attribute*), 368
PaddingTest (*class in pytext.data.test.utils_test*), 219
PairClassificationDataHandler (*class in pytext.data*), 264
PairClassificationDataHandler (*class in pytext.data.pair_classification_data_handler*), 243
PairClassificationModel (*class in pytext.models.pair_classification_model*), 381
PairClassificationTask (*class in pytext.task.tasks*), 396

PairRepresentation (*class in pytext.models.representations.pair_rep*), 362
PairwiseClassificationModel (*class in pytext.models.pair_classification_model*), 382
PairwiseClassificationTask (*class in pytext.task.new_task*), 393
PairwiseRankingLoss (*class in pytext.loss*), 284
PairwiseRankingLoss (*class in pytext.loss.loss*), 282
PairwiseRankingMetricReporter (*class in pytext.metric_reporters*), 301
PairwiseRankingMetricReporter (*class in pytext.metric_reporters.pairwise_ranking_metric_reporter*), 294
PairwiseRankingMetrics (*class in pytext.metrics*), 310
PairwiseRankingOutputLayer (*class in pytext.models.output_layers*), 351
PairwiseRankingOutputLayer (*class in pytext.models.output_layers.pairwise_ranking_output_layer*), 344
parse_config() (*in module pytext.config.serialize*), 202
parse_json_array() (*in module pytext.utils.data*), 405
parse_slot_string() (*in module pytext.utils.data*), 405
parse_token() (*in module pytext.utils.data*), 405
ParserState (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 369
pass_index (*pytext.data.data_handler.DataHandler attribute*), 232
pass_index (*pytext.data.DataHandler attribute*), 258
PassThroughRepresentation (*class in pytext.models.representations.pass_through*), 363
pearson_correlation (*pytext.metrics.RegressionMetrics attribute*), 312
per_label_confusions (*pytext.metrics.AllConfusions attribute*), 307
per_label_scores (*pytext.metrics.MacroPRF1Metrics attribute*), 309
per_label_scores (*pytext.metrics.PRF1Metrics attribute*), 310
per_label_soft_scores (*pytext.metrics.ClassificationMetrics attribute*), 308
PerLabelConfusions (*class in pytext.metrics*), 311
perplexity_per_word (*pytext.metrics.language_model_metrics.LanguageModelMetric attribute*), 307
PlaceHolder (*class in pytext.config.pytext_config*), 201
pooling (*pytext.models.representations.bilstm_doc_attention.BiLSTMDoc attribute*), 123
pooling_type (*pytext.models.embeddings.dict_embedding.DictEmbedd attribute*), 323
pooling_type (*pytext.models.embeddings.DictEmbedding attribute*), 330
PoolingBatcher (*class in pytext.data*), 264
PoolingBatcher (*class in pytext.data.data*), 229
PoolingType (*class in pytext.config.module_config*), 200
pop() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackL method*), 369
pop() (*pytext.utils.timing.HierarchicalTimer method*), 410
POS_RESPONSE (*pytext.config.query_document_pairwise_ranking.Modell attribute*), 201
pos_response (*pytext.config.query_document_pairwise_ranking.Modell attribute*), 202
postprocess_output() (*pytext.exporters.exporter.ModelExporter method*), 269
postprocess_output() (*pytext.exporters.ModelExporter method*), 271
precision (*pytext.metrics.MacroPRF1Scores attribute*), 309
precision (*pytext.metrics.PRF1Scores attribute*), 310
precision_range_lower (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 80
precision_range_upper (*pytext.loss.loss.AUCPRHingeLoss.Config attribute*), 81
predict() (*pytext.task.task.TaskBase method*), 394
predict() (*pytext.task.TaskBase method*), 398
predicted_frame (*pytext.metrics.intent_slot_metrics.FramePredictionPair attribute*), 303
predicted_label (*pytext.metrics.LabelPrediction attribute*), 308, 309
predicted_nodes (*pytext.metrics.intent_slot_metrics.NodesPredictionPair attribute*), 304
PREDICTOR (*pytext.config.component.ComponentType attribute*), 196
prepare() (*pytext.optimizer.scheduler.BatchScheduler method*), 388
prepare() (*pytext.optimizer.scheduler.Scheduler method*), 390
prepare() (*pytext.optimizer.scheduler.WarmupScheduler method*), 390
prepare_for_onnx_export_() (*pytext.models.BaseModel method*), 388

prepare_for_onnx_export_() (*pytext.models.model.BaseModel method*), 378
prepare_task() (*in module pytext.workflow*), 411
prepare_task_metadata() (*in module pytext.workflow*), 411
prepend_operators() (*pytext.exporters.exporter.ModelExporter method*), 269
prepend_operators() (*pytext.exporters.ModelExporter method*), 271
preprocess() (*pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler method*), 222
preprocess() (*pytext.data.BPTTLanguageModelDataHandler method*), 253
preprocess() (*pytext.data.data_handler.DataHandler method*), 233
preprocess() (*pytext.data.DataHandler method*), 259
preprocess_row() (*pytext.data.bptt_lm_data_handler.BPTTLanguageModelDataHandler method*), 222
preprocess_row() (*pytext.data.BPTTLanguageModelDataHandler method*), 253
preprocess_row() (*pytext.data.compositional_data_handler.CompositionalDataHandler method*), 225
preprocess_row() (*pytext.data.CompositionalDataHandler method*), 254
preprocess_row() (*pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotModelDataHandler method*), 227
preprocess_row() (*pytext.data.ContextualIntentSlotModelDataHandler method*), 255
preprocess_row() (*pytext.data.data_handler.DataHandler method*), 233
preprocess_row() (*pytext.data.DataHandler method*), 259
preprocess_row() (*pytext.data.doc_classification_data_handler.DocClassificationDataHandler method*), 239
preprocess_row() (*pytext.data.DocClassificationDataHandler method*), 262
preprocess_row() (*pytext.data.joint_data_handler.JointModelDataHandler method*), 240
preprocess_row() (*pytext.data.JointModelDataHandler method*), 263
preprocess_row() (*pytext.data.language_model_data_handler.LanguageModelDataHandler method*), 242
preprocess_row() (*pytext.data.LanguageModelDataHandler method*), 264
preprocess_row() (*pytext.data.pair_classification_data_handler.PairClassificationDataHandler method*), 244
preprocess_row() (*pytext.data.PairClassificationDataHandler method*), 264
preprocess_row() (*pytext.data.query_document_pairwise_ranking_data_handler.QueryDocumentPairwiseRankingDataHandler method*), 246
preprocess_row() (*pytext.data.QueryDocumentPairwiseRankingDataHandler method*), 266
preprocess_row() (*pytext.data.seq_data_handler.SeqModelDataHandler method*), 246
preprocess_row() (*pytext.data.SeqModelDataHandler method*), 267
PretrainedEmbedding (*class in pytext.utils.embeddings*), 406
print_config_class() (*in module pytext.utils.documentation*), 406
PRF1Metrics (*class in pytext.metrics*), 309
PRF1Scores (*class in pytext.metrics*), 310
print_metrics() (*pytext.metrics.ClassificationMetrics method*), 308
print_metrics() (*pytext.metrics.intent_slot_metrics.AllMetrics method*), 302
print_metrics() (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics method*), 303
print_metrics() (*pytext.metrics.language_model_metrics.LanguageModelMetric method*), 307
print_metrics() (*pytext.metrics.MacroPRF1Metrics method*), 309
print_metrics() (*pytext.metrics.PairwiseRankingMetrics method*), 311
print_metrics() (*pytext.metrics.PRF1Metrics method*), 310
print_metrics() (*pytext.metrics.RegressionMetrics method*), 312
print_tree() (*pytext.data.data_structures.annotation.Tree method*), 204
process_pred() (*py-*

text.metric_reporters.intent_slot_detection_metric_reporter.IntentSlotMetricReporter method), 291
process_pred() (*pytext.metric_reporters.IntentSlotMetricReporter method*), 299
process_pred() (*pytext.metric_reporters.word_tagging_metric_reporter.WordTaggingMetricReporter method*), 296
process_pred() (*pytext.metric_reporters.WordTaggingMetricReporter method*), 300
projection (*pytext.models.embeddings.char_embedding.CharacterEmbedding attribute*), 322
projection (*pytext.models.embeddings.CharacterEmbedding attribute*), 331
projection_d (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 358
projection_w (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 358
PureDocAttention (*class in pytext.models.representations.pure_doc_attention*), 365
push() (*pytext.models.semantic_parsers.rnng.rnng_data_structures.StackLSTM method*), 370
push() (*pytext.utils.timing.HierarchicalTimer method*), 410
push_action() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 371
pytext (*module*), 412
pytext.builtin_task (*module*), 410
pytext.common (*module*), 195
pytext.common.constants (*module*), 193
pytext.config (*module*), 202
pytext.config.component (*module*), 195
pytext.config.config_adapter (*module*), 197
pytext.config.contextual_intent_slot (*module*), 197
pytext.config.doc_classification (*module*), 198
pytext.config.field_config (*module*), 198
pytext.config.module_config (*module*), 200
pytext.config.pair_classification (*module*), 200
pytext.config.pytext_config (*module*), 200
pytext.config.query_document_pairwise_ranking (*module*), 201
pytext.config.serialize (*module*), 202
pytext.data (*module*), 252
pytext.data.batch_sampler (*module*), 220
pytext.data.bptt_lm_data_handler (*module*), 222
pytext.data.compositional_data_handler (*module*), 224
pytext.data.contextual_intent_slot_data_handler (*module*), 226
pytext.data.data (*module*), 228
pytext.data.data_handler (*module*), 230
pytext.data.data_structures (*module*), 205
pytext.data.data_structures.annotation (*module*), 205
pytext.data.data_structures.node (*module*), 205
pytext.data.disjoint_multitask_data (*module*), 234
pytext.data.disjoint_multitask_data_handler (*module*), 235
pytext.data.doc_classification_data_handler (*module*), 238
pytext.data.featurizer (*module*), 207
pytext.data.featurizer.featurizer (*module*), 207
pytext.data.featurizer.simple_featurizer (*module*), 207
pytext.data.joint_data_handler (*module*), 240
pytext.data.language_model_data_handler (*module*), 241
pytext.data.pair_classification_data_handler (*module*), 243
pytext.data.query_document_pairwise_ranking_data_handler (*module*), 245
pytext.data.seq_data_handler (*module*), 246
pytext.data.sources (*module*), 213
pytext.data.sources.data_source (*module*), 209
pytext.data.sources.squad (*module*), 211
pytext.data.sources.tsv (*module*), 212
pytext.data.tensorizers (*module*), 247
pytext.data.test (*module*), 219
pytext.data.test.batch_sampler_test (*module*), 214
pytext.data.test.bptt_lm_data_handler_test (*module*), 214
pytext.data.test.compositional_datahandler_test (*module*), 214
pytext.data.test.contextual_intent_slot_data_handler (*module*), 215
pytext.data.test.data_test (*module*), 215
pytext.data.test.datahandler_test (*module*), 216
pytext.data.test.doc_classification_data_handler_test (*module*), 216
pytext.data.test.joint_data_handler_test (*module*), 216
pytext.data.test.kd_doc_classification_data_handler (*module*), 216
pytext.data.test.language_model_data_handler_test

(*module*), 217
pytext.data.test.query_document_pairwise_ranking_data_handler_test (*module*), 217
pytext.data.test.round_robin_batchiterator_test (*module*), 217
pytext.data.test.seq_data_handler_test (*module*), 217
pytext.data.test.simple_featurizer_test (*module*), 217
pytext.data.test.tensorizers_test (*module*), 218
pytext.data.test.tokenizers_test (*module*), 218
pytext.data.test.tsv_data_source_test (*module*), 218
pytext.data.test.utils_test (*module*), 219
pytext.data.tokenizers (*module*), 220
pytext.data.tokenizers.tokenizer (*module*), 219
pytext.data.utils (*module*), 251
pytext.exporters (*module*), 270
pytext.exporters.custom_exporters (*module*), 268
pytext.exporters.exporter (*module*), 268
pytext.fields (*module*), 277
pytext.fields.char_field (*module*), 272
pytext.fields.contextual_token_embedding_field (*module*), 273
pytext.fields.dict_field (*module*), 273
pytext.fields.field (*module*), 274
pytext.fields.text_field_with_special_unk (*module*), 276
pytext.loss (*module*), 282
pytext.loss.loss (*module*), 281
pytext.main (*module*), 411
pytext.metric_reporters (*module*), 296
pytext.metric_reporters.channel (*module*), 284
pytext.metric_reporters.classification_metric_reporter (*module*), 288
pytext.metric_reporters.compositional_metric_reporter (*module*), 289
pytext.metric_reporters.disjoint_multitask_metric_reporter (*module*), 290
pytext.metric_reporters.intent_slot_detection_metric_reporter (*module*), 291
pytext.metric_reporters.language_model_metric_reporter (*module*), 292
pytext.metric_reporters.metric_reporter (*module*), 293
pytext.metric_reporters.pairwise_ranking_metric_reporter (*module*), 294
pytext.metric_reporters.regression_metric_reporter (*module*), 295
pytext.metric_reporters.word_tagging_metric_reporter (*module*), 295
pytext.metrics (*module*), 307
pytext.metrics.intent_slot_metrics (*module*), 302
pytext.metrics.language_model_metrics (*module*), 307
pytext.models (*module*), 385
pytext.models.crf (*module*), 373
pytext.models.decoders (*module*), 318
pytext.models.decoders.decoder_base (*module*), 315
pytext.models.decoders.intent_slot_model_decoder (*module*), 316
pytext.models.decoders.mlp_decoder (*module*), 317
pytext.models.decoders.mlp_decoder_query_response (*module*), 318
pytext.models.disjoint_multitask_model (*module*), 374
pytext.models.distributed_model (*module*), 375
pytext.models.doc_model (*module*), 376
pytext.models.embeddings (*module*), 327
pytext.models.embeddings.char_embedding (*module*), 321
pytext.models.embeddings.contextual_token_embedding (*module*), 323
pytext.models.embeddings.dict_embedding (*module*), 323
pytext.models.embeddings.embedding_base (*module*), 324
pytext.models.embeddings.embedding_list (*module*), 325
pytext.models.embeddings.word_embedding (*module*), 326
pytext.models.ensembles (*module*), 334
pytext.models.ensembles.bagging_doc_ensemble (*module*), 332
pytext.models.ensembles.bagging_intent_slot_ensemble (*module*), 333
pytext.models.ensembles.ensemble (*module*), 333
pytext.models.joint_model (*module*), 377
pytext.models.language_models (*module*), 336
pytext.models.language_models.lmlstm (*module*), 335
pytext.models.model (*module*), 378
pytext.models.module (*module*), 380
pytext.models.output_layers (*module*), 347
pytext.models.output_layers.doc_classification_output (*module*), 337
pytext.models.output_layers.doc_regression_output_l

(*module*), 339
pytext.models.output_layers.intent_slot_output_layer (*module*), 340
pytext.models.output_layers.lm_output_layer (*module*), 341
pytext.models.output_layers.output_layer_base (*module*), 342
pytext.models.output_layers.pairwise_ranking_output_layer (*module*), 344
pytext.models.output_layers.utils (*module*), 344
pytext.models.output_layers.word_tagging_output_layer (*module*), 344
pytext.models.pair_classification_model (*module*), 381
pytext.models.query_document_pairwise_ranking_model (*module*), 384
pytext.models.representations (*module*), 368
pytext.models.representations.augmented_lstm (*module*), 352
pytext.models.representations.bilstm (*module*), 355
pytext.models.representations.bilstm_doc_attention (*module*), 356
pytext.models.representations.bilstm_doc_slot_attention (*module*), 357
pytext.models.representations.bilstm_slot_attn (*module*), 359
pytext.models.representations.biseqcnn (*module*), 360
pytext.models.representations.contextual_intent_slot_rep (*module*), 360
pytext.models.representations.docnn (*module*), 361
pytext.models.representations.jointcnn_rep (*module*), 362
pytext.models.representations.pair_rep (*module*), 362
pytext.models.representations.pass_through (*module*), 363
pytext.models.representations.pooling (*module*), 363
pytext.models.representations.pure_doc_attention (*module*), 365
pytext.models.representations.query_document_pairwise_rep (*module*), 365
pytext.models.representations.representation_base (*module*), 366
pytext.models.representations.seq_rep (*module*), 366
pytext.models.representations.slot_attention (*module*), 367
pytext.models.representations.stacked_bidirectional_rnn (*module*), 367
pytext.models.semantic_parsers (*module*), 372
pytext.models.semantic_parsers.rnng (*module*), 372
pytext.models.semantic_parsers.rnng.rnng_data_structure (*module*), 368
pytext.models.semantic_parsers.rnng.rnng_parser (*module*), 370
pytext.models.seq_models (*module*), 373
pytext.models.seq_models.contextual_intent_slot (*module*), 372
pytext.models.seq_models.seqnn (*module*), 373
pytext.models.word_model (*module*), 385
pytext.optimizer (*module*), 390
pytext.optimizer.scheduler (*module*), 388
pytext.task (*module*), 397
pytext.task.disjoint_multitask (*module*), 391
pytext.task.new_task (*module*), 392
pytext.task.serialize (*module*), 393
pytext.task.task (*module*), 393
pytext.task.tasks (*module*), 395
pytext.trainers (*module*), 402
pytext.trainers.ensemble_trainer (*module*), 399
pytext.trainers.hogwild_trainer (*module*), 399
pytext.trainers.trainer (*module*), 400
pytext.utils (*module*), 410
pytext.utils.data (*module*), 404
pytext.utils.cuda (*module*), 404
pytext.utils.data (*module*), 404
pytext.utils.distributed (*module*), 405
pytext.utils.documentation (*module*), 405
pytext.utils.embeddings (*module*), 406
pytext.utils.loss (*module*), 406
pytext.utils.meter (*module*), 408
pytext.utils.model (*module*), 409
pytext.utils.onnx (*module*), 409
pytext.utils.precision (*module*), 409
pytext.utils.test (*module*), 409
pytext.utils.timing (*module*), 410
pytext.utils.torch (*module*), 410
pytext.workflow (*module*), 411
pytext_config_from_json() (*in module pytext.config.serialize*), 202
PyTextConfig (*class in pytext.config.pytext_config*), 201
pytorch_to_caffe2() (*in module pytext.utils.onnx*), 409

Q

QUERY (*pytext.config.query_document_pairwise_ranking.ModelInput attribute*), 202
query (*pytext.config.query_document_pairwise_ranking.ModelInputConfig attribute*), 202
QueryDocumentPairwiseRankingDataHandler (*class in pytext.data*), 265
QueryDocumentPairwiseRankingDataHandler (*class in pytext.data.query_document_pairwise_ranking_data_handler*), 245
QueryDocumentPairwiseRankingDataHandlerTest (*class in pytext.data.test.query_document_pairwise_ranking_data_handler_test*), 217
QueryDocumentPairwiseRankingModel (*class in pytext.models.query_document_pairwise_ranking_model*), 384
QueryDocumentPairwiseRankingRep (*class in pytext.models.representations.query_document_pairwise_ranking_rep*), 365
QueryDocumentPairwiseRankingTask (*class in pytext.task.tasks*), 396

R

RANDOM (*pytext.config.field_config.EmbedInitStrategy attribute*), 199
random_seed (*pytext.config.pytext_config.PyTextConfig attribute*), 201
RandomizedBatchSampler (*class in pytext.data*), 265
RandomizedBatchSampler (*class in pytext.data.batch_sampler*), 220
range_to_anchors_and_delta() (*in module pytext.utils.loss*), 407
rank (*pytext.trainers.trainer.TrainingState attribute*), 401
rank (*pytext.trainers.TrainingState attribute*), 403
raw_columns (*pytext.data.contextual_intent_slot_data_handler.ContextualIntentSlotModelDataHandler attribute*), 227
raw_columns (*pytext.data.ContextualIntentSlotModelDataHandler attribute*), 255
raw_columns (*pytext.data.data_handler.DataHandler attribute*), 231
raw_columns (*pytext.data.DataHandler attribute*), 257
RAW_DICT_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
RAW_EMBED (*pytext.common.constants.PackageFileName attribute*), 194
raw_eval_data_generator() (*pytext.data.sources.data_source.RootDataSource method*), 210
raw_eval_data_generator() (*pytext.data.sources.tsv.TSVDataSource method*), 213
raw_eval_data_generator() (*pytext.data.sources.TSVDataSource method*), 214
RAW_FEATS (*pytext.common.constants.DFColumn attribute*), 193
raw_gazetteer_feats (*pytext.data.featurizer.featurizer.InputRecord attribute*), 206
raw_gazetteer_feats (*pytext.data.featurizer.InputRecord attribute*), 208
RAW_SEQUENCE (*pytext.common.constants.DatasetFieldName attribute*), 194
raw_test_data_generator() (*pytext.data.sources.data_source.RootDataSource method*), 210
raw_test_data_generator() (*pytext.data.sources.tsv.TSVDataSource method*), 213
raw_test_data_generator() (*pytext.data.sources.TSVDataSource method*), 214
RAW_TEXT (*pytext.config.doc_classification.ExtraField attribute*), 198
raw_text (*pytext.data.featurizer.featurizer.InputRecord attribute*), 206
raw_text (*pytext.data.featurizer.InputRecord attribute*), 208
raw_train_data_generator() (*pytext.data.sources.data_source.RootDataSource method*), 210
raw_train_data_generator() (*pytext.data.sources.tsv.TSVDataSource method*), 213
raw_train_data_generator() (*pytext.data.sources.TSVDataSource method*), 214
RAW_WORD_LABEL (*pytext.common.constants.DatasetFieldName attribute*), 194
RAW_WORD_LABEL (*pytext.config.contextual_intent_slot.ExtraField attribute*), 197
RawData (*class in pytext.data*), 266
RawData (*class in pytext.data.contextual_intent_slot_data_handler*), 228
RawData (*class in py-*

text.data.doc_classification_data_handler), 239
RawData (*class in pytext.data.pair_classification_data_handler*), 244
RawExample (*class in pytext.data.sources*), 213
RawExample (*class in pytext.data.sources.data_source*), 209
RawField (*class in pytext.fields*), 279
RawField (*class in pytext.fields.field*), 275
RawJson (*class in pytext.data.tensorizers*), 249
RawString (*class in pytext.data.tensorizers*), 249
read_from_file() (*pytext.data.data_handler.DataHandler method*), 233
read_from_file() (*pytext.data.DataHandler method*), 260
real_trainer (*pytext.trainers.ensemble_trainer.EnsembleTrainer attribute*), 399
real_trainer (*pytext.trainers.EnsembleTrainer attribute*), 403
realtime_report_freq (*pytext.metric_reporters.metric_reporter.MetricReporter attribute*), 294
realtime_report_freq (*pytext.metric_reporters.MetricReporter attribute*), 298
RealtimeMetrics (*class in pytext.metrics*), 311
recall (*pytext.metrics.MacroPRF1Scores attribute*), 309
recall (*pytext.metrics.PRF1Scores attribute*), 310
recall_at_precision (*pytext.metrics.SoftClassificationMetrics attribute*), 312
recall_at_precision() (*in module pytext.metrics*), 314
RECALL_AT_PRECISION_THREHOLDS (*in module pytext.metrics*), 311
recursive_validation() (*pytext.data.data_structures.annotation.Tree method*), 204
ReduceLROnPlateau (*class in pytext.optimizer.scheduler*), 389
register_adapter() (*in module pytext.config.config_adapter*), 197
register_builtin_tasks() (*in module pytext.builtin_task*), 410
register_tasks() (*in module pytext.config.component*), 197
register_type() (*pytext.data.sources.data_source.RootDataSource class method*), 210
Registry (*class in pytext.config.component*), 196
RegistryError, 196
RegressionMetricReporter (*class in pytext.metric_reporters*), 299
RegressionMetricReporter (*class in pytext.metric_reporters.regression_metric_reporter*), 295
RegressionMetrics (*class in pytext.metrics*), 312
RegressionOutputLayer (*class in pytext.models.output_layers*), 350
RegressionOutputLayer (*class in pytext.models.output_layers.doc_regression_output_layer*), 339
relu (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDo attribute*), 358
remove() (*pytext.data.data_structures.annotation.Token method*), 204
repackage_hidden() (*in module pytext.models.language_models.lmlstm*), 336
replace_components() (*in module pytext.utils.documentation*), 406
replace_tokens() (*pytext.data.utils.Vocabulary method*), 251
report() (*pytext.metric_reporters.Channel method*), 296
report() (*pytext.metric_reporters.channel.Channel method*), 284
report() (*pytext.metric_reporters.channel.ConsoleChannel method*), 285
report() (*pytext.metric_reporters.channel.FileChannel method*), 285
report() (*pytext.metric_reporters.channel.TensorBoardChannel method*), 286
report() (*pytext.utils.timing.Snapshot method*), 410
report_eval_results (*pytext.config.pytext_config.PyTextConfig attribute*), 201
report_metric() (*pytext.metric_reporters.disjoint_multitask_metric_reporter.DisjointI method*), 290
report_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 294
report_metric() (*pytext.metric_reporters.MetricReporter method*), 298
report_realtime_metric() (*pytext.metric_reporters.metric_reporter.MetricReporter method*), 294
report_realtime_metric() (*pytext.metric_reporters.MetricReporter method*), 298
report_snapshot() (*in module pytext.utils.timing*), 410
report_train_metrics (*pytext.trainers.Trainer attribute*), 402

report_train_metrics (*pytext.trainers.trainer.Trainer attribute*), 400
representation (*pytext.models.language_models.lmlstm.LMLSTM.Config attribute*), 105
representation (*pytext.models.Model attribute*), 386
representation (*pytext.models.model.Model attribute*), 379
representation_dim (*pytext.models.representations.augmented_lstm.AugmentedLSTM attribute*), 353
representation_dim (*pytext.models.representations.bilstm.BiLSTM attribute*), 355
representation_dim (*pytext.models.representations.bilstm_doc_attention.BiLSTMDocAttention attribute*), 357
representation_dim (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDocSlotAttention attribute*), 358
representation_dim (*pytext.models.representations.bilstm_slot_attn.BiLSTMSlotAttention attribute*), 359
representation_dim (*pytext.models.representations.stacked_bidirectional_rnn.StackedBidirectionalRNN attribute*), 368
RepresentationBase (*class in pytext.models.representations.representation_base*), 366
reset() (*pytext.utils.meter.Meter method*), 408
reset() (*pytext.utils.meter.TimeMeter method*), 409
reset_parameters() (*pytext.models.crf.CRF method*), 374
reset_parameters() (*pytext.models.embeddings.char_embedding.Highway method*), 322
reset_parameters() (*pytext.models.representations.augmented_lstm.AugmentedLSTMCell method*), 354
ResultRow (*class in pytext.utils.data*), 404
ResultTable (*class in pytext.utils.data*), 404
RNN (*pytext.models.representations.stacked_bidirectional_rnn.RnnType attribute*), 367
RNNGParser (*class in pytext.models.semantic_parsers.rnng.rnng_parser*), 370
RnnType (*class in pytext.models.representations.stacked_bidirectional_rnn*), 367
ROC_AUC (*pytext.metric_reporters.classification_metric_reporter.ComparableClassificationMetric attribute*), 289
roc_auc (*pytext.metrics.ClassificationMetrics attribute*), 308
roc_auc (*pytext.metrics.SoftClassificationMetrics attribute*), 312
Root (*class in pytext.data.data_structures.annotation*), 203
RootDataSource (*class in pytext.data.sources.data_source*), 209
RoundRobinBatchIterator (*class in pytext.data.disjoint_multitask_data_handler*), 236
RoundRobinBatchIteratorTest (*class in pytext.data.test.round_robin_batchiterator_test*), 217
RoundRobinBatchSampler (*class in pytext.data*), 266
RoundRobinBatchSampler (*class in pytext.data.batch_sampler*), 221
RowDataSource (*class in pytext.data.sources.data_source*), 210
run_epoch() (*pytext.task.new_task.NewTaskTrainer method*), 393
run_epoch() (*pytext.trainers.hogwild_trainer.HogwildTrainer method*), 400
run_epoch() (*pytext.trainers.HogwildTrainer method*), 404
run_epoch() (*pytext.trainers.Trainer method*), 402
run_epoch() (*pytext.trainers.trainer.Trainer method*), 400
run_single() (*in module pytext.main*), 411

S

safe_division() (*in module pytext.metrics*), 314
SafeFileWrapper (*class in pytext.data.sources.data_source*), 210
samples (*pytext.metrics.RealtimeMetrics attribute*), 311, 312
save() (*in module pytext.task*), 399
save() (*in module pytext.task.serialize*), 393
save_all_checkpoints (*pytext.config.pytext_config.PyTextConfig attribute*), 201
save_and_export() (*in module pytext.workflow*), 411
save_checkpoint() (*pytext.trainers.Trainer method*), 402
save_checkpoint() (*pytext.trainers.trainer.Trainer method*), 400
save_module_checkpoints (*pytext.config.pytext_config.PyTextConfig attribute*), 201
save_modules() (*pytext.models.BaseModel method*), 388
save_modules() (*pytext.models.disjoint_multitask_model.DisjointMultitaskModel method*), 374

save_modules() (*pytext.models.ensembles.ensemble.Ensemble method*), 334
save_modules() (*pytext.models.model.BaseModel method*), 378
save_modules() (*pytext.models.pair_classification_model.BasePairwiseClassificationModel method*), 381
save_modules() (*pytext.models.pair_classification_model.PairClassificationModel method*), 382
save_modules() (*pytext.models.query_document_pairwise_ranking_model.QueryDocumentPairwiseRankingModel method*), 385
save_modules() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 371
save_snapshot_path (*pytext.config.pytext_config.PyTextConfig attribute*), 201
Scheduler (*class in pytext.optimizer.scheduler*), 389
SCHEDULER (*pytext.config.component.ComponentType attribute*), 196
SelfAttention (*class in pytext.models.representations.pooling*), 364
SemanticParsingTask (*class in pytext.task.tasks*), 397
SEQ (*pytext.config.contextual_intent_slot.ModelInput attribute*), 198
SEQ_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
SEQ_LENS (*pytext.common.constants.DatasetFieldName attribute*), 194
SEQ_LENS (*pytext.config.doc_classification.ModelInput attribute*), 198
seq_word_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 198
seq_word_feat (*pytext.config.field_config.FeatureConfig attribute*), 199
SeqFeatureField (*class in pytext.fields*), 280
SeqFeatureField (*class in pytext.fields.field*), 275
SEQLOGICAL (*pytext.common.constants.DFColumn attribute*), 193
SeqModelDataHandler (*class in pytext.data*), 267
SeqModelDataHandler (*class in pytext.data.seq_data_handler*), 246
SeqModelDataHandlerTest (*class in pytext.data.test.seq_data_handler_test*), 217
SeqNNModel (*class in pytext.models.seq_models.seqnn*), 373
SeqNNTask (*class in pytext.task.tasks*), 397
SeqRepresentation (*class in pytext.models.representations.seq_rep*), 366
SERIALIZED_EMBED (*pytext.common.constants.PackageFileName attribute*), 194
SessionTSVDataSource (*class in pytext.data.sources.tsv*), 212
SessionTSVDataSourceTest (*class in pytext.data.test.tsv_data_source_test*), 218
set_fp16() (*in module pytext.utils.precision*), 409
set_random_seeds() (*in module pytext.utils*), 410
set_transitions() (*pytext.models.crf.CRF method*), 374
set_up_training() (*pytext.trainers.hogwild_trainer.HogwildTrainer method*), 400
set_up_training() (*pytext.trainers.HogwildTrainer method*), 404
set_up_training() (*pytext.trainers.Trainer method*), 402
set_up_training() (*pytext.trainers.trainer.Trainer method*), 400
setUp() (*pytext.data.test.batch_sampler_test.BatchSamplerTest method*), 214
setUp() (*pytext.data.test.compositional_datahandler_test.CompositionalD method*), 214
setUp() (*pytext.data.test.contextual_intent_slot_data_handler_test.Conte method*), 215
setUp() (*pytext.data.test.data_test.DataTest method*), 215
setUp() (*pytext.data.test.doc_classification_data_handler_test.DocClass method*), 216
setUp() (*pytext.data.test.joint_data_handler_test.JointDataHandlerTest method*), 216
setUp() (*pytext.data.test.kd_doc_classification_data_handler_test.KDDo method*), 216
setUp() (*pytext.data.test.query_document_pairwise_ranking_data_handl method*), 217
setUp() (*pytext.data.test.seq_data_handler_test.SeqModelDataHandlerT method*), 217
setUp() (*pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method*), 217
setUp() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
setUp() (*pytext.data.test.tsv_data_source_test.SessionTSVDataSourceTes method*), 218
setUp() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 218
SGD (*class in pytext.optimizer*), 390
shard() (*in module pytext.data.utils*), 251
ShardedDataSource (*class in pytext.data.sources.data_source*), 211
should_iter() (*in module pytext.data.utils*), 251
shuffle (*pytext.data.data_handler.DataHandler attribute*), 231

shuffle (*pytext.data.DataHandler attribute*), 258
simple_tokenize() (*in module pytext.utils.data*), 405
SimpleFeaturizer (*class in pytext.data.featurizer*), 208
SimpleFeaturizer (*class in pytext.data.featurizer.simple_featurizer*), 207
SimpleFeaturizerTest (*class in pytext.data.test.simple_featurizer_test*), 217
SimpleWordTaggingMetricReporter (*class in pytext.metric_reporters*), 301
SimpleWordTaggingMetricReporter (*class in pytext.metric_reporters.word_tagging_metric_reporter*), 295
Slot (*class in pytext.data.data_structures.annotation*), 203
Slot (*class in pytext.utils.data*), 404
slot_confusions (*pytext.metrics.intent_slot_metrics.IntentSlotConfusions attribute*), 303
slot_metrics (*pytext.metrics.intent_slot_metrics.IntentSlotMetrics attribute*), 303
SlotAttention (*class in pytext.models.representations.slot_attention*), 367
SlotAttentionType (*class in pytext.config.module_config*), 200
slots (*pytext.metrics.intent_slot_metrics.IntentsAndSlots attribute*), 303
Snapshot (*class in pytext.utils.timing*), 410
snapshot() (*pytext.utils.timing.HierarchicalTimer method*), 410
SnapshotList (*class in pytext.utils.timing*), 410
SoftClassificationMetrics (*class in pytext.metrics*), 312
SoftHardBCELoss (*class in pytext.loss*), 283
SoftHardBCELoss (*class in pytext.loss.loss*), 282
sort_by_score() (*in module pytext.metrics*), 314
sort_key() (*pytext.data.data_handler.DataHandler method*), 233
sort_key() (*pytext.data.DataHandler method*), 260
sort_key() (*pytext.data.pair_classification_data_handler.PairClassificationDataHandler method*), 244
sort_key() (*pytext.data.PairClassificationDataHandler method*), 264
sort_key() (*pytext.data.query_document_pairwise_ranking_data_handler.QueryDocumentPairwiseRankingDataHandler method*), 246
sort_key() (*pytext.data.QueryDocumentPairwiseRankingDataHandler method*), 266
sort_key() (*pytext.data.tensorizers.ByteTensorizer method*), 247
sort_key() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 247
sort_key() (*pytext.data.tensorizers.Tensorizer method*), 250
sort_key() (*pytext.data.tensorizers.TokenTensorizer method*), 250
sort_within_batch (*pytext.data.data_handler.DataHandler attribute*), 231
sort_within_batch (*pytext.data.DataHandler attribute*), 258
SOURCE_FEATS (*pytext.common.constants.DFColumn attribute*), 193
SOURCE_SEQ_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
SOURCE_SEQUENCE (*pytext.common.constants.DFColumn attribute*), 193
Span (*class in pytext.data.data_structures.node*), 205
span (*pytext.data.data_structures.node.Node attribute*), 205
span (*pytext.metrics.intent_slot_metrics.Node attribute*), 304
SpecialToken (*class in pytext.data.utils*), 251
SquadDataSource (*class in pytext.data.sources*), 213
SquadDataSource (*class in pytext.data.sources.squad*), 211
StackedBidirectionalRNN (*class in pytext.models.representations.stacked_bidirectional_rnn*), 367
StackLSTM (*class in pytext.models.semantic_parsers.rnng.rnng_data_structures*), 369
Stage (*class in pytext.common.constants*), 195
stage (*pytext.trainers.trainer.TrainingState attribute*), 401
stage (*pytext.trainers.TrainingState attribute*), 403
stages (*pytext.metric_reporters.Channel attribute*), 296
stages (*pytext.metric_reporters.channel.Channel attribute*), 284
start (*pytext.data.data_structures.node.Span attribute*), 205
start (*pytext.data.tokenizers.tokenizer.Token attribute*), 220
start (*pytext.data.tokenizers.tokenizer.Token attribute*), 219
state_dict() (*pytext.models.distributed_model.DistributedModel method*), 375
state_linearity (*pytext.models.representations.augmented_lstm.AugmentedLSTMCell attribute*), 353
stateful (*pytext.models.language_models.lmlstm.LMLSTM.Config attribute*), 105
step_batch() (*pytext.optimizer.scheduler.CosineAnnealingLR method*), 388

step_batch() (*pytext.optimizer.scheduler.LmFineTuning method*), 389
step_batch() (*pytext.optimizer.scheduler.Scheduler method*), 390
step_batch() (*pytext.optimizer.scheduler.WarmupScheduler method*), 390
step_epoch() (*pytext.optimizer.scheduler.ExponentialLR method*), 389
step_epoch() (*pytext.optimizer.scheduler.ReduceLROnPlateau method*), 389
step_epoch() (*pytext.optimizer.scheduler.Scheduler method*), 390
step_epoch() (*pytext.optimizer.scheduler.StepLR method*), 390
StepLR (*class in pytext.optimizer.scheduler*), 390
strip_bio_prefix() (*in module pytext.utils.data*), 405
subconfigs() (*pytext.config.component.Registry class method*), 196
subrepresentation (*pytext.models.representations.pair_rep.PairRepresentation.Config attribute*), 130
subrepresentation_right (*pytext.models.representations.pair_rep.PairRepresentation.Config attribute*), 130
summary_writer (*pytext.metric_reporters.channel.TensorBoardChannel attribute*), 286
SUPPORT_FP16_OPTIMIZER (*pytext.models.BaseModel attribute*), 387
SUPPORT_FP16_OPTIMIZER (*pytext.models.model.BaseModel attribute*), 378
suppress_output() (*in module pytext.utils.distributed*), 405

T

Target (*class in pytext.config.field_config*), 199
TARGET_LABEL_FIELD (*pytext.config.field_config.Target attribute*), 199
TARGET_LABELS (*pytext.common.constants.DFColumn attribute*), 193
TARGET_LOGITS (*pytext.common.constants.DFColumn attribute*), 193
TARGET_LOGITS_FIELD (*pytext.config.field_config.Target attribute*), 199
target_prob (*pytext.config.field_config.DocLabelConfig attribute*), 199
TARGET_PROB_FIELD (*pytext.config.field_config.Target attribute*), 199
TARGET_PROBS (*pytext.common.constants.DFColumn attribute*), 193
TARGET_SEQ_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
TARGET_SEQ_LENS (*pytext.common.constants.DatasetFieldName attribute*), 194
TARGET_SEQUENCE (*pytext.common.constants.DFColumn attribute*), 194
target_task_name (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataH attribute*), 235
target_task_name (*pytext.data.DisjointMultitaskDataHandler attribute*), 261
target_time_limit_seconds (*pytext.trainers.Trainer attribute*), 402
target_time_limit_seconds (*pytext.trainers.trainer.Trainer attribute*), 400
TARGET_TOKENS (*pytext.common.constants.DFColumn attribute*), 194
TargetTest (*class in pytext.data.test.utils_test*), 219
Task (*class in pytext.task*), 398
Task (*class in pytext.task.task*), 393
TASK (*pytext.config.component.ComponentType attribute*), 196
TASK_NAME (*pytext.common.constants.BatchContext attribute*), 193
TaskBase (*class in pytext.task*), 398
TaskBase (*class in pytext.task.task*), 393
tensor() (*in module pytext.utils.cuda*), 404
TensorBoardChannel (*class in pytext.metric_reporters.channel*), 286
tensorize() (*pytext.data.tensorizers.ByteTensorizer method*), 247
tensorize() (*pytext.data.tensorizers.CharacterTokenTensorizer method*), 247
tensorize() (*pytext.data.tensorizers.FloatListTensorizer method*), 248
tensorize() (*pytext.data.tensorizers.LabelTensorizer method*), 248
tensorize() (*pytext.data.tensorizers.MetricTensorizer method*), 248
tensorize() (*pytext.data.tensorizers.NtokensTensorizer method*), 249
tensorize() (*pytext.data.tensorizers.NumericLabelTensorizer method*), 249
tensorize() (*pytext.data.tensorizers.Tensorizer method*), 250
tensorize() (*pytext.data.tensorizers.TokenTensorizer*

method), 250
tensorize() (*pytext.data.tensorizers.WordLabelTensorizer method*), 251
Tensorizer (*class in pytext.data.tensorizers*), 249
TENSORIZER (*pytext.config.component.ComponentType attribute*), 196
TensorizersTest (*class in pytext.data.test.tensorizers_test*), 218
TEST (*pytext.common.constants.Stage attribute*), 195
test (*pytext.data.sources.data_source.DataSource attribute*), 209
test (*pytext.data.sources.data_source.RootDataSource attribute*), 210
test (*pytext.data.sources.DataSource attribute*), 213
test (*pytext.data.sources.squad.SquadDataSource attribute*), 211
test (*pytext.data.sources.SquadDataSource attribute*), 213
test (*pytext.data.sources.tsv.MultilingualTSVDataSource attribute*), 212
test() (*pytext.task.task.TaskBase method*), 394
test() (*pytext.task.TaskBase method*), 398
test() (*pytext.trainers.Trainer method*), 402
test() (*pytext.trainers.trainer.Trainer method*), 400
test_align_target_label() (*pytext.data.test.utils_test.TargetTest method*), 219
test_batch_iterator() (*pytext.data.test.round_robin_batchiterator_test.RoundRobinBatchIterator method*), 217
test_batch_size (*pytext.data.data_handler.DataHandler attribute*), 232
test_batch_size (*pytext.data.DataHandler attribute*), 258
test_batcher() (*pytext.data.test.data_test.BatcherTest method*), 215
test_convert_to_bytes() (*pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method*), 217
test_create_batches() (*pytext.data.test.data_test.DataTest method*), 215
test_create_batches_different_tensorizers() (*pytext.data.test.data_test.DataTest method*), 215
test_create_byte_tensors() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_create_data_no_batcher_provided() (*pytext.data.test.data_test.DataTest method*), 215
test_create_float_list_tensor() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_create_from_config() (*pytext.data.test.contextual_intent_slot_data_handler_test.Contextua method*), 215
test_create_from_config() (*pytext.data.test.doc_classification_data_handler_test.DocClassifica method*), 216
test_create_from_config() (*pytext.data.test.joint_data_handler_test.JointDataHandlerTest method*), 216
test_create_from_config() (*pytext.data.test.kd_doc_classification_data_handler_test.KDDocCla method*), 216
test_create_from_config() (*pytext.data.test.query_document_pairwise_ranking_data_handler_t method*), 217
test_create_label_tensors() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_create_word_character_tensors() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_create_word_tensors() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_data_handler() (*pytext.data.test.bptt_lm_data_handler_test.BPTTLanguageModelDa method*), 217
test_data_handler() (*pytext.data.test.language_model_data_handler_test.LanguageMode method*), 217
test_data_initializes_tensorsizers() (*pytext.data.test.data_test.DataTest method*), 215
test_data_iterate_multiple_times() (*pytext.data.test.data_test.DataTest method*), 215
test_eval_batch_sampler() (*pytext.data.test.batch_sampler_test.BatchSamplerTest method*), 214
test_init_feature_metadata() (*pytext.data.test.datahandler_test.DataHandlerTest method*), 216
test_initialize_label_tensorizer() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_initialize_tensorizers() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_initialize_word_tensorizer() (*pytext.data.test.tensorizers_test.TensorizersTest method*), 218
test_intermediate_result() (*py-*

text.data.test.compositional_datahandler_test.CompositionalDataHandlerTest method), 214
test_intermediate_result() (*pytext.data.test.contextual_intent_slot_data_handler_test.ContextualIntentSlotModelDataHandlerTest method*), 215
test_intermediate_result() (*pytext.data.test.seq_data_handler_test.SeqModelDataHandlerTest method*), 217
test_iterate_training_data_multiple_times() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 218
test_min_freq() (*pytext.data.test.compositional_datahandler_test.CompositionalDataHandlerTest method*), 215
test_model() (*in module pytext.workflow*), 411
test_model_from_snapshot_path() (*in module pytext.workflow*), 411
test_out_path (*pytext.config.pytext_config.PyTextConfig attribute*), 201
test_out_path (*pytext.config.pytext_config.TestConfig attribute*), 201
test_path (*pytext.config.pytext_config.TestConfig attribute*), 201
test_path (*pytext.data.data_handler.DataHandler attribute*), 231
test_path (*pytext.data.DataHandler attribute*), 258
test_pooling_batcher() (*pytext.data.test.data_test.BatcherTest method*), 215
test_prob_batch_sampler() (*pytext.data.test.batch_sampler_test.BatchSamplerTest method*), 214
test_process_data() (*pytext.data.test.seq_data_handler_test.SeqModelDataHandlerTest method*), 217
test_quoting() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 218
test_read_data_source() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 218
test_read_data_source_with_column_remapping() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 219
test_read_data_source_with_utf8_issues() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 219
test_read_eval_data_source() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 219
test_read_file_with_dense_features() (*pytext.data.test.contextual_intent_slot_data_handler_test.ContextualIntentSlotModelDataHandlerDenseTest method*), 215
test_read_from_csv() (*pytext.data.test.datahandler_test.DataHandlerTest method*), 216
test_read_from_csv() (*pytext.data.test.contextual_intent_slot_data_handler_test.ContextualIntentSlotModelDataHandlerTest method*), 215
test_read_from_file() (*pytext.data.test.doc_classification_data_handler_test.DocClassificationDataHandlerTest method*), 216
test_read_from_file() (*pytext.data.test.joint_data_handler_test.JointDataHandlerTest method*), 216
test_read_from_file() (*pytext.data.test.kd_doc_classification_data_handler_test.KDDocClassificationDataHandlerTest method*), 216
test_read_from_file() (*pytext.data.test.query_document_pairwise_ranking_data_handler_test.QueryDocumentPairwiseRankingDataHandlerTest method*), 217
test_read_partially_from_csv() (*pytext.data.test.datahandler_test.DataHandlerTest method*), 216
test_read_session_data() (*pytext.data.test.tsv_data_source_test.SessionTSVDataSourceTest method*), 218
test_read_test_data_source() (*pytext.data.test.tsv_data_source_test.TSVDataSourceTest method*), 219
test_round_robin_batch_sampler() (*pytext.data.test.batch_sampler_test.BatchSamplerTest method*), 214
test_sharding() (*pytext.data.test.language_model_data_handler_test.LanguageModelDataHandlerTest method*), 217
test_sort() (*pytext.data.test.data_test.DataTest method*), 215
test_split_with_regex() (*pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method*), 217
test_split_with_regex() (*pytext.data.test.tokenizers_test.TokenizeTest method*), 218
test_test_tensors() (*pytext.data.test.compositional_datahandler_test.CompositionalDataHandlerTest method*), 215
test_tokenization() (*pytext.data.test.doc_classification_data_handler_test.DocClassificationDataHandlerTest method*), 216
test_tokenization() (*pytext.data.test.joint_data_handler_test.JointDataHandlerTest method*), 216
test_tokenization() (*pytext.data.test.kd_doc_classification_data_handler_test.KDDocClassificationDataHandlerTest method*), 216

test_tokenization() (pytext.data.test.query_document_pairwise_ranking_data_handler_test.QueryDocumentPairwiseRankingDataHandlerTest method), 217
test_tokenize() (pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method), 217
test_tokenize() (pytext.data.test.tokenizers_test.TokenizeTest method), 218
test_tokenize_add_sentence_markers() (pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method), 218
test_tokenize_dont_lowercase() (pytext.data.test.simple_featurizer_test.SimpleFeaturizerTest method), 218
test_tokenize_dont_lowercase() (pytext.data.test.tokenizers_test.TokenizeTest method), 218
test_train_tensors() (pytext.data.test.compositional_datahandler_test.CompositionalDataHandlerTest method), 215
test_uppercase_tokens() (pytext.data.test.compositional_datahandler_test.CompositionalDataHandlerTest method), 215
testBuildVocabulary() (pytext.data.test.utils_test.VocabularyTest method), 219
TestConfig (class in pytext.config.pytext_config), 201
testPadding() (pytext.data.test.utils_test.PaddingTest method), 219
testPaddingProvideShape() (pytext.data.test.utils_test.PaddingTest method), 219
TEXT (pytext.config.contextual_intent_slot.ModelInput attribute), 198
TEXT (pytext.data.contextual_intent_slot_data_handler.RawData attribute), 228
TEXT (pytext.data.doc_classification_data_handler.RawData attribute), 239
TEXT (pytext.data.RawData attribute), 266
TEXT1 (pytext.config.pair_classification.ModelInput attribute), 200
text1 (pytext.config.pair_classification.ModelInputConfig attribute), 200
TEXT1 (pytext.data.pair_classification_data_handler.RawData attribute), 244
TEXT2 (pytext.config.pair_classification.ModelInput attribute), 200
text2 (pytext.config.pair_classification.ModelInputConfig attribute), 200
TEXT2 (pytext.data.pair_classification_data_handler.RawData attribute), 244
text_feature_name (pytext.data.data_handler.DataHandler attribute), 221
text_feature_name (pytext.data.DataHandler attribute), 257
TEXT_FIELD (pytext.common.constants.DatasetFieldName attribute), 194
TextFeatureField (class in pytext.fields), 279
TextFeatureField (class in pytext.fields.field), 275
TextFeatureFieldWithSpecialUnk (class in pytext.fields), 280
TextFeatureFieldWithSpecialUnk (class in pytext.fields.text_field_with_special_unk), 276
tied_representation (pytext.models.pair_classification_model.PairwiseClassificationModel attribute), 118
tied_weights (pytext.models.language_models.lmlstm.LMLSTM.Config attribute), 105
time() (pytext.utils.timing.HierarchicalTimer method), 410
TimeMeter (class in pytext.utils.meter), 408
Timings (class in pytext.utils.timing), 410
to_actions() (pytext.data.data_structures.annotation.Tree method), 204
to_onehot() (in module pytext.utils.model), 409
Token (class in pytext.data.data_structures.annotation), 203
Token (class in pytext.data.tokenizers), 220
Token (class in pytext.data.tokenizers.tokenizer), 219
Token_Info (class in pytext.data.data_structures.annotation), 204
token_label() (pytext.utils.data.Slot method), 405
token_overlap() (pytext.utils.data.Slot method), 405
TOKEN_RANGE (pytext.common.constants.DatasetFieldName attribute), 194
TOKEN_RANGE (pytext.common.constants.DFColumn attribute), 194
TOKEN_RANGE (pytext.config.contextual_intent_slot.ExtraField attribute), 197
token_ranges (pytext.data.featurizer.featurizer.OutputRecord attribute), 206
token_ranges (pytext.data.featurizer.OutputRecord attribute), 208
tokenize() (pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method), 207
tokenize() (pytext.data.featurizer.SimpleFeaturizer method), 208
tokenize() (pytext.data.tokenizers.Tokenizer method), 220
tokenize() (pytext.data.tokenizers.tokenizer.Tokenizer method), 219
tokenize_batch() (pytext.data.featurizer.simple_featurizer.SimpleFeaturizer method), 207

tokenize_batch() (*pytext.data.featurizer.SimpleFeaturizer method*), 208
Tokenizer (*class in pytext.data.tokenizers*), 220
Tokenizer (*class in pytext.data.tokenizers.tokenizer*), 219
TOKENIZER (*pytext.config.component.ComponentType attribute*), 196
TokenizeTest (*class in pytext.data.test.tokenizers_test*), 218
TOKENS (*pytext.common.constants.DatasetFieldName attribute*), 194
tokens (*pytext.data.featurizer.featurizer.OutputRecord attribute*), 207
tokens (*pytext.data.featurizer.OutputRecord attribute*), 208
TokenTensorizer (*class in pytext.data.tensorizers*), 250
top_intent_accuracy (*pytext.metrics.intent_slot_metrics.AllMetrics attribute*), 302
torchscript_export() (*pytext.task.disjoint_multitask.NewDisjointMultitask method*), 392
torchscript_predictions() (*pytext.models.output_layers.doc_classification_output_layer.BinaryDocClassificationOutputLayer method*), 337
torchscript_predictions() (*pytext.models.output_layers.doc_classification_output_layer.MultiLabelDocClassificationOutputLayer method*), 338
torchscriptify() (*pytext.models.doc_model.NewDocModel method*), 377
TP (*pytext.metrics.Confusions attribute*), 308
tps (*pytext.metrics.RealtimeMetrics attribute*), 311, 312
TRAIN (*pytext.common.constants.Stage attribute*), 195
train (*pytext.data.sources.data_source.DataSource attribute*), 209
train (*pytext.data.sources.data_source.RootDataSource attribute*), 210
train (*pytext.data.sources.data_source.RowShardedDataSource attribute*), 210
train (*pytext.data.sources.DataSource attribute*), 213
train (*pytext.data.sources.squad.SquadDataSource attribute*), 211
train (*pytext.data.sources.SquadDataSource attribute*), 213
train (*pytext.data.sources.tsv.MultilingualTSVDataSource attribute*), 212
train() (*pytext.models.BaseModel method*), 388
train() (*pytext.models.distributed_model.DistributedModel method*), 376
train() (*pytext.models.model.BaseModel method*), 378
train() (*pytext.task.task.TaskBase method*), 394
train() (*pytext.task.TaskBase method*), 399
train() (*pytext.trainers.ensemble_trainer.EnsembleTrainer method*), 399
train() (*pytext.trainers.EnsembleTrainer method*), 403
train() (*pytext.trainers.Trainer method*), 402
train() (*pytext.trainers.trainer.Trainer method*), 400
train_batch() (*pytext.models.BaseModel class method*), 388
train_batch() (*pytext.models.model.BaseModel class method*), 378
train_batch_size (*pytext.data.data_handler.DataHandler attribute*), 232
train_batch_size (*pytext.data.DataHandler attribute*), 258
train_model() (*in module pytext.workflow*), 411
train_model_distributed() (*in module pytext.main*), 411
train_path (*pytext.data.data_handler.DataHandler attribute*), 231
train_path (*pytext.data.DataHandler attribute*), 258
train_single_model() (*pytext.task.tasks.EnsembleTask method*), 395
train_tensorboard (*pytext.config.pytext_config.TensorBoardChannel attribute*), 286
train_unsharded (*pytext.data.sources.data_source.RowShardedDataSource attribute*), 210
train_unsharded (*pytext.data.sources.tsv.BlockShardedTSVDataSource attribute*), 212
Trainer (*class in pytext.trainers*), 402
Trainer (*class in pytext.trainers.trainer*), 400
TRAINER (*pytext.config.component.ComponentType attribute*), 196
TrainerBase (*class in pytext.trainers.trainer*), 401
TrainingState (*class in pytext.trainers*), 403
TrainingState (*class in pytext.trainers.trainer*), 401
Tree (*class in pytext.data.data_structures.annotation*), 204
tree_from_tokens_and_indx_actions() (*pytext.metric_reporters.compositional_metric_reporter.Composition static method*), 289
tree_from_tokens_and_indx_actions() (*pytext.metric_reporters.CompositionalMetricReporter static method*), 300
tree_metrics (*pytext.metrics.intent_slot_metrics.AllMetrics attribute*), 302
tree_to_metric_node() (*pytext.metric_reporters.compositional_metric_reporter.Composition static method*), 289
tree_to_metric_node() (*py-*

text.metric_reporters.CompositionalMetricReporter static method), 301
TreeBuilder (*class in pytext.data.data_structures.annotation*), 204
true_positives (*pytext.metrics.PRF1Scores attribute*), 310
true_positives_lower_bound() (*in module pytext.utils.loss*), 408
TSV (*class in pytext.data.sources.tsv*), 212
TSVDataSource (*class in pytext.data.sources*), 213
TSVDataSource (*class in pytext.data.sources.tsv*), 212
TSVDataSourceTest (*class in pytext.data.test.tsv_data_source_test*), 218

U

unflatten() (*in module pytext.data.sources.squad*), 211
UnionTypeError, 202
UNK_BYTE (*pytext.data.tensorizers.ByteTensorizer attribute*), 247
UNK_NUM_TOKEN (*pytext.common.constants.VocabMeta attribute*), 195
UNK_TOKEN (*pytext.common.constants.VocabMeta attribute*), 195
unkify() (*in module pytext.utils.data*), 405
unwrap_optimizer() (*in module pytext.utils.precision*), 409
update() (*pytext.metrics.PerLabelConfusions method*), 311
update() (*pytext.utils.meter.Meter method*), 408
update() (*pytext.utils.meter.TimeMeter method*), 409
update_best_model() (*pytext.trainers.Trainer method*), 403
update_best_model() (*pytext.trainers.trainer.Trainer method*), 401
update_tree() (*pytext.data.data_structures.annotation.TreeBuilder method*), 204
upgrade_one_version() (*in module pytext.config.config_adapter*), 197
upgrade_to_latest() (*in module pytext.config.config_adapter*), 197
ups (*pytext.metrics.RealtimeMetrics attribute*), 312
upsample (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler attribute*), 235
upsample (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler.Config attribute*), 62
upsample (*pytext.data.disjoint_multitask_data_handler.RoundRobinBatchIterator attribute*), 237
upsample (*pytext.data.DisjointMultitaskDataHandler attribute*), 261
use_action (*pytext.models.semantic_parsers.rnng.rnng_parser.AblationParams attribute*), 139
use_bio_labels (*pytext.config.field_config.WordLabelConfig attribute*), 199
use_buffer (*pytext.models.semantic_parsers.rnng.rnng_parser.Ablation attribute*), 139
use_crf (*pytext.models.ensembles.bagging_intent_slot_ensemble.Bagging attribute*), 333
use_crf (*pytext.models.ensembles.BaggingIntentSlotEnsemble attribute*), 335
use_cuda_if_available (*pytext.config.pytext_config.PyTextConfig attribute*), 201
use_cuda_if_available (*pytext.config.pytext_config.TestConfig attribute*), 201
use_doc_attention (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDo attribute*), 358
use_doc_probs_in_word (*pytext.models.decoders.intent_slot_model_decoder.IntentSlotModel attribute*), 316
use_doc_probs_in_word (*pytext.models.decoders.intent_slot_model_decoder.IntentSlotModel attribute*), 89
use_doc_probs_in_word (*pytext.models.decoders.IntentSlotModelDecoder attribute*), 320
use_fp16 (*pytext.config.pytext_config.PyTextConfig attribute*), 201
use_last_open_NT_feature (*pytext.models.semantic_parsers.rnng.rnng_parser.AblationParams attribute*), 139
use_stack (*pytext.models.semantic_parsers.rnng.rnng_parser.AblationP attribute*), 139
use_tensorboard (*pytext.config.pytext_config.PyTextConfig attribute*), 201
use_tensorboard (*pytext.config.pytext_config.TestConfig attribute*), 201
use_word_attention (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDo attribute*), 358
UTTERANCE (*pytext.common.constants.DFColumn attribute*), 194
UTTERANCE (*pytext.config.contextual_intent_slot.ExtraField attribute*), 107
UTTERANCE (*pytext.data.disjoint_multitask_data_handler.DisjointMultitaskDataHandler.Config attribute*), 62
UTTERANCE_COLUMN (*pytext.metric_reporters.classification_metric_reporter.Classification attribute*), 288
UTTERANCE_COLUMN (*pytext.metric_reporters.ClassificationMetricReporter attribute*), 298
UTTERANCE_FIELD (*py-*

text.common.constants.DatasetFieldName attribute), 194
UTTERANCE_PAIR (*pytext.config.pair_classification.ExtraField attribute*), 200

V

v0_to_v1() (*in module pytext.config.config_adapter*), 197
v1_to_v2() (*in module pytext.config.config_adapter*), 197
v2_to_v3() (*in module pytext.config.config_adapter*), 197
v3_to_v4() (*in module pytext.config.config_adapter*), 197
valid_actions() (*pytext.models.semantic_parsers.rnng.rnng_parser.RNNGParser method*), 371
validate_node() (*pytext.data.data_structures.annotation.Intent method*), 203
validate_node() (*pytext.data.data_structures.annotation.Node method*), 203
validate_node() (*pytext.data.data_structures.annotation.Root method*), 203
validate_node() (*pytext.data.data_structures.annotation.Slot method*), 203
validate_node() (*pytext.data.data_structures.annotation.Token method*), 204
validate_tree() (*pytext.data.data_structures.annotation.Tree method*), 204
value (*pytext.data.tokenizers.Token attribute*), 220
value (*pytext.data.tokenizers.tokenizer.Token attribute*), 219
values() (*pytext.config.component.Registry class method*), 196
var_to_numpy() (*in module pytext.utils.cuda*), 404
Variable() (*in module pytext.utils.cuda*), 404
vocab_map (*pytext.exporters.exporter.ModelExporter attribute*), 268
vocab_map (*pytext.exporters.ModelExporter attribute*), 270
vocab_to_export() (*pytext.models.doc_model.NewDocModel method*), 377
VocabBuilder (*class in pytext.data.utils*), 251
VocabMeta (*class in pytext.common.constants*), 195
Vocabulary (*class in pytext.data.utils*), 251
Vocabulary (*class in pytext.utils.torch*), 410
VocabularyTest (*class in pytext.data.test.utils_test*), 219
VocabUsingField (*class in pytext.fields*), 279
VocabUsingField (*class in pytext.fields.field*), 275
VocabUsingNestedField (*class in pytext.fields*), 280
VocabUsingNestedField (*class in pytext.fields.field*), 276

W

WarmupScheduler (*class in pytext.optimizer.scheduler*), 390
weighted_hinge_loss() (*in module pytext.utils.loss*), 408
word_attention (*pytext.models.representations.bilstm_doc_slot_attention.BiLSTMDo attribute*), 358
word_decoder (*pytext.models.decoders.intent_slot_model_decoder.Inten attribute*), 316
word_decoder (*pytext.models.decoders.IntentSlotModelDecoder attribute*), 320
word_feat (*pytext.config.contextual_intent_slot.ModelInputConfig attribute*), 198
WORD_FEAT (*pytext.config.doc_classification.ModelInput attribute*), 198
word_feat (*pytext.config.doc_classification.ModelInputConfig attribute*), 198
word_feat (*pytext.config.field_config.FeatureConfig attribute*), 199
WORD_LABEL (*pytext.common.constants.DFColumn attribute*), 194
WORD_LABEL (*pytext.data.contextual_intent_slot_data_handler.RawData attribute*), 228
WORD_LABEL_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
WORD_LABEL_PAD (*pytext.common.constants.Padding attribute*), 194
WORD_LABEL_PAD_IDX (*pytext.common.constants.Padding attribute*), 195
word_output (*pytext.models.output_layers.intent_slot_output_layer.Inte attribute*), 340
WORD_WEIGHT (*pytext.common.constants.DFColumn attribute*), 194
WORD_WEIGHT (*pytext.config.contextual_intent_slot.ExtraField attribute*), 197
WORD_WEIGHT (*pytext.data.contextual_intent_slot_data_handler.RawDat attribute*), 228
WORD_WEIGHT_FIELD (*pytext.common.constants.DatasetFieldName attribute*), 194
WordEmbedding (*class in pytext.models.embeddings*), 328

WordEmbedding (*class in pytext.models.embeddings.word_embedding*), 326
WordFeatConfig (*in module pytext.config.field_config*), 199
WordLabelConfig (*class in pytext.config.field_config*), 199
WordLabelField (*class in pytext.fields*), 280
WordLabelField (*class in pytext.fields.field*), 276
WordLabelTensorizer (*class in pytext.data.tensorizers*), 250
WordTaggingMetricReporter (*class in pytext.metric_reporters*), 300
WordTaggingMetricReporter (*class in pytext.metric_reporters.word_tagging_metric_reporter*), 295
WordTaggingModel (*class in pytext.models.word_model*), 385
WordTaggingOutputLayer (*class in pytext.models.output_layers*), 350
WordTaggingOutputLayer (*class in pytext.models.output_layers.word_tagging_output_layer*), 345
WordTaggingTask (*class in pytext.task.tasks*), 397
wrap_optimizer() (*in module pytext.utils.precision*), 409

X

xaviervar() (*in module pytext.utils.cuda*), 404

Z

ZERO (*pytext.config.field_config.EmbedInitStrategy attribute*), 199
zero_grads() (*pytext.trainers.Trainer method*), 403
zero_grads() (*pytext.trainers.trainer.Trainer method*), 401
zerovar() (*in module pytext.utils.cuda*), 404
zip_dicts() (*in module pytext.data.data*), 230