xnmt Documentation

xnmt team

Feb 01, 2019

Contents

This is a repository for the extensible neural machine translation toolkit *xnmt*. It is coded in Python based on DyNet.

CHAPTER 1

Getting Started

1.1 Prerequisites

xnmt requires Python 3.6.

Before running *xnmt* you must install the required packages, including Python bindings for DyNet. This can be done by running `pip install -r requirements.txt`. (There are also optional package requirements under `requirements-extra/` for features that are non-central to *xnmt*.)

Next, install *xnmt* by running `python setup.py install` for normal usage or `python setup.py develop` for development.

1.2 Command line tools

xnmt comes with the following command line interfaces:

- `xnmt` runs experiments given a configuration file that can specify preprocessing, model training, and evaluation. The corresponding Python file is `xnmt/xnmt_run_experiments.py`. Typical example call:

```
xnmt --dynet-gpu my-training.yaml
```

- `xnmt_decode` decodes a hypothesis using a specified model. The corresponding Python file is `xnmt/xnmt_decode.py`. Typical example call:

```
xnmt_decode --src src.txt --hyp out.txt --mod saved-model.mod
```

- `xnmt_evaluate` computes an evaluation metric given hypothesis and reference files. The corresponding Python file is `xnmt/xnmt_evaluate.py`. Typical example call:

```
xnmt_evaluate --hyp out.txt --ref ref.txt --metric bleu
```

1.3 Running the examples

xnmt includes a series of tutorial-style examples in the `examples/` subfolder. These are a good starting point to get familiarized with specifying models and experiments. To run the first experiment, use the following:

```
xnmt examples/01_standard.yaml
```

This is a shortcut for typing `python -m xnmt.xnmt_run_experiments examples/01_standard.yaml`. Make sure to read the comments provided in the *example configuration*.

See the *Experiment configuration file format* documentation entry for more details about writing experiment configuration files.

1.4 Running recipes

xnmt includes several self-contained recipes on publically available data with competitive model settings, and including scripts for data preparation, in the `recipes/` subfolder.

1.5 Running unit tests

From the main directory, run: `python -m unittest`

Or, to run a specific test, use e.g. `python -m unittest test.test_run.TestRunningConfig.test_standard`

1.6 Cython modules

If you wish to use all the modules in *xnmt* that need cython, you need to build the cython extensions by this command:

```
python setup.py build_ext --inplace --use-cython-extensions
```

CHAPTER 2

Experiment configuration file format

2.1 Intro

Configuration files are in YAML format.

At the top-level, a config file consists of a dictionary where keys are experiment names and values are the experiment specifications. By default, all experiments are run in lexicographical ordering, but `xnmt_run_experiments` can also be told to run only a selection of the specified experiments. An example template with 2 experiments looks like this

```
exp1: !Experiment
  exp_global: ...
  preproc: ...
  model: ...
  train: ...
  evaluate: ...
exp2: !Experiment
  exp_global: ...
  preproc: ...
  model: ...
  train: ...
  evaluate: ...
```

`!Experiment` is YAML syntax specifying a Python object of the same name, and its parameters will be passed on to the Python constructor. There can be a special top-level entry named `defaults`; this experiment will never be run, but can be used as a template where components are partially shared using YAML anchors or the !Ref mechanism (more on this later).

The usage of `exp_global`, `preproc`, `model`, `train`, `evaluate` are explained below. Not all of them need to be specified, depending on the use case.

2.1.1 Experiment

This specifies settings that are global to this experiment. An example

```
exp_global: !ExpGlobal
  model_file: '{EXP_DIR}/models/{EXP}.mod'
  log_file: '{EXP_DIR}/logs/{EXP}.log'
  default_layer_dim: 512
  dropout: 0.3
```

Not that for any strings used here or anywhere in the config file `{EXP}` will be over-written by the name of the experiment, `{EXP_DIR}` will be overwritten by the directory the config file lies in, `{PID}` by the process id, and `{GIT_REV}` by the current git revision.

To obtain a full list of allowed parameters, please check the documentation for *ExpGlobal*.

2.1.2 Preprocessing

xnmt supports a variety of data preprocessing features. Please refer to *Preprocessing* for details.

2.1.3 Model

This specifies the model architecture. An typical example looks like this

```
model: !DefaultTranslator
  src_reader: !PlainTextReader
    vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
  trg_reader: !PlainTextReader
    vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  encoder: !BiLSTMSeqTransducer
    layers: 1
  attender: !MlpAttender
    hidden_dim: 512
    state_dim: 512
    input_dim: 512
  decoder: !AutoRegressiveDecoder
    embedder: !SimpleWordEmbedder
      emb_dim: 512
    rnn_layer: !UniLSTMSeqTransducer
      layers: 1
    transform: !NonLinear
      output_dim: 512
    bridge: !CopyBridge {}
```

The top level entry is typically DefaultTranslator, which implements a standard attentional sequence-to-sequence model. It allows flexible specification of encoder, attender, source / target embedder, and other settings. Again, to obtain the full list of supported options, please refer to the corresponding class in the *API Doc*.

Note that some of this Python objects are passed to their parent object's initializer method, which requires that the children are initialized first. *xnmt* therefore uses a bottom-up initialization strategy, where siblings are initialized in the order they appear in the constructor. Among others, this guarantees that preprocessing is carried out before the model training.

2.1.4 Training

A typical example looks like this

```
train: !SimpleTrainingRegimen
  trainer: !AdamTrainer
    alpha: 0.001
  run_for_epochs: 2
  src_file: examples/data/head.ja
  trg_file: examples/data/head.en
  dev_tasks:
    - !LossEvalTask
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
```

The expected object here is a subclass of TrainingRegimen. Besides `xnmt.training_regimen.SimpleTrainingRegimen`, multi-task style training regimens are supported. For multi task training, each training regimen uses their own model, so in this case models must be specified as sub-components of the training regimen. An example *Multi-task* configuration can be refered to for more details on this.

2.1.5 Evaluation

If specified, the model is tested after training finished.

2.2 Config files vs. saved model files

Saved model files are written out in the exact same YAML format as the config files (with the addition of some .data directories that contain DyNet weights). This means that it is possible to specify a saved model as the configuration file. There is one subtle difference: In a config file, placeholders such as `{EXP_DIR}` are resolved based on the current context, which will be different when directly specifying the saved model file as config file. For this purpose a `--resume` option exists that makes sure to use the context from the saved model file: `xnmt --resume /path/to/saved-model.mod`.

This feature is currently implemented only in a very basic form: When resuming a crashed experiment, this will cause the whole experiment to be carried out from the start. When resuming a finished experiment, *xnmt* will return without performing any action. In the future, this will be extended to support resuming from the most recent saved checkpoint, etc.

2.3 Examples

Here are more elaborate examples from the github repository.

2.3.1 Standard

```
# A standard setup, specifying model architecture, training parameters,
# and evaluation of the trained model
!Experiment # 'standard' is the name given to the experiment
  name: standard # every experiment needs a name
  # global parameters shared throughout the experiment
  exp_global: !ExpGlobal
    # {EXP_DIR} is a placeholder for the directory in which the config file lies.
    # {EXP} is a placeholder for the experiment name (here: 'standard')
    model_file: '{EXP_DIR}/models/{EXP}.mod'
```

(continues on next page)

(continued from previous page)

```
    log_file: '{EXP_DIR}/logs/{EXP}.log'
    default_layer_dim: 512
    dropout: 0.3
  # model architecture
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    src_embedder: !SimpleWordEmbedder
      emb_dim: 512
    encoder: !BiLSTMSeqTransducer
      layers: 1
    attender: !MlpAttender
      hidden_dim: 512
      state_dim: 512
      input_dim: 512
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
        emb_dim: 512
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 512
        activation: 'tanh'
      bridge: !CopyBridge {}
      scorer: !Softmax {}
  # training parameters
  train: !SimpleTrainingRegimen
    batcher: !SrcBatcher
      batch_size: 32
    trainer: !AdamTrainer
      alpha: 0.001
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  # final evaluation
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.2 Minimal

```
# Most entries in the config file have default values and don't need to be
# specified explicitly. This config file produces the same results as
# 01_standard.yaml.
# Default parameters are specified and documented directly in the __init__()
# method of the corresponding classes.
```

(continues on next page)

(continued from previous page)

```
# For example,xnmt.translator.DefaultTranslator.__init__()
# specifies MlpAttender as the default attender, which will be used in this
# examples since nothing is specified.
!Experiment
  name: minimal
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.3 Multiple experiments

```
# A config file can contain multiple experiments.
# These are run in sequence.
# It's also possible to run experiments in parallel:
# by default, experiments are skipped when the corresponding log file already
# exists, i.e. when the experiment is currently running or has already finnished.
# That means it's safe to run ``xnmt my_config.yaml`` on the same config file
# multiple times.
#
# This particular examples runs the same experiment, changing only the amount
# of dropout. model, train, evaluate settings are shared using YAML anchors,
# see here for more information: http://yaml.readthedocs.io/en/latest/example.html
#
# There are two ways of specifying multiple experiments: the dictionary-way and the
# list-way. The dictionary-way is shown below. Here, dictionary keys are experiment
# names and the values are !Experiment objects. The order is determined by␣
↪lexicographic
# ordering of the experiment names.
exp1_dropout: !Experiment
  exp_global: !ExpGlobal
    dropout: 0.5
  model: &my_model !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: &my_train !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
```

(continues on next page)

(continued from previous page)

```
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  evaluate: &my_eval
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp

exp2_no_dropout: !Experiment
  exp_global: !ExpGlobal
    dropout: 0.0
  model: *my_model
  train: *my_train
  evaluate: *my_eval
```

```
# This example demonstrates specifying multiple experiments as a list.
# Here, the list makes the order of experiments explicit.
# Experiment names have to be passed as arguments to !Experiment
- !Experiment
  name: exp1_dropout
  exp_global: !ExpGlobal
    dropout: 0.5
  model: &my_model !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: &my_train !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  evaluate: &my_eval
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp

- !Experiment
  name: exp2_no_dropout
  exp_global: !ExpGlobal
    dropout: 0.0
  model: *my_model
  train: *my_train
  evaluate: *my_eval
```

```
# Finally, it's possible to specify a single experiment as top-level entry,
# where again the experiment name has to be passed as an argument.
```

(continues on next page)

(continued from previous page)

```
!Experiment
  name: exp1_dropout
  exp_global: !ExpGlobal
    dropout: 0.5
  model: &my_model !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: &my_train !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  evaluate: &my_eval
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.4 Settings

```
# The basic XNMT behavior can be controlled via predefined configurations.
# These are defined under xnmt/settings.py, and include "standard", "debug", and
↪"unittest" settings.
# These specify things like verbosity, default paths, whether experiments should be␣
↪skipped if the log file already
# exists, and whether to activate the DyNet check_validity and immediate_compute␣
↪options.
#
# As the name suggests, e.g. when debugging one might use XNMT as follows:
# ``xnmt --settings=debug examples/04_settings.yaml``
#
# It is easy to change behavior by either changing these configurations, or adding a␣
↪new configuration to the module.
!Experiment
  name: settings-exp
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  evaluate:
```

(continues on next page)

(continued from previous page)

```
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.5 Preprocessing

```
# XNMT supports various ways to preprocess data as demonstrated in this example.
# Note that some preprocessing functionality relies on third-party tools.
!Experiment
  name: preproc
  exp_global: !ExpGlobal
    # define some named strings that can be used throughout the experiment config:
    placeholders:
      DATA_IN: examples/data/
      DATA_OUT: examples/preproc/
  preproc: !PreprocRunner
    overwrite: False
    tasks:
    - !PreprocTokenize
      in_files:
      - '{DATA_IN}/train.ja'
      - '{DATA_IN}/train.en'
      - '{DATA_IN}/dev.ja'
      - '{DATA_IN}/dev.en'
      - '{DATA_IN}/test.ja'
      - '{DATA_IN}/test.en'
      out_files:
      - '{DATA_OUT}/train.tok.ja'
      - '{DATA_OUT}/train.tok.en'
      - '{DATA_OUT}/dev.tok.ja'
      - '{DATA_OUT}/dev.tok.en'
      - '{DATA_OUT}/test.tok.ja'
      - '{DATA_OUT}/test.tok.en'
      specs:
      - filenum: all
        tokenizers:
        - !UnicodeTokenizer {}
    - !PreprocNormalize
      in_files:
      - '{DATA_OUT}/train.tok.ja'
      - '{DATA_OUT}/train.tok.en'
      - '{DATA_OUT}/dev.tok.ja'
      - '{DATA_OUT}/dev.tok.en'
      - '{DATA_OUT}/test.tok.ja'
      - '{DATA_OUT}/test.tok.en'
      - '{DATA_IN}/dev.en'
      - '{DATA_IN}/test.en'
      out_files:
      - '{DATA_OUT}/train.tok.norm.ja'
      - '{DATA_OUT}/train.tok.norm.en'
      - '{DATA_OUT}/dev.tok.norm.ja'
      - '{DATA_OUT}/dev.tok.norm.en'
      - '{DATA_OUT}/test.tok.norm.ja'
```

(continues on next page)

(continued from previous page)

```
    - '{DATA_OUT}/test.tok.norm.en'
    - '{DATA_OUT}/dev.norm.en'
    - '{DATA_OUT}/test.norm.en'
    specs:
    - filenum: all
      normalizers:
      - !NormalizerLower {}
  - !PreprocFilter
    in_files:
    - '{DATA_OUT}/train.tok.norm.ja'
    - '{DATA_OUT}/train.tok.norm.en'
    out_files:
    - '{DATA_OUT}/train.tok.norm.filter.ja'
    - '{DATA_OUT}/train.tok.norm.filter.en'
    specs:
    - !SentenceFiltererLength
      min_all: 1
      max_all: 60
  - !PreprocVocab
    in_files:
    - '{DATA_OUT}/train.tok.norm.ja'
    - '{DATA_OUT}/train.tok.norm.en'
    out_files:
    - '{DATA_OUT}/train.vocab.ja'
    - '{DATA_OUT}/train.vocab.en'
    specs:
    - filenum: all
      filters:
      - !VocabFiltererFreq
          min_freq: 2
model: !DefaultTranslator
  src_reader: !PlainTextReader
    vocab: !Vocab
      vocab_file: examples/preproc/train.vocab.ja
  trg_reader: !PlainTextReader
    vocab: !Vocab
      vocab_file: examples/preproc/train.vocab.en
  src_embedder: !SimpleWordEmbedder
    emb_dim: 512
  encoder: !BiLSTMSeqTransducer
    layers: 1
  attender: !MlpAttender
    hidden_dim: 512
    state_dim: 512
    input_dim: 512
  decoder: !AutoRegressiveDecoder
    embedder: !SimpleWordEmbedder
      emb_dim: 512
    rnn: !UniLSTMSeqTransducer
      layers: 1
    transform: !AuxNonLinear
      output_dim: 512
    bridge: !NoBridge {}
  inference: !AutoRegressiveInference
    post_process: join-piece
train: !SimpleTrainingRegimen
  run_for_epochs: 20
```

(continues on next page)

(continued from previous page)

```
    src_file: '{DATA_OUT}/dev.tok.norm.ja'
    trg_file: '{DATA_OUT}/dev.tok.norm.en'
    dev_tasks:
      - !AccuracyEvalTask
        eval_metrics: bleu
        src_file: '{DATA_OUT}/dev.tok.norm.ja'
        ref_file: '{DATA_OUT}/dev.norm.en'
        hyp_file: examples/output/{EXP}.dev_hyp
      - !LossEvalTask
        src_file: '{DATA_OUT}/dev.tok.norm.ja'
        ref_file: '{DATA_OUT}/dev.tok.norm.en'
  evaluate:
  - !AccuracyEvalTask
    eval_metrics: bleu
    src_file: '{DATA_OUT}/test.tok.norm.ja'
    ref_file: '{DATA_OUT}/test.norm.en'
    hyp_file: examples/output/{EXP}.test_hyp
```

2.3.6 Early stopping

```
# Early stopping is achieved by configuring SimpleTrainingRegimen, with the following␣
↪options:
# - run_for_epochs
# - lr_decay
# - lr_decay_times
# - patience
# - initial_patience
# - dev_tasks (to configure the metric used to determine lr decay or early stopping)
!Experiment
  name: minimal-early-stopping
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: !SimpleTrainingRegimen
    run_for_epochs: 100 # maximum number of epochs, but might stop earlier depending␣
↪on the following settings.
    lr_decay: 0.5
    lr_decay_times: 3
    patience: 1
    initial_patience: 2
    dev_tasks: # the first metric (here: bleu) is used for checking whether LR should␣
↪be decayed.
      - !AccuracyEvalTask
        eval_metrics: bleu,gleu
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
        hyp_file: examples/output/{EXP}.test_hyp
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
  evaluate:
```

(continues on next page)

(continued from previous page)

```
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.7 Fine-tuning

```
# Saving and loading models is a key feature demonstrated in this config file.
# This example shows how to load a trained model for fine tuning.
# pretrained model.
exp1-pretrain-model: !Experiment
  exp_global: !ExpGlobal
    # The model file contain the whole contents of this experiment in YAML
    # format. Note that {EXP} expressions are left intact when saving.
    default_layer_dim: 64
    dropout: 0.3
    weight_noise: 0.1
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    src_embedder: !SimpleWordEmbedder
      emb_dim: 64
    encoder: !BiLSTMSeqTransducer
      layers: 2
      input_dim: 64
    attender: !MlpAttender
      state_dim: 64
      hidden_dim: 64
      input_dim: 64
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
        emb_dim: 64
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 64
      input_feeding: True
      bridge: !CopyBridge {}
    inference: !AutoRegressiveInference {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.dev_hyp
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
```

(continues on next page)

(continued from previous page)

```
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp

exp2-finetune-model: !LoadSerialized
  # This will load the contents of the above experiments that were saved to the
  # YAML file specified after filename:
  # This will carry out the exact same thing, except that {EXP} is resolved to
  # a different value (making sure we don't overwrite the previous model),
  # and except for the things explicitly overwritten in the overwrite: section.
  # It's possible to change any settings as long as these don't change the number
  # or nature of DyNet parameters allocated for the component.
  filename: examples/models/exp1-pretrain-model.mod
  path: ''
  overwrite: # list of [path, value] pairs. Value can be scalar or an arbitrary object
  - path: train.trainer
    val: !AdamTrainer
        alpha: 0.0002
  - path: exp_global.dropout
    val: 0.5
  - path: train.dev_zero
    val: True
  - path: status
    val: null
```

2.3.8 Beam search

```
# This example shows how to configure beam search, and how use the loading mechanism␣
↪for the purpose of evaluating a
# model.
exp1-train-model: !Experiment
  exp_global: !ExpGlobal
    # The model file contain the whole contents of this experiment in YAML
    # format. Note that {EXP} expressions are left intact when saving.
    model_file: examples/output/{EXP}.mod
    log_file: examples/output/{EXP}.log
    default_layer_dim: 64
    dropout: 0.5
    weight_noise: 0.1
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    src_embedder: !SimpleWordEmbedder
      emb_dim: 64
    encoder: !BiLSTMSeqTransducer
      layers: 2
      input_dim: 64
    attender: !MlpAttender
      state_dim: 64
      hidden_dim: 64
      input_dim: 64
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
```

(continues on next page)

(continued from previous page)

```
        emb_dim: 64
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 64
      input_feeding: True
      bridge: !CopyBridge {}
    inference: !AutoRegressiveInference
      search_strategy: !BeamSearch
        beam_size: 5
        len_norm: !PolynomialNormalization
          apply_during_search: true
          m: 0.8
  train: !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.dev_hyp
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp

exp2-eval-model: !LoadSerialized
  filename: examples/output/exp1-train-model.mod
  overwrite: # list of [path, value] pairs. Value can be scalar or an arbitrary object
  - path: train # skip the training loop
    val: null
  - path: status
    val: null
  - path: model.inference.search_strategy.beam_size # try some new beam settings
    val: 10
  - path: evaluate
    val: # (re-)define test data and other evaluation settings
    - !AccuracyEvalTask
      eval_metrics: bleu,gleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.9 Programmatic usage

```
# It is also possible to configure model training using Python code rather than
# YAML config files. This is less convenient and usually not necessary, but there
# may be cases where the added flexibility is needed. This basically works by
# using XNMT as a library of components that are initialized and run in this
# config file.
#
```

(continues on next page)

(continued from previous page)

```
# This demonstrates a standard model training, including set up of logging, model
# saving, etc.; models are saved into YAML files that can again be loaded using
# the standard YAML  way (examples/07_load_finetune.yaml) or the Python way
# (10_programmatic_load.py)
#
# To launch this, use ``python -m examples.09_programmatic``, making sure that XNMT
# setup.py has been run properly.


import os
import random

import numpy as np

from xnmt.modelparts.attenders import MlpAttender
from xnmt.batchers import SrcBatcher, InOrderBatcher
from xnmt.modelparts.bridges import CopyBridge
from xnmt.modelparts.decoders import AutoRegressiveDecoder
from xnmt.modelparts.embedders import SimpleWordEmbedder
from xnmt.eval.tasks import LossEvalTask, AccuracyEvalTask
from xnmt.experiments import Experiment
from xnmt.inferences import AutoRegressiveInference
from xnmt.input_readers import PlainTextReader
from xnmt.transducers.recurrent import BiLSTMSeqTransducer, UniLSTMSeqTransducer
from xnmt.modelparts.transforms import AuxNonLinear
from xnmt.modelparts.scorers import Softmax
from xnmt.optimizers import AdamTrainer
from xnmt.param_collections import ParamManager
from xnmt.persistence import save_to_file
import xnmt.tee
from xnmt.train.regimens import SimpleTrainingRegimen
from xnmt.models.translators.default import DefaultTranslator
from xnmt.vocabs import Vocab

seed=13
random.seed(seed)
np.random.seed(seed)

EXP_DIR = os.path.dirname(__file__)
EXP = "programmatic"

model_file = f"{EXP_DIR}/models/{EXP}.mod"
log_file = f"{EXP_DIR}/logs/{EXP}.log"

xnmt.tee.set_out_file(log_file, EXP)

ParamManager.init_param_col()
ParamManager.param_col.model_file = model_file

src_vocab = Vocab(vocab_file="examples/data/head.ja.vocab")
trg_vocab = Vocab(vocab_file="examples/data/head.en.vocab")

batcher = SrcBatcher(batch_size=64)

inference = AutoRegressiveInference(batcher=InOrderBatcher(batch_size=1))

layer_dim = 512
```

(continues on next page)

(continued from previous page)

```
model = DefaultTranslator(
  src_reader=PlainTextReader(vocab=src_vocab),
  trg_reader=PlainTextReader(vocab=trg_vocab),
  src_embedder=SimpleWordEmbedder(emb_dim=layer_dim, vocab_size=len(src_vocab)),

  encoder=BiLSTMSeqTransducer(input_dim=layer_dim, hidden_dim=layer_dim, layers=1),
  attender=MlpAttender(hidden_dim=layer_dim, state_dim=layer_dim, input_dim=layer_
↪dim),
  decoder=AutoRegressiveDecoder(input_dim=layer_dim,
                                embedder=SimpleWordEmbedder(emb_dim=layer_dim, vocab_
↪size=len(trg_vocab)),
                                rnn=UniLSTMSeqTransducer(input_dim=layer_dim, hidden_
↪dim=layer_dim,
                                                         decoder_input_dim=layer_dim,␣
↪yaml_path="decoder"),
                                transform=AuxNonLinear(input_dim=layer_dim, output_
↪dim=layer_dim,
                                                       aux_input_dim=layer_dim),
                                scorer=Softmax(vocab_size=len(trg_vocab), input_
↪dim=layer_dim),
                                bridge=CopyBridge(dec_dim=layer_dim, dec_layers=1)),
  inference=inference
)

train = SimpleTrainingRegimen(
  name=f"{EXP}",
  model=model,
  batcher=batcher,
  trainer=AdamTrainer(alpha=0.001),
  run_for_epochs=2,
  src_file="examples/data/head.ja",
  trg_file="examples/data/head.en",
  dev_tasks=[LossEvalTask(src_file="examples/data/head.ja",
                          ref_file="examples/data/head.en",
                          model=model,
                          batcher=batcher)],
)

evaluate = [AccuracyEvalTask(eval_metrics="bleu,wer",
                             src_file="examples/data/head.ja",
                             ref_file="examples/data/head.en",
                             hyp_file=f"examples/output/{EXP}.test_hyp",
                             inference=inference,
                             model=model)]

standard_experiment = Experiment(
  name="programmatic",
  model=model,
  train=train,
  evaluate=evaluate
)

# run experiment
standard_experiment(save_fct=lambda: save_to_file(model_file, standard_experiment))

exit()
```

2.3.10 Programmatic loading

```
# This demonstrates how to load the model trained using ``09_programmatic.py``
# the programmatic way and for the purpose of evaluating the model.

import os

import xnmt.tee
from xnmt.param_collections import ParamManager
from xnmt.persistence import initialize_if_needed, YamlPreloader, LoadSerialized,␣
↪save_to_file

EXP_DIR = os.path.dirname(__file__)
EXP = "programmatic-load"

model_file = f"{EXP_DIR}/models/{EXP}.mod"
log_file = f"{EXP_DIR}/logs/{EXP}.log"

xnmt.tee.set_out_file(log_file, EXP)

ParamManager.init_param_col()

load_experiment = LoadSerialized(
  filename=f"{EXP_DIR}/models/programmatic.mod",
  overwrite=[
    {"path" : "train", "val" : None},
    {"path": "status", "val": None},
  ]
)

uninitialized_experiment = YamlPreloader.preload_obj(load_experiment, exp_dir=EXP_DIR,
↪ exp_name=EXP)
loaded_experiment = initialize_if_needed(uninitialized_experiment)

# if we were to continue training, we would need to set a save model file like this:
# ParamManager.param_col.model_file = model_file
ParamManager.populate()

# run experiment
loaded_experiment(save_fct=lambda: None)
```

2.3.11 Parameter sharing

```
# This illustrates component and parameter sharing. This is useful for making
# config files less verbose, and more importantly makes it possible to realize
# weight-sharing between components, which will also be demonstrated in the
# multi-task example later.
#
# There are 2 ways to achieve sharing:
# - YAML's anchor system where '&' denotes a named anchor, '*' denotes a reference to␣
↪an anchor.
#   This essentially copies values or subcomponents from one place to another.
#   It can be combined with the << operator that allows copying parts of a dictionary,
↪ but overwriting other parts.
#   More info is found here: http://yaml.readthedocs.io/en/latest/example.html
```

(continues on next page)

(continued from previous page)

```
# - XNMT's !Ref object creates a reference, meaning both places will point to the␣
↪exact same Python object,
#   and that DyNet parameters will be shared.
#   References can be made by path or by name, as illustrated below. The name refers␣
↪to a _xnmt_id that can
#   be set in any component and must be unique.
#   Note that references do not work across experiments (e.g. we cannot refer to exp2.
↪load from within exp1.pretrain)

exp1.pretrain: !Experiment
  exp_global: !ExpGlobal
    default_layer_dim: 32
    model_file: 'examples/output/{EXP}.mod'
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    src_embedder: !SimpleWordEmbedder
      emb_dim: 32
    encoder: !BiLSTMSeqTransducer
      layers: 1
    attender: !MlpAttender {}
    # reference-sharing between softmax projection and target embedder. This means␣
↪both layers share DyNet parameters!
    decoder: !AutoRegressiveDecoder
      embedder: !DenseWordEmbedder
        _xnmt_id: trg_emb # this id must be unique and is needed to create a␣
↪reference-by-name below.
        emb_dim: 32
      rnn: !UniLSTMSeqTransducer
        layers: 1
      scorer: !Softmax
        output_projector: !Ref { name: trg_emb }
        #                 alternatively, the same could be achieved like this,
        #                 in which case model.decoder.embedder._xnmt_id is not␣
↪required:
        #                 !Ref { path: model.decoder.embedder }
      bridge: !CopyBridge {}
    inference: !AutoRegressiveInference {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: &dev_src examples/data/head.ja  # value-sharing between train.
↪training_corpus.dev_src and inference.src_file
        ref_file: &dev_trg examples/data/head.en  # value-sharing between train.
↪training_corpus.dev_trg and evaluate.ref_file
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: *dev_src # Copy over the file path from the dev tasks using YAML␣
↪anchors.
      ref_file: *dev_trg # The same could also be done for more complex objects.
      hyp_file: examples/output/{EXP}.test_hyp
```

(continues on next page)

(continued from previous page)

```
exp2.load: !LoadSerialized
  filename: examples/output/exp1.pretrain.mod
```

2.3.12 Multi-task

```
# XNMT offers a very flexible way of multi-task training by specifying multiple
# models and using the !Ref mechanism for weight sharing, as demonstrated
# in this config file.
# The possible multi-task training strategies can be looked up in
# xnmt/regimens.py and include same-batch, alternating-batch, and serial
# strategies.
exp1-multi_task: !Experiment
  exp_global: !ExpGlobal
    model_file: examples/output/{EXP}.mod
    log_file: examples/output/{EXP}.log
    default_layer_dim: 64
  train: !SameBatchMultiTaskTrainingRegimen
    trainer: !AdamTrainer {}
    n_task_steps: [2,1]
    tasks:
    - !SimpleTrainingTask # first task is the main task: it will control early␣
↪stopping, learning rate schedule, model checkpoints, ..
      name: first_task
      run_for_epochs: 6
      batcher: !SrcBatcher
        batch_size: 6
      src_file: examples/data/head.ja
      trg_file: examples/data/head.en
      model: !DefaultTranslator
        _xnmt_id: first_task_model
        src_reader: !PlainTextReader
          vocab: !Vocab
            _xnmt_id: src_vocab
            vocab_file: examples/data/head.ja.vocab
        trg_reader: !PlainTextReader
          vocab: !Vocab
            _xnmt_id: trg_vocab
            vocab_file: examples/data/head.en.vocab
        src_embedder: !SimpleWordEmbedder
          emb_dim: 64
          vocab: !Ref {name: src_vocab}
        encoder: !BiLSTMSeqTransducer # the encoder shares parameters between tasks
          _xnmt_id: first_task_encoder
          layers: 1
        attender: !MlpAttender
          state_dim: 64
          hidden_dim: 64
          input_dim: 64
        decoder: !AutoRegressiveDecoder
          embedder: !SimpleWordEmbedder
            emb_dim: 64
            vocab: !Ref {name: trg_vocab}
          rnn: !UniLSTMSeqTransducer
```

(continues on next page)

(continued from previous page)

```
            layers: 1
            hidden_dim: 64
          bridge: !CopyBridge {}
          scorer: !Softmax
            vocab: !Ref {name: trg_vocab}
      dev_tasks:
        - !AccuracyEvalTask
          model: !Ref { name: first_task_model }
          src_file: &first_task_dev_src examples/data/head.ja  # value-sharing␣
↪between first task dev and final eval
          ref_file: &first_task_dev_trg examples/data/head.en  # value-sharing␣
↪between first task dev and final eval
          hyp_file: examples/output/{EXP}.first_dev_hyp
          eval_metrics: bleu # tasks can specify different dev_metrics
    - !SimpleTrainingTask
      name: second_task
      batcher: !SrcBatcher
        batch_size: 6
      src_file: examples/data/head.ja
      trg_file: examples/data/head.en
      model: !DefaultTranslator
        _xnmt_id: second_task_model
        src_reader: !PlainTextReader
          vocab: !Ref {name: src_vocab}
        trg_reader: !PlainTextReader
          vocab: !Ref {name: trg_vocab}
        src_embedder: !SimpleWordEmbedder
          emb_dim: 64
          vocab: !Ref {name: src_vocab}
        encoder: !Ref { name: first_task_encoder }
        attender: !MlpAttender
          state_dim: 64
          hidden_dim: 64
          input_dim: 64
        decoder: !AutoRegressiveDecoder
          embedder: !SimpleWordEmbedder
            emb_dim: 64
            vocab: !Ref {name: trg_vocab}
          bridge: !CopyBridge {}
          scorer: !Softmax
            vocab: !Ref {name: trg_vocab}
      dev_tasks:
        - !AccuracyEvalTask
          model: !Ref { name: second_task_model }
          src_file: examples/data/head.ja
          ref_file: examples/data/head.en
          hyp_file: examples/output/{EXP}.second_dev_hyp
          eval_metrics: gleu # tasks can specify different dev_metrics
  evaluate:
    - !AccuracyEvalTask
      model: !Ref { name: first_task_model }
      eval_metrics: bleu
      src_file: *first_task_dev_src
      ref_file: *first_task_dev_trg
      hyp_file: examples/output/{EXP}.test_hyp

exp2-finetune-model: !LoadSerialized
```

(continues on next page)

(continued from previous page)

```
  filename: examples/output/exp1-multi_task.mod
```

2.3.13 Speech

```
# This config file demonstrates how to specify a speech recognition model
# using the Listen-Attend-Spell architecture: https://arxiv.org/pdf/1508.01211.pdf
# Compared to the conventional attentional model, we remove input embeddings,
# instead directly read in a feature vector the pyramidal LSTM reduces length of
# the input sequence by a factor of 2 per layer (except for the first layer).
# Output units should be characters according to the paper.
!Experiment
  name: speech
  exp_global: !ExpGlobal
    save_num_checkpoints: 2
    default_layer_dim: 32
    dropout: 0.4
  preproc: !PreprocRunner
    overwrite: False
    tasks:
    - !PreprocExtract
      in_files:
      - examples/data/LDC94S13A.yaml
      out_files:
      - examples/data/LDC94S13A.h5
      specs: !MelFiltExtractor {}
  model: !DefaultTranslator
    src_embedder: !NoopEmbedder
      emb_dim: 40
    encoder: !PyramidalLSTMSeqTransducer
      layers: 3
      downsampling_method: concat
      reduce_factor: 2
      input_dim: 40
      hidden_dim: 64
    attender: !MlpAttender
      state_dim: 64
      hidden_dim: 64
      input_dim: 64
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
        emb_dim: 64
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 64
      bridge: !CopyBridge {}
    src_reader: !H5Reader
      transpose: True
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/char.vocab}
      output_proc: join-char
  train: !SimpleTrainingRegimen
    run_for_epochs: 1
    batcher: !SrcBatcher
      pad_src_to_multiple: 4
```

(continues on next page)

(continued from previous page)

```
    batch_size: 3
  trainer: !AdamTrainer {}
  src_file: examples/data/LDC94S13A.h5
  trg_file: examples/data/LDC94S13A.char
  dev_tasks:
    - !LossEvalTask
      src_file: examples/data/LDC94S13A.h5
      ref_file: examples/data/LDC94S13A.char
    - !AccuracyEvalTask
      eval_metrics: cer,wer
      src_file: examples/data/LDC94S13A.h5
      ref_file: examples/data/LDC94S13A.char
      hyp_file: examples/output/{EXP}.dev_hyp
      inference: !AutoRegressiveInference
        batcher: !InOrderBatcher
          _xnmt_id: inference_batcher
          pad_src_to_multiple: 4
          batch_size: 1
evaluate:
  - !AccuracyEvalTask
    eval_metrics: cer,wer
    src_file: examples/data/LDC94S13A.h5
    ref_file: examples/data/LDC94S13A.words
    hyp_file: examples/output/{EXP}.test_hyp
    inference: !AutoRegressiveInference
      batcher: !Ref { name: inference_batcher }
```

2.3.14 Reporting attention matrices

```
# XNMT supports writing out reports, such as attention matrices generated during␣
↪inference or difference highlighting
# between outputs and references.
# These are generally created by setting exp_global.compute_report to True, and␣
↪adding one or several reporters
# to the inference class.
!Experiment
  name: report
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  train: !SimpleTrainingRegimen
    run_for_epochs: 0
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
  evaluate:
```

(continues on next page)

(continued from previous page)

```
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
      inference: !AutoRegressiveInference
        reporter:
        - !AttentionReporter {} # plot attentions
        - !ReferenceDiffReporter {} # difference highlighting
        - !CompareMtReporter {} # analyze MT outputs
        - !OOVStatisticsReporter # report on recovered OOVs, fantasized new words,␣
↪etc.
            train_trg_file: examples/data/head.en
```

2.3.15 Scoring N-best lists

```
# Using a trained model to add hypothesis score for an nbest list
# First, exp1-model trains a model which is saved at examples/output/exp1-model.mod
# Then, exp2-score loads the exp1-model, and use it to score an nbest list
# The nbest list example used here is located at examples/data/head.nbest.en
# exp2-score outputs a new nbest list with hypothesis score.
# The output file will be in examples/output/exp2-score.test_hyp

exp1-model: !Experiment
  exp_global: !ExpGlobal
    model_file: examples/output/{EXP}.mod
    log_file: examples/output/{EXP}.log
    default_layer_dim: 64
    dropout: 0.5
    weight_noise: 0.1
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    src_embedder: !SimpleWordEmbedder
      emb_dim: 64
    encoder: !BiLSTMSeqTransducer
      layers: 2
      input_dim: 64
    attender: !MlpAttender
      state_dim: 64
      hidden_dim: 64
      input_dim: 64
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
        emb_dim: 64
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 64
      input_feeding: True
      bridge: !CopyBridge {}
    inference: !AutoRegressiveInference {}
  train: !SimpleTrainingRegimen
```

(continues on next page)

(continued from previous page)

```
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.dev_hyp
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp

exp2-score: !LoadSerialized
  filename: examples/output/exp1-model.mod
  overwrite:
  - path: train
    val: ~
  - path: model.inference
    val: !AutoRegressiveInference
      mode: score
      ref_file: examples/data/head.nbest.en
      src_file: examples/data/head.ja
  - path: evaluate.0
    val: !AccuracyEvalTask
      src_file: examples/data/head.ja
      ref_file: examples/data/head.nbest.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.16 Ensembling

```
# This example shows different ways to perform model ensembling

# First, let's a define a simple experiment with a single model
exp1-single: !Experiment
  exp_global: &globals !ExpGlobal
    model_file: examples/output/{EXP}.mod
    log_file: examples/output/{EXP}.log
    default_layer_dim: 32
  # Just use default model settings here
  model: &model1 !DefaultTranslator
    src_reader: &src_reader !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: &trg_reader !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
  train: &train !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
```

(continues on next page)

(continued from previous page)

```
        ref_file: examples/data/head.en

# Another single model, but with a different number of layers and some other
# different settings
exp2-single: !Experiment
  exp_global: *globals
  model: &model2 !DefaultTranslator
    src_reader: *src_reader
    trg_reader: *trg_reader
    encoder: !BiLSTMSeqTransducer
      layers: 3
      hidden_dim: 64
    decoder: !AutoRegressiveDecoder
      embedder: !DenseWordEmbedder
        _xnmt_id: dense_embed
        emb_dim: 64
      rnn: !UniLSTMSeqTransducer
        hidden_dim: 64
      transform: !AuxNonLinear
        output_dim: 64
      scorer: !Softmax
        output_projector: !Ref {name: dense_embed}
  train: *train

# Load the previously trained models and combine them to an ensemble
exp3-ensemble-load: !Experiment
  exp_global: *globals
  model: !EnsembleTranslator
    src_reader: !Ref {path: model.models.0.src_reader}
    trg_reader: !Ref {path: model.models.0.trg_reader}
    models:
      - !LoadSerialized
        filename: 'examples/output/exp1-single.mod'
        path: model
      - !LoadSerialized
        filename: 'examples/output/exp2-single.mod'
        path: model
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu,wer
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp

# Alternatively, we can also hook up the models during training time already
exp4-ensemble-train: !Experiment
  exp_global: *globals
  model: !EnsembleTranslator
    src_reader: *src_reader
    trg_reader: *trg_reader
    models:
      - *model1
      - *model2
  train: *train
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu,wer
```

(continues on next page)

(continued from previous page)

```
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.17 Minimum risk training

```
# Saving and loading models is a key feature demonstrated in this config file.
# This example shows how to load a trained model for fine tuning.
# pretrained model.
exp1-pretrain-model: !Experiment
  exp_global: !ExpGlobal
    # The model file contain the whole contents of this experiment in YAML
    # format. Note that {EXP} expressions are left intact when saving.
    default_layer_dim: 64
    dropout: 0.3
    weight_noise: 0.1
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    src_embedder: !SimpleWordEmbedder
      emb_dim: 64
    encoder: !BiLSTMSeqTransducer
      layers: 2
      input_dim: 64
    attender: !MlpAttender
      state_dim: 64
      hidden_dim: 64
      input_dim: 64
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
        emb_dim: 64
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 64
      input_feeding: True
      bridge: !CopyBridge {}
    inference: !AutoRegressiveInference {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 2
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.dev_hyp
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
```

(continues on next page)

(continued from previous page)

```
    hyp_file: examples/output/{EXP}.test_hyp

exp2-finetune-minrisk: !LoadSerialized
  # This will perform minimum risk training with SamplingSearch.
  # Same as above, the pretrained model will be loaded and an appropriate search_
↪strategy
  # will be used during minimum risk training.
  filename: examples/models/exp1-pretrain-model.mod
  path: ''
  overwrite:
  - path: train.loss_calculator
    val: !MinRiskLoss
      alpha: 0.005
  - path: model.inference.search_strategy
    val: !SamplingSearch
      sample_size: 10
      max_len: 50
  - path: train.run_for_epochs
    val: 1
```

2.3.18 Biased Lexicon

(this is currently broken)

```
lexbias: !Experiment # 'standard' is the name given to the experiment
  exp_global: !ExpGlobal
    model_file: '{EXP_DIR}/models/{EXP}.mod'
    log_file: '{EXP_DIR}/logs/{EXP}.log'
    default_layer_dim: 512
    dropout: 0.3
  # model architecture
  model: !DefaultTranslator
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    src_embedder: !SimpleWordEmbedder
      emb_dim: 512
    encoder: !BiLSTMSeqTransducer
      layers: 1
    attender: !MlpAttender
      hidden_dim: 512
      state_dim: 512
      input_dim: 512
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
        emb_dim: 512
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 512
      bridge: !CopyBridge {}
      scorer: !LexiconSoftmax
        lexicon_file: examples/data/head-ja_given_en.lex
        # can choose between bias/linear
```

(continues on next page)

(continued from previous page)

```
        lexicon_type: bias
        # The small epsilon value to be added to the bias
        lexicon_alpha: 0.001
  # training parameters
  train: !SimpleTrainingRegimen
    batcher: !SrcBatcher
      batch_size: 32
    trainer: !AdamTrainer
      alpha: 0.001
    run_for_epochs: 2
    src_file: examples/data/head.en
    trg_file: examples/data/head.ja
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.en
        ref_file: examples/data/head.ja
  # final evaluation
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.en
      ref_file: examples/data/head.ja
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.19 Subword Sampling

```
# Sampling subword units for subword regularization
# Note that this requires 'sentencepiece' as an extra dependency
!Experiment
  name: subword_sample
  exp_global: !ExpGlobal
    model_file: '{EXP_DIR}/models/{EXP}.mod'
    log_file: '{EXP_DIR}/logs/{EXP}.log'
    default_layer_dim: 512
    dropout: 0.3
  model: !DefaultTranslator
    # Here we set the sample_train and alpha parameters to turn on sampling
    src_reader: !SentencePieceTextReader
      sample_train: True
      alpha: 0.1
      vocab: !Vocab
        vocab_file: examples/data/big-ja.vocab
        sentencepiece_vocab: True
      model_file: examples/data/big-ja.model
    trg_reader: !SentencePieceTextReader
      sample_train: True
      alpha: 0.1
      vocab: !Vocab
        vocab_file: examples/data/big-en.vocab
        sentencepiece_vocab: True
      model_file: examples/data/big-en.model
    src_embedder: !SimpleWordEmbedder
      emb_dim: 512
    encoder: !BiLSTMSeqTransducer
      layers: 1
```

(continues on next page)

(continued from previous page)

```
    attender: !MlpAttender
      hidden_dim: 512
      state_dim: 512
      input_dim: 512
    decoder: !AutoRegressiveDecoder
      embedder: !SimpleWordEmbedder
        emb_dim: 512
      rnn: !UniLSTMSeqTransducer
        layers: 1
      transform: !AuxNonLinear
        output_dim: 512
        activation: 'tanh'
      bridge: !CopyBridge {}
    inference: !AutoRegressiveInference
      post_process: join-piece
  # training parameters
  train: !SimpleTrainingRegimen
    batcher: !SrcBatcher
      batch_size: 32
    trainer: !AdamTrainer
      alpha: 0.001
    run_for_epochs: 20
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  # final evaluation
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.20 Self Attention

```
# A setup using self-attention
!Experiment
  name: self_attention
  exp_global: !ExpGlobal
    model_file: '{EXP_DIR}/models/{EXP}.mod'
    log_file: '{EXP_DIR}/logs/{EXP}.log'
    default_layer_dim: 512
    dropout: 0.3
    placeholders:
      DATA_IN: examples/data
      DATA_OUT: examples/preproc
  preproc: !PreprocRunner
    overwrite: False
    tasks:
    - !PreprocVocab
      in_files:
      - '{DATA_IN}/train.ja'
```

(continues on next page)

(continued from previous page)

```
    - '{DATA_IN}/train.en'
    out_files:
    - '{DATA_OUT}/train.ja.vocab'
    - '{DATA_OUT}/train.en.vocab'
    specs:
    - filenum: all
      filters:
      - !VocabFiltererFreq
          min_freq: 2
model: !DefaultTranslator
  src_reader: !PlainTextReader
    vocab: !Vocab {vocab_file: '{DATA_OUT}/train.ja.vocab'}
  trg_reader: !PlainTextReader
    vocab: !Vocab {vocab_file: '{DATA_OUT}/train.en.vocab'}
  src_embedder: !SimpleWordEmbedder
    emb_dim: 512
  encoder: !ModularSeqTransducer
    modules:
    - !PositionalSeqTransducer
      input_dim: 512
      max_pos: 100
      dropout: 0.1
    - !ModularSeqTransducer
      modules: !Repeat
        times: 2
        content: !ModularSeqTransducer
          modules:
          - !ResidualSeqTransducer
            input_dim: 512
            child: !MultiHeadAttentionSeqTransducer
              num_heads: 8
              dropout: 0.1
            layer_norm: True
            dropout: 0.1
          - !ResidualSeqTransducer
            input_dim: 512
            child: !TransformSeqTransducer
              transform: !MLP
                activation: relu
            layer_norm: True
            dropout: 0.1
  attender: !MlpAttender
    hidden_dim: 512
    state_dim: 512
    input_dim: 512
  decoder: !AutoRegressiveDecoder
    embedder: !SimpleWordEmbedder
      emb_dim: 512
    rnn: !UniLSTMSeqTransducer
      layers: 1
    transform: !AuxNonLinear
      output_dim: 512
      activation: 'tanh'
    bridge: !CopyBridge {}
train: !SimpleTrainingRegimen
  batcher: !SrcBatcher
    batch_size: 32
```

(continues on next page)

(continued from previous page)

```
    trainer: !NoamTrainer
      alpha: 1.0
      warmup_steps: 4000
    run_for_epochs: 2
    src_file: examples/data/train.ja
    trg_file: examples/data/train.en
    dev_tasks:
      - !LossEvalTask
        src_file: examples/data/head.ja
        ref_file: examples/data/head.en
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: examples/output/{EXP}.test_hyp
```

2.3.21 Char Segment

```
# Examples of using SegmentingSeqTransducer
# Look available composition functions at xnmt/specialized_encoders/segmenting_
↪encoder/segmenting_composer.py

# Looking up characters from word vocabulary
# Basically this is the same as 01_standard.yaml
seg_lookup: !Experiment
  exp_global: !ExpGlobal {}
  model: !DefaultTranslator
    src_reader: !CharFromWordTextReader
      # Can be produced by script/vocab/make_vocab.py --char_vocab < [CORPUS]
      vocab: !Vocab {vocab_file: examples/data/head.ja.charvocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    # It reads in characters and produce word embeddings
    encoder: !SegmentingSeqTransducer
      segment_composer: !LookupComposer
        word_vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
      final_transducer: !BiLSTMSeqTransducer {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 1
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu,wer
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: test/tmp/{EXP}.test_hyp
      inference: !AutoRegressiveInference {}

# Summing together character composition functions.
seg_sum: !Experiment
  exp_global: !ExpGlobal {}
  model: !DefaultTranslator
    src_reader: !CharFromWordTextReader
```

(continues on next page)

(continued from previous page)

```
      vocab: !Vocab {vocab_file: examples/data/head.ja.charvocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    encoder: !SegmentingSeqTransducer
      ### Pay attention to this part
      segment_composer: !SumComposer {}
      ###
      final_transducer: !BiLSTMSeqTransducer {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 1
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu,wer
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: test/tmp/{EXP}.test_hyp
      inference: !AutoRegressiveInference {}

# Using BiLSTM to predict word embeddings.
seg_bilstm: !Experiment
  exp_global: !ExpGlobal {}
  model: !DefaultTranslator
    src_reader: !CharFromWordTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.charvocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    encoder: !SegmentingSeqTransducer
      ### Pay attention to this part
      segment_composer: !SeqTransducerComposer
        seq_transducer: !BiLSTMSeqTransducer {}
      ###
      final_transducer: !BiLSTMSeqTransducer {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 1
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu,wer
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: test/tmp/{EXP}.test_hyp
      inference: !AutoRegressiveInference {}

# Using CHARAGRAM composition function
seg_charagram: !Experiment
  exp_global: !ExpGlobal {}
  model: !DefaultTranslator
    src_reader: !CharFromWordTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.charvocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    encoder: !SegmentingSeqTransducer
      ### Pay attention to this part
      segment_composer: !CharNGramComposer
```

(continues on next page)

(continued from previous page)

```
        ngram_size: 4
        word_vocab: !Vocab {vocab_file: examples/data/head.ngramcount.ja}
      ###
      final_transducer: !BiLSTMSeqTransducer {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 1
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu,wer
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: test/tmp/{EXP}.test_hyp
      inference: !AutoRegressiveInference {}

# Using Composition of CHARAGRAM and Lookup
seg_lookup_charagram: !Experiment
  exp_global: !ExpGlobal {}
  model: !DefaultTranslator
    src_reader: !CharFromWordTextReader
      vocab: !Vocab {vocab_file: examples/data/head.ja.charvocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    encoder: !SegmentingSeqTransducer
      ### Pay attention to this part
      segment_composer: !SumMultipleComposer
        composers:
        - !LookupComposer
          word_vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
        - !CharNGramComposer
          ngram_size: 4
          word_vocab: !Vocab {vocab_file: examples/data/head.ngramcount.ja}
      ###
      final_transducer: !BiLSTMSeqTransducer {}
  train: !SimpleTrainingRegimen
    run_for_epochs: 1
    src_file: examples/data/head.ja
    trg_file: examples/data/head.en
  evaluate:
    - !AccuracyEvalTask
      eval_metrics: bleu,wer
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
      hyp_file: test/tmp/{EXP}.test_hyp
      inference: !AutoRegressiveInference {}
```

2.3.22 Switchout

```
# Implements SwitchOut, a data augmentation strategy for NMT
# RAML corrupts target side only, while SwitchOut corrupts both source and target
# https://arxiv.org/pdf/1808.07512.pdf
switchout: !Experiment
  # global parameters shared throughout the experiment
  exp_global: !ExpGlobal
```

(continues on next page)

(continued from previous page)

```
  # {EXP_DIR} is a placeholder for the directory in which the config file lies.
  # {EXP} is a placeholder for the experiment name (here: 'standard')
  model_file: '{EXP_DIR}/models/{EXP}.mod'
  log_file: '{EXP_DIR}/logs/{EXP}.log'
  default_layer_dim: 512
  dropout: 0.3
# model architecture
model: !DefaultTranslator
  src_reader: !RamlTextReader
    vocab: !Vocab {vocab_file: examples/data/head.ja.vocab}
    tau: 0.8
  trg_reader: !RamlTextReader
    vocab: !Vocab {vocab_file: examples/data/head.en.vocab}
    tau: 0.8
  src_embedder: !SimpleWordEmbedder
    emb_dim: 512
  encoder: !BiLSTMSeqTransducer
    layers: 1
  attender: !MlpAttender
    hidden_dim: 512
    state_dim: 512
    input_dim: 512
  decoder: !AutoRegressiveDecoder
    embedder: !SimpleWordEmbedder
      emb_dim: 512
    rnn: !UniLSTMSeqTransducer
      layers: 1
    transform: !AuxNonLinear
      output_dim: 512
      activation: 'tanh'
    bridge: !CopyBridge {}
    scorer: !Softmax {}
# training parameters
train: !SimpleTrainingRegimen
  batcher: !SrcBatcher
    batch_size: 32
  trainer: !AdamTrainer
    alpha: 0.001
  run_for_epochs: 2
  src_file: examples/data/head.ja
  trg_file: examples/data/head.en
  dev_tasks:
    - !LossEvalTask
      src_file: examples/data/head.ja
      ref_file: examples/data/head.en
# final evaluation
evaluate:
  - !AccuracyEvalTask
    eval_metrics: bleu
    src_file: examples/data/head.ja
    ref_file: examples/data/head.en
    hyp_file: examples/output/{EXP}.test_hyp
```

2.3.23 Autobatching

CHAPTER 3

Translator Structure

If you want to dig in to using *xnmt* for your research it is necessary to understand the overall structure. The main class that you need to be aware of is `Translator`, which can calculate the conditional probability of the target sentence given the source sentence. This is useful for calculating losses at training time, or generating sentences at test time. Basically it consists of 4 major components:

1. **Source `Embedder`: This converts input symbols into continuous-space vectors. Usually this** is done by looking up the word in a lookup table, but it could be done any other way.
2. **Encoder `SeqTransducer`: Takes the embedded input and encodes it, for example using a bi-directional** LSTM to calculate context-sensitive embeddings.
3. **`Attender`: This is the "attention" module, which takes the encoded input and decoder** state, then calculates attention.
4. **`Decoder`: This calculates a probability distribution over the words in the output,** either to calculate a loss function during training, or to generate outputs at test time.

In addition, given this `Translator`, we have a `SearchStrategy` that takes the calculated probabilities calculated by the decoder and actually generates outputs at test time.

Note that the target `Embedder`, if necessary, lives inside the `Decoder` object. Some decoders may use non-standard embedding schemes.

There are a bunch of auxiliary classes as well to handle saving/loading of the inputs, etc. However, if you're interested in using *xnmt* to develop a new method, most of your work will probably go into one or a couple of the classes listed above.

CHAPTER 4

Preprocessing

In machine translation, and neural MT in particular, properly pre-processing input before passing it to the learner can greatly increase translation accuracy. This document describes the preprocessing options available within *xnmt*, and documents where external executables can be plugged into the experiment framework.

4.1 Tokenization

A number of tokenization methods are available out of the box; others can be plugged in either with some help (like sentencepiece) or by passing parameters through the experiment framework through to the external decoders.

Multiple tokenizers can be run on the same text; for example, it may be (is there a citation?) that running the Moses tokenizer before performing Byte-pair encoding (BPE) is preferable to either one or the other. It is worth noting, however, that if you want to exactly specify your vocabulary size at tokenization first, an exact-size tokenizer like BPE should be specified (and thus run) *last*.

1. **Sentencepiece: An external tokenizer library that permits a large number of tokenization** options, is written in C++, and is very fast. It is a optional dependency for xnmt (install via `pip install sentencepiece`, see requirements-extra.txt). Specification of the training file is set through the experiment framework, but that (and all other) options can be passed transparently by adding them to the experiment config. See the Sentencepiece section for more specific information on this tokenizer.
2. **External Tokenizers: Any external tokenizer can be used as long as it tokenizes stdin and outputs** to stdout. A single Yaml dictionary labelled `tokenizer_args` is used to pass all (and any) options to the external tokenizer. The option `detokenizer_path`, and its option dictionary, `detokenizer_args`, can optionally be used to specify a detokenizer.
3. **Byte-Pair Encoding: A compression-inspired unsupervised sub-word unit encoding** that performs well (Sennrich, 2016) and permits specification of an exact vocabulary size. Native to *xnmt*; written in Python. Invoked with tokenizer type `bpe`. Right now there is no separate bpe implementation (contributions are welcome), however sentencepiece provides a `bpe` options that performs something similar for a fixed vocabulary size see the following section for more details.

4.1.1 Sentencepiece

The YAML options supported by the SentencepieceTokenizer are almost exactly those presented in the Sentencepiece readme, namely:

- `model_type`: Either `unigram` (default), `bpe`, `char` or `word`. Please refer to the sentencepiece documentation for more details
- `model_prefix`: The trained bpe model will be saved under `{model_prefix}.model/.vocab`
- `vocab_size`: fixes the vocabulary size
- `hard_vocab_limit`: setting this to `False` will make the vocab size a soft limit. Useful for small datasets. This is `True` by default.

Some notable exceptions are below:

- Instead of `extra_options`, since one must be able to pass separate options to the encoder and the decoder, use `encode_extra_options` and `decode_extra_options`, respectively.
- When specifying extra options as above, note that `eos` and `bos` are both off-limits, and will produce odd errors in `vocab.py`. This is because these options add `<s>` and `</s>` to the output, which are already addded by *xnmt*, and are reserved types.
- **Unfortunately, right now, if tokenizers are chained together we see the following behavior:**
  - If the Moses tokenizer is run first, and tokenizes files that are to be used for training BPE in Sentencepiece, Sentencepiece will learn off of the *original* files, not the Moses-tokenized ones.

CHAPTER 5

API Doc

5.1 Experiment

class xnmt.experiments.**ExpGlobal**(*model_file='{EXP_DIR}/models/{EXP}.mod', log_file='{EXP_DIR}/logs/{EXP}.log', dropout=0.3, weight_noise=0.0, default_layer_dim=512, param_init=bare(GlorotInitializer), bias_init=bare(ZeroInitializer), truncate_dec_batches=False, save_num_checkpoints=1, loss_comb_method='sum', commandline_args={}, placeholders={}*)

Bases: *xnmt.persistence.Serializable*

An object that holds global settings that can be referenced by components wherever appropriate.

Parameters

- **model_file** (str) – Location to write model file to
- **log_file** (str) – Location to write log file to
- **dropout** (Real) – Default dropout probability that should be used by supporting components but can be overwritten
- **weight_noise** (Real) – Default weight noise level that should be used by supporting components but can be overwritten
- **default_layer_dim** (Integral) – Default layer dimension that should be used by supporting components but can be overwritten
- **param_init** (*ParamInitializer*) – Default parameter initializer that should be used by supporting components but can be overwritten
- **bias_init** (*ParamInitializer*) – Default initializer for bias parameters that should be used by supporting components but can be overwritten

- **truncate_dec_batches** (bool) – whether the decoder drops batch elements as soon as these are masked at some time step.
- **save_num_checkpoints** (Integral) – save DyNet parameters for the most recent n checkpoints, useful for model averaging/ensembling
- **loss_comb_method** (str) – method for combining loss across batch elements ('sum' or 'avg').
- **commandline_args** (dict) – Holds commandline arguments with which XNMT was launched
- **placeholders** (Dict[str, Any]) – these will be used as arguments for a format() call applied to every string in the config. For example, `placeholders: {"PATH":"/some/path"} will cause each occurence of ``"{PATH}"` in a string to be replaced by `"/some/path"`. As a special variable, `EXP_DIR` can be specified to overwrite the default location for writing models, logs, and other files.

class xnmt.experiments.**Experiment**(*name*, *exp_global=bare(ExpGlobal)*, *preproc=None*, *model=None*, *train=None*, *evaluate=None*, *random_search_report=None*, *status=None*)

Bases: *xnmt.persistence.Serializable*

A default experiment that performs preprocessing, training, and evaluation.

The initializer calls ParamManager.populate(), meaning that model construction should be finalized at this point. __call__() runs the individual steps.

Parameters

- **name** (str) – name of experiment
- **exp_global** (Optional[*ExpGlobal*]) – global experiment settings
- **preproc** (Optional[*PreprocRunner*]) – carry out preprocessing if specified
- **model** (Optional[*TrainableModel*]) – The main model. In the case of multitask training, several models must be specified, in which case the models will live not here but inside the training task objects.
- **train** (Optional[*TrainingRegimen*]) – The training regimen defines the training loop.
- **evaluate** (Optional[List[*EvalTask*]]) – list of tasks to evaluate the model after training finishes.
- **random_search_report** (Optional[dict]) – When random search is used, this holds the settings that were randomly drawn for documentary purposes.
- **status** (Optional[str]) – Status of the experiment, will be automatically set to "done" in saved model if the experiment has finished running.

5.2 Model

5.2.1 Model Base Classes

class xnmt.models.base.**TrainableModel**

Bases: object

A template class for a basic trainable model, implementing a loss function.

calc_nll(**args*, ***kwargs*)

Calculate loss based on input-output pairs.

Losses are accumulated only across unmasked timesteps in each batch element.

Arguments are to be defined by subclasses

Return type `Expression`

Returns A (possibly batched) expression representing the loss.

class xnmt.models.base.**UnconditionedModel**(*trg_reader*)

Bases: *xnmt.models.base.TrainableModel*

A template class for trainable model that computes target losses without conditioning on other inputs.

Parameters trg_reader (*InputReader*) – target reader

calc_nll(*trg*)

Calculate loss based on target inputs.

Losses are accumulated only across unmasked timesteps in each batch element.

Parameters trg (Union[*Batch*, *Sentence*]) – The target, a sentence or a batch of sentences.

Return type `Expression`

Returns A (possibly batched) expression representing the loss.

class xnmt.models.base.**ConditionedModel**(*src_reader*, *trg_reader*)

Bases: *xnmt.models.base.TrainableModel*

A template class for a trainable model that computes target losses conditioned on a source input.

Parameters

- **src_reader** (*InputReader*) – source reader
- **trg_reader** (*InputReader*) – target reader

calc_nll(*src*, *trg*)

Calculate loss based on input-output pairs.

Losses are accumulated only across unmasked timesteps in each batch element.

Parameters

- **src** (Union[*Batch*, *Sentence*]) – The source, a sentence or a batch of sentences.
- **trg** (Union[*Batch*, *Sentence*]) – The target, a sentence or a batch of sentences.

Return type `Expression`

Returns A (possibly batched) expression representing the loss.

class xnmt.models.base.**GeneratorModel**(*src_reader*, *trg_reader=None*)

Bases: `object`

A template class for models that can perform inference to generate some kind of output.

Parameters

- **src_reader** (*InputReader*) – source input reader
- **trg_reader** (Optional[*InputReader*]) – an optional target input reader, needed in some cases such as n-best scoring

generate (*src*, **args*, ***kwargs*)
Generate outputs.

Parameters

- **src** (*Batch*) – batch of source-side inputs
- ***args** –
- ****kwargs** – Further arguments to be specified by subclasses

Return type Sequence[*ReadableSentence*]

Returns output objects

class xnmt.models.base.**CascadeGenerator** (*generators*)
Bases: *xnmt.models.base.GeneratorModel*, *xnmt.persistence.Serializable*

A cascade that chains several generator models.

This generator does not support calling `generate()` directly. Instead, it's sub-generators should be accessed and used to generate outputs one by one.

Parameters **generators** (Sequence[*GeneratorModel*]) – list of generators

generate (**args*, ***kwargs*)
Generate outputs.

Parameters

- **src** – batch of source-side inputs
- ***args** –
- ****kwargs** – Further arguments to be specified by subclasses

Return type Sequence[*ReadableSentence*]

Returns output objects

5.2.2 Translator

5.2.3 Embedder

class xnmt.modelparts.embedders.**Embedder**
Bases: `object`

An embedder takes in word IDs and outputs continuous vectors.

This can be done on a word-by-word basis, or over a sequence.

embed (*word*)
Embed a single word.

Parameters **word** (Any) – This will generally be an integer word ID, but could also be something like a string. It could also be batched, in which case the input will be a `xnmt.batcher.Batch` of integers or other things.

Return type Expression

Returns Expression corresponding to the embedding of the word(s).

embed_sent (*x*)
Embed a full sentence worth of words. By default, just do a for loop.

Parameters **x** (Any) – This will generally be a list of word IDs, but could also be a list of strings or some other format. It could also be batched, in which case it will be a (possibly masked) `xnmt.batcher.Batch` object

Return type `ExpressionSequence`

Returns An expression sequence representing vectors of each word in the input.

choose_vocab (*vocab*, *yaml_path*, *src_reader*, *trg_reader*)

Choose the vocab for the embedder basd on the passed arguments

This is done in order of priority of vocab, model+yaml_path

Parameters

- **vocab** (*Vocab*) – If None, try to obtain from `src_reader` or `trg_reader`, depending on the `yaml_path`
- **yaml_path** (*Path*) – Path of this embedder in the component hierarchy. Automatically determined when deserializing the YAML model.
- **src_reader** (*InputReader*) – Model's src_reader, if exists and unambiguous.
- **trg_reader** (*InputReader*) – Model's trg_reader, if exists and unambiguous.

Return type *Vocab*

Returns chosen vocab

choose_vocab_size (*vocab_size*, *vocab*, *yaml_path*, *src_reader*, *trg_reader*)

Choose the vocab size for the embedder based on the passed arguments

This is done in order of priority of vocab_size, vocab, model+yaml_path

Parameters

- **vocab_size** (Integral) – vocab size or None
- **vocab** (*Vocab*) – vocab or None
- **yaml_path** (*Path*) – Path of this embedder in the component hierarchy. Automatically determined when YAML-deserializing.
- **src_reader** (*InputReader*) – Model's src_reader, if exists and unambiguous.
- **trg_reader** (*InputReader*) – Model's trg_reader, if exists and unambiguous.

Return type `int`

Returns chosen vocab size

class xnmt.modelparts.embedders.**DenseWordEmbedder** (*emb_dim=Ref(path=exp_global.default_layer_dim), weight_noise=Ref(path=exp_global.weight_noise, default=0.0), word_dropout=0.0, fix_norm=None, param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704012410608), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704012452080), vocab_size=None, vocab=None, yaml_path='', src_reader=Ref(path=model.src_reader, default=None), trg_reader=Ref(path=model.trg_reader, default=None)*)

Bases: *xnmt.modelparts.embedders.Embedder*, *xnmt.modelparts.transforms.Linear*, *xnmt.persistence.Serializable*

Word embeddings via full matrix.

Parameters

- **emb_dim** (Integral) – embedding dimension
- **weight_noise** (Real) – apply Gaussian noise with given standard deviation to embeddings
- **word_dropout** (Real) – drop out word types with a certain probability, sampling word types on a per-sentence level, see https://arxiv.org/abs/1512.05287
- **fix_norm** (Optional[Real]) – fix the norm of word vectors to be radius r, see https://arxiv.org/abs/1710.01329
- **param_init** (*ParamInitializer*) – how to initialize weight matrices
- **bias_init** (*ParamInitializer*) – how to initialize bias vectors
- **vocab_size** (Optional[Integral]) – vocab size or None
- **vocab** (Optional[*Vocab*]) – vocab or None
- **yaml_path** (*Path*) – Path of this embedder in the component hierarchy. Automatically set by the YAML deserializer.
- **src_reader** (Optional[*InputReader*]) – A reader for the source side. Automatically set by the YAML deserializer.
- **trg_reader** (Optional[*InputReader*]) – A reader for the target side. Automatically set by the YAML deserializer.

embed(*x*)

Embed a single word.

Parameters word – This will generally be an integer word ID, but could also be something like a string. It could also be batched, in which case the input will be a xnmt.batcher.Batch of integers or other things.

Return type Expression

Returns Expression corresponding to the embedding of the word(s).

class xnmt.modelparts.embedders.**SimpleWordEmbedder**(*emb_dim=Ref(path=exp_global.default_layer_dim), weight_noise=Ref(path=exp_global.weight_noise, default=0.0), word_dropout=0.0, fix_norm=None, param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704012453032), vocab_size=None, vocab=None, yaml_path=, src_reader=Ref(path=model.src_reader, default=None), trg_reader=Ref(path=model.trg_reader, default=None)*)

Bases: *xnmt.modelparts.embedders.Embedder*, *xnmt.persistence.Serializable*

Simple word embeddings via lookup.

Parameters

- **emb_dim** (Integral) – embedding dimension
- **weight_noise** (Real) – apply Gaussian noise with given standard deviation to embeddings
- **word_dropout** (Real) – drop out word types with a certain probability, sampling word types on a per-sentence level, see https://arxiv.org/abs/1512.05287
- **fix_norm** (Optional[Real]) – fix the norm of word vectors to be radius r, see https://arxiv.org/abs/1710.01329
- **param_init** (*ParamInitializer*) – how to initialize lookup matrices
- **vocab_size** (Optional[Integral]) – vocab size or None
- **vocab** (Optional[*Vocab*]) – vocab or None
- **yaml_path** (*Path*) – Path of this embedder in the component hierarchy. Automatically set by the YAML deserializer.
- **src_reader** (Optional[*InputReader*]) – A reader for the source side. Automatically set by the YAML deserializer.
- **trg_reader** (Optional[*InputReader*]) – A reader for the target side. Automatically set by the YAML deserializer.

embed (*x*)

Embed a single word.

Parameters **word** – This will generally be an integer word ID, but could also be something like a string. It could also be batched, in which case the input will be a `xnmt.batcher.Batch` of integers or other things.

Return type `Expression`

Returns Expression corresponding to the embedding of the word(s).

class xnmt.modelparts.embedders.**NoopEmbedder** (*emb_dim*)

Bases: *xnmt.modelparts.embedders.Embedder*, *xnmt.persistence.Serializable*

This embedder performs no lookups but only passes through the inputs.

Normally, the input is a Sentence object, which is converted to an expression.

Parameters **emb_dim** (Optional[Integral]) – Size of the inputs

embed (*x*)

Embed a single word.

Parameters **word** – This will generally be an integer word ID, but could also be something like a string. It could also be batched, in which case the input will be a `xnmt.batcher.Batch` of integers or other things.

Return type `Expression`

Returns Expression corresponding to the embedding of the word(s).

embed_sent (*x*)

Embed a full sentence worth of words. By default, just do a for loop.

Parameters **x** (*Sentence*) – This will generally be a list of word IDs, but could also be a list of strings or some other format. It could also be batched, in which case it will be a (possibly masked) `xnmt.batcher.Batch` object

Return type `ExpressionSequence`

Returns An expression sequence representing vectors of each word in the input.

class xnmt.modelparts.embedders.**PretrainedSimpleWordEmbedder**(*filename, emb_dim=Ref(path=exp_global.default_lay weight_noise=Ref(path=exp_global.weight_ default=0.0), word_dropout=0.0, fix_norm=None, vocab=None, yaml_path=, src_reader=Ref(path=model.src_reader, default=None), trg_reader=Ref(path=model.trg_reader, default=None)*)

Bases: *xnmt.modelparts.embedders.SimpleWordEmbedder*, *xnmt.persistence.Serializable*

Simple word embeddings via lookup. Initial pretrained embeddings must be supplied in FastText text format.

Parameters

- **filename** (str) – Filename for the pretrained embeddings
- **emb_dim** (Integral) – embedding dimension; if None, use exp_global.default_layer_dim
- **weight_noise** (Real) – apply Gaussian noise with given standard deviation to embeddings; if None, use exp_global.weight_noise
- **word_dropout** (Real) – drop out word types with a certain probability, sampling word types on a per-sentence level, see https://arxiv.org/abs/1512.05287
- **fix_norm** (Optional[Real]) – fix the norm of word vectors to be radius r, see https://arxiv.org/abs/1710.01329
- **vocab** (Optional[*Vocab*]) – vocab or None
- **yaml_path** (*Path*) – Path of this embedder in the component hierarchy. Automatically set by the YAML deserializer.
- **src_reader** (Optional[*InputReader*]) – A reader for the source side. Automatically set by the YAML deserializer.
- **trg_reader** (Optional[*InputReader*]) – A reader for the target side. Automatically set by the YAML deserializer.

class xnmt.modelparts.embedders.**PositionEmbedder**(*max_pos, emb_dim=Ref(path=exp_global.default_layer_dim), param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704012454544)*)

Bases: *xnmt.modelparts.embedders.Embedder*, *xnmt.persistence.Serializable*

embed(*word*)

Embed a single word.

Parameters word – This will generally be an integer word ID, but could also be something like a string. It could also be batched, in which case the input will be a xnmt.batcher.Batch of integers or other things.

Returns Expression corresponding to the embedding of the word(s).

embed_sent(*sent_len*)

Embed a full sentence worth of words. By default, just do a for loop.

Parameters **x** – This will generally be a list of word IDs, but could also be a list of strings or some other format. It could also be batched, in which case it will be a (possibly masked) `xnmt.batcher.Batch` object

Return type `ExpressionSequence`

Returns An expression sequence representing vectors of each word in the input.

5.2.4 Transducer

class `xnmt.transducers.base.`**FinalTransducerState**(*main_expr*, *cell_expr=None*)

Bases: `object`

Represents the final encoder state; Currently handles a main (hidden) state and a cell state. If cell state is not provided, it is created as tanh^{-1}(hidden state). Could in the future be extended to handle dimensions other than h and c.

Parameters

- **main_expr** (`Expression`) – expression for hidden state
- **cell_expr** (`Optional[Expression]`) – expression for cell state, if exists

cell_expr()

Returns: dy.Expression: cell state; if not given, it is inferred as inverse tanh of main expression

Return type `Expression`

class `xnmt.transducers.base.`**SeqTransducer**

Bases: `object`

A class that transforms one sequence of vectors into another, using `expression_seqs.ExpressionSequence` objects as inputs and outputs.

transduce(*seq*)

Parameters should be `expression_seqs.ExpressionSequence` objects wherever appropriate

Parameters **seq** (`ExpressionSequence`) – An expression sequence representing the input to the transduction

Return type `ExpressionSequence`

Returns result of transduction, an expression sequence

get_final_states()

Returns: A list of FinalTransducerState objects corresponding to a fixed-dimension representation of the input, after having invoked transduce()

Return type `List[`*`FinalTransducerState`*`]`

class `xnmt.transducers.base.`**ModularSeqTransducer**(*input_dim*, *modules*)

Bases: *`xnmt.transducers.base.SeqTransducer`*, *`xnmt.persistence.Serializable`*

A sequence transducer that stacks several `xnmt.transducer.SeqTransducer` objects, all of which must accept exactly one argument (an `expression_seqs.ExpressionSequence`) in their transduce method.

Parameters

- **input_dim** (`Integral`) – input dimension (not required)
- **modules** (`List[`*`SeqTransducer`*`]`) – list of SeqTransducer modules

shared_params()

Return the shared parameters of this Serializable class.

This can be overwritten to specify what parameters of this component and its subcomponents are shared. Parameter sharing is performed before any components are initialized, and can therefore only include basic data types that are already present in the YAML file (e.g. # dimensions, etc.) Sharing is performed if at least one parameter is specified and multiple shared parameters don't conflict. In case of conflict a warning is printed, and no sharing is performed. The ordering of shared parameters is irrelevant. Note also that if a submodule is replaced by a reference, its shared parameters are ignored.

Returns

objects referencing params of this component or a subcompononent e.g.:

```
return [set([".input_dim",
             ".sub_module.input_dim",
             ".submodules_list.0.input_dim"])]
```

transduce(*seq*)

Parameters should be `expression_seqs.ExpressionSequence` objects wherever appropriate

Parameters **seq** (`ExpressionSequence`) – An expression sequence representing the input to the transduction

Return type `ExpressionSequence`

Returns result of transduction, an expression sequence

get_final_states()

Returns: A list of FinalTransducerState objects corresponding to a fixed-dimension representation of the input, after having invoked transduce()

Return type `List[FinalTransducerState]`

class `xnmt.transducers.base.`**IdentitySeqTransducer**

Bases: *xnmt.transducers.base.SeqTransducer*, *xnmt.persistence.Serializable*

A transducer that simply returns the input.

transduce(*seq*)

Parameters should be `expression_seqs.ExpressionSequence` objects wherever appropriate

Parameters **seq** (`ExpressionSequence`) – An expression sequence representing the input to the transduction

Return type `ExpressionSequence`

Returns result of transduction, an expression sequence

class `xnmt.transducers.base.`**TransformSeqTransducer**(*transform*, *downsample_by=1*)

Bases: *xnmt.transducers.base.SeqTransducer*, *xnmt.persistence.Serializable*

A sequence transducer that applies a given transformation to the sequence's tensor representation

Parameters

- **transform** (*Transform*) – the Transform to apply to the sequence
- **downsample_by** (`Integral`) – if > 1, downsample the sequence via appropriate reshapes. The transform must accept a respectively larger hidden dimension.

get_final_states()

Returns: A list of FinalTransducerState objects corresponding to a fixed-dimension representation of the input, after having invoked transduce()

Return type `List[`*`FinalTransducerState`*`]`

`transduce`(*src*)

Parameters should be `expression_seqs.ExpressionSequence` objects wherever appropriate

Parameters **`seq`** – An expression sequence representing the input to the transduction

Return type `ExpressionSequence`

Returns result of transduction, an expression sequence

5.2.5 RNN

class `xnmt.transducers.recurrent.`**`UniLSTMState`**(*network*, *prev=None*, *c=None*, *h=None*)

Bases: `object`

State object for UniLSTMSeqTransducer.

class `xnmt.transducers.recurrent.`**`UniLSTMSeqTransducer`**(*layers=1*, *input_dim=Ref(path=exp_global.default_layer_dim)*, *hidden_dim=Ref(path=exp_global.default_layer_dim)*, *dropout=Ref(path=exp_global.dropout, default=0.0)*, *weightnoise_std=Ref(path=exp_global.weight_noise, default=0.0)*, *param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704080633984)*, *bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704080634432)*, *yaml_path=*, *decoder_input_dim=Ref(path=exp_global.default_layer_* *default=None)*, *decoder_input_feeding=True*)

Bases: *`xnmt.transducers.base.SeqTransducer`*, *`xnmt.persistence.Serializable`*

This implements a single LSTM layer based on the memory-friendly dedicated DyNet nodes. It works similar to DyNet's CompactVanillaLSTMBuilder, but in addition supports taking multiple inputs that are concatenated on-the-fly.

Parameters

- **`layers`** (*`int`*) – number of layers
- **`input_dim`** (*`int`*) – input dimension
- **`hidden_dim`** (*`int`*) – hidden dimension
- **`dropout`** (*`float`*) – dropout probability
- **`weightnoise_std`** (*`float`*) – weight noise standard deviation
- **`param_init`** (`ParamInitializer`) – how to initialize weight matrices
- **`bias_init`** (`ParamInitializer`) – how to initialize bias vectors
- **`yaml_path`** (*`str`*) –
- **`decoder_input_dim`** (*`int`*) – input dimension of the decoder; if `yaml_path` contains 'decoder' and `decoder_input_feeding` is True, this will be added to `input_dim`

- **decoder_input_feeding** (*bool*) – whether this transducer is part of an input-feeding decoder; cf. `decoder_input_dim`

get_final_states()

Returns: A list of FinalTransducerState objects corresponding to a fixed-dimension representation of the input, after having invoked transduce()

Return type `List[FinalTransducerState]`

transduce(*expr_seq*)

transduce the sequence, applying masks if given (masked timesteps simply copy previous h / c)

Parameters **expr_seq** (`ExpressionSequence`) – expression sequence or list of expression sequences (where each inner list will be concatenated)

Return type `ExpressionSequence`

Returns expression sequence

class xnmt.transducers.recurrent.**BiLSTMSeqTransducer**(*layers=1, input_dim=Ref(path=exp_global.default_layer_dim), hidden_dim=Ref(path=exp_global.default_layer_dim), dropout=Ref(path=exp_global.dropout, default=0.0), weightnoise_std=Ref(path=exp_global.weight_noise, default=0.0), param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704080635440), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704080635888), forward_layers=None, backward_layers=None*)

Bases: `xnmt.transducers.base.SeqTransducer`, `xnmt.persistence.Serializable`

This implements a bidirectional LSTM and requires about 8.5% less memory per timestep than DyNet's CompactVanillaLSTMBuilder due to avoiding concat operations. It uses 2 `xnmt.lstm.UniLSTMSeqTransducer` objects in each layer.

Parameters

- **layers** (*int*) – number of layers
- **input_dim** (*int*) – input dimension
- **hidden_dim** (*int*) – hidden dimension
- **dropout** (*float*) – dropout probability
- **weightnoise_std** (*float*) – weight noise standard deviation
- **param_init** (*ParamInitializer*) – a `xnmt.param_init.ParamInitializer` or list of `xnmt.param_init.ParamInitializer` objects specifying how to initialize weight matrices. If a list is given, each entry denotes one layer.
- **bias_init** (*ParamInitializer*) – a `xnmt.param_init.ParamInitializer` or list of `xnmt.param_init.ParamInitializer` objects specifying how to initialize bias vectors. If a list is given, each entry denotes one layer.
- **forward_layers** (`Optional[Sequence[UniLSTMSeqTransducer]]`) – set automatically

- **backward_layers** (Optional[Sequence[*UniLSTMSeqTransducer*]]) – set automatically

get_final_states()

Returns: A list of FinalTransducerState objects corresponding to a fixed-dimension representation of the input, after having invoked transduce()

Return type List[*FinalTransducerState*]

transduce(*es*)

Parameters should be `expression_seqs.ExpressionSequence` objects wherever appropriate

Parameters **seq** – An expression sequence representing the input to the transduction

Return type ExpressionSequence

Returns result of transduction, an expression sequence

class xnmt.transducers.recurrent.**CustomLSTMSeqTransducer**(*layers*, *input_dim*, *hidden_dim*, *param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704080636336)*, *bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704080636672)*)

Bases: *xnmt.transducers.base.SeqTransducer*, *xnmt.persistence.Serializable*

This implements an LSTM builder based on elementary DyNet operations. It is more memory-hungry than the compact LSTM, but can be extended more easily. It currently does not support dropout or multiple layers and is mostly meant as a starting point for LSTM extensions.

Parameters

- **layers** (*int*) – number of layers
- **input_dim** (*int*) – input dimension; if None, use exp_global.default_layer_dim
- **hidden_dim** (*int*) – hidden dimension; if None, use exp_global.default_layer_dim
- **param_init** (*ParamInitializer*) – a `xnmt.param_init.ParamInitializer` or list of `xnmt.param_init.ParamInitializer` objects specifying how to initialize weight matrices. If a list is given, each entry denotes one layer. If None, use `exp_global.param_init`
- **bias_init** (*ParamInitializer*) – a `xnmt.param_init.ParamInitializer` or list of `xnmt.param_init.ParamInitializer` objects specifying how to initialize bias vectors. If a list is given, each entry denotes one layer. If None, use `exp_global.param_init`

transduce(*xs*)

Parameters should be `expression_seqs.ExpressionSequence` objects wherever appropriate

Parameters **seq** – An expression sequence representing the input to the transduction

Return type ExpressionSequence

Returns result of transduction, an expression sequence

class xnmt.transducers.pyramidal.**PyramidalLSTMSeqTransducer**(*layers=1, input_dim=Ref(path=exp_global.default_layer_dim), hidden_dim=Ref(path=exp_global.default_layer_dim), downsampling_method='concat', reduce_factor=2, dropout=Ref(path=exp_global.dropout, default=0.0), builder_layers=None*)

Bases: *xnmt.transducers.base.SeqTransducer*, *xnmt.persistence.Serializable*

Builder for pyramidal RNNs that delegates to `UniLSTMSeqTransducer` objects and wires them together. See https://arxiv.org/abs/1508.01211

Every layer (except the first) reduces sequence length by the specified factor.

Parameters

- **layers** (`Integral`) – number of layers
- **input_dim** (`Integral`) – input dimension
- **hidden_dim** (`Integral`) – hidden dimension
- **downsampling_method** (`str`) – how to perform downsampling (concat|skip)
- **reduce_factor** (`Union[Integral, Sequence[Integral]]`) – integer, or list of ints (different skip for each layer)
- **dropout** (`float`) – dropout probability; if None, use exp_global.dropout
- **builder_layers** (`Optional[Any]`) – set automatically

get_final_states()

Returns: A list of FinalTransducerState objects corresponding to a fixed-dimension representation of the input, after having invoked transduce()

Return type `List[`*FinalTransducerState*`]`

transduce(*es*)

returns the list of output Expressions obtained by adding the given inputs to the current state, one by one, to both the forward and backward RNNs, and concatenating.

Parameters **es** (`ExpressionSequence`) – an ExpressionSequence

Return type `ExpressionSequence`

class xnmt.transducers.residual.**ResidualSeqTransducer**(*child, input_dim, layer_norm=False, dropout=Ref(path=exp_global.dropout, default=0.0)*)

Bases: *xnmt.transducers.base.SeqTransducer*, *xnmt.persistence.Serializable*

A sequence transducer that wraps a *xnmt.transducers.base.SeqTransducer* in an additive residual connection, and optionally performs some variety of normalization.

Parameters

- **the child transducer to wrap** (*child*) –
- **layer_norm** (`bool`) – whether to perform layer normalization
- **dropout** – whether to apply residual dropout

transduce (*seq*)

Parameters should be `expression_seqs.ExpressionSequence` objects wherever appropriate

Parameters **seq** (`ExpressionSequence`) – An expression sequence representing the input to the transduction

Return type `ExpressionSequence`

Returns result of transduction, an expression sequence

get_final_states ()

Returns: A list of FinalTransducerState objects corresponding to a fixed-dimension representation of the input, after having invoked transduce()

Return type `List[FinalTransducerState]`

5.2.6 Attender

class `xnmt.modelparts.attenders.`**Attender**

Bases: `object`

A template class for functions implementing attention.

init_sent (*sent*)

Args: sent: the encoder states, aka keys and values. Usually but not necessarily an `expression_seqs.ExpressionSequence`

Return type `None`

calc_attention (*state*)

Compute attention weights.

Parameters **state** (`Expression`) – the current decoder state, aka query, for which to compute the weights.

Return type `Expression`

Returns DyNet expression containing normalized attention scores

calc_context (*state*, *attention=None*)

Compute weighted sum.

Parameters

- **state** (`Expression`) – the current decoder state, aka query, for which to compute the weighted sum.
- **attention** (`Optional[Expression]`) – the attention vector to use. if not given it is calculated from the state.

Return type `Expression`

class `xnmt.modelparts.attenders.`**MlpAttender** (*input_dim=Ref(path=exp_global.default_layer_dim), state_dim=Ref(path=exp_global.default_layer_dim), hidden_dim=Ref(path=exp_global.default_layer_dim), param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139703657786896), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139703657787288), truncate_dec_batches=Ref(path=exp_global.truncate_dec_batches, default=False)*)

Bases: `xnmt.modelparts.attenders.Attender`, `xnmt.persistence.Serializable`

Implements the attention model of Bahdanau et. al (2014)

Parameters

- **input_dim** (`Integral`) – input dimension
- **state_dim** (`Integral`) – dimension of state inputs
- **hidden_dim** (`Integral`) – hidden MLP dimension
- **param_init** (*ParamInitializer*) – how to initialize weight matrices
- **bias_init** (*ParamInitializer*) – how to initialize bias vectors
- **truncate_dec_batches** (`bool`) – whether the decoder drops batch elements as soon as these are masked at some time step.

init_sent (*sent*)

Args: sent: the encoder states, aka keys and values. Usually but not necessarily an `expression_seqs.ExpressionSequence`

Return type `None`

calc_attention (*state*)

Compute attention weights.

Parameters **state** (`Expression`) – the current decoder state, aka query, for which to compute the weights.

Return type `Expression`

Returns DyNet expression containing normalized attention scores

class xnmt.modelparts.attenders.**DotAttender** (*scale=True, truncate_dec_batches=Ref(path=exp_global.truncate_dec_batches, default=False)*)

Bases: *xnmt.modelparts.attenders.Attender*, *xnmt.persistence.Serializable*

Implements dot product attention of https://arxiv.org/abs/1508.04025 Also (optionally) perform scaling of https://arxiv.org/abs/1706.03762

Parameters

- **scale** (`bool`) – whether to perform scaling
- **truncate_dec_batches** (`bool`) – currently unsupported

init_sent (*sent*)

Args: sent: the encoder states, aka keys and values. Usually but not necessarily an `expression_seqs.ExpressionSequence`

Return type `None`

calc_attention (*state*)

Compute attention weights.

Parameters **state** (`Expression`) – the current decoder state, aka query, for which to compute the weights.

Return type `Expression`

Returns DyNet expression containing normalized attention scores

class xnmt.modelparts.attenders.**BilinearAttender**(*input_dim=Ref(path=exp_global.default_layer_dim), state_dim=Ref(path=exp_global.default_layer_dim), param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139703655097120), truncate_dec_batches=Ref(path=exp_global.truncate_dec_ba default=False)*)

Bases: *xnmt.modelparts.attenders.Attender*, *xnmt.persistence.Serializable*

Implements a bilinear attention, equivalent to the 'general' linear attention of https://arxiv.org/abs/1508.04025

Parameters

- **input_dim** (Integral) – input dimension; if None, use exp_global.default_layer_dim
- **state_dim** (Integral) – dimension of state inputs; if None, use exp_global.default_layer_dim
- **param_init** (*ParamInitializer*) – how to initialize weight matrices; if None, use exp_global.param_init
- **truncate_dec_batches** (bool) – currently unsupported

init_sent(*sent*)

Args: sent: the encoder states, aka keys and values. Usually but not necessarily an expression_seqs.ExpressionSequence

Return type None

calc_attention(*state*)

Compute attention weights.

Parameters state (Expression) – the current decoder state, aka query, for which to compute the weights.

Return type Expression

Returns DyNet expression containing normalized attention scores

class xnmt.modelparts.attenders.**LatticeBiasedMlpAttender**(*input_dim=Ref(path=exp_global.default_layer_di state_dim=Ref(path=exp_global.default_layer_di hidden_dim=Ref(path=exp_global.default_layer_dim param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139703655097904), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139703655098352), truncate_dec_batches=Ref(path=exp_global.truncate_ default=False)*)

Bases: *xnmt.modelparts.attenders.MlpAttender*, *xnmt.persistence.Serializable*

Modified MLP attention, where lattices are assumed as input and the attention is biased toward confident nodes.

Parameters

- **input_dim** (Integral) – input dimension
- **state_dim** (Integral) – dimension of state inputs
- **hidden_dim** (Integral) – hidden MLP dimension
- **param_init** (*ParamInitializer*) – how to initialize weight matrices

- **bias_init** (*ParamInitializer*) – how to initialize bias vectors
- **truncate_dec_batches** (bool) – whether the decoder drops batch elements as soon as these are masked at some time step.

calc_attention (*state*)
Compute attention weights.

Parameters state (Expression) – the current decoder state, aka query, for which to compute the weights.

Return type Expression

Returns DyNet expression containing normalized attention scores

5.2.7 Decoder

class xnmt.modelparts.decoders.**Decoder**
Bases: object

A template class to convert a prefix of previously generated words and a context vector into a probability distribution over possible next words.

class xnmt.modelparts.decoders.**DecoderState**
Bases: object

A state that holds whatever information is required for the decoder. Child classes must implement the as_vector() method, which will be used by e.g. the attention mechanism

class xnmt.modelparts.decoders.**AutoRegressiveDecoderState** (*rnn_state=None, context=None*)
Bases: *xnmt.modelparts.decoders.DecoderState*

A state holding all the information needed for AutoRegressiveDecoder

Parameters

- **rnn_state** – a DyNet RNN state
- **context** – a DyNet expression

class xnmt.modelparts.decoders.**AutoRegressiveDecoder** (*input_dim=Ref(path=exp_global.default_layer_dim), embedder=bare(SimpleWordEmbedder), input_feeding=True, bridge=bare(CopyBridge), rnn=bare(UniLSTMSeqTransducer), transform=bare(AuxNonLinear), scorer=bare(Softmax), truncate_dec_batches=Ref(path=exp_global.truncate_dec_b default=False)*)
Bases: *xnmt.modelparts.decoders.Decoder*, *xnmt.persistence.Serializable*

Standard autoregressive-decoder.

Parameters

- **input_dim** (Integral) – input dimension
- **embedder** (*Embedder*) – embedder for target words
- **input_feeding** (bool) – whether to activate input feeding

- **bridge** (*Bridge*) – how to initialize decoder state
- **rnn** (*UniLSTMSeqTransducer*) – recurrent decoder
- **transform** (*Transform*) – a layer of transformation between rnn and output scorer
- **scorer** (*Scorer*) – the method of scoring the output (usually softmax)
- **truncate_dec_batches** (bool) – whether the decoder drops batch elements as soon as these are masked at some time step.

shared_params()

Return the shared parameters of this Serializable class.

This can be overwritten to specify what parameters of this component and its subcomponents are shared. Parameter sharing is performed before any components are initialized, and can therefore only include basic data types that are already present in the YAML file (e.g. # dimensions, etc.) Sharing is performed if at least one parameter is specified and multiple shared parameters don't conflict. In case of conflict a warning is printed, and no sharing is performed. The ordering of shared parameters is irrelevant. Note also that if a submodule is replaced by a reference, its shared parameters are ignored.

Returns

objects referencing params of this component or a subcompononent e.g.:

```
return [set([".input_dim",
             ".sub_module.input_dim",
             ".submodules_list.0.input_dim"])]
```

initial_state(*enc_final_states*, *ss*)

Get the initial state of the decoder given the encoder final states.

Parameters

- **enc_final_states** (Any) – The encoder final states. Usually but not necessarily an xnmt.expression_sequence.ExpressionSequence
- **ss** (Any) – first input

Return type *AutoRegressiveDecoderState*

Returns initial decoder state

add_input(*dec_state*, *trg_word*)

Add an input and return a *new* update the state.

Parameters

- **dec_state** (*AutoRegressiveDecoderState*) – An object containing the current state.
- **trg_word** (Any) – The word to input.

Return type *AutoRegressiveDecoderState*

Returns The updated decoder state.

5.2.8 Bridge

class xnmt.modelparts.bridges.**Bridge**

Bases: object

Responsible for initializing the decoder LSTM, based on the final encoder state

decoder_init (*enc_final_states*)

Parameters **enc_final_states** (Sequence[*FinalTransducerState*]) – list of final states for each encoder layer

Return type List[Expression]

Returns list of initial hidden and cell expressions for each layer. List indices 0..n-1 hold hidden states, n..2n-1 hold cell states.

class xnmt.modelparts.bridges.**NoBridge** (*dec_layers=1*, *dec_dim=Ref(path=exp_global.default_layer_dim)*)

Bases: *xnmt.modelparts.bridges.Bridge*, *xnmt.persistence.Serializable*

This bridge initializes the decoder with zero vectors, disregarding the encoder final states.

Parameters

- **dec_layers** (Integral) – number of decoder layers to initialize
- **dec_dim** (Integral) – hidden dimension of decoder states

decoder_init (*enc_final_states*)

Parameters **enc_final_states** (Sequence[*FinalTransducerState*]) – list of final states for each encoder layer

Return type List[Expression]

Returns list of initial hidden and cell expressions for each layer. List indices 0..n-1 hold hidden states, n..2n-1 hold cell states.

class xnmt.modelparts.bridges.**CopyBridge** (*dec_layers=1*, *dec_dim=Ref(path=exp_global.default_layer_dim)*)

Bases: *xnmt.modelparts.bridges.Bridge*, *xnmt.persistence.Serializable*

This bridge copies final states from the encoder to the decoder initial states. Requires that: - encoder / decoder dimensions match for every layer - num encoder layers >= num decoder layers (if unequal, we disregard final states at the encoder bottom)

Parameters

- **dec_layers** (Integral) – number of decoder layers to initialize
- **dec_dim** (Integral) – hidden dimension of decoder states

decoder_init (*enc_final_states*)

Parameters **enc_final_states** (Sequence[*FinalTransducerState*]) – list of final states for each encoder layer

Return type List[Expression]

Returns list of initial hidden and cell expressions for each layer. List indices 0..n-1 hold hidden states, n..2n-1 hold cell states.

class xnmt.modelparts.bridges.**LinearBridge** (*dec_layers=1*, *enc_dim=Ref(path=exp_global.default_layer_dim)*, *dec_dim=Ref(path=exp_global.default_layer_dim)*, *param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704012823912)*, *bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704012824248)*, *projector=None*)

Bases: *xnmt.modelparts.bridges.Bridge*, *xnmt.persistence.Serializable*

This bridge does a linear transform of final states from the encoder to the decoder initial states. Requires that num encoder layers >= num decoder layers (if unequal, we disregard final states at the encoder bottom)

Parameters

- **dec_layers** (Integral) – number of decoder layers to initialize
- **enc_dim** (Integral) – hidden dimension of encoder states
- **dec_dim** (Integral) – hidden dimension of decoder states
- **param_init** (*ParamInitializer*) – how to initialize weight matrices; if None, use `exp_global.param_init`
- **bias_init** (*ParamInitializer*) – how to initialize bias vectors; if None, use `exp_global.bias_init`
- **projector** (Optional[*Linear*]) – linear projection (created automatically)

decoder_init (*enc_final_states*)

Parameters **enc_final_states** (Sequence[*FinalTransducerState*]) – list of final states for each encoder layer

Return type List[Expression]

Returns list of initial hidden and cell expressions for each layer. List indices 0..n-1 hold hidden states, n..2n-1 hold cell states.

5.2.9 Transform

class xnmt.modelparts.transforms.**Transform**

Bases: `object`

A class of transforms that change a dynet expression into another.

class xnmt.modelparts.transforms.**Identity**

Bases: *xnmt.modelparts.transforms.Transform*, *xnmt.persistence.Serializable*

Identity transform. For use when you think it might be a better idea to not perform a specific transform in a place where you would normally do one.

class xnmt.modelparts.transforms.**Linear** (*input_dim=Ref(path=exp_global.default_layer_dim), output_dim=Ref(path=exp_global.default_layer_dim), bias=True, param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704081089368), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704081088976)*)

Bases: *xnmt.modelparts.transforms.Transform*, *xnmt.persistence.Serializable*

Linear projection with optional bias.

Parameters

- **input_dim** (Integral) – input dimension
- **output_dim** (Integral) – hidden dimension
- **bias** (bool) – whether to add a bias
- **param_init** (*ParamInitializer*) – how to initialize weight matrices
- **bias_init** (*ParamInitializer*) – how to initialize bias vectors

class xnmt.modelparts.transforms.**NonLinear**(*input_dim=Ref(path=exp_global.default_layer_dim), output_dim=Ref(path=exp_global.default_layer_dim), bias=True, activation='tanh', param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704081089088), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704081090264)*)

Bases: *xnmt.modelparts.transforms.Transform*, *xnmt.persistence.Serializable*

Linear projection with optional bias and non-linearity.

Parameters

- **input_dim** (Integral) – input dimension
- **output_dim** (Integral) – hidden dimension
- **bias** (bool) – whether to add a bias
- **activation** (str) – One of tanh, relu, sigmoid, elu, selu, asinh or identity.
- **param_init** (*ParamInitializer*) – how to initialize weight matrices
- **bias_init** (*ParamInitializer*) – how to initialize bias vectors

class xnmt.modelparts.transforms.**AuxNonLinear**(*input_dim=Ref(path=exp_global.default_layer_dim), output_dim=Ref(path=exp_global.default_layer_dim), aux_input_dim=Ref(path=exp_global.default_layer_dim), bias=True, activation='tanh', param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704081090824), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704081091272)*)

Bases: *xnmt.modelparts.transforms.NonLinear*, *xnmt.persistence.Serializable*

NonLinear with an additional auxiliary input.

Parameters

- **input_dim** (Integral) – input dimension
- **output_dim** (Integral) – hidden dimension
- **aux_input_dim** (Integral) – auxiliary input dimension. The actual input dimension is aux_input_dim + input_dim. This is useful for when you want to do something like input feeding.
- **bias** (bool) – whether to add a bias
- **activation** (str) – One of tanh, relu, sigmoid, elu, selu, asinh or identity.
- **param_init** (*ParamInitializer*) – how to initialize weight matrices
- **bias_init** (*ParamInitializer*) – how to initialize bias vectors

class xnmt.modelparts.transforms.**MLP**(*input_dim=Ref(path=exp_global.default_layer_dim), hidden_dim=Ref(path=exp_global.default_layer_dim), output_dim=Ref(path=exp_global.default_layer_dim), bias=True, activation='tanh', hidden_layers=1, param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704081091832), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704081092168), layers=None*)

Bases: *xnmt.modelparts.transforms.Transform*, *xnmt.persistence.Serializable*

A multi-layer perceptron. Defined as one or more NonLinear transforms of equal hidden dimension and type, then a Linear transform to the output dimension.

class xnmt.modelparts.transforms.**Cwise**(*op='rectify'*)

Bases: *xnmt.modelparts.transforms.Transform*, *xnmt.persistence.Serializable*

A component-wise transformation that can be an arbitrary unary DyNet operation.

Parameters **op** (`str`) – arbitrary unary DyNet node

5.2.10 Scorer

class xnmt.modelparts.scorers.**Scorer**

Bases: `object`

A template class of things that take in a vector and produce a score over discrete output items.

calc_scores(*x*)

Calculate the score of each discrete decision, where the higher the score is the better the model thinks a decision is. These often correspond to unnormalized log probabilities.

Parameters **x** (`Expression`) – The vector used to make the prediction

Return type `Expression`

best_k(*x, k, normalize_scores=False*)

Returns a list of the k items with the highest scores. The items may not be in sorted order.

Parameters

- **x** (`Expression`) – The vector used to make the prediction
- **k** (`Integral`) – Number of items to return
- **normalize_scores** (`bool`) – whether to normalize the scores

sample(*x, n*)

Return samples from the scores that are treated as probability distributions.

calc_probs(*x*)

Calculate the normalized probability of a decision.

Parameters **x** (`Expression`) – The vector used to make the prediction

Return type `Expression`

calc_log_probs(*x*)

Calculate the log probability of a decision

log(calc_prob()) == calc_log_prob()

Both functions exist because it might help save memory.

Parameters **x** (Expression) – The vector used to make the prediction

Return type Expression

calc_loss (*x*, *y*)

Calculate the loss incurred by making a particular decision.

Parameters

- **x** (Expression) – The vector used to make the prediction
- **y** (Union[int, List[int]]) – The correct label(s)

Return type Expression

class xnmt.modelparts.scorers.**Softmax** (*input_dim=Ref(path=exp_global.default_layer_dim), vocab_size=None, vocab=None, trg_reader=Ref(path=model.trg_reader, default=None), label_smoothing=0.0, param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704012876152), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704012876600), output_projector=None*)

Bases: *xnmt.modelparts.scorers.Scorer*, *xnmt.persistence.Serializable*

A class that does an affine transform from the input to the vocabulary size, and calculates a softmax.

Note that all functions in this class rely on calc_scores(), and thus this class can be sub-classed by any other class that has an alternative method for calculating un-normalized log probabilities by simply overloading the calc_scores() function.

Parameters

- **input_dim** (Integral) – Size of the input vector
- **vocab_size** (Optional[Integral]) – Size of the vocab to predict
- **vocab** (Optional[*Vocab*]) – A vocab object from which the vocab size can be derived automatically
- **trg_reader** (Optional[*InputReader*]) – An input reader for the target, which can be used to derive the vocab size
- **label_smoothing** (Real) – Whether to apply label smoothing (a value of 0.1 is good if so)
- **param_init** (*ParamInitializer*) – How to initialize the parameters
- **bias_init** (*ParamInitializer*) – How to initialize the bias
- **output_projector** (Optional[*Linear*]) – The projection to be used before the output

calc_scores (*x*)

Calculate the score of each discrete decision, where the higher the score is the better the model thinks a decision is. These often correspond to unnormalized log probabilities.

Parameters **x** (Expression) – The vector used to make the prediction

Return type Expression

best_k (*x*, *k*, *normalize_scores=False*)

Returns a list of the k items with the highest scores. The items may not be in sorted order.

Parameters

- **x** (Expression) – The vector used to make the prediction
- **k** (Integral) – Number of items to return
- **normalize_scores** (bool) – whether to normalize the scores

sample (*x*, *n*, *temperature=1.0*)
Return samples from the scores that are treated as probability distributions.

can_loss_be_derived_from_scores ()
This method can be used to determine whether dy.pickneglogsoftmax can be used to quickly calculate the loss value. If False, then the calc_loss method should (1) calc log_softmax, (2) perform necessary modification, (3) pick the loss

calc_loss (*x*, *y*)
Calculate the loss incurred by making a particular decision.

Parameters

- **x** (Expression) – The vector used to make the prediction
- **y** (Union[Integral, List[Integral]]) – The correct label(s)

Return type Expression

calc_probs (*x*)
Calculate the normalized probability of a decision.

Parameters **x** (Expression) – The vector used to make the prediction

Return type Expression

calc_log_probs (*x*)
Calculate the log probability of a decision

log(calc_prob()) == calc_log_prob()

Both functions exist because it might help save memory.

Parameters **x** (Expression) – The vector used to make the prediction

Return type Expression

class xnmt.modelparts.scorers.**LexiconSoftmax** (*input_dim=Ref(path=exp_global.default_layer_dim), vocab_size=None, vocab=None, trg_reader=Ref(path=model.trg_reader, default=None), attender=Ref(path=model.attender), label_smoothing=0.0, param_init=Ref(path=exp_global.param_init, default=GlorotInitializer@139704012877160), bias_init=Ref(path=exp_global.bias_init, default=ZeroInitializer@139704012877496), output_projector=None, lexicon_file=None, lexicon_alpha=0.001, lexicon_type='bias', coef_predictor=None, src_vocab=Ref(path=model.src_reader.vocab, default=None)*)

Bases: *xnmt.modelparts.scorers.Softmax*, *xnmt.persistence.Serializable*

A subclass of the softmax class that can make use of an external lexicon probability as described in: http://anthology.aclweb.org/D/D16/D16-1162.pdf

Parameters

- **input_dim** (Integral) – Size of the input vector
- **vocab_size** (Optional[Integral]) – Size of the vocab to predict
- **vocab** (Optional[*Vocab*]) – A vocab object from which the vocab size can be derived automatically
- **trg_reader** (Optional[*InputReader*]) – An input reader for the target, which can be used to derive the vocab size
- **label_smoothing** (Real) – Whether to apply label smoothing (a value of 0.1 is good if so)
- **param_init** (*ParamInitializer*) – How to initialize the parameters
- **bias_init** (*ParamInitializer*) – How to initialize the bias
- **output_projector** (Optional[*Linear*]) – The projection to be used before the output
- **lexicon_file** – A file containing "trg src p(trg|src)"
- **lexicon_alpha** – smoothing constant for bias method
- **lexicon_type** – Either bias or linear method

calc_scores (*x*)

Calculate the score of each discrete decision, where the higher the score is the better the model thinks a decision is. These often correspond to unnormalized log probabilities.

Parameters **x** (Expression) – The vector used to make the prediction

Return type Expression

calc_probs (*x*)

Calculate the normalized probability of a decision.

Parameters **x** (Expression) – The vector used to make the prediction

Return type Expression

calc_log_probs (*x*)

Calculate the log probability of a decision

log(calc_prob()) == calc_log_prob()

Both functions exist because it might help save memory.

Parameters **x** (Expression) – The vector used to make the prediction

Return type Expression

can_loss_be_derived_from_scores ()

This method can be used to determine whether dy.pickneglogsoftmax can be used to quickly calculate the loss value. If False, then the calc_loss method should (1) calc log_softmax, (2) perform necessary modification, (3) pick the loss

5.2.11 SequenceLabeler

class xnmt.models.sequence_labelers.**SeqLabeler**(*src_reader*, *trg_reader*, *src_embedder=bare(SimpleWordEmbedder)*, *encoder=bare(BiLSTMSeqTransducer)*, *transform=bare(NonLinear)*, *scorer=bare(Softmax)*, *inference=bare(IndependentOutputInference)*, *auto_cut_pad=False*)

Bases: *xnmt.models.base.ConditionedModel*, *xnmt.models.base.GeneratorModel*, *xnmt.persistence.Serializable*, *xnmt.reports.Reportable*

A simple sequence labeler based on an encoder and an output softmax layer.

Parameters

- **src_reader** (*InputReader*) – A reader for the source side.
- **trg_reader** (*InputReader*) – A reader for the target side.
- **src_embedder** (*Embedder*) – A word embedder for the input language
- **encoder** (*SeqTransducer*) – An encoder to generate encoded inputs
- **transform** (*Transform*) – A transform to be applied before making predictions
- **scorer** (*Scorer*) – The class to actually make predictions
- **inference** (*Inference*) – The inference method used for this model
- **auto_cut_pad** (bool) – If True, cut or pad target sequences so the match the length of the encoded inputs. If False, an error is thrown if there is a length mismatch.

shared_params()

Return the shared parameters of this Serializable class.

This can be overwritten to specify what parameters of this component and its subcomponents are shared. Parameter sharing is performed before any components are initialized, and can therefore only include basic data types that are already present in the YAML file (e.g. # dimensions, etc.) Sharing is performed if at least one parameter is specified and multiple shared parameters don't conflict. In case of conflict a warning is printed, and no sharing is performed. The ordering of shared parameters is irrelevant. Note also that if a submodule is replaced by a reference, its shared parameters are ignored.

Return type Sequence[Set[str]]

Returns

objects referencing params of this component or a subcomponent e.g.:

```
return [set([".input_dim",
             ".sub_module.input_dim",
             ".submodules_list.0.input_dim"])]
```

calc_nll(*src*, *trg*)

Calculate loss based on input-output pairs.

Losses are accumulated only across unmasked timesteps in each batch element.

Parameters

- **src** (Union[*Batch*, *Sentence*]) – The source, a sentence or a batch of sentences.
- **trg** (Union[*Batch*, *Sentence*]) – The target, a sentence or a batch of sentences.

Return type Expression

Returns A (possibly batched) expression representing the loss.

generate (*src*, *normalize_scores=False*)

Generate outputs.

Parameters

- **src** (*Batch*) – batch of source-side inputs
- ***args** –
- ****kwargs** – Further arguments to be specified by subclasses

Return type Sequence[*ReadableSentence*]

Returns output objects

set_trg_vocab (*trg_vocab=None*)

Set target vocab for generating outputs. If not specified, word IDs are generated instead.

Parameters **trg_vocab** (Optional[*Vocab*]) – target vocab, or None to generate word IDs

Return type None

5.2.12 Classifier

class xnmt.models.classifiers.**SequenceClassifier** (*src_reader*, *trg_reader*, *src_embedder=bare(SimpleWordEmbedder)*, *encoder=bare(BiLSTMSeqTransducer)*, *inference=bare(IndependentOutputInference)*, *transform=bare(NonLinear)*, *scorer=bare(Softmax)*)

Bases: *xnmt.models.base.ConditionedModel*, *xnmt.models.base.GeneratorModel*, *xnmt.persistence.Serializable*

A sequence classifier.

Runs embeddings through an encoder, feeds the average over all encoder outputs to a transform and scoring layer.

Parameters

- **src_reader** (*InputReader*) – A reader for the source side.
- **trg_reader** (*InputReader*) – A reader for the target side.
- **src_embedder** (*Embedder*) – A word embedder for the input language
- **encoder** (*SeqTransducer*) – An encoder to generate encoded inputs
- **inference** – how to perform inference
- **transform** (*Transform*) – A transform performed before the scoring function
- **scorer** (*Scorer*) – A scoring function over the multiple choices

shared_params ()

Return the shared parameters of this Serializable class.

This can be overwritten to specify what parameters of this component and its subcomponents are shared. Parameter sharing is performed before any components are initialized, and can therefore only include basic data types that are already present in the YAML file (e.g. # dimensions, etc.) Sharing is performed if at least one parameter is specified and multiple shared parameters don't conflict. In case of conflict a warning is printed, and no sharing is performed. The ordering of shared parameters is irrelevant. Note also that if a submodule is replaced by a reference, its shared parameters are ignored.

Returns

objects referencing params of this component or a subcompononent e.g.:

```
return [set([".input_dim",
             ".sub_module.input_dim",
             ".submodules_list.0.input_dim"])]
```

calc_nll (*src*, *trg*)

Calculate loss based on input-output pairs.

Losses are accumulated only across unmasked timesteps in each batch element.

Parameters

- **src** (Union[*Batch*, *Sentence*]) – The source, a sentence or a batch of sentences.
- **trg** (Union[*Batch*, *Sentence*]) – The target, a sentence or a batch of sentences.

Return type Expression

Returns A (possibly batched) expression representing the loss.

generate (*src*, *normalize_scores=False*)

Generate outputs.

Parameters

- **src** (Union[*Batch*, *Sentence*]) – batch of source-side inputs
- ***args** –
- ****kwargs** – Further arguments to be specified by subclasses

Returns output objects

5.3 Loss

5.3.1 Loss

class xnmt.losses.**FactoredLossExpr** (*init_loss=None*)

Bases: object

Loss consisting of (possibly batched) DyNet expressions, with one expression per loss factor.

Used to represent losses within a training step.

Parameters **init_loss** (Optional[Dict[str, Expression]]) – initial loss values

compute (*comb_method='sum'*)

Compute loss as DyNet expression by summing over factors and batch elements.

Parameters **comb_method** (str) – method for combining loss across batch elements ('sum' or 'avg').

Return type Expression

Returns Scalar DyNet expression.

value()
Get list of per-batch-element loss values, summed over factors.

Return type List[float]

Returns List of same length as batch-size.

get_factored_loss_val(*comb_method='sum'*)
Create factored loss values by calling .value() for each DyNet loss expression and applying batch combination.

Parameters **comb_method** (str) – method for combining loss across batch elements ('sum' or 'avg').

Return type *FactoredLossVal*

Returns Factored loss values.

get_nobackprop_loss()
Get dictionary of named non-backpropagating loss expressions

Return type Dict[str, Expression]

Returns Loss expressions

class xnmt.losses.**FactoredLossVal**(*loss_dict=None*)
Bases: object

Loss consisting of (unbatched) float values, with one value per loss factor.

Used to represent losses accumulated across several training steps.

sum_factors()
Return the sum of all loss factors.

Return type float

Returns A float value.

items()
Get name/value tuples for loss factors.

Return type List[Tuple[str, float]]

Returns Name/value tuples.

clear()
Clears all loss factors.

Return type None

5.3.2 LossCalculator

class xnmt.loss_calculators.**LossCalculator**
Bases: object

A template class implementing the training strategy and corresponding loss calculation.

class xnmt.loss_calculators.**MLELoss**
Bases: *xnmt.persistence.Serializable*, *xnmt.loss_calculators.LossCalculator*

Max likelihood loss calculator.

class xnmt.loss_calculators.**GlobalFertilityLoss**
Bases: *xnmt.persistence.Serializable*, *xnmt.loss_calculators.LossCalculator*

A fertility loss according to Cohn+, 2016. Incorporating Structural Alignment Biases into an Attentional Neural Translation Model

https://arxiv.org/pdf/1601.01085.pdf

class xnmt.loss_calculators.**CompositeLoss**(*pt_losses*, *loss_weight=None*)
Bases: *xnmt.persistence.Serializable*, *xnmt.loss_calculators.LossCalculator*

Summing losses from multiple LossCalculator.

class xnmt.loss_calculators.**ReinforceLoss**(*baseline=None*, *evaluation_metric=bare(FastBLEUEvaluator)*, *search_strategy=bare(SamplingSearch)*, *inv_eval=True*, *decoder_hidden_dim=Ref(path=exp_global.default_layer_dim)*)
Bases: *xnmt.persistence.Serializable*, *xnmt.loss_calculators.LossCalculator*

Reinforce Loss according to Ranzato+, 2015. SEQUENCE LEVEL TRAINING WITH RECURRENT NEURAL NETWORKS.

(This is not the MIXER algorithm)

https://arxiv.org/pdf/1511.06732.pdf

class xnmt.loss_calculators.**MinRiskLoss**(*evaluation_metric=bare(FastBLEUEvaluator)*, *alpha=0.005*, *inv_eval=True*, *unique_sample=True*, *search_strategy=bare(SamplingSearch)*)
Bases: *xnmt.persistence.Serializable*, *xnmt.loss_calculators.LossCalculator*

class xnmt.loss_calculators.**FeedbackLoss**(*child_loss=bare(MLELoss)*, *repeat=1*)
Bases: *xnmt.persistence.Serializable*, *xnmt.loss_calculators.LossCalculator*

A loss that first calculates a standard loss function, then feeds it back to the model using the model.additional_loss function.

Parameters

- **child_loss** (*LossCalculator*) – The loss that will be fed back to the model
- **repeat** (Integral) – Repeat the process multiple times and use the sum of the losses. This is useful when there is some non-determinism (such as sampling in the encoder, etc.)

5.4 Training

5.4.1 TrainingRegimen

class xnmt.train.regimens.**TrainingRegimen**
Bases: object

A training regimen is a class that implements a training loop.

run_training(*save_fct*)
Run training steps in a loop until stopping criterion is reached.

Parameters **save_fct** (Callable) – function to be invoked to save a model at dev checkpoints

Return type None

backward (*loss*, *dynet_profiling*)
Perform backward pass to accumulate gradients.

Parameters

- **loss** (Expression) – Result of self.training_step(…)
- **dynet_profiling** (Integral) – if > 0, print the computation graph

Return type None

update (*trainer*)
Update DyNet weights using the given optimizer.

Parameters **trainer** (*XnmtOptimizer*) – DyNet trainer

Return type None

class xnmt.train.regimens.**SimpleTrainingRegimen** (*model=Ref(path=model), src_file=None, trg_file=None, dev_every=0, dev_zero=False, batcher=bare(SrcBatcher{'batch_size': 32}), loss_calculator=bare(MLELoss), trainer=bare(SimpleSGDTrainer{'e0': 0.1}), run_for_epochs=None, lr_decay=1.0, lr_decay_times=3, patience=1, initial_patience=None, dev_tasks=None, dev_combinator=None, restart_trainer=False, reload_command=None, name='{EXP}', sample_train_sents=None, max_num_train_sents=None, max_src_len=None, max_trg_len=None, loss_comb_method=Ref(path=exp_global.loss_comb_method, default=sum), update_every=1, commandline_args=Ref(path=exp_global.commandline_args, default={})*)

Bases: *xnmt.train.tasks.SimpleTrainingTask*, *xnmt.train.regimens.TrainingRegimen*, *xnmt.persistence.Serializable*

Parameters

- **model** (*ConditionedModel*) – the model
- **src_file** (Union[None, str, Sequence[str]]) – the source training file
- **trg_file** (Optional[str]) – the target training file
- **dev_every** (Integral) – dev checkpoints every n sentences (0 for only after epoch)
- **dev_zero** (bool) – if True, add a checkpoint before training loop is entered (useful with pretrained models).
- **batcher** (*Batcher*) – Type of batcher
- **loss_calculator** (*LossCalculator*) – The method for calculating the loss.
- **trainer** (*XnmtOptimizer*) – Trainer object, default is SGD with learning rate 0.1

- **run_for_epochs** (Optional[Integral]) –
- **lr_decay** (Real) –
- **lr_decay_times** (Integral) – Early stopping after decaying learning rate a certain number of times
- **patience** (Integral) – apply LR decay after dev scores haven't improved over this many checkpoints
- **initial_patience** (Optional[Integral]) – if given, allows adjusting patience for the first LR decay
- **dev_tasks** (Optional[Sequence[*EvalTask*]]) – A list of tasks to use during the development stage.
- **dev_combinator** (Optional[str]) – A formula to combine together development scores into a single score to choose whether to perform learning rate decay, etc. e.g. 'x[0]-x[1]' would say that the first dev task score minus the second dev task score is our measure of how well we're doing. If not specified, only the score from the first dev task will be used.
- **restart_trainer** (bool) – Restart trainer (useful for Adam) and revert weights to best dev checkpoint when applying LR decay (https://arxiv.org/pdf/1706.09733.pdf)
- **reload_command** (Optional[str]) – Command to change the input data after each epoch. –epoch EPOCH_NUM will be appended to the command. To just reload the data after each epoch set the command to True.
- **name** (str) – will be prepended to log outputs if given
- **sample_train_sents** (Optional[Integral]) –
- **max_num_train_sents** (Optional[Integral]) –
- **max_src_len** (Optional[Integral]) –
- **max_trg_len** (Optional[Integral]) –
- **loss_comb_method** (str) – method for combining loss across batch elements (sum or avg).
- **update_every** (Integral) – simulate large-batch training by accumulating gradients over several steps before updating parameters
- **commandline_args** (dict) –

run_training (*save_fct*)
Main training loop (overwrites TrainingRegimen.run_training())

Return type None

update (*trainer*)
Update DyNet weights using the given optimizer.

Parameters trainer (*XnmtOptimizer*) – DyNet trainer

Return type None

class xnmt.train.regimens.**AutobatchTrainingRegimen**(*model=Ref(path=model), src_file=None, trg_file=None, dev_every=0, dev_zero=False, batcher=bare(SrcBatcher{'batch_size': 32}), loss_calculator=bare(MLELoss), trainer=bare(SimpleSGDTrainer{'e0': 0.1}), run_for_epochs=None, lr_decay=1.0, lr_decay_times=3, patience=1, initial_patience=None, dev_tasks=None, dev_combinator=None, restart_trainer=False, reload_command=None, name='{EXP}', sample_train_sents=None, max_num_train_sents=None, max_src_len=None, max_trg_len=None, loss_comb_method=Ref(path=exp_global.loss_comb_meth default=sum), update_every=1, commandline_args=Ref(path=exp_global.commandline_args, default={})*)

Bases: *xnmt.train.regimens.SimpleTrainingRegimen*

This regimen overrides SimpleTrainingRegimen by accumulating (summing) losses into a FactoreLossExpr *before* running forward/backward in the computation graph. It is designed to work with DyNet autobatching and when parts of architecture make batching difficult (such as structured encoders like TreeLSTMS or Graph Networks). The actual batch size is set through the "update_every" parameter, while the underlying Batcher is expected to have "batch_size" equal to 1.

Parameters

- **model** (*ConditionedModel*) – the model
- **src_file** (Union[None, str, Sequence[str]]) – the source training file
- **trg_file** (Optional[str]) – the target training file
- **dev_every** (Integral) – dev checkpoints every n sentences (0 for only after epoch)
- **dev_zero** (bool) – if True, add a checkpoint before training loop is entered (useful with pretrained models).
- **batcher** (*Batcher*) – Type of batcher
- **loss_calculator** (*LossCalculator*) – The method for calculating the loss.
- **trainer** (*XnmtOptimizer*) – Trainer object, default is SGD with learning rate 0.1
- **run_for_epochs** (Optional[Integral]) –
- **lr_decay** (Real) –
- **lr_decay_times** (Integral) – Early stopping after decaying learning rate a certain number of times
- **patience** (Integral) – apply LR decay after dev scores haven't improved over this many checkpoints

- **initial_patience** (Optional[Integral]) – if given, allows adjusting patience for the first LR decay
- **dev_tasks** (Optional[Sequence[*EvalTask*]]) – A list of tasks to use during the development stage.
- **dev_combinator** (Optional[str]) – A formula to combine together development scores into a single score to choose whether to perform learning rate decay, etc. e.g. 'x[0]-x[1]' would say that the first dev task score minus the second dev task score is our measure of how good we're doing. If not specified, only the score from the first dev task will be used.
- **restart_trainer** (bool) – Restart trainer (useful for Adam) and revert weights to best dev checkpoint when applying LR decay (https://arxiv.org/pdf/1706.09733.pdf)
- **reload_command** (Optional[str]) – Command to change the input data after each epoch. –epoch EPOCH_NUM will be appended to the command. To just reload the data after each epoch set the command to True.
- **name** (str) – will be prepended to log outputs if given
- **sample_train_sents** (Optional[Integral]) –
- **max_num_train_sents** (Optional[Integral]) –
- **max_src_len** (Optional[Integral]) –
- **max_trg_len** (Optional[Integral]) –
- **loss_comb_method** (str) – method for combining loss across batch elements (sum or avg).
- **update_every** (Integral) – how many instances to accumulate before updating parameters. This effectively sets the batch size under DyNet autobatching.
- **commandline_args** (dict) –

run_training (*save_fct*)

Main training loop (overwrites TrainingRegimen.run_training())

Return type None

class xnmt.train.regimens.**MultiTaskTrainingRegimen** (*tasks, trainer=bare(SimpleSGDTrainer{'e0': 0.1}), dev_zero=False, update_every=1, commandline_args=Ref(path=exp_global.commandline_args, default=None)*)

Bases: *xnmt.train.regimens.TrainingRegimen*

Base class for multi-task training classes. Mainly initializes tasks, performs sanity-checks, and manages set_train events.

Parameters

- **tasks** (Sequence[*TrainingTask*]) – list of training tasks. The first item takes on the role of the main task, meaning it will control early stopping, learning rate schedule, and model checkpoints.
- **trainer** (*XnmtOptimizer*) – Trainer object, default is SGD with learning rate 0.1
- **dev_zero** (bool) – if True, add a checkpoint before training loop is entered (useful with pretrained models).
- **update_every** (Integral) – simulate large-batch training by accumulating gradients over several steps before updating parameters

- **commandline_args** (dict) –

trigger_train_event (*value*)

Trigger set_train event, but only if that would lead to a change of the value of set_train. :type value: bool :param value: True or False

Return type None

update (*trainer*)

Update DyNet weights using the given optimizer.

Parameters trainer (*XnmtOptimizer*) – DyNet trainer

Return type None

class xnmt.train.regimens.**SameBatchMultiTaskTrainingRegimen** (*tasks, trainer=bare(SimpleSGDTrainer{'e0': 0.1}), dev_zero=False, per_task_backward=True, loss_comb_method=Ref(path=exp_global.los default=sum), update_every=1, n_task_steps=None, commandline_args=Ref(path=exp_global.commandlin default=None)*)

Bases: *xnmt.train.regimens.MultiTaskTrainingRegimen*, *xnmt.persistence.Serializable*

Multi-task training where gradients are accumulated and weight updates are thus performed jointly for each task. The relative weight between tasks can be configured setting the number of steps to accumulate over for each task. Note that the batch size for each task also has an influence on task weighting. The stopping criterion of the first task is used (other tasks' stopping criteria are ignored).

Parameters

- **tasks** (Sequence[*TrainingTask*]) – Training tasks
- **trainer** (*XnmtOptimizer*) – The trainer is shared across tasks
- **dev_zero** (bool) – If True, add a checkpoint before training loop is entered (useful with pretrained models).
- **per_task_backward** (bool) – If True, call backward() for each task separately and renew computation graph between tasks. Yields the same results, but True uses less memory while False may be faster when using autobatching.
- **loss_comb_method** (str) – Method for combining loss across batch elements ('sum' or 'avg').
- **update_every** (Integral) – Simulate large-batch training by accumulating gradients over several steps before updating parameters. This is implemented as an outer loop, i.e. we first accumulate gradients from steps for each task, and then loop according to this parameter so that we collect multiple steps for each task and always according to the same ratio.
- **n_task_steps** (Optional[Sequence[Integral]]) – The number steps to accumulate for each task, useful for weighting tasks.
- **commandline_args** (dict) –

run_training (*save_fct*)

Run training steps in a loop until stopping criterion is reached.

Parameters **save_fct** (`Callable`) – function to be invoked to save a model at dev checkpoints

Return type `None`

class `xnmt.train.regimens.`**`AlternatingBatchMultiTaskTrainingRegimen`**(*tasks, task_weights=None, trainer=bare(SimpleSGDTrainer{'e 0.1}), dev_zero=False, loss_comb_method=Ref(path=exp_ default=sum), update_every_within=1, update_every_across=1, commandline_args=Ref(path=exp_global.co default=None)*)

Bases: *`xnmt.train.regimens.MultiTaskTrainingRegimen`*, *`xnmt.persistence.Serializable`*

Multi-task training where training steps are performed one after another.

The relative weight between tasks are explicitly specified explicitly, and for each step one task is drawn at random accordingly. The stopping criterion of the first task is used (other tasks' stopping criteria are ignored).

Parameters

- **tasks** (`Sequence`[*`TrainingTask`*]) – training tasks
- **trainer** (*`XnmtOptimizer`*) – the trainer is shared across tasks
- **dev_zero** (`bool`) – if True, add a checkpoint before training loop is entered (useful with pretrained models).
- **loss_comb_method** (`str`) – method for combining loss across batch elements ('sum' or 'avg').
- **update_every_within** (`Integral`) – Simulate large-batch training by accumulating gradients over several steps before updating parameters. The behavior here is to draw multiple times from the same task until update is invoked.
- **update_every_across** (`Integral`) – Simulate large-batch training by accumulating gradients over several steps before updating parameters. The behavior here is to draw tasks randomly several times before doing parameter updates.
- **commandline_args** –

run_training(*save_fct*)

Run training steps in a loop until stopping criterion is reached.

Parameters **save_fct** (`Callable`) – function to be invoked to save a model at dev checkpoints

Return type `None`

class xnmt.train.regimens.**SerialMultiTaskTrainingRegimen**(*tasks, trainer=bare(SimpleSGDTrainer{'e0': 0.1}), dev_zero=False, loss_comb_method=Ref(path=exp_global.loss_co default=sum), update_every=1, commandline_args=Ref(path=exp_global.commandline_ar default=None)*)

Bases: *xnmt.train.regimens.MultiTaskTrainingRegimen*, *xnmt.persistence.Serializable*

Trains only first task until stopping criterion met, then the same for the second task, etc.

Useful to realize a pretraining-finetuning strategy.

Parameters

- **tasks** (Sequence[*TrainingTask*]) – training tasks. The currently active task is treated as main task.
- **trainer** (*XnmtOptimizer*) – the trainer is shared across tasks
- **dev_zero** (bool) – if True, add a checkpoint before training loop is entered (useful with pretrained models).
- **loss_comb_method** (str) – method for combining loss across batch elements ('sum' or 'avg').
- **update_every** (Integral) – simulate large-batch training by accumulating gradients over several steps before updating parameters
- **commandline_args** (dict) –

run_training(*save_fct*)

Run training steps in a loop until stopping criterion is reached.

Parameters **save_fct** (Callable) – function to be invoked to save a model at dev checkpoints

Return type None

5.4.2 TrainingTask

class xnmt.train.tasks.**TrainingTask**(*model*)

Bases: object

Base class for a training task. Training tasks can perform training steps and keep track of the training state, but may not implement the actual training loop.

Parameters **model** (*TrainableModel*) – The model to train

should_stop_training()

Returns True iff training is finished, i.e. training_step(. . .) should not be called again

training_step(***kwargs*)

Perform forward pass for the next training step and handle training logic (switching epoch, reshuffling, ..)

Parameters ****kwargs** – depends on subclass implementations

Return type *FactoredLossExpr*

Returns Loss

next_minibatch()
Infinitely loop over training minibatches.

Return type `Iterator[+T_co]`

Returns Generator yielding (src_batch,trg_batch) tuples

checkpoint(*control_learning_schedule=False*)
Perform a dev checkpoint.

Parameters **control_learning_schedule** (`bool`) – If `False`, only evaluate dev data. If `True`, also perform model saving, LR decay etc. if needed.

Return type `bool`

Returns `True` iff the model needs saving

cur_num_minibatches()
Current number of minibatches (may change between epochs, e.g. for randomizing batchers or if reload_command is given)

Return type `int`

cur_num_sentences()
Current number of parallel sentences (may change between epochs, e.g. if reload_command is given)

Return type `int`

class xnmt.train.tasks.**SimpleTrainingTask**(*model, src_file=None, trg_file=None, dev_every=0, batcher=bare(SrcBatcher{'batch_size': 32}), loss_calculator=bare(MLELoss), run_for_epochs=None, lr_decay=1.0, lr_decay_times=3, patience=1, initial_patience=None, dev_tasks=None, dev_combinator=None, restart_trainer=False, reload_command=None, name=None, sample_train_sents=None, max_num_train_sents=None, max_src_len=None, max_trg_len=None*)
Bases: *xnmt.train.tasks.TrainingTask*, *xnmt.persistence.Serializable*

Parameters

- **model** (*ConditionedModel*) – a trainable supervised model
- **src_file** (`Union[str, Sequence[str], None]`) – The file for the source data.
- **trg_file** (`Optional[str]`) – The file for the target data.
- **dev_every** (`Integral`) – dev checkpoints every n sentences (0 for only after epoch)
- **batcher** (*Batcher*) – Type of batcher
- **loss_calculator** (*LossCalculator*) –
- **run_for_epochs** (`Optional[Integral]`) – number of epochs (None for unlimited epochs)
- **lr_decay** (`Real`) – decay learning rate by multiplying by this factor
- **lr_decay_times** (`Integral`) – Early stopping after decaying learning rate a certain number of times

- **patience** (Integral) – apply LR decay after dev scores haven't improved over this many checkpoints
- **initial_patience** (Optional[Integral]) – if given, allows adjusting patience for the first LR decay
- **dev_tasks** (Optional[Sequence[*EvalTask*]]) – A list of tasks to run on the development set
- **dev_combinator** – A formula to combine together development scores into a single score to choose whether to perform learning rate decay, etc. e.g. 'x[0]-x[1]' would say that the first dev task score minus the second dev task score is our measure of how good we're doing. If not specified, only the score from the first dev task will be used.
- **restart_trainer** (bool) – Restart trainer (useful for Adam) and revert weights to best dev checkpoint when applying LR decay (https://arxiv.org/pdf/1706.09733.pdf)
- **reload_command** (Optional[str]) – Command to change the input data after each epoch. –epoch EPOCH_NUM will be appended to the command. To just reload the data after each epoch set the command to 'true'.
- **sample_train_sents** (Optional[Integral]) – If given, load a random subset of training sentences before each epoch. Useful when training data does not fit in memory.
- **max_num_train_sents** (Optional[Integral]) – Train only on the first n sentences
- **max_src_len** (Optional[Integral]) – Discard training sentences with source-side longer than this
- **max_trg_len** (Optional[Integral]) – Discard training sentences with target-side longer than this
- **name** (Optional[str]) – will be prepended to log outputs if given

should_stop_training()

Signal stopping if self.early_stopping_reached is marked or we exhausted the number of requested epochs.

Return type bool

cur_num_minibatches()

Current number of minibatches (may change between epochs, e.g. for randomizing batchers or if reload_command is given)

Return type Integral

cur_num_sentences()

Current number of parallel sentences (may change between epochs, e.g. if reload_command is given)

Return type Integral

next_minibatch()

Infinitely loops over training minibatches and advances internal epoch state after every complete sweep over the corpus.

Return type Iterator[+T_co]

Returns Generator yielding (src_batch,trg_batch) tuples

training_step(*src*, *trg*)

Perform forward pass for the next training step and handle training logic (switching epoch, reshuffling, ..)

Parameters

- **src** (*Batch*) – src minibatch

- **trg** (*Batch*) – trg minibatch

Returns Loss

checkpoint (*control_learning_schedule=True*)
Performs a dev checkpoint

Parameters **control_learning_schedule** (bool) – If False, only evaluate dev data. If True, also perform model saving, LR decay etc. if needed.

Returns True if the model needs saving, False otherwise

class xnmt.train.tasks.**TrainingState**
Bases: object

This holds the state of the training loop.

5.5 Parameters

5.5.1 ParamManager

class xnmt.param_collections.**ParamManager**
Bases: object

A static class that manages the currently loaded DyNet parameters of all components.

Responsibilities are registering of all components that use DyNet parameters and loading pretrained parameters. Components can register parameters by calling ParamManager.my_params(self) from within their __init__() method. This allocates a subcollection with a unique identifier for this component. When loading previously saved parameters, one or several paths are specified to look for the corresponding saved DyNet collection named after this identifier.

static init_param_col ()
Initializes or resets the parameter collection.

This must be invoked before every time a new model is loaded (e.g. on startup and between consecutive experiments).

Return type None

static add_load_path (*data_file*)
Add new data directory path to load from.

When calling populate(), pretrained parameters from all directories added in this way are searched for the requested component identifiers.

Parameters **data_file** (str) – a data directory (usually named *.data) containing DyNet parameter collections.

Return type None

static populate ()
Populate the parameter collections.

Searches the given data paths and loads parameter collections if they exist, otherwise leave parameters in their randomly initialized state.

Return type None

static my_params (*subcol_owner*)
Creates a dedicated parameter subcollection for a serializable object.

This should only be called from the __init__ method of a Serializable.

Parameters **subcol_owner** (Serializable) – The object which is requesting to be assigned a subcollection.

Return type ParameterCollection

Returns The assigned subcollection.

static global_collection()

Access the top-level parameter collection, including all parameters.

Return type ParameterCollection

Returns top-level DyNet parameter collection

exception xnmt.param_collections.**RevertingUnsavedModelException**

Bases: Exception

5.5.2 Optimizer

class xnmt.optimizers.**XnmtOptimizer**(*optimizer*, *skip_noisy=False*)

Bases: object

A base classe for trainers. Trainers are mostly simple wrappers of DyNet trainers but can add extra functionality.

Parameters

- **optimizer** (Trainer) – the underlying DyNet optimizer (trainer)
- **skip_noisy** (bool) – keep track of a moving average and a moving standard deviation of the log of the gradient norm values, and abort a step if the norm of the gradient exceeds four standard deviations of the moving average. Reference: https://arxiv.org/pdf/1804.09849.pdf

update()

Update the parameters.

Return type None

status()

Outputs information about the trainer in the stderr.

(number of updates since last call, number of clipped gradients, learning rate, etc…)

Return type None

set_clip_threshold(*thr*)

Set clipping thershold

To deactivate clipping, set the threshold to be <=0

Parameters **thr** (Real) – Clipping threshold

Return type None

get_clip_threshold()

Get clipping threshold

Return type Real

Returns Gradient clipping threshold

restart()

Restarts the optimizer

Clears all momentum values and assimilate (if applicable)

Return type `None`

class `xnmt.optimizers.`**`SimpleSGDTrainer`**(*e0=0.1*, *skip_noisy=False*)

Bases: *`xnmt.optimizers.XnmtOptimizer`*, *`xnmt.persistence.Serializable`*

Stochastic gradient descent trainer

This trainer performs stochastic gradient descent, the goto optimization procedure for neural networks.

Parameters

- **`e0`** (`Real`) – Initial learning rate
- **`skip_noisy`** (`bool`) – keep track of a moving average and a moving standard deviation of the log of the gradient norm values, and abort a step if the norm of the gradient exceeds four standard deviations of the moving average. Reference: https://arxiv.org/pdf/1804.09849.pdf

class `xnmt.optimizers.`**`MomentumSGDTrainer`**(*e0=0.01*, *mom=0.9*, *skip_noisy=False*)

Bases: *`xnmt.optimizers.XnmtOptimizer`*, *`xnmt.persistence.Serializable`*

Stochastic gradient descent with momentum

This is a modified version of the SGD algorithm with momentum to stablize the gradient trajectory.

Parameters

- **`e0`** (`Real`) – Initial learning rate
- **`mom`** (`Real`) – Momentum
- **`skip_noisy`** (`bool`) – keep track of a moving average and a moving standard deviation of the log of the gradient norm values, and abort a step if the norm of the gradient exceeds four standard deviations of the moving average. Reference: https://arxiv.org/pdf/1804.09849.pdf

class `xnmt.optimizers.`**`AdagradTrainer`**(*e0=0.1*, *eps=1e-20*, *skip_noisy=False*)

Bases: *`xnmt.optimizers.XnmtOptimizer`*, *`xnmt.persistence.Serializable`*

Adagrad optimizer

The adagrad algorithm assigns a different learning rate to each parameter.

Parameters

- **`e0`** (`Real`) – Initial learning rate
- **`eps`** (`Real`) – Epsilon parameter to prevent numerical instability
- **`skip_noisy`** (`bool`) – keep track of a moving average and a moving standard deviation of the log of the gradient norm values, and abort a step if the norm of the gradient exceeds four standard deviations of the moving average. Reference: https://arxiv.org/pdf/1804.09849.pdf

class `xnmt.optimizers.`**`AdadeltaTrainer`**(*eps=1e-06*, *rho=0.95*, *skip_noisy=False*)

Bases: *`xnmt.optimizers.XnmtOptimizer`*, *`xnmt.persistence.Serializable`*

AdaDelta optimizer

The AdaDelta optimizer is a variant of Adagrad aiming to prevent vanishing learning rates.

Parameters

- **`eps`** (`Real`) – Epsilon parameter to prevent numerical instability
- **`rho`** (`Real`) – Update parameter for the moving average of updates in the numerator
- **`skip_noisy`** (`bool`) – keep track of a moving average and a moving standard deviation of the log of the gradient norm values, and abort a step if the norm of the gradient exceeds four standard deviations of the moving average. Reference: https://arxiv.org/pdf/1804.09849.pdf

class xnmt.optimizers.**AdamTrainer**(*alpha=0.001*, *beta_1=0.9*, *beta_2=0.999*, *eps=1e-08*, *skip_noisy=False*)

Bases: *xnmt.optimizers.XnmtOptimizer*, *xnmt.persistence.Serializable*

Adam optimizer

The Adam optimizer is similar to RMSProp but uses unbiased estimates of the first and second moments of the gradient

Parameters

- **alpha** (Real) – Initial learning rate
- **beta_1** (Real) – Moving average parameter for the mean
- **beta_2** (Real) – Moving average parameter for the variance
- **eps** (Real) – Epsilon parameter to prevent numerical instability
- **skip_noisy** (bool) – keep track of a moving average and a moving standard deviation of the log of the gradient norm values, and abort a step if the norm of the gradient exceeds four standard deviations of the moving average. Reference: https://arxiv.org/pdf/1804.09849.pdf

class xnmt.optimizers.**NoamTrainer**(*alpha=1.0*, *dim=512*, *warmup_steps=4000*, *beta_1=0.9*, *beta_2=0.98*, *eps=1e-09*, *skip_noisy=False*)

Bases: *xnmt.optimizers.XnmtOptimizer*, *xnmt.persistence.Serializable*

Proposed in the paper "Attention is all you need" (https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf) [Page 7, Eq. 3] In this the learning rate of Adam Optimizer is increased for the first warmup steps followed by a gradual decay

Parameters

- **alpha** (Real) –
- **dim** (Integral) –
- **warmup_steps** (Optional[Integral]) –
- **beta_1** (Real) –
- **beta_2** (Real) –
- **eps** (Real) –
- **skip_noisy** (bool) – keep track of a moving average and a moving standard deviation of the log of the gradient norm values, and abort a step if the norm of the gradient exceeds four standard deviations of the moving average. Reference: https://arxiv.org/pdf/1804.09849.pdf

update()

Update the parameters.

Return type None

class xnmt.optimizers.**DummyTrainer**

Bases: *xnmt.optimizers.XnmtOptimizer*, *xnmt.persistence.Serializable*

A dummy trainer that does not perform any parameter updates.

update()

Update the parameters.

Return type None

status()

Outputs information about the trainer in the stderr.

(number of updates since last call, number of clipped gradients, learning rate, etc…)

Return type None

set_clip_threshold(*thr*)

Set clipping thershold

To deactivate clipping, set the threshold to be <=0

Parameters **thr** – Clipping threshold

Return type None

get_clip_threshold()

Get clipping threshold

Return type None

Returns Gradient clipping threshold

restart()

Restarts the optimizer

Clears all momentum values and assimilate (if applicable)

Return type None

5.5.3 ParamInitializer

class xnmt.param_initializers.**ParamInitializer**

Bases: object

A parameter initializer that delegates to the DyNet initializers and possibly performs some extra configuration.

initializer(*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** – dimension of parameter tensor
- **is_lookup** – True if parameters are a lookup matrix
- **num_shared** – Indicates if one parameter object holds multiple matrices

Returns a dynet initializer object

class xnmt.param_initializers.**NormalInitializer**(*mean=0*, *var=1*)

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Wraps DyNet's NormalInitializer: http://dynet.readthedocs.io/en/latest/python_ref.html#dynet.NormalInitializer

Initialize the parameters with a gaussian distribution.

Parameters

- **mean** (Real) – Mean of the distribution
- **var** (Real) – Variance of the distribution

initializer(*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimension of parameter tensor

- **is_lookup** (bool) – True if parameters are a lookup matrix
- **num_shared** (Integral) – Indicates if one parameter object holds multiple matrices

Return type NormalInitializer

Returns a dynet initializer object

class xnmt.param_initializers.**UniformInitializer**(*scale*)

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Wraps DyNet's UniformInitializer: http://dynet.readthedocs.io/en/latest/python_ref.html#dynet.UniformInitializer

Initialize the parameters with a uniform distribution. :type scale: Real :param scale: Parameters are sampled from $\mathcal{U}([-\texttt{scale}, \texttt{scale}])$

initializer(*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimension of parameter tensor
- **is_lookup** (bool) – True if parameters are a lookup matrix
- **num_shared** (Integral) – Indicates if one parameter object holds multiple matrices

Return type UniformInitializer

Returns a dynet initializer object

class xnmt.param_initializers.**ConstInitializer**(*c*)

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Wraps DyNet's ConstInitializer: http://dynet.readthedocs.io/en/latest/python_ref.html#dynet.ConstInitializer

Initialize the parameters with a constant value.

Parameters **c** (Real) – Value to initialize the parameters

initializer(*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimension of parameter tensor
- **is_lookup** (bool) – True if parameters are a lookup matrix
- **num_shared** (Integral) – Indicates if one parameter object holds multiple matrices

Return type ConstInitializer

Returns a dynet initializer object

class xnmt.param_initializers.**GlorotInitializer**(*gain=1.0*)

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Wraps DyNet's GlorotInitializer: http://dynet.readthedocs.io/en/latest/python_ref.html#dynet.GlorotInitializer

Initializes the weights according to Glorot & Bengio (2011)

If the dimensions of the parameter matrix are m, n, the weights are sampled from $\mathcal{U}([-g\sqrt{\frac{6}{m+n}}, g\sqrt{\frac{6}{m+n}}])$

The gain g depends on the activation function :

- tanh : 1.0
- ReLU : 0.5
- sigmoid : 4.0
- Any smooth function f : $\frac{1}{f'(0)}$

In addition to the DyNet class, this also supports the case where one parameter object stores several matrices (as is popular for computing LSTM gates, for instance).

Note: This is also known as **Xavier initialization**

Parameters **gain** (Real) – Gain (Depends on the activation function)

initializer (*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimensions of parameter tensor
- **is_lookup** (bool) – Whether the parameter is a lookup parameter
- **num_shared** (Integral) – If > 1, treat the first dimension as spanning multiple matrices, each of which is initialized individually

Return type UniformInitializer

Returns a dynet initializer object

class xnmt.param_initializers.**FromFileInitializer** (*fname*)

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Wraps DyNet's FromFileInitializer: http://dynet.readthedocs.io/en/latest/python_ref.html#dynet.FromFileInitializer

Initialize parameter from file.

Parameters **fname** (str) – File name

initializer (*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimension of parameter tensor
- **is_lookup** (bool) – True if parameters are a lookup matrix
- **num_shared** (Integral) – Indicates if one parameter object holds multiple matrices

Return type FromFileInitializer

Returns a dynet initializer object

class xnmt.param_initializers.**NumpyInitializer** (*array*)

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Wraps DyNet's NumpyInitializer: http://dynet.readthedocs.io/en/latest/python_ref.html#dynet.NumpyInitializer

Initialize from numpy array

Alternatively, use ParameterCollection.parameters_from_numpy()

Parameters **array** (ndarray) – Numpy array

initializer(*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimension of parameter tensor
- **is_lookup** (bool) – True if parameters are a lookup matrix
- **num_shared** (Integral) – Indicates if one parameter object holds multiple matrices

Return type NumpyInitializer

Returns a dynet initializer object

class xnmt.param_initializers.**ZeroInitializer**

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Initializes parameter matrix to zero (most appropriate for bias parameters).

initializer(*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimension of parameter tensor
- **is_lookup** (bool) – True if parameters are a lookup matrix
- **num_shared** (Integral) – Indicates if one parameter object holds multiple matrices

Return type ConstInitializer

Returns a dynet initializer object

class xnmt.param_initializers.**LeCunUniformInitializer**(*scale=1.0*)

Bases: *xnmt.param_initializers.ParamInitializer*, *xnmt.persistence.Serializable*

Reference: LeCun 98, Efficient Backprop http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Parameters **scale** (Real) – scale

initializer(*dim*, *is_lookup=False*, *num_shared=1*)

Parameters

- **dim** (Tuple[Integral]) – dimension of parameter tensor
- **is_lookup** (bool) – True if parameters are a lookup matrix
- **num_shared** (Integral) – Indicates if one parameter object holds multiple matrices

Return type UniformInitializer

Returns a dynet initializer object

5.6 Inference

5.6.1 AutoRegressiveInference

class xnmt.inferences.**Inference**(*src_file=None*, *trg_file=None*, *ref_file=None*, *max_src_len=None*, *max_num_sents=None*, *mode='onebest'*, *batcher=bare(InOrderBatcher{'batch_size': 1})*, *reporter=None*)

Bases: object

A template class for classes that perform inference.

Parameters

- **src_file** (Optional[str]) – path of input src file to be translated
- **trg_file** (Optional[str]) – path of file where trg translatons will be written
- **ref_file** (Optional[str]) – path of file with reference translations, e.g. for forced decoding
- **max_src_len** (Optional[int]) – Remove sentences from data to decode that are longer than this on the source side
- **max_num_sents** (Optional[int]) – Stop decoding after the first n sentences.
- **mode** (str) – type of decoding to perform.
  - onebest: generate one best.
  - score: output scores, useful for rescoring
  - forced: perform forced decoding.
  - forceddebug: perform forced decoding, calculate training loss, and make sure the scores are identical for debugging purposes.
- **batcher** (*InOrderBatcher*) – inference batcher, needed e.g. in connection with pad_src_token_to_multiple
- **reporter** (Union[None, *Reporter*, Sequence[*Reporter*]]) – a reporter to create reports for each decoded sentence

perform_inference(*generator*, *src_file=None*, *trg_file=None*, *ref_file=None*)
Perform inference.

Parameters

- **generator** (*GeneratorModel*) – the model to be used
- **src_file** (Optional[str]) – path of input src file to be translated
- **trg_file** (Optional[str]) – path of file where trg translatons will be written

Return type None

class xnmt.inferences.**IndependentOutputInference**(*src_file=None*, *trg_file=None*, *ref_file=None*, *max_src_len=None*, *max_num_sents=None*, *post_process=None*, *mode='onebest'*, *batcher=bare(InOrderBatcher{'batch_size': 1})*, *reporter=None*)

Bases: *xnmt.inferences.Inference*, *xnmt.persistence.Serializable*

Inference when outputs are produced independently, including for classifiers that produce only a single output.

Assumes that generator.generate() takes arguments src, idx

Parameters

- **src_file** (Optional[str]) – path of input src file to be translated
- **trg_file** (Optional[str]) – path of file where trg translatons will be written
- **ref_file** (Optional[str]) – path of file with reference translations, e.g. for forced decoding

- **max_src_len** (Optional[int]) – Remove sentences from data to decode that are longer than this on the source side
- **max_num_sents** (Optional[int]) – Stop decoding after the first n sentences.
- **post_process** (Union[None, str, OutputProcessor, Sequence[OutputProcessor]]) – post-processing of translation outputs (available string shortcuts: none, join-char, join-bpe, join-piece)
- **mode** (str) – type of decoding to perform.
  - onebest: generate one best.
  - score: output scores, useful for rescoring
- **batcher** (*InOrderBatcher*) – inference batcher, needed e.g. in connection with pad_src_token_to_multiple
- **reporter** (Union[None, *Reporter*, Sequence[*Reporter*]]) – a reporter to create reports for each decoded sentence

class xnmt.inferences.**AutoRegressiveInference**(*src_file=None, trg_file=None, ref_file=None, max_src_len=None, max_num_sents=None, post_process=[], search_strategy=bare(BeamSearch), mode='onebest', batcher=bare(InOrderBatcher{'batch_size': 1}), reporter=None*)

Bases: *xnmt.inferences.Inference*, *xnmt.persistence.Serializable*

Performs inference for auto-regressive models that expand based on their own previous outputs.

Assumes that generator.generate() takes arguments src, idx, search_strategy, forced_trg_ids

Parameters

- **src_file** (Optional[str]) – path of input src file to be translated
- **trg_file** (Optional[str]) – path of file where trg translatons will be written
- **ref_file** (Optional[str]) – path of file with reference translations, e.g. for forced decoding
- **max_src_len** (Optional[int]) – Remove sentences from data to decode that are longer than this on the source side
- **max_num_sents** (Optional[int]) – Stop decoding after the first n sentences.
- **post_process** (Union[str, OutputProcessor, Sequence[OutputProcessor]]) – post-processing of translation outputs (available string shortcuts: none,``join-char``,``join-bpe``,``join-piece``)
- **search_strategy** (*SearchStrategy*) – a search strategy used during decoding.
- **mode** (str) – type of decoding to perform.
  - onebest: generate one best.
  - score: output scores, useful for rescoring
- **batcher** (*InOrderBatcher*) – inference batcher, needed e.g. in connection with pad_src_token_to_multiple
- **reporter** (Union[None, *Reporter*, Sequence[*Reporter*]]) – a reporter to create reports for each decoded sentence

class xnmt.inferences.**CascadeInference**(*steps*)

Bases: *xnmt.inferences.Inference*, *xnmt.persistence.Serializable*

Inference class that performs inference as a series of independent inference steps.

Steps are performed using a list of inference sub-objects and a list of models. Intermediate outputs are written out to disk and then read by the next time step.

The generator passed to `perform_inference` must be a `xnmt.models.CascadeGenerator`.

Parameters **steps** (Sequence[*Inference*]) – list of inference objects

perform_inference(*generator*, *src_file=None*, *trg_file=None*, *ref_file=None*)

Perform inference.

Parameters

- **generator** (*CascadeGenerator*) – the model to be used
- **src_file** (Optional[str]) – path of input src file to be translated
- **trg_file** (Optional[str]) – path of file where trg translatons will be written

Return type None

5.6.2 SearchStrategy

class xnmt.search_strategies.**SearchOutput**(*word_ids*, *attentions*, *score*, *state*, *mask*)

Bases: `tuple`

Output of the search words_ids: list of generated word ids attentions: list of corresponding attention vector of word_ids score: a single value of log(p(E|F)) logsoftmaxes: a corresponding softmax vector of the score. score = logsoftmax[word_id] state: a NON-BACKPROPAGATEABLE state that is used to produce the logsoftmax layer

state is usually used to generate 'baseline' in reinforce loss

masks: whether the particular word id should be ignored or not (1 for not, 0 for yes)

attentions

Alias for field number 1

mask

Alias for field number 4

score

Alias for field number 2

state

Alias for field number 3

word_ids

Alias for field number 0

class xnmt.search_strategies.**SearchStrategy**

Bases: `object`

A template class to generate translation from the output probability model. (Non-batched operation)

generate_output(*translator*, *initial_state*, *src_length=None*)

Parameters

- **translator** (*xnmt.models.translators.AutoRegressiveTranslator*) – a translator

- **initial_state** (*AutoRegressiveDecoderState*) – initial decoder state
- **src_length** (Optional[Integral]) – length of src sequence, required for some types of length normalization

Return type List[*SearchOutput*]

Returns List of (word_ids, attentions, score, logsoftmaxes)

class xnmt.search_strategies.**GreedySearch**(*max_len=100*)
Bases: *xnmt.persistence.Serializable*, *xnmt.search_strategies.SearchStrategy*

Performs greedy search (aka beam search with beam size 1)

Parameters **max_len** (Integral) – maximum number of tokens to generate.

generate_output(*translator*, *initial_state*, *src_length=None*)

Parameters

- **translator** (*xnmt.models.translators.AutoRegressiveTranslator*) – a translator
- **initial_state** (*AutoRegressiveDecoderState*) – initial decoder state
- **src_length** (Optional[Integral]) – length of src sequence, required for some types of length normalization

Return type List[*SearchOutput*]

Returns List of (word_ids, attentions, score, logsoftmaxes)

class xnmt.search_strategies.**BeamSearch**(*beam_size=1*, *max_len=100*, *len_norm=bare(NoNormalization)*, *one_best=True*, *scores_proc=None*)
Bases: *xnmt.persistence.Serializable*, *xnmt.search_strategies.SearchStrategy*

Performs beam search.

Parameters

- **beam_size** (Integral) – number of beams
- **max_len** (Integral) – maximum number of tokens to generate.
- **len_norm** (*LengthNormalization*) – type of length normalization to apply
- **one_best** (bool) – Whether to output the best hyp only or all completed hyps.
- **scores_proc** (Optional[Callable[[ndarray], None]]) – apply an optional operation on all scores prior to choosing the top k. E.g. use with xnmt.length_normalization.EosBooster.

class Hypothesis(*score*, *output*, *parent*, *word*)
Bases: tuple

output
Alias for field number 1

parent
Alias for field number 2

score
Alias for field number 0

word
Alias for field number 3

generate_output (*translator*, *initial_state*, *src_length=None*)

Parameters

- **translator** (*xnmt.models.translators.AutoRegressiveTranslator*) – a translator
- **initial_state** (*AutoRegressiveDecoderState*) – initial decoder state
- **src_length** (Optional[Integral]) – length of src sequence, required for some types of length normalization

Return type List[*SearchOutput*]

Returns List of (word_ids, attentions, score, logsoftmaxes)

class xnmt.search_strategies.**SamplingSearch** (*max_len=100*, *sample_size=5*)
Bases: *xnmt.persistence.Serializable*, *xnmt.search_strategies.SearchStrategy*

Performs search based on the softmax probability distribution. Similar to greedy searchol

Parameters

- **max_len** (Integral) –
- **sample_size** (Integral) –

generate_output (*translator*, *initial_state*, *src_length=None*)

Parameters

- **translator** (*xnmt.models.translators.AutoRegressiveTranslator*) – a translator
- **initial_state** (*AutoRegressiveDecoderState*) – initial decoder state
- **src_length** (Optional[Integral]) – length of src sequence, required for some types of length normalization

Return type List[*SearchOutput*]

Returns List of (word_ids, attentions, score, logsoftmaxes)

class xnmt.search_strategies.**MctsSearch** (*visits=200*, *max_len=100*)
Bases: *xnmt.persistence.Serializable*, *xnmt.search_strategies.SearchStrategy*

Performs search with Monte Carlo Tree Search

generate_output (*translator*, *dec_state*, *src_length=None*)

Parameters

- **translator** (*xnmt.models.translators.AutoRegressiveTranslator*) – a translator
- **initial_state** – initial decoder state
- **src_length** (Optional[Integral]) – length of src sequence, required for some types of length normalization

Return type List[*SearchOutput*]

Returns List of (word_ids, attentions, score, logsoftmaxes)

5.6.3 LengthNormalization

class xnmt.length_norm.**LengthNormalization**

Bases: object

A template class to adjust scores for length normalization during search.

normalize_completed(*completed_hyps*, *src_length=None*)

Apply normalization step to completed hypotheses after search and return the normalized scores.

Parameters

- **completed_hyps** (Sequence[Hypothesis]) – list of completed Hypothesis objects, will be normalized in-place
- **src_length** (Optional[int]) – length of source sequence (None if not given)

Return type Sequence[float]

Returns normalized scores

normalize_partial_topk(*score_so_far*, *score_to_add*, *new_len*)

Apply normalization step after expanding a partial hypothesis and selecting the top k scores.

Parameters

- **score_so_far** – log score of the partial hypothesis
- **score_to_add** – log score of the top-k item that is to be added
- **new_len** – new length of partial hypothesis with current word already appended

Returns new score after applying score_to_add to score_so_far

class xnmt.length_norm.**NoNormalization**

Bases: *xnmt.length_norm.LengthNormalization*, *xnmt.persistence.Serializable*

Adding no form of length normalization.

normalize_completed(*completed_hyps*, *src_length=None*)

Apply normalization step to completed hypotheses after search and return the normalized scores.

Parameters

- **completed_hyps** (Sequence[Hypothesis]) – list of completed Hypothesis objects, will be normalized in-place
- **src_length** (Optional[int]) – length of source sequence (None if not given)

Return type Sequence[float]

Returns normalized scores

class xnmt.length_norm.**AdditiveNormalization**(*penalty=-0.1*, *apply_during_search=False*)

Bases: *xnmt.length_norm.LengthNormalization*, *xnmt.persistence.Serializable*

Adding a fixed word penalty everytime the word is added.

normalize_completed(*completed_hyps*, *src_length=None*)

Apply normalization step to completed hypotheses after search and return the normalized scores.

Parameters

- **completed_hyps** (Sequence[Hypothesis]) – list of completed Hypothesis objects, will be normalized in-place
- **src_length** (Optional[int]) – length of source sequence (None if not given)

Return type `Sequence[float]`

Returns normalized scores

normalize_partial_topk(*score_so_far*, *score_to_add*, *new_len*)

Apply normalization step after expanding a partial hypothesis and selecting the top k scores.

Parameters

- **score_so_far** – log score of the partial hypothesis
- **score_to_add** – log score of the top-k item that is to be added
- **new_len** – new length of partial hypothesis with current word already appended

Returns new score after applying score_to_add to score_so_far

class xnmt.length_norm.**PolynomialNormalization**(*m=1*, *apply_during_search=False*)

Bases: *xnmt.length_norm.LengthNormalization*, *xnmt.persistence.Serializable*

Dividing by the length (raised to some power)

normalize_completed(*completed_hyps*, *src_length=None*)

Apply normalization step to completed hypotheses after search and return the normalized scores.

Parameters

- **completed_hyps** (`Sequence[Hypothesis]`) – list of completed Hypothesis objects, will be normalized in-place
- **src_length** (`Optional[int]`) – length of source sequence (None if not given)

Return type `Sequence[float]`

Returns normalized scores

normalize_partial_topk(*score_so_far*, *score_to_add*, *new_len*)

Apply normalization step after expanding a partial hypothesis and selecting the top k scores.

Parameters

- **score_so_far** – log score of the partial hypothesis
- **score_to_add** – log score of the top-k item that is to be added
- **new_len** – new length of partial hypothesis with current word already appended

Returns new score after applying score_to_add to score_so_far

class xnmt.length_norm.**MultinomialNormalization**(*sent_stats*)

Bases: *xnmt.length_norm.LengthNormalization*, *xnmt.persistence.Serializable*

The algorithm followed by: Tree-to-Sequence Attentional Neural Machine Translation https://arxiv.org/pdf/1603.06075.pdf

normalize_completed(*completed_hyps*, *src_length=None*)

Parameters

- **completed_hyps** (`Sequence[Hypothesis]`) –
- **src_length** (`Optional[int]`) – length of the src sent

Return type `Sequence[float]`

class xnmt.length_norm.**GaussianNormalization**(*sent_stats*)

Bases: *xnmt.length_norm.LengthNormalization*, *xnmt.persistence.Serializable*

The Gaussian regularization encourages the inference to select sents that have similar lengths as the sents in the training set. refer: https://arxiv.org/pdf/1509.04942.pdf

normalize_completed(*completed_hyps*, *src_length=None*)
Apply normalization step to completed hypotheses after search and return the normalized scores.

Parameters

- **completed_hyps** (Sequence[Hypothesis]) – list of completed Hypothesis objects, will be normalized in-place
- **src_length** (Optional[int]) – length of source sequence (None if not given)

Return type Sequence[float]

Returns normalized scores

class xnmt.length_norm.**EosBooster**(*boost_val*)
Bases: *xnmt.persistence.Serializable*

Callable that applies boosting of end-of-sequence token, can be used with xnmt.search_strategy.BeamSearch.

Parameters **boost_val** (Real) – value to add to the eos token's log probability. Positive values make sentences shorter, negative values make sentences longer.

5.7 Evaluation

5.7.1 EvalTasks

class xnmt.eval.tasks.**EvalTask**
Bases: object

An EvalTask is a task that does evaluation and returns one or more EvalScore objects.

class xnmt.eval.tasks.**LossEvalTask**(*src_file*, *ref_file=None*, *model=Ref(path=model)*, *batcher=Ref(path=train.batcher*, *default=SrcBatcher@139703660330568)*, *loss_calculator=bare(MLELoss)*, *max_src_len=None*, *max_trg_len=None*, *max_num_sents=None*, *loss_comb_method=Ref(path=exp_global.loss_comb_method*, *default=sum)*, *desc=None*)
Bases: *xnmt.eval.tasks.EvalTask*, *xnmt.persistence.Serializable*

A task that does evaluation of the loss function.

Parameters

- **src_file** (Union[str, Sequence[str]]) – source file name
- **ref_file** (Optional[str]) – reference file name
- **model** (*GeneratorModel*) – generator model to use for inference
- **batcher** (*Batcher*) – batcher to use
- **loss_calculator** (*LossCalculator*) – loss calculator
- **max_src_len** (Optional[int]) – omit sentences with source length greater than specified number

- **max_trg_len** (Optional[int]) – omit sentences with target length greater than specified number
- **max_num_sents** (Optional[int]) – compute loss only for the first n sentences in the given corpus
- **loss_comb_method** (str) – method for combining loss across batch elements ('sum' or 'avg').
- **desc** (Optional[Any]) – description to pass on to computed score objects

eval()

Perform evaluation task.

Return type *EvalScore*

Returns Evaluated score

class xnmt.eval.tasks.**AccuracyEvalTask**(*src_file*, *ref_file*, *hyp_file*, *model=Ref(path=model)*, *eval_metrics='bleu'*, *inference=None*, *perform_inference=True*, *desc=None*)

Bases: *xnmt.eval.tasks.EvalTask*, *xnmt.persistence.Serializable*

A task that does evaluation of some measure of accuracy.

Parameters

- **src_file** (Union[str, Sequence[str]]) – path(s) to read source file(s) from
- **ref_file** (Union[str, Sequence[str]]) – path(s) to read reference file(s) from
- **hyp_file** (str) – path to write hypothesis file to
- **model** (*GeneratorModel*) – generator model to generate hypothesis with
- **eval_metrics** (Union[str, *Evaluator*, Sequence[*Evaluator*]]) – list of evaluation metrics (list of Evaluator objects or string of comma-separated shortcuts)
- **inference** (Optional[*Inference*]) – inference object
- **perform_inference** (bool) – Whether to generate the output or not. One eval task can use an already existing hyp_file that was generated by the previous eval tasks.
- **desc** (Optional[Any]) – human-readable description passed on to resulting score objects

class xnmt.eval.tasks.**DecodingEvalTask**(*src_file*, *hyp_file*, *model=Ref(path=model)*, *inference=None*)

Bases: *xnmt.eval.tasks.EvalTask*, *xnmt.persistence.Serializable*

A task that does performs decoding without comparing against a reference.

Parameters

- **src_file** (Union[str, Sequence[str]]) – path(s) to read source file(s) from
- **hyp_file** (str) – path to write hypothesis file to
- **model** (*GeneratorModel*) – generator model to generate hypothesis with
- **inference** (Optional[*Inference*]) – inference object

5.7.2 Eval Metrics

This module contains classes to compute evaluation metrics and to hold the resulting scores.

EvalScore subclasses represent a computed score, including useful statistics, and can be printed with an informative string representation.

Evaluator subclasses are used to compute these scores. Currently the following are implemented:

- *LossScore* (created directly by the model)
- *BLEUEvaluator* and *FastBLEUEvaluator* create *BLEUScore* objects
- *GLEUEvaluator* creates *GLEUScore* objects
- *WEREvaluator* creates *WERScore* objects
- *CEREvaluator* creates *CERScore* objects
- *ExternalEvaluator* creates *ExternalScore* objects
- *SequenceAccuracyEvaluator* creates *SequenceAccuracyScore* objects

class xnmt.eval.metrics.**EvalScore**(*desc=None*)

Bases: `object`

A template class for scores as resulting from using an *Evaluator*.

Parameters **desc** (`Optional[Any]`) – human-readable description to include in log outputs

higher_is_better()

Return `True` if higher values are favorable, `False` otherwise.

Return type `bool`

Returns Whether higher values are favorable.

value()

Get the numeric value of the evaluated metric.

Return type `float`

Returns Numeric evaluation score.

metric_name()

Get the metric name.

Return type `str`

Returns Metric name as string.

score_str()

A string representation of the evaluated score, potentially including additional statistics.

Return type `str`

Returns String representation of score.

better_than(*another_score*)

Compare score against another score and return `True` iff this score is better.

Parameters **another_score** (*EvalScore*) – score to _compare against.

Return type `bool`

Returns Whether this score is better than `another_score`.

class xnmt.eval.metrics.**SentenceLevelEvalScore**(*desc=None*)

Bases: *xnmt.eval.metrics.EvalScore*

A template class for scores that work on a sentence-level and can be aggregated to corpus-level.

static aggregate(*scores*, *desc=None*)

Aggregate a sequence of sentence-level scores into a corpus-level score.

Parameters

- **scores** (Sequence[*SentenceLevelEvalScore*]) – list of sentence-level scores.
- **desc** (Optional[Any]) – human-readable description.

Return type *SentenceLevelEvalScore*

Returns Score object that is the aggregate of all sentence-level scores.

class xnmt.eval.metrics.**LossScore**(*loss*, *loss_stats=None*, *num_ref_words=None*, *desc=None*)

Bases: *xnmt.eval.metrics.EvalScore*, *xnmt.persistence.Serializable*

Score indicating the value of the loss function of a neural network.

Parameters

- **loss** (Real) – the (primary) loss value
- **loss_stats** (Optional[Dict[str, Real]]) – info on additional loss values
- **num_ref_words** (Optional[Integral]) – number of reference tokens
- **desc** (Optional[Any]) – human-readable description to include in log outputs

value()

Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

metric_name()

Get the metric name.

Returns Metric name as string.

higher_is_better()

Return True if higher values are favorable, False otherwise.

Returns Whether higher values are favorable.

score_str()

A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

class xnmt.eval.metrics.**BLEUScore**(*bleu*, *frac_score_list=None*, *brevity_penalty_score=None*, *hyp_len=None*, *ref_len=None*, *ngram=4*, *desc=None*)

Bases: *xnmt.eval.metrics.EvalScore*, *xnmt.persistence.Serializable*

Class to keep a BLEU score.

Parameters

- **bleu** (Real) – actual BLEU score between 0 and 1
- **frac_score_list** (Optional[Sequence[Real]]) – list of fractional scores for each n-gram order
- **brevity_penalty_score** (Optional[Real]) – brevity penalty that was multiplied to the precision score.
- **hyp_len** (Optional[Integral]) – length of hypothesis
- **ref_len** (Optional[Integral]) – length of reference
- **ngram** (Integral) – match n-grams up to this order (usually 4)

- **desc** (Optional[Any]) – human-readable description to include in log outputs

value()

Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

metric_name()

Get the metric name.

Returns Metric name as string.

higher_is_better()

Return True if higher values are favorable, False otherwise.

Returns Whether higher values are favorable.

score_str()

A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

class xnmt.eval.metrics.**GLEUScore**(*corpus_n_match*, *corpus_total*, *hyp_len*, *ref_len*, *desc=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvalScore*, *xnmt.persistence.Serializable*

Class to keep a GLEU (Google BLEU) score.

Parameters

- **gleu** – actual GLEU score between 0 and 1
- **hyp_len** (Integral) – length of hypothesis
- **ref_len** (Integral) – length of reference
- **desc** (Optional[Any]) – human-readable description to include in log outputs

value()

Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

metric_name()

Get the metric name.

Returns Metric name as string.

higher_is_better()

Return True if higher values are favorable, False otherwise.

Returns Whether higher values are favorable.

score_str()

A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

static aggregate(*scores*, *desc=None*)

Aggregate a sequence of sentence-level scores into a corpus-level score.

Parameters

- **scores** (Sequence[*SentenceLevelEvalScore*]) – list of sentence-level scores.
- **desc** (Optional[Any]) – human-readable description.

Returns Score object that is the aggregate of all sentence-level scores.

class xnmt.eval.metrics.**LevenshteinScore**(*correct*, *substitutions*, *insertions*, *deletions*, *desc=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvalScore*

A template class for Levenshtein-based scores.

Parameters

- **correct** (Integral) – number of correct matches
- **substitutions** (Integral) – number of substitution errors
- **insertions** (Integral) – number of insertion errors
- **deletions** (Integral) – number of deletion errors
- **desc** (Optional[Any]) – human-readable description to include in log outputs

value()

Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

higher_is_better()

Return True if higher values are favorable, False otherwise.

Returns Whether higher values are favorable.

score_str()

A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

static aggregate(*scores*, *desc=None*)

Aggregate a sequence of sentence-level scores into a corpus-level score.

Parameters

- **scores** (Sequence[*LevenshteinScore*]) – list of sentence-level scores.
- **desc** (Optional[Any]) – human-readable description.

Return type *LevenshteinScore*

Returns Score object that is the aggregate of all sentence-level scores.

class xnmt.eval.metrics.**WERScore**(*correct*, *substitutions*, *insertions*, *deletions*, *desc=None*)

Bases: *xnmt.eval.metrics.LevenshteinScore*, *xnmt.persistence.Serializable*

Class to keep a word error rate.

metric_name()

Get the metric name.

Returns Metric name as string.

class xnmt.eval.metrics.**CERScore**(*correct*, *substitutions*, *insertions*, *deletions*, *desc=None*)

Bases: *xnmt.eval.metrics.LevenshteinScore*, *xnmt.persistence.Serializable*

Class to keep a character error rate.

metric_name()

Get the metric name.

Returns Metric name as string.

class xnmt.eval.metrics.**RecallScore**(*recall*, *hyp_len*, *ref_len*, *nbest=5*, *desc=None*)
Bases: *xnmt.eval.metrics.SentenceLevelEvalScore*, *xnmt.persistence.Serializable*

Class to keep a recall score.

Parameters

- **recall** (Real) – recall score value between 0 and 1
- **hyp_len** (Integral) – length of hypothesis
- **ref_len** (Integral) – length of reference
- **nbest** (Integral) – recall computed within n-best of specified n
- **desc** (Optional[Any]) – human-readable description to include in log outputs

higher_is_better()
Return True if higher values are favorable, False otherwise.

Returns Whether higher values are favorable.

score_str()
A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

value()
Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

metric_name()
Get the metric name.

Returns Metric name as string.

static aggregate(*scores*, *desc=None*)
Aggregate a sequence of sentence-level scores into a corpus-level score.

Parameters

- **scores** (Sequence[*RecallScore*]) – list of sentence-level scores.
- **desc** (Optional[Any]) – human-readable description.

Return type *RecallScore*

Returns Score object that is the aggregate of all sentence-level scores.

class xnmt.eval.metrics.**ExternalScore**(*value*, *higher_is_better=True*, *desc=None*)
Bases: *xnmt.eval.metrics.EvalScore*, *xnmt.persistence.Serializable*

Class to keep a score computed with an external tool.

Parameters

- **value** (Real) – score value
- **higher_is_better** (bool) – whether higher scores or lower scores are favorable
- **desc** (Optional[Any]) – human-readable description to include in log outputs

value()
Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

metric_name()
Get the metric name.

Returns Metric name as string.

higher_is_better()
Return `True` if higher values are favorable, `False` otherwise.

Returns Whether higher values are favorable.

score_str()
A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

class xnmt.eval.metrics.**SequenceAccuracyScore**(*num_correct*, *num_total*, *desc=None*)
Bases: *xnmt.eval.metrics.SentenceLevelEvalScore*, *xnmt.persistence.Serializable*

Class to keep a sequence accuracy score.

Parameters

- **num_correct** (`Integral`) – number of correct outputs
- **num_total** (`Integral`) – number of total outputs
- **desc** (`Optional[Any]`) – human-readable description to include in log outputs

higher_is_better()
Return `True` if higher values are favorable, `False` otherwise.

Returns Whether higher values are favorable.

value()
Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

metric_name()
Get the metric name.

Returns Metric name as string.

score_str()
A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

static aggregate(*scores*, *desc=None*)
Aggregate a sequence of sentence-level scores into a corpus-level score.

Parameters

- **scores** (`Sequence[`*SentenceLevelEvalScore*`]`) – list of sentence-level scores.
- **desc** (`Optional[Any]`) – human-readable description.

Returns Score object that is the aggregate of all sentence-level scores.

class xnmt.eval.metrics.**FMeasure**(*true_pos*, *false_neg*, *false_pos*, *desc=None*)
Bases: *xnmt.eval.metrics.SentenceLevelEvalScore*, *xnmt.persistence.Serializable*

higher_is_better()
Return `True` if higher values are favorable, `False` otherwise.

Returns Whether higher values are favorable.

value()
Get the numeric value of the evaluated metric.

Returns Numeric evaluation score.

metric_name()
Get the metric name.

Returns Metric name as string.

score_str()
A string representation of the evaluated score, potentially including additional statistics.

Returns String representation of score.

static aggregate(*scores*, *desc=None*)
Aggregate a sequence of sentence-level scores into a corpus-level score.

Parameters

- **scores** (Sequence[*SentenceLevelEvalScore*]) – list of sentence-level scores.
- **desc** (Optional[Any]) – human-readable description.

Returns Score object that is the aggregate of all sentence-level scores.

class xnmt.eval.metrics.**Evaluator**
Bases: object

A template class to evaluate the quality of output.

evaluate(*ref*, *hyp*, *desc=None*)
Calculate the quality of output given a reference.

Parameters

- **ref** (Sequence[+T_co]) – list of reference sents (a sentence is a list of tokens)
- **hyp** (Sequence[+T_co]) – list of hypothesis sents (a sentence is a list of tokens)
- **desc** (Optional[Any]) – optional description that is passed on to score objects

Returns:

Return type *EvalScore*

evaluate_multi_ref(*ref*, *hyp*, *desc=None*)
Calculate the quality of output given multiple references.

Parameters

- **ref** (Sequence[Sequence[+T_co]]) – list of tuples of reference sentences (a sentence is a list of tokens)
- **hyp** (Sequence[+T_co]) – list of hypothesis sentences (a sentence is a list of tokens)
- **desc** (Optional[Any]) – optional description that is passed on to score objects

Return type *EvalScore*

class xnmt.eval.metrics.**SentenceLevelEvaluator**(*write_sentence_scores=None*)
Bases: *xnmt.eval.metrics.Evaluator*

A template class for sentence-level evaluators.

Parameters **write_sentence_scores** (Optional[str]) – path of file to write sentence-level scores to (in YAML format)

evaluate (*ref*, *hyp*, *desc=None*)
Calculate the quality of output given a reference.

Parameters

- **ref** (Sequence[+T_co]) – list of reference sents (a sentence is a list of tokens)
- **hyp** (Sequence[+T_co]) – list of hypothesis sents (a sentence is a list of tokens)
- **desc** (Optional[Any]) – optional description that is passed on to score objects

Returns:

Return type *SentenceLevelEvalScore*

evaluate_multi_ref (*ref*, *hyp*, *desc=None*)
Calculate the quality of output given multiple references.

Parameters

- **ref** (Sequence[Sequence[+T_co]]) – list of tuples of reference sentences (a sentence is a list of tokens)
- **hyp** (Sequence[+T_co]) – list of hypothesis sentences (a sentence is a list of tokens)
- **desc** (Optional[Any]) – optional description that is passed on to score objects

Return type *EvalScore*

class xnmt.eval.metrics.**FastBLEUEvaluator** (*ngram=4*, *smooth=1*)
Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

Class for computing BLEU scores using a fast Cython implementation.

Does not support multiple references. BLEU scores are computed according to K Papineni et al "BLEU: a method for automatic evaluation of machine translation"

Parameters

- **ngram** (Integral) – consider ngrams up to this order (usually 4)
- **smooth** (Real) –

class xnmt.eval.metrics.**BLEUEvaluator** (*ngram=4*)
Bases: *xnmt.eval.metrics.Evaluator*, *xnmt.persistence.Serializable*

Compute BLEU scores against one or several references.

BLEU scores are computed according to K Papineni et al "BLEU: a method for automatic evaluation of machine translation"

Parameters **ngram** (Integral) – consider ngrams up to this order (usually 4)

evaluate (*ref*, *hyp*, *desc=None*)

Parameters

- **ref** (Sequence[Sequence[str]]) – reference sentences (single-reference case: sentence is list of strings;
- **hyp** (Sequence[Sequence[str]]) – list of hypothesis sentences (a sentence is a list of tokens)
- **desc** (Optional[Any]) – description to pass on to returned score

Return type *BLEUScore*

Returns Score, including intermediate results such as ngram ratio, sentence length, brevity penalty

evaluate_multi_ref(*ref*, *hyp*, *desc=None*)

Parameters

- **ref** (Sequence[Sequence[Sequence[str]]]) – list of tuples of reference sentences (a sentence is a list of tokens)
- **hyp** (Sequence[Sequence[str]]) – list of hypothesis sentences (a sentence is a list of tokens)
- **desc** (Optional[Any]) – optional description that is passed on to score objects

Return type *BLEUScore*

Returns Score, including intermediate results such as ngram ratio, sentence length, brevity penalty

class xnmt.eval.metrics.**GLEUEvaluator**(*min_length=1*, *max_length=4*, *write_sentence_scores=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

Class for computing GLEU (Google BLEU) Scores.

GLEU scores are described in https://arxiv.org/pdf/1609.08144v2.pdf as follows:

> "The BLEU score has some undesirable properties when used for single sentences, as it was designed to be a corpus measure. We therefore use a slightly different score for our RL experiments which we call the 'GLEU score'. For the GLEU score, we record all sub-sequences of 1, 2, 3 or 4 tokens in output and target sequence (n-grams). We then compute a recall, which is the ratio of the number of matching n-grams to the number of total n-grams in the target (ground truth) sequence, and a precision, which is the ratio of the number of matching n-grams to the number of total n-grams in the generated output sequence. Then GLEU score is simply the minimum of recall and precision. This GLEU score's range is always between 0 (no matches) and 1 (all match) and it is symmetrical when switching output and target. According to our experiments, GLEU score correlates quite well with the BLEU metric on a corpus level but does not have its drawbacks for our per sentence reward objective."

Parameters

- **min_length** (Integral) – minimum n-gram order to consider
- **max_length** (Integral) – maximum n-gram order to consider
- **write_sentence_scores** (Optional[str]) – path of file to write sentence-level scores to (in YAML format)

evaluate_one_sent(*ref*, *hyp*)

Parameters

- **ref** (Sequence[str]) – reference sentence (a sent is a list of tokens)
- **hyp** (Sequence[str]) – hypothesis sentence (a sent is a list of tokens)

Returns GLEU score object

class xnmt.eval.metrics.**WEREvaluator**(*case_sensitive=False*, *write_sentence_scores=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

A class to evaluate the quality of output in terms of word error rate.

Parameters

- **case_sensitive** (bool) – whether scoring should be case-sensitive
- **write_sentence_scores** (Optional[str]) – path of file to write sentence-level scores to (in YAML format)

class xnmt.eval.metrics.**CEREvaluator**(*case_sensitive=False*, *write_sentence_scores=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

A class to evaluate the quality of output in terms of character error rate.

Parameters

- **case_sensitive** (bool) – whether scoring should be case-sensitive
- **write_sentence_scores** (Optional[str]) – path of file to write sentence-level scores to (in YAML format)

evaluate_one_sent(*ref*, *hyp*)

Calculate the quality of output sentence given a reference.

Parameters

- **ref** (Sequence[str]) – list of reference words
- **hyp** (Sequence[str]) – list of decoded words

Returns (ins+del+sub) / (ref_len)

Return type character error rate

class xnmt.eval.metrics.**ExternalEvaluator**(*path=None*, *higher_better=True*)

Bases: *xnmt.eval.metrics.Evaluator*, *xnmt.persistence.Serializable*

A class to evaluate the quality of the output according to an external evaluation script.

Does not support multiple references. The external script should only print a number representing the calculated score.

Parameters

- **path** (Optional[str]) – path to external command line tool.
- **higher_better** (bool) – whether to interpret higher scores as favorable.

evaluate(*ref*, *hyp*, *desc=None*)

Calculate the quality of output according to an external script.

Parameters

- **ref** – (ignored)
- **hyp** – (ignored)
- **desc** – description to pass on to returned score

Returns external eval script score

class xnmt.eval.metrics.**RecallEvaluator**(*nbest=5*, *write_sentence_scores=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

Compute recall by counting true positives.

Parameters

- **nbest** (Integral) – compute recall within n-best of specified n
- **write_sentence_scores** (Optional[str]) – path of file to write sentence-level scores to (in YAML format)

evaluate(*ref*, *hyp*, *desc=None*)

Calculate the quality of output given a reference.

Parameters

- **ref** – list of reference sents (a sentence is a list of tokens)
- **hyp** – list of hypothesis sents (a sentence is a list of tokens)
- **desc** – optional description that is passed on to score objects

Returns:

class xnmt.eval.metrics.**SequenceAccuracyEvaluator**(*case_sensitive=False*, *write_sentence_scores=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

A class to evaluate the quality of output in terms of sequence accuracy.

Parameters

- **case_sensitive** – whether differences in capitalization are to be considered
- **write_sentence_scores** (Optional[str]) – path of file to write sentence-level scores to (in YAML format)

evaluate_one_sent(*ref*, *hyp*)

Calculate the accuracy of output given a references.

Parameters

- **ref** (Sequence[str]) – list of list of reference words
- **hyp** (Sequence[str]) – list of list of decoded words

Return: formatted string

class xnmt.eval.metrics.**FMeasureEvaluator**(*pos_token='1'*, *write_sentence_scores=None*)

Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

A class to evaluate the quality of output in terms of classification F-score.

Parameters

- **pos_token** (str) – token for the 'positive' class
- **write_sentence_scores** (Optional[str]) – path of file to write sentence-level scores to (in YAML format)

evaluate_one_sent(*ref*, *hyp*)

Calculate the accuracy of output given a references.

Parameters

- **ref** (Sequence[str]) – list of list of reference words
- **hyp** (Sequence[str]) – list of list of decoded words

Return: formatted string

class xnmt.eval.metrics.**SegmentationFMeasureEvaluator**(*write_sentence_scores=None*)
Bases: *xnmt.eval.metrics.SentenceLevelEvaluator*, *xnmt.persistence.Serializable*

5.8 Data

5.8.1 Sentence

class xnmt.sent.**Sentence**(*idx=None*, *score=None*)
Bases: object

A template class to represent a single data example of any type, used for both model input and output.

Parameters

- **idx** (Optional[int]) – running sentence number (0-based; unique among sentences loaded from the same file, but not across files)
- **score** (Optional[Real]) – a score given to this sentence by a model

sent_len()
Return length of input, included padded tokens.

Returns: length

Return type int

len_unpadded()
Return length of input prior to applying any padding.

Returns: unpadded length

Return type int

create_padded_sent(*pad_len*)
Return a new, padded version of the sentence (or self if pad_len is zero).

Parameters **pad_len** (Integral) – number of tokens to append

Return type *Sentence*

Returns padded sentence

create_truncated_sent(*trunc_len*)
Create a new, right-truncated version of the sentence (or self if trunc_len is zero).

Parameters **trunc_len** (Integral) – number of tokens to truncate

Return type *Sentence*

Returns truncated sentence

get_unpadded_sent()
Return the unpadded sentence.

If self is unpadded, return self, if not return reference to original unpadded sentence if possible, otherwise create a new sentence.

Return type *Sentence*

class xnmt.sent.**ReadableSentence**(*idx*, *score=None*, *output_procs=[]*)
Bases: *xnmt.sent.Sentence*

A base class for sentences based on readable strings.

Parameters

- **idx** (Integral) – running sentence number (0-based; unique among sentences loaded from the same file, but not across files)
- **score** (Optional[Real]) – a score given to this sentence by a model
- **output_procs** (Union[OutputProcessor, Sequence[OutputProcessor]]) – output processors to be applied when calling sent_str()

str_tokens(***kwargs*)
Return list of readable string tokens.

Parameters ****kwargs** – should accept arbitrary keyword args

Returns: list of tokens.

Return type List[str]

sent_str(*custom_output_procs=None*, ***kwargs*)
Return a single string containing the readable version of the sentence.

Parameters

- **custom_output_procs** – if not None, overwrite the sentence's default output processors
- ****kwargs** – should accept arbitrary keyword args

Returns: readable string

Return type str

class xnmt.sent.**ScalarSentence**(*value*, *idx=None*, *vocab=None*, *score=None*)
Bases: *xnmt.sent.ReadableSentence*

A sentence represented by a single integer value, optionally interpreted via a vocab.

This is useful for classification-style problems.

Parameters

- **value** (Integral) – scalar value
- **idx** (Optional[Integral]) – running sentence number (0-based; unique among sentences loaded from the same file, but not across files)
- **vocab** (Optional[*Vocab*]) – optional vocab to give different scalar values a string representation.
- **score** (Optional[Real]) – a score given to this sentence by a model

sent_len()
Return length of input, included padded tokens.

Returns: length

Return type int

len_unpadded()
Return length of input prior to applying any padding.

Returns: unpadded length

Return type int

create_padded_sent(*pad_len*)
Return a new, padded version of the sentence (or self if pad_len is zero).

Parameters **pad_len** (Integral) – number of tokens to append

Return type *ScalarSentence*

Returns padded sentence

create_truncated_sent(*trunc_len*)
Create a new, right-truncated version of the sentence (or self if trunc_len is zero).

Parameters **trunc_len** (Integral) – number of tokens to truncate

Return type *ScalarSentence*

Returns truncated sentence

get_unpadded_sent()
Return the unpadded sentence.

If self is unpadded, return self, if not return reference to original unpadded sentence if possible, otherwise create a new sentence.

str_tokens(***kwargs*)
Return list of readable string tokens.

Parameters ****kwargs** – should accept arbitrary keyword args

Returns: list of tokens.

Return type List[str]

class xnmt.sent.**CompoundSentence**(*sents*)
Bases: *xnmt.sent.Sentence*

A compound sentence contains several sentence objects that present different 'views' on the same data examples.

Parameters **sents** (Sequence[*Sentence*]) – a list of sentences

sent_len()
Return length of input, included padded tokens.

Returns: length

Return type int

len_unpadded()
Return length of input prior to applying any padding.

Returns: unpadded length

Return type int

create_padded_sent(*pad_len*)
Return a new, padded version of the sentence (or self if pad_len is zero).

Parameters **pad_len** – number of tokens to append

Returns padded sentence

create_truncated_sent(*trunc_len*)
Create a new, right-truncated version of the sentence (or self if trunc_len is zero).

Parameters **trunc_len** – number of tokens to truncate

Returns truncated sentence

get_unpadded_sent()
Return the unpadded sentence.

If self is unpadded, return self, if not return reference to original unpadded sentence if possible, otherwise create a new sentence.

class xnmt.sent.**SimpleSentence**(*words*, *idx=None*, *vocab=None*, *score=None*, *output_procs=[]*, *pad_token=1*, *unpadded_sent=None*)
Bases: *xnmt.sent.ReadableSentence*

A simple sentence, represented as a list of tokens

Parameters

- **words** (Sequence[Integral]) – list of integer word ids
- **idx** (Optional[Integral]) – running sentence number (0-based; unique among sentences loaded from the same file, but not across files)
- **vocab** (Optional[*Vocab*]) – optionally vocab mapping word ids to strings
- **score** (Optional[Real]) – a score given to this sentence by a model
- **output_procs** (Union[OutputProcessor, Sequence[OutputProcessor]]) – output processors to be applied when calling sent_str()
- **pad_token** (Integral) – special token used for padding
- **unpadded_sent** (Optional[*SimpleSentence*]) – reference to original, unpadded sentence if available

sent_len()
Return length of input, included padded tokens.

Returns: length

create_padded_sent(*pad_len*)
Return a new, padded version of the sentence (or self if pad_len is zero).

Parameters **pad_len** (Integral) – number of tokens to append

Return type *SimpleSentence*

Returns padded sentence

create_truncated_sent(*trunc_len*)
Create a new, right-truncated version of the sentence (or self if trunc_len is zero).

Parameters **trunc_len** (Integral) – number of tokens to truncate

Return type *SimpleSentence*

Returns truncated sentence

get_unpadded_sent()
Return the unpadded sentence.

If self is unpadded, return self, if not return reference to original unpadded sentence if possible, otherwise create a new sentence.

str_tokens (*exclude_ss_es=True*, *exclude_unk=False*, *exclude_padded=True*, ***kwargs*)
Return list of readable string tokens.

Parameters ****kwargs** – should accept arbitrary keyword args

Returns: list of tokens.

Return type List[str]

class xnmt.sent.**SegmentedSentence** (*segment=[]*, ***kwargs*)
Bases: *xnmt.sent.SimpleSentence*

class xnmt.sent.**ArraySentence** (*nparr*, *idx=None*, *padded_len=0*, *score=None*, *unpadded_sent=None*)
Bases: *xnmt.sent.Sentence*

A sentence based on a numpy array containing a continuous-space vector for each token.

Parameters

- **idx** (Optional[Integral]) – running sentence number (0-based; unique among sentences loaded from the same file, but not across files)
- **nparr** (ndarray) – numpy array of dimension num_tokens x token_size
- **padded_len** (Integral) – how many padded tokens are contained in the given nparr
- **score** (Optional[Real]) – a score given to this sentence by a model

sent_len ()
Return length of input, included padded tokens.

Returns: length

len_unpadded ()
Return length of input prior to applying any padding.

Returns: unpadded length

create_padded_sent (*pad_len*)
Return a new, padded version of the sentence (or self if pad_len is zero).

Parameters **pad_len** (Integral) – number of tokens to append

Return type *ArraySentence*

Returns padded sentence

create_truncated_sent (*trunc_len*)
Create a new, right-truncated version of the sentence (or self if trunc_len is zero).

Parameters **trunc_len** (Integral) – number of tokens to truncate

Return type *ArraySentence*

Returns truncated sentence

get_unpadded_sent ()
Return the unpadded sentence.

If self is unpadded, return self, if not return reference to original unpadded sentence if possible, otherwise create a new sentence.

class xnmt.sent.**NbestSentence** (*base_sent*, *nbest_id*, *print_score=False*)
Bases: *xnmt.sent.SimpleSentence*

Output in the context of an nbest list.

Parameters

- **base_sent** (*SimpleSentence*) – The base sent object
- **nbest_id** (Integral) – The sentence id in the nbest list
- **print_score** (bool) – If True, print nbest_id, score, content separated by `|||`. If False, drop the score.

sent_str(*custom_output_procs=None*, ***kwargs*)

Return a single string containing the readable version of the sentence.

Parameters

- **custom_output_procs** – if not None, overwrite the sentence's default output processors
- ****kwargs** – should accept arbitrary keyword args

Returns: readable string

Return type str

class xnmt.sent.**GraphSentence**(*idx*, *graph*, *vocab*, *num_padded=0*, *unpadded_sent=None*)

Bases: *xnmt.sent.ReadableSentence*

A graph structure.

This is a wrapper for a graph datastructure.

Parameters

- **idx** (Optional[Integral]) – running sentence number (0-based; unique among sentences loaded from the same file, but not across files)
- **graph** (HyperGraph) – hypergraph containing graphs
- **vocab** (*Vocab*) – vocabulary for word IDs
- **num_padded** (Integral) – denoting that this many words are padded (without adding any physical nodes)
- **unpadded_sent** (Optional[*GraphSentence*]) – reference to original, unpadded sentence if available

sent_len()

Return number of nodes in the graph, including padded words.

Return type int

Returns Number of nodes in graph.

len_unpadded()

Return number of nodes in the graph, without counting padded words.

Return type int

Returns Number of nodes in graph.

create_padded_sent(*pad_len*)

Return padded graph.

Parameters **pad_len** (Integral) – Number of tokens to pad.

Return type *GraphSentence*

Returns New padded graph, or self if pad_len==0.

create_truncated_sent (*trunc_len*)

Return self, as truncation is not supported.

Parameters **trunc_len** (Integral) – Number of tokens to truncate, must be 0.

Return type *GraphSentence*

Returns self.

get_unpadded_sent ()

Return the unpadded sentence.

If self is unpadded, return self, if not return reference to original unpadded sentence if possible, otherwise create a new sentence.

Return type *GraphSentence*

reversed ()

Create a graph with reversed direction.

The new graph will have graph nodes in reversed order and switched successors/predecessors. It will have the same number of padded nodes (again at the end of the nodes!).

Return type *GraphSentence*

Returns Reversed graph.

str_tokens (***kwargs*)

Return list of readable string tokens.

Parameters ****kwargs** – ignored

Returns: list of tokens of linearized graph.

Return type List[str]

sent_str (*custom_output_procs=None*, ***kwargs*)

Return a single string containing the readable version of the sentence.

Parameters

- **custom_output_procs** – ignored
- ****kwargs** – ignored

Returns: readable string

Return type str

class xnmt.sent.**LatticeNode** (*node_id*, *value*, *fwd_log_prob=0*, *marginal_log_prob=0*, *bwd_log_prob=0*)

Bases: xnmt.graph.HyperNode

A lattice node.

Parameters

- **node_id** (int) – Unique identifier for node
- **value** (Integral) – Word id assigned to this node.
- **fwd_log_prob** (Optional[Real]) – Lattice log probability normalized in forward-direction (successors sum to 1)
- **marginal_log_prob** (Optional[Real]) – Lattice log probability globally normalized

- **bwd_log_prob** (Optional[Real]) – Lattice log probability normalized in backward-direction (predecessors sum to 1)

class xnmt.sent.**SyntaxTreeNode**(*node_id*, *value*, *head*, *node_type=<Type.NONE: 0>*)
Bases: xnmt.graph.HyperNode

class Type
Bases: enum.Enum

An enumeration.

class xnmt.sent.**RNNGSequenceSentence**(*idx*, *graph*, *surface_vocab*, *nt_vocab*, *all_surfaces=False*, *num_padded=0*, *unpadded_sent=None*)
Bases: *xnmt.sent.ReadableSentence*

sent_len()
Return length of input, included padded tokens.

Returns: length

Return type int

len_unpadded()
Return length of input prior to applying any padding.

Returns: unpadded length

Return type int

create_padded_sent(*pad_len*)
Return a new, padded version of the sentence (or self if pad_len is zero).

Parameters pad_len (Integral) – number of tokens to append

Return type *ScalarSentence*

Returns padded sentence

create_truncated_sent(*trunc_len*)
Create a new, right-truncated version of the sentence (or self if trunc_len is zero).

Parameters trunc_len (Integral) – number of tokens to truncate

Return type *ScalarSentence*

Returns truncated sentence

get_unpadded_sent()
Return the unpadded sentence.

If self is unpadded, return self, if not return reference to original unpadded sentence if possible, otherwise create a new sentence.

str_tokens(***kwargs*)
Return list of readable string tokens.

Parameters **kwargs – should accept arbitrary keyword args

Returns: list of tokens.

Return type List[str]

sent_str()
Return a single string containing the readable version of the sentence.

Parameters

- **custom_output_procs** – if not None, overwrite the sentence's default output processors
- ****kwargs** – should accept arbitrary keyword args

Returns: readable string

5.8.2 InputReader

class xnmt.input_readers.**InputReader**
Bases: `object`

A base class to read in a file and turn it into an input

read_sents (*filename*, *filter_ids=None*)
Read sentences and return an iterator.

Parameters

- **filename** (`str`) – data file
- **filter_ids** (`Optional[Sequence[Integral]]`) – only read sentences with these ids (0-indexed)

Returns: iterator over sentences from filename

Return type `Iterator[Sentence]`

count_sents (*filename*)
Count the number of sentences in a data file.

Parameters **filename** (`str`) – data file

Returns: number of sentences in the data file

Return type `int`

needs_reload ()
Overwrite this method if data needs to be reload for each epoch

Return type `bool`

class xnmt.input_readers.**BaseTextReader**
Bases: *xnmt.input_readers.InputReader*

read_sent (*line*, *idx*)
Convert a raw text line into an input object.

Parameters

- **line** (`str`) – a single input string
- **idx** (`Integral`) – sentence number

Returns: a SentenceInput object for the input sentence

Return type *Sentence*

iterate_filtered (*filename*, *filter_ids=None*)

Parameters

- **filename** (`str`) – data file (text file)
- **filter_ids** (`Optional[Sequence[Integral]]`) –

Returns: iterator over lines as strings (useful for subclasses to implement read_sents)

Return type `Iterator`[+T_co]

class `xnmt.input_readers.`**`PlainTextReader`**(*vocab=None*, *read_sent_len=False*, *output_proc=[]*)

Bases: *xnmt.input_readers.BaseTextReader*, *xnmt.persistence.Serializable*

Handles the typical case of reading plain text files, with one sent per line.

Parameters

- **vocab** (`Optional`[*Vocab*]) – Vocabulary to convert string tokens to integer ids. If not given, plain text will be assumed to contain space-separated integer ids.
- **read_sent_len** (`bool`) – if set, read the length of each sentence instead of the sentence itself. EOS is not counted.
- **output_proc** (`Sequence`[`OutputProcessor`]) – output processors to revert the created sentences back to a readable string

read_sent(*line*, *idx*)

Convert a raw text line into an input object.

Parameters

- **line** (`str`) – a single input string
- **idx** (`Integral`) – sentence number

Returns: a SentenceInput object for the input sentence

Return type *Sentence*

class `xnmt.input_readers.`**`CompoundReader`**(*readers*, *vocab=None*)

Bases: *xnmt.input_readers.InputReader*, *xnmt.persistence.Serializable*

A compound reader reads inputs using several input readers at the same time.

The resulting inputs will be of type `sent.CompoundSentence`, which holds the results from the different readers as a tuple. Inputs can be read from different locations (if input file name is a sequence of filenames) or all from the same location (if it is a string). The latter can be used to read the same inputs using several input different readers which might capture different aspects of the input data.

Parameters

- **readers** (`Sequence`[*InputReader*]) – list of input readers to use
- **vocab** (`Optional`[*Vocab*]) – not used by this reader, but some parent components may require access to the vocab.

read_sents(*filename*, *filter_ids=None*)

Read sentences and return an iterator.

Parameters

- **filename** (`Union`[`str`, `Sequence`[`str`]]) – data file
- **filter_ids** (`Optional`[`Sequence`[`Integral`]]) – only read sentences with these ids (0-indexed)

Returns: iterator over sentences from filename

Return type `Iterator`[*Sentence*]

count_sents(*filename*)

Count the number of sentences in a data file.

Parameters **filename** (`str`) – data file

Returns: number of sentences in the data file

Return type int

needs_reload()

Overwrite this method if data needs to be reload for each epoch

Return type bool

class xnmt.input_readers.**SentencePieceTextReader**(*model_file, sample_train=False, l=-1, alpha=0.1, vocab=None, output_proc=[<class 'xnmt.output.JoinPieceTextOutputProcessor'>]*)

Bases: *xnmt.input_readers.BaseTextReader*, *xnmt.persistence.Serializable*

Read in text and segment it with sentencepiece. Optionally perform sampling for subword regularization, only at training time. https://arxiv.org/pdf/1804.10959.pdf

read_sent(*line*, *idx*)

Convert a raw text line into an input object.

Parameters

- **line** (str) – a single input string
- **idx** (Integral) – sentence number

Returns: a SentenceInput object for the input sentence

Return type *SimpleSentence*

class xnmt.input_readers.**RamlTextReader**(*tau=1.0*, *vocab=None*, *output_proc=[]*)

Bases: *xnmt.input_readers.BaseTextReader*, *xnmt.persistence.Serializable*

Handles the RAML sampling, can be used on the target side, or on both the source and target side. Randomly replaces words according to Hamming Distance. https://arxiv.org/pdf/1808.07512.pdf https://arxiv.org/pdf/1609.00150.pdf

read_sent(*line*, *idx*)

Convert a raw text line into an input object.

Parameters

- **line** (str) – a single input string
- **idx** (Integral) – sentence number

Returns: a SentenceInput object for the input sentence

Return type *SimpleSentence*

needs_reload()

Overwrite this method if data needs to be reload for each epoch

Return type bool

class xnmt.input_readers.**CharFromWordTextReader**(*vocab=None*, *read_sent_len=False*, *output_proc=[]*)

Bases: *xnmt.input_readers.PlainTextReader*, *xnmt.persistence.Serializable*

Read in word based corpus and turned that into SegmentedSentence. SegmentedSentece's words are characters, but it contains the information of the segmentation.

x = SegmentedSentence("i code today") (TRUE) x.words == ["i", "c", "o", "d", "e", "t", "o", "d", "a", "y"] (TRUE) x.segment == [0, 4, 9]

It means that the segmentation (end of words) happen in the 0th, 4th and 9th position of the char sequence.

read_sent (*line*, *idx*)
Convert a raw text line into an input object.

Parameters

- **line** (str) – a single input string
- **idx** (Integral) – sentence number

Returns: a SentenceInput object for the input sentence

Return type *SegmentedSentence*

class xnmt.input_readers.**H5Reader** (*transpose=False*, *feat_from=None*, *feat_to=None*, *feat_skip=None*, *timestep_skip=None*, *timestep_truncate=None*)
Bases: *xnmt.input_readers.InputReader*, *xnmt.persistence.Serializable*

Handles the case where sents are sequences of continuous-space vectors.

The input is a ".h5" file, which can be created for example using xnmt.preproc.MelFiltExtractor

The data items are assumed to be labeled with integers 0, 1, .. (converted to strings).

Each data item will be a 2D matrix representing a sequence of vectors. They can be in either order, depending on the value of the "transpose" variable: * sents[sent_id][feat_ind,timestep] if transpose=False * sents[sent_id][timestep,feat_ind] if transpose=True

Parameters

- **transpose** (bool) – whether inputs are transposed or not.
- **feat_from** (Optional[Integral]) – use feature dimensions in a range, starting at this index (inclusive)
- **feat_to** (Optional[Integral]) – use feature dimensions in a range, ending at this index (exclusive)
- **feat_skip** (Optional[Integral]) – stride over features
- **timestep_skip** (Optional[Integral]) – stride over timesteps
- **timestep_truncate** (Optional[Integral]) – cut off timesteps if sequence is longer than specified value

read_sents (*filename*, *filter_ids=None*)
Read sentences and return an iterator.

Parameters

- **filename** (str) – data file
- **filter_ids** (Optional[Sequence[Integral]]) – only read sentences with these ids (0-indexed)

Returns: iterator over sentences from filename

Return type Iterator[*ArraySentence*]

count_sents (*filename*)
Count the number of sentences in a data file.

Parameters **filename** (str) – data file

Returns: number of sentences in the data file

Return type Integral

class xnmt.input_readers.**NpzReader**(*transpose=False*, *feat_from=None*, *feat_to=None*, *feat_skip=None*, *timestep_skip=None*, *timestep_truncate=None*)

Bases: *xnmt.input_readers.InputReader*, *xnmt.persistence.Serializable*

Handles the case where sents are sequences of continuous-space vectors.

The input is a ".npz" file, which consists of multiply ".npy" files, each corresponding to a single sequence of continuous features. This can be created in two ways: * Use the builtin function numpy.savez_compressed() * Create a bunch of .npy files, and run "zip" on them to zip them into an archive.

The file names should be named XXX_0, XXX_1, etc., where the final number after the underbar indicates the order of the sequence in the corpus. This is done automatically by numpy.savez_compressed(), in which case the names will be arr_0, arr_1, etc.

Each numpy file will be a 2D matrix representing a sequence of vectors. They can be in either order, depending on the value of the "transpose" variable. * sents[sent_id][feat_ind,timestep] if transpose=False * sents[sent_id][timestep,feat_ind] if transpose=True

Parameters

- **transpose** (`bool`) – whether inputs are transposed or not.
- **feat_from** (`Optional[Integral]`) – use feature dimensions in a range, starting at this index (inclusive)
- **feat_to** (`Optional[Integral]`) – use feature dimensions in a range, ending at this index (exclusive)
- **feat_skip** (`Optional[Integral]`) – stride over features
- **timestep_skip** (`Optional[Integral]`) – stride over timesteps
- **timestep_truncate** (`Optional[Integral]`) – cut off timesteps if sequence is longer than specified value

read_sents(*filename*, *filter_ids=None*)

Read sentences and return an iterator.

Parameters

- **filename** (`str`) – data file
- **filter_ids** (`Optional[Sequence[Integral]]`) – only read sentences with these ids (0-indexed)

Returns: iterator over sentences from filename

Return type `None`

count_sents(*filename*)

Count the number of sentences in a data file.

Parameters **filename** (`str`) – data file

Returns: number of sentences in the data file

Return type `Integral`

class xnmt.input_readers.**IDReader**

Bases: *xnmt.input_readers.BaseTextReader*, *xnmt.persistence.Serializable*

Handles the case where we need to read in a single ID (like retrieval problems).

Files must be text files containing a single integer per line.

read_sent (*line*, *idx*)
Convert a raw text line into an input object.

Parameters

- **line** (str) – a single input string
- **idx** (Integral) – sentence number

Returns: a SentenceInput object for the input sentence

Return type *ScalarSentence*

read_sents (*filename*, *filter_ids=None*)
Read sentences and return an iterator.

Parameters

- **filename** (str) – data file
- **filter_ids** (Optional[Sequence[Integral]]) – only read sentences with these ids (0-indexed)

Returns: iterator over sentences from filename

Return type list

class xnmt.input_readers.**CoNLLToRNNGActionsReader** (*surface_vocab*, *nt_vocab*)
Bases: *xnmt.input_readers.BaseTextReader*, *xnmt.persistence.Serializable*

Handles the reading of CoNLL File Format:

ID FORM LEMMA POS FEAT HEAD DEPREL

A single line represents a single edge of dependency parse tree.

read_sents (*filename*, *filter_ids=None*)
Read sentences and return an iterator.

Parameters

- **filename** (str) – data file
- **filter_ids** (Optional[Sequence[Integral]]) – only read sentences with these ids (0-indexed)

Returns: iterator over sentences from filename

class xnmt.input_readers.**LatticeReader** (*vocab*, *text_input=False*, *flatten=False*)
Bases: *xnmt.input_readers.BaseTextReader*, *xnmt.persistence.Serializable*

Reads lattices from a text file.

The expected lattice file format is as follows: * 1 line per lattice * lines are serialized python lists / tuples * 2 lists per lattice: - list of nodes, with every node a 4-tuple: (lexicon_entry, fwd_log_prob, marginal_log_prob, bwd_log_prob) - list of arcs, each arc a tuple: (node_id_start, node_id_end) - node_id references the nodes and is 0-indexed - node_id_start < node_id_end * All paths must share a common start and end node, i.e. <s> and </s> need to be contained in the lattice

A simple example lattice: [('<s>', 0.0, 0.0, 0.0), ('buenas', 0, 0.0, 0.0), ('tardes', 0, 0.0, 0.0), ('</s>', 0.0, 0.0, 0.0)],[(0, 1), (1, 2), (2, 3)]

Parameters

- **vocab** (*Vocab*) – Vocabulary to convert string tokens to integer ids. If not given, plain text will be assumed to contain space-separated integer ids.

- **text_input** (bool) – If True, assume a standard text file as input and convert it to a flat lattice.
- **flatten** – If True, convert to a flat lattice, with all probabilities set to 1.

read_sent (*line*, *idx*)
Convert a raw text line into an input object.

Parameters

- **line** – a single input string
- **idx** – sentence number

Returns: a SentenceInput object for the input sentence

xnmt.input_readers.**read_parallel_corpus** (*src_reader*, *trg_reader*, *src_file*, *trg_file*, *batcher=None*, *sample_sents=None*, *max_num_sents=None*, *max_src_len=None*, *max_trg_len=None*)

A utility function to read a parallel corpus.

Parameters

- **src_reader** (*InputReader*) –
- **trg_reader** (*InputReader*) –
- **src_file** (str) –
- **trg_file** (str) –
- **batcher** (Optional[*Batcher*]) –
- **sample_sents** (Optional[Integral]) – if not None, denote the number of sents that should be randomly chosen from all available sents.
- **max_num_sents** (Optional[Integral]) – if not None, read only the first this many sents
- **max_src_len** (Optional[Integral]) – skip pair if src side is too long
- **max_trg_len** (Optional[Integral]) – skip pair if trg side is too long

Return type tuple

Returns A tuple of (src_data, trg_data, src_batches, trg_batches) where `*_batches = *_data` if `batcher=None`

5.8.3 Vocab

class xnmt.vocabs.**Vocab** (*i2w=None*, *vocab_file=None*, *sentencepiece_vocab=False*)
Bases: *xnmt.persistence.Serializable*

An open vocabulary that converts between strings and integer ids.

The open vocabulary is realized via a special unknown-word token that is used whenever a word is not inside the list of known tokens. This class is immutable, i.e. its contents are not to change after the vocab has been initialized.

For initialization, i2w or vocab_file must be specified, but not both.

Parameters

- **i2w** (Optional[Sequence[str]]) – complete list of known words, including <s> and </s>.
- **vocab_file** (Optional[str]) – file containing one word per line, and not containing <s>, </s>, <unk>
- **sentencepiece_vocab** (bool) – Set to True if vocab_file is the output of the sentencepiece tokenizer. Defaults to False.

static i2w_from_vocab_file(*vocab_file*, *sentencepiece_vocab=False*)
Load the vocabulary from a file.

If sentencepiece_vocab is set to True, this will accept a sentencepiece vocabulary file

Parameters

- **vocab_file** (str) – file containing one word per line, and not containing <s>, </s>, <unk>
- **sentencepiece_vocab** (*bool*) – Set to True if vocab_file is the output of the sentencepiece tokenizer. Defaults to False.

Return type List[str]

is_compatible(*other*)
Check if this vocab produces the same conversions as another one.

Return type bool

5.8.4 Batcher

class xnmt.batchers.**Batch**
Bases: abc.ABC

An abstract base class for minibatches of things.

class xnmt.batchers.**ListBatch**(*batch_elements*, *mask=None*)
Bases: list, *xnmt.batchers.Batch*

A class containing a minibatch of things.

This class behaves like a Python list, but adds semantics that the contents form a (mini)batch of things. An optional mask can be specified to indicate padded parts of the inputs. Should be treated as an immutable object.

Parameters

- **batch_elements** (list) – list of things
- **mask** (Optional[*Mask*]) – optional mask when batch contains items of unequal size

class xnmt.batchers.**CompoundBatch**(**batch_elements*)
Bases: *xnmt.batchers.Batch*

A compound batch contains several parallel batches.

Parameters ***batch_elements** – one or several batches

class xnmt.batchers.**Mask**(*np_arr*)
Bases: object

An immutable mask specifies padded parts in a sequence or batch of sequences.

Masks are represented as numpy array of dimensions batchsize x seq_len, with parts belonging to the sequence set to 0, and parts that should be masked set to 1

Parameters **np_arr** (ndarray) – numpy array

cmult_by_timestep_expr (*expr*, *timestep*, *inverse=False*)

Parameters

- **expr** (Expression) – a dynet expression corresponding to one timestep
- **timestep** (Integral) – index of current timestep
- **inverse** (bool) – True will keep the unmasked parts, False will zero out the unmasked parts

Return type Expression

class xnmt.batchers.**Batcher** (*batch_size*, *granularity='sent'*, *pad_src_to_multiple=1*, *sort_within_by_trg_len=True*)

Bases: object

A template class to convert a list of sentences to several batches of sentences.

Parameters

- **batch_size** (Integral) – batch size
- **granularity** (str) – 'sent' or 'word'
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.
- **sort_within_by_trg_len** (bool) – whether to sort by reverse trg len inside a batch

is_random ()

Return type bool

Returns True if there is some randomness in the batching process, False otherwise.

create_single_batch (*src_sents*, *trg_sents=None*, *sort_by_trg_len=False*)

Create a single batch, either source-only or source-and-target.

Parameters

- **src_sents** (Sequence[*Sentence*]) – list of source-side inputs
- **trg_sents** (Optional[Sequence[*Sentence*]]) – optional list of target-side inputs
- **sort_by_trg_len** (bool) – if True (and targets are specified), sort source- and target batches by target length

Return type Union[*Batch*, Tuple[*Batch*]]

Returns a tuple of batches if targets were given, otherwise a single batch

pack (*src*, *trg*)

Create a list of src/trg batches based on provided src/trg inputs.

Parameters

- **src** (Sequence[*Sentence*]) – list of src-side inputs
- **trg** (Sequence[*Sentence*]) – list of trg-side inputs

Return type Tuple[Sequence[*Batch*], Sequence[*Batch*]]

Returns tuple of lists of src and trg batches

class xnmt.batchers.**InOrderBatcher** (*batch_size=1*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.Batcher*, *xnmt.persistence.Serializable*

A class to create batches in order of the original corpus, both across and within batches.

Parameters

- **batch_size** (Integral) – batch size
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

pack (*src*, *trg*)

Pack batches. Unlike other batches, the trg sentences are optional.

Parameters

- **src** (Sequence[*Sentence*]) – list of src-side inputs
- **trg** (Optional[Sequence[*Sentence*]]) – optional list of trg-side inputs

Return type Tuple[Sequence[*Batch*], Sequence[*Batch*]]

Returns src batches if trg was not given; tuple of src batches and trg batches if trg was given

class xnmt.batchers.**ShuffleBatcher** (*batch_size*, *granularity='sent'*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.Batcher*

A template class to create batches through randomly shuffling without sorting.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **batch_size** (Integral) – batch size
- **granularity** (str) – 'sent' or 'word'
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

pack (*src*, *trg*)

Create a list of src/trg batches based on provided src/trg inputs.

Parameters

- **src** (Sequence[*Sentence*]) – list of src-side inputs
- **trg** (Optional[Sequence[*Sentence*]]) – list of trg-side inputs

Return type Tuple[Sequence[*Batch*], Sequence[*Batch*]]

Returns tuple of lists of src and trg batches

is_random ()

Returns: True if there is some randomness in the batching process, False otherwise.

Return type bool

class xnmt.batchers.**SortBatcher** (*batch_size*, *granularity='sent'*, *sort_key=<function SortBatcher.<lambda>>*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.Batcher*

A template class to create batches through bucketing sentence length.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **batch_size** (Integral) – batch size
- **granularity** (str) – 'sent' or 'word'
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

pack (*src*, *trg*)
Create a list of src/trg batches based on provided src/trg inputs.

Parameters

- **src** (Sequence[*Sentence*]) – list of src-side inputs
- **trg** (Optional[Sequence[*Sentence*]]) – list of trg-side inputs

Return type Tuple[Sequence[*Batch*], Sequence[*Batch*]]

Returns tuple of lists of src and trg batches

is_random ()
Returns: True if there is some randomness in the batching process, False otherwise.

Return type bool

xnmt.batchers.**mark_as_batch** (*data*, *mask=None*)
Mark a sequence of items as batch

Parameters

- **data** (Sequence[+T_co]) – sequence of things
- **mask** (Optional[*Mask*]) – optional mask

Returns: a batch of things

Return type *Batch*

xnmt.batchers.**is_batched** (*data*)
Check whether some data is batched.

Parameters **data** (Sequence[+T_co]) – data to check

Return type bool

Returns True iff data is batched.

xnmt.batchers.**pad** (*batch*, *pad_to_multiple=1*)
Apply padding to sentences in a batch.

Parameters

- **batch** (Sequence[+T_co]) – batch of sentences
- **pad_to_multiple** (Integral) – pad sentences so their length is a multiple of this integer.

Return type *Batch*

Returns batch containing padded items and a corresponding batch mask.

class xnmt.batchers.**SrcBatcher** (*batch_size*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)
Bases: *xnmt.batchers.SortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates fixed-size batches, grouped by src len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **batch_size** (Integral) – batch size
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**TrgBatcher**(*batch_size*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.SortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates fixed-size batches, grouped by trg len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **batch_size** (Integral) – batch size
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**SrcTrgBatcher**(*batch_size*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.SortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates fixed-size batches, grouped by src len, then trg len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **batch_size** (Integral) – batch size
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**TrgSrcBatcher**(*batch_size*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.SortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates fixed-size batches, grouped by trg len, then src len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **batch_size** (Integral) – batch size
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**SentShuffleBatcher**(*batch_size*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.ShuffleBatcher*, *xnmt.persistence.Serializable*

A batcher that creates fixed-size batches of random order.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **batch_size** (Integral) – batch size
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**WordShuffleBatcher**(*words_per_batch*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.ShuffleBatcher*, *xnmt.persistence.Serializable*

A batcher that creates fixed-size batches, grouped by src len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **words_per_batch** (Integral) – number of src+trg words in each batch
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**WordSortBatcher**(*words_per_batch*, *avg_batch_size*, *sort_key*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.SortBatcher*

Base class for word sort-based batchers.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **words_per_batch** (Optional[Integral]) – number of src+trg words in each batch
- **avg_batch_size** (Optional[Real]) – avg number of sentences in each batch (if words_per_batch not given)
- **sort_key** (Callable) –
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**WordSrcBatcher**(*words_per_batch=None*, *avg_batch_size=None*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.WordSortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates variable-sized batches with given average (src+trg) words per batch, grouped by src len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **words_per_batch** (Optional[Integral]) – number of src+trg words in each batch
- **avg_batch_size** (Optional[Real]) – avg number of sentences in each batch (if words_per_batch not given)
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**WordTrgBatcher**(*words_per_batch=None*, *avg_batch_size=None*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.WordSortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates variable-sized batches with given average (src+trg) words per batch, grouped by trg len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **words_per_batch** (Optional[Integral]) – number of src+trg words in each batch
- **avg_batch_size** (Optional[Real]) – avg number of sentences in each batch (if words_per_batch not given)
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**WordSrcTrgBatcher**(*words_per_batch=None*, *avg_batch_size=None*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.WordSortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates variable-sized batches with given average number of src + trg words per batch, grouped by src len, then trg len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **words_per_batch** (Optional[Integral]) – number of src+trg words in each batch
- **avg_batch_size** (Optional[Real]) – avg number of sentences in each batch (if words_per_batch not given)
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

class xnmt.batchers.**WordTrgSrcBatcher**(*words_per_batch=None*, *avg_batch_size=None*, *break_ties_randomly=True*, *pad_src_to_multiple=1*)

Bases: *xnmt.batchers.WordSortBatcher*, *xnmt.persistence.Serializable*

A batcher that creates variable-sized batches with given average number of src + trg words per batch, grouped by trg len, then src len.

Sentences inside each batch are sorted by reverse trg length.

Parameters

- **words_per_batch** (Optional[Integral]) – number of src+trg words in each batch
- **avg_batch_size** (Optional[Real]) – avg number of sentences in each batch (if words_per_batch not given)
- **break_ties_randomly** (bool) – if True, randomly shuffle sentences of the same src length before creating batches.
- **pad_src_to_multiple** (Integral) – pad source sentences so its length is multiple of this integer.

xnmt.batchers.**truncate_batches**(**xl*)

Truncate a list of batched items so that all items have the batch size of the input with the smallest batch size.

Inputs can be of various types and would usually correspond to a single time step. Assume that the batch elements with index 0 correspond across the inputs, so that batch elements will be truncated from the top, i.e. starting with the highest-indexed batch elements. Masks are not considered even if attached to a input of *Batch* type.

Parameters ***xl** – batched timesteps of various types

Return type Sequence[Union[Expression, *Batch*, *Mask*, *UniLSTMState*]]

Returns Copies of the inputs, truncated to consistent batch size.

5.8.5 Preprocessing

class xnmt.preproc.**PreprocRunner**(*tasks=None*, *overwrite=False*)

Bases: *xnmt.persistence.Serializable*

Preprocess and filter the input files, and create the vocabulary.

Parameters

- **tasks** (Optional[List[PreprocTask]]) – A list of preprocessing steps, usually parametrized by in_files (the input files), out_files (the output files), and spec for that particular preprocessing type The types of arguments that preproc_spec expects: * Option("in_files", help_str="list of paths to the input files"), * Option("out_files", help_str="list of paths for the output files"), * Option("spec", help_str="The specifications describing which type of processing to use. For normalize and vocab, should consist of the 'lang' and 'spec', where 'lang' can either be 'all' to apply the same type of processing to all languages, or a zero-indexed integer indicating which language to process."),
- **overwrite** (bool) – Whether to overwrite files if they already exist.

class xnmt.preproc.**PreprocExtract**(*in_files*, *out_files*, *specs*)

Bases: xnmt.preproc.PreprocTask, *xnmt.persistence.Serializable*

class xnmt.preproc.**PreprocTokenize**(*in_files*, *out_files*, *specs*)

Bases: xnmt.preproc.PreprocTask, *xnmt.persistence.Serializable*

class xnmt.preproc.**PreprocNormalize**(*in_files*, *out_files*, *specs*)

Bases: xnmt.preproc.PreprocTask, *xnmt.persistence.Serializable*

class xnmt.preproc.**PreprocFilter**(*in_files*, *out_files*, *specs*)

Bases: xnmt.preproc.PreprocTask, *xnmt.persistence.Serializable*

class xnmt.preproc.**PreprocVocab**(*in_files*, *out_files*, *specs*)

Bases: xnmt.preproc.PreprocTask, *xnmt.persistence.Serializable*

class xnmt.preproc.**Normalizer**

Bases: object

A type of normalization to perform to a file. It is initialized first, then expanded.

normalize(*sent*)

Takes a plain text string and converts it into another plain text string after preprocessing.

Return type str

class xnmt.preproc.**NormalizerLower**

Bases: *xnmt.preproc.Normalizer*, *xnmt.persistence.Serializable*

Lowercase the text.

normalize(*sent*)
Takes a plain text string and converts it into another plain text string after preprocessing.

Return type str

class xnmt.preproc.**NormalizerRemovePunct**(*remove_inside_word=False*, *allowed_chars=''*)
Bases: *xnmt.preproc.Normalizer*, *xnmt.persistence.Serializable*

Remove punctuation from the text.

Parameters

- **remove_inside_word** (bool) – If False, only remove punctuation appearing adjacent to white space.
- **allowed_chars** (str) – Specify punctuation that is allowed and should not be removed.

normalize(*sent*)
Takes a plain text string and converts it into another plain text string after preprocessing.

Return type str

class xnmt.preproc.**Tokenizer**
Bases: *xnmt.preproc.Normalizer*

Pass the text through an internal or external tokenizer.

TODO: only StreamTokenizers are supported by the preproc runner right now.

tokenize_stream(*stream*)
Tokenize a file-like text stream.

Parameters **stream** – A file-like stream of untokenized text

Returns A file-like stream of tokenized text

class xnmt.preproc.**BPETokenizer**(*vocab_size*, *train_files*)
Bases: *xnmt.preproc.Tokenizer*, *xnmt.persistence.Serializable*

Class for byte-pair encoding tokenizer.

TODO: Unimplemented

tokenize(*sent*)
Tokenizes a single sentence according to the determined BPE.

class xnmt.preproc.**CharacterTokenizer**
Bases: *xnmt.preproc.Tokenizer*, *xnmt.persistence.Serializable*

Tokenize into characters, with __ indicating blank spaces

tokenize(*sent*)
Tokenizes a single sentence into characters.

Return type str

class xnmt.preproc.**UnicodeTokenizer**(*use_merge_symbol=True*, *merge_symbol=''*, *reverse=False*)
Bases: *xnmt.preproc.Tokenizer*, *xnmt.persistence.Serializable*

Tokenizer that inserts whitespace between words and punctuation.

This tokenizer is language-agnostic and (optionally) reversible, and is based on unicode character categories. See appendix of https://arxiv.org/pdf/1804.08205

Parameters

- **use_merge_symbol** (bool) – whether to prepend a merge-symbol so that the tokenization becomes reversible
- **merge_symbol** (str) – the merge symbol to use
- **reverse** (bool) – whether to reverse tokenization (assumes use_merge_symbol=True was used in forward direction)

tokenize (*sent*)
Tokenizes a single sentence.

Parameters **sent** (str) – input sentence

Return type str

Returns output sentence

class xnmt.preproc.**ExternalTokenizer** (*path*, *tokenizer_args=None*, *arg_separator=' '*)
Bases: *xnmt.preproc.Tokenizer*, *xnmt.persistence.Serializable*

Class for arbitrary external tokenizer that accepts untokenized text to stdin and emits tokenized tezt to stdout, with passable parameters.

It is assumed that in general, external tokenizers will be more efficient when run once per file, so are run as such (instead of one-execution-per-line.)

Parameters

- **path** (str) –
- **tokenizer_args** (Optional[Sequence[str]]) –
- **arg_separator** (str) –

tokenize (*sent*)
Pass the sentence through the external tokenizer.

Parameters **sent** (str) – An untokenized sentence

Return type str

Returns A tokenized sentence

class xnmt.preproc.**SentencepieceTokenizer** (*train_files*, *vocab_size*, *overwrite=False*, *model_prefix='sentpiece'*, *output_format='piece'*, *model_type='bpe'*, *hard_vocab_limit=True*, *encode_extra_options=None*, *decode_extra_options=None*)
Bases: *xnmt.preproc.Tokenizer*, *xnmt.persistence.Serializable*

Sentencepiece tokenizer The options supported by the SentencepieceTokenizer are almost exactly those presented in the Sentencepiece readme, namely:

Parameters

- **train_files** (Sequence[str]) –
- **vocab_size** (Integral) – fixes the vocabulary size
- **overwrite** (bool) –
- **model_prefix** (str) – The trained bpe model will be saved under {model_prefix}.model/.vocab
- **output_format** (str) –

- **model_type** (str) – Either unigram (default), bpe, char or word. Please refer to the sentencepiece documentation for more details
- **hard_vocab_limit** (bool) – setting this to False will make the vocab size a soft limit. Useful for small datasets. This is True by default.
- **encode_extra_options** (Optional[str]) –
- **decode_extra_options** (Optional[str]) –

tokenize (*sent*)

Tokenizes a single sentence into pieces.

Return type str

class xnmt.preproc.**SentenceFilterer** (*spec*)

Bases: object

Filters sentences that don't match a criterion.

keep (*sents*)

Takes a list of inputs/outputs for a single sentence and decides whether to keep them.

In general, these inputs/outpus should already be segmented into words, so len() will return the number of words, not the number of characters.

Parameters **sents** (list) – A list of parallel sentences.

Return type bool

Returns True if they should be used or False if they should be filtered.

class xnmt.preproc.**SentenceFiltererMatchingRegex** (*regex_src*, *regex_trg*, *regex_all*)

Bases: *xnmt.preproc.SentenceFilterer*

Filters sentences via regular expressions. A sentence must match the expression to be kept.

keep (*sents*)

Keep only sentences that match the regex.

Return type bool

class xnmt.preproc.**SentenceFiltererLength** (*min_src=None*, *max_src=None*, *min_trg=None*, *max_trg=None*, *min_all=None*, *max_all=None*)

Bases: *xnmt.preproc.SentenceFilterer*, *xnmt.persistence.Serializable*

Filters sentences by length

keep (*sents*)

Filter sentences by length.

Return type bool

class xnmt.preproc.**VocabFilterer** (*spec*)

Bases: object

Filters vocabulary by some criterion

filter (*vocab*)

Filter a vocabulary.

Parameters **vocab** (Dict[str, Integral]) – A dictionary of vocabulary words with their frequecies.

Return type Dict[str, Integral]

Returns A new dictionary with frequencies containing only the words to leave in the vocabulary.

class xnmt.preproc.**VocabFiltererFreq**(*min_freq*)
Bases: *xnmt.preproc.VocabFilterer*, *xnmt.persistence.Serializable*

Filter the vocabulary, removing words below a particular minimum frequency

filter(*vocab*)
Filter a vocabulary.

Parameters **vocab** – A dictionary of vocabulary words with their frequecies.

Returns A new dictionary with frequencies containing only the words to leave in the vocabulary.

class xnmt.preproc.**VocabFiltererRank**(*max_rank*)
Bases: *xnmt.preproc.VocabFilterer*, *xnmt.persistence.Serializable*

Filter the vocabulary, removing words above a particular frequency rank

filter(*vocab*)
Filter a vocabulary.

Parameters **vocab** – A dictionary of vocabulary words with their frequecies.

Returns A new dictionary with frequencies containing only the words to leave in the vocabulary.

class xnmt.preproc.**Extractor**
Bases: object

A type of extraction task to perform.

class xnmt.preproc.**MelFiltExtractor**(*nfilt=40*, *delta=False*)
Bases: *xnmt.preproc.Extractor*, *xnmt.persistence.Serializable*

extract_to(*in_file*, *out_file*)

Parameters

- **in_file** (str) – yaml file that contains a list of dictionaries. Each dictionary contains: - wav (str): path to wav file - offset (float): start time stamp (optional) - duration (float): stop time stamp (optional) - speaker: speaker id for normalization (optional; if not given, the filename is used as speaker id)
- **out_file** (str) – a filename ending in ".h5"

Return type None

class xnmt.preproc.**LatticeFromPlfExtractor**
Bases: *xnmt.preproc.Extractor*, *xnmt.persistence.Serializable*

Creates node-labeled lattices that can be read by the LatticeInputReader.

The input to this extractor is a list of edge-labeled lattices in PLF format. The PLF format is described here: http://www.statmt.org/moses/?n=Moses.WordLattices It is used, among others, in the Fisher/Callhome Spanish-to-English Speech Translation Corpus (Post et al, 2013).

5.9 Persistence

This module takes care of loading and saving YAML files. Both configuration files and saved models are stored in the same YAML file format.

The main objects to be aware of are:

- *Serializable*: must be subclassed by all components that are specified in a YAML file.

- *Ref*: a reference that points somewhere in the object hierarchy, for both convenience and to realize parameter sharing.
- *Repeat*: a syntax for creating a list components with same configuration but without parameter sharing.
- *YamlPreloader*: pre-loads YAML contents so that some infrastructure can be set up, but does not initialize components.
- *initialize_if_needed()*, *initialize_object()*: initialize a preloaded YAML tree, taking care of resolving references etc.
- *save_to_file()*: saves a YAML file along with registered DyNet parameters
- *LoadSerialized*: can be used to load, modify, and re-assemble pretrained models.
- *bare()*: create uninitialized objects, usually for the purpose of specifying them as default arguments.
- *RandomParam*: a special Serializable subclass that realizes random parameter search.

class xnmt.persistence.**Serializable**

Bases: `yaml.YAMLObject`

All model components that appear in a YAML file must inherit from Serializable. Implementing classes must specify a unique yaml_tag class attribute, e.g. `yaml_tag = "!Serializable"`

shared_params()

Return the shared parameters of this Serializable class.

This can be overwritten to specify what parameters of this component and its subcomponents are shared. Parameter sharing is performed before any components are initialized, and can therefore only include basic data types that are already present in the YAML file (e.g. # dimensions, etc.) Sharing is performed if at least one parameter is specified and multiple shared parameters don't conflict. In case of conflict a warning is printed, and no sharing is performed. The ordering of shared parameters is irrelevant. Note also that if a submodule is replaced by a reference, its shared parameters are ignored.

Return type `List[Set[Union[str, Path]]]`

Returns

objects referencing params of this component or a subcompononent e.g.:

```
return [set([".input_dim",
             ".sub_module.input_dim",
             ".submodules_list.0.input_dim"])]
```

save_processed_arg(*key*, *val*)

Save a new value for an init argument (call from within `__init__()`).

Normally, the serialization mechanism makes sure that the same arguments are passed when creating the class initially based on a config file, and when loading it from a saved model. This method can be called from inside `__init__()` to save a new value that will be passed when loading the saved model. This can be useful when one doesn't want to recompute something every time (like a vocab) or when something has been passed via implicit referencing which might yield inconsistent result when loading the model to assemble a new model of different structure.

Parameters

- **key** (`str`) – name of property, must match an argument of `__init__()`
- **val** (`Any`) – new value; a *Serializable* or basic Python type or list or dict of these

Return type `None`

add_serializable_component (*name*, *passed*, *create_fct*)

Create a *Serializable* component, or a container component with several *Serializable*-s.

Serializable sub-components should always be created using this helper to make sure DyNet parameters are assigned properly and serialization works properly. The components must also be accepted as init arguments, defaulting to `None`. The helper makes sure that components are only created if `None` is passed, otherwise the passed component is reused.

The idiom for using this for an argument named `my_comp` would be:

```
def __init__(self, my_comp=None, other_args, ...):
  ...
  my_comp = self.add_serializable_component("my_comp", my_comp, lambda:␣
↪SomeSerializable(other_args))
  # now, do something with my_comp
  ...
```

Parameters

- **name** (`str`) – name of the object
- **passed** (`Any`) – object as passed in the constructor. If `None`, will be created using create_fct.
- **create_fct** (`Callable[[], Any]`) – a callable with no arguments that returns a *Serializable* or a collection of *Serializable*-s. When loading a saved model, this same object will be passed via the `passed` argument, and `create_fct` is not invoked.

Return type `Any`

Returns reused or newly created object(s).

class `xnmt.persistence.`**UninitializedYamlObject**(*data*)

Bases: `object`

Wrapper class to indicate an object created by the YAML parser that still needs initialization.

Parameters **data** (`Any`) – uninitialized object

`xnmt.persistence.`**bare**(*class_type*, ***kwargs*)

Create an uninitialized object of arbitrary type.

This is useful to specify XNMT components as default arguments. `__init__()` commonly requires DyNet parameters, component referencing, etc., which are not yet set up at the time the default arguments are loaded. In this case, a bare class can be specified with the desired arguments, and will be properly initialized when passed as arguments into a component.

Parameters

- **class_type** (`Type[~T]`) – class type (must be a subclass of *Serializable*)
- **kwargs** (`Any`) – will be passed to class's `__init__()`

Return type ~T

Returns uninitialized object

class `xnmt.persistence.`**Ref**(*path=None*, *name=None*, *default=1928437192847*)

Bases: *xnmt.persistence.Serializable*

A reference to somewhere in the component hierarchy.

Components can be referenced by path or by name.

Parameters

- **path** (Union[None, *Path*, str]) – reference by path
- **name** (Optional[str]) – reference by name. The name refers to a unique _xnmt_id property that must be set in exactly one component.

get_name()
Return name, or None if this is not a named reference

Return type str

get_path()
Return path, or None if this is a named reference

Return type Optional[*Path*]

is_required()
Return True iff there exists no default value and it is mandatory that this reference be resolved.

Return type bool

get_default()
Return default value, or Ref.NO_DEFAULT if no default value is set (i.e., this is a required reference).

Return type Any

resolve_path(*named_paths*)
Get path, resolving paths properly in case this is a named reference.

Return type *Path*

class xnmt.persistence.**Path**(*path_str=''*)
Bases: object

A relative or absolute path in the component hierarchy.

Paths are immutable: Operations that change the path always return a new Path object.

Parameters path_str (str) – path string, with period . as separator. If prefixed by ., marks a relative path, otherwise absolute.

append(*link*)
Return a new path by appending a link.

Parameters link (str) – link to append

Returns: new path

Return type *Path*

add_path(*path_to_add*)
Concatenates a path

Parameters path_to_add (*Path*) – path to concatenate

Returns: concatenated path

Return type *Path*

class xnmt.persistence.**Repeat**(*times*, *content*)
Bases: *xnmt.persistence.Serializable*

A special object that is replaced by a list of components with identical configuration but not with shared params.

This can be specified anywhere in the config hierarchy where normally a list is expected. A common use case is a multi-layer neural architecture, where layer configurations are repeated many times. It is replaced in the preloader and cannot be instantiated directly.

exception xnmt.persistence.**PathError**(*message*)
Bases: Exception

class xnmt.persistence.**SavedFormatString**(*value*, *unformatted_value*)
Bases: str, *xnmt.persistence.Serializable*

class xnmt.persistence.**FormatString**(*value*, *serialize_as*)
Bases: str, yaml.YAMLObject

Used to handle the {EXP} string formatting syntax. When passed around it will appear like the properly resolved string, but writing it back to YAML will use original version containing {EXP}

class xnmt.persistence.**RandomParam**(*values*)
Bases: yaml.YAMLObject

class xnmt.persistence.**LoadSerialized**(*filename*, *path=”*, *overwrite=None*)
Bases: *xnmt.persistence.Serializable*

Load content from an external YAML file.

This object points to an object in an external YAML file and will be replaced by the corresponding content by the YAMLPreloader.

Parameters

- **filename** (str) – YAML file name to load from
- **path** (str) – path inside the YAML file to load from, with . separators. Empty string denotes root.
- **overwrite** (Optional[List[Dict[str, Any]]]) – allows overwriting parts of the loaded model with new content. A list of path/val dictionaries, where path is a path string relative to the loaded sub-object following the syntax of *Path*, and val is a Yaml-serializable specifying the new content. E.g.:

```
[{"path" : "model.trainer", "val":AdamTrainer()},
 {"path" : ..., "val":...}]
```

  It is possible to specify the path to point to a new key to a dictionary. If path points to a list, it's possible append to that list by using append_val instead of val.

class xnmt.persistence.**YamlPreloader**
Bases: object

Loads experiments from YAML and performs basic preparation, but does not initialize objects.

Has the following responsibilities:

- takes care of extracting individual experiments from a YAML file
- replaces !LoadSerialized by loading the corresponding content
- resolves kwargs syntax (items from a kwargs dictionary are moved to the owner where they become object attributes)
- implements random search (draws proper random values when !RandomParam is encountered)
- finds and replaces placeholder strings such as {EXP}, {EXP_DIR}, {GIT_REV}, and {PID}
- copies bare default arguments into the corresponding objects where appropriate.

Typically, *initialize_object()* would be invoked by passing the result from the YamlPreloader.

static experiment_names_from_file(*filename*)
Return list of experiment names.

Parameters **filename** (str) – path to YAML file

Return type List[str]

Returns experiment names occuring in the given file in lexicographic order.

static preload_experiment_from_file(*filename*, *exp_name*, *resume=False*)
Preload experiment from YAML file.

Parameters

- **filename** (str) – YAML config file name
- **exp_name** (str) – experiment name to load
- **resume** (bool) – set to True if we are loading a saved model file directly and want to restore all formatted strings.

Return type *UninitializedYamlObject*

Returns Preloaded but uninitialized object.

static preload_obj(*root*, *exp_name*, *exp_dir*, *resume=False*)
Preload a given object.

Preloading a given object, usually an xnmt.experiment.Experiment or *LoadSerialized* object as parsed by pyyaml, includes replacing !LoadSerialized, resolving kwargs syntax, and instantiating random search.

Parameters

- **root** (Any) – object to preload
- **exp_name** (str) – experiment name, needed to replace {EXP}
- **exp_dir** (str) – directory of the corresponding config file, needed to replace {EXP_DIR}
- **resume** (bool) – if True, keep the formatted strings, e.g. set {EXP} to the value of the previous run if possible

Return type *UninitializedYamlObject*

Returns Preloaded but uninitialized object.

xnmt.persistence.**save_to_file**(*fname*, *mod*)
Save a component hierarchy and corresponding DyNet parameter collection to disk.

Parameters

- **fname** (str) – Filename to save to.
- **mod** (Any) – Component hierarchy.

Return type None

xnmt.persistence.**initialize_if_needed**(*root*)
Initialize if obj has not yet been initialized.

This includes parameter sharing and resolving of references.

Parameters **root** (Union[Any, *UninitializedYamlObject*]) – object to be potentially serialized

Return type Any

Returns initialized object

xnmt.persistence.**initialize_object**(*root*)
Initialize an uninitialized object.

This includes parameter sharing and resolving of references.

Parameters **root** (*UninitializedYamlObject*) – object to be serialized

Return type Any

Returns initialized object

exception xnmt.persistence.**ComponentInitError**
Bases: Exception

xnmt.persistence.**check_type**(*obj*, *desired_type*)
Checks argument types using isinstance, or some custom logic if type hints from the 'typing' module are given.

Regarding type hints, only a few major ones are supported. This should cover almost everything that would be expected in a YAML config file, but might miss a few special cases. For unsupported types, this function evaluates to True. Most notably, forward references such as 'SomeType' (with apostrophes around the type) are not supported. Note also that typing.Tuple is among the unsupported types because tuples aren't supported by the XNMT serializer.

Parameters

- **obj** – object whose type to check
- **desired_type** – desired type of obj

Returns False if types don't match or desired_type is unsupported, True otherwise.

5.10 Reportable

Reports gather inputs, outputs, and intermediate computations in a nicely formatted way for convenient manual inspection.

To support reporting, the models providing the data to be reported must subclass Reportable and call self.report_sent_info(d) with key/value pairs containing the data to be reported at the appropriate times. If this causes a computational overhead, the boolean compute_report field should queried and extra computations skipped if this field is False.

Next, a Reporter needs to be specified that supports reports based on the previously created key/value pairs. Reporters are passed to inference classes, so it's possible e.g. to report only at the final test decoding, or specify a special reporting inference object that only looks at a handful of sentences, etc.

Note that currently reporting is only supported at test-time, not at training time.

class xnmt.reports.**ReportInfo**(*sent_info=[]*, *glob_info={}*)
Bases: object

Info to pass to reporter

Parameters

- **sent_info** (Sequence[Dict[str, Any]]) – list of dicts, one dict per sentence
- **glob_info** (Dict[str, Any]) – a global dict applicable to each sentence

class xnmt.reports.**Reportable**

Bases: object

Base class for classes that contribute information to a report.

Making an arbitrary class reportable requires to do the following:

- specify Reportable as base class
- call this super class's `__init__()`, or do @register_xnmt_handler manually
- pass either global info or per-sentence info or both: - call self.report_sent_info(d) for each sentence, where d is a dictionary containing info to pass on to the

  reporter

  - call self.report_corpus_info(d) once, where d is a dictionary containing info to pass on to the reporter

report_sent_info(*sent_info*)

Add key/value pairs belonging to the current sentence for reporting.

This should be called consistently for every sentence and in order.

Parameters **sent_info** (Dict[str, Any]) – A dictionary of key/value pairs. The keys must match (be a subset of) the arguments in the reporter's create_sent_report() method, and the values must be of the corresponding types.

Return type None

report_corpus_info(*glob_info*)

Add key/value pairs for reporting that are relevant to all reported sentences.

Parameters **glob_info** (Dict[str, Any]) – A dictionary of key/value pairs. The keys must match (be a subset of) the arguments in the reporter's create_sent_report() method, and the values must be of the corresponding types.

Return type None

class xnmt.reports.**Reporter**

Bases: object

A base class for a reporter that collects reportable information, formats it and writes it to disk.

create_sent_report(***kwargs*)

Create the report.

The reporter should specify the arguments it needs explicitly, and should specify kwargs in addition to handle extra (unused) arguments without crashing.

Parameters ****kwargs** – additional arguments

Return type None

class xnmt.reports.**ReferenceDiffReporter**(*match_size=3*, *alt_norm=False*, *report_path='{EXP_DIR}/reports/{EXP}'*)

Bases: *xnmt.reports.Reporter*, *xnmt.persistence.Serializable*

Reporter that uses the CharCut tool for nicely displayed difference highlighting between outputs and references.

The stand-alone tool can be found at https://github.com/alardill/CharCut

Parameters

- **match_size** (Integral) – min match size in characters (set < 3 e.g. for Japanese or Chinese)

- **alt_norm** (bool) – alternative normalization scheme: use only the candidate's length for normalization
- **report_path** (str) – Path of directory to write HTML files to

create_sent_report (*src*, *output*, *ref_file=None*, ***kwargs*)
Create report.

Parameters

- **src** (*Sentence*) – source-side input
- **output** (*ReadableSentence*) – generated output
- **ref_file** (Optional[str]) – path to reference file
- ****kwargs** – arguments to be ignored

Return type None

class xnmt.reports.**CompareMtReporter** (*out2_file=None*, *train_file=None*, *train_counts=None*, *alpha=1.0*, *ngram=4*, *ngram_size=50*, *sent_size=10*, *report_path='{EXP_DIR}/reports/{EXP}'*)
Bases: *xnmt.reports.Reporter*, *xnmt.persistence.Serializable*

Reporter that uses the compare-mt.py script to analyze and compare MT results.

The stand-alone tool can be found at https://github.com/neubig/util-scripts

Parameters

- **out2_file** (Optional[str]) – A path to another system output. Add only if you want to compare outputs from two systems.
- **train_file** (Optional[str]) – A link to the training corpus target file
- **train_counts** (Optional[str]) – A link to the training word frequency counts as a tab-separated "wordtfreq" file
- **alpha** (Real) – A smoothing coefficient to control how much the model focuses on low- and high-frequency events. 1.0 should be fine most of the time.
- **ngram** (Integral) – Maximum length of n-grams.
- **sent_size** (Integral) – How many sentences to print.
- **ngram_size** (Integral) – How many n-grams to print.
- **report_path** (str) – Path of directory to write report files to

create_sent_report (*output*, *ref_file*, ***kwargs*)
Create report.

Parameters

- **output** (*ReadableSentence*) – generated output
- **ref_file** (str) – path to reference file
- ****kwargs** – arguments to be ignored

Return type None

class xnmt.reports.**HtmlReporter** (*report_name*, *report_path='{EXP_DIR}/reports/{EXP}'*)
Bases: *xnmt.reports.Reporter*

A base class for reporters that produce HTML outputs that takes care of some common functionality.

Parameters

- **report_name** (str) – prefix for report files
- **report_path** (str) – Path of directory to write HTML and image files to

class xnmt.reports.**AttentionReporter**(*max_num_sents=100*, *report_name='attention'*, *report_path='{EXP_DIR}/reports/{EXP}'*)

Bases: *xnmt.reports.HtmlReporter*, *xnmt.persistence.Serializable*

Reporter that writes attention matrices to HTML.

Parameters

- **max_num_sents** (Optional[Integral]) – create attention report for only the first n sentences
- **report_name** (str) – prefix for output files
- **report_path** (str) – Path of directory to write HTML and image files to

create_sent_report(*src*, *output*, *attentions*, *ref_file*, ***kwargs*)

Create report.

Parameters

- **src** (*Sentence*) – source-side input
- **output** (*ReadableSentence*) – generated output
- **attentions** (ndarray) – attention matrices
- **ref_file** (Optional[str]) – path to reference file
- ****kwargs** – arguments to be ignored

Return type None

add_atts(*attentions*, *src_tokens*, *trg_tokens*, *idx*, *desc='Attentions'*)

Add attention matrix to HTML code.

Parameters

- **attentions** (ndarray) – numpy array of dimensions (src_len x trg_len)
- **src_tokens** (Union[Sequence[str], ndarray]) – list of strings (case of src text) or numpy array of dims (nfeat x speech_len) (case of src speech)
- **trg_tokens** (Sequence[str]) – list of string tokens
- **idx** (Integral) – sentence no
- **desc** (str) – readable description

Return type None

class xnmt.reports.**SegmentationReporter**(*report_path='{EXP_DIR}/reports/{EXP}'*)

Bases: *xnmt.reports.Reporter*, *xnmt.persistence.Serializable*

A reporter to be used with the segmenting encoder.

Parameters **report_path** (str) – Path of directory to write text files to

create_sent_report(*segment_actions*, *src*, ***kwargs*)

Create the report.

The reporter should specify the arguments it needs explicitly, and should specify kwargs in addition to handle extra (unused) arguments without crashing.

Parameters ****kwargs** – additional arguments

class xnmt.reports.**OOVStatisticsReporter**(*train_trg_file*, *report_path='{EXP_DIR}/reports/{EXP}'*)

Bases: *xnmt.reports.Reporter*, *xnmt.persistence.Serializable*

A reporter that prints OOV statistics: recovered OOVs, fantasized new words, etc.

Some models such as character- or subword-based models can produce words that are not in the training. This is desirable when we produce a correct word that would have been an OOV with a word-based model but undesirable when we produce something that's not a correct word. The reporter prints some statistics that help analyze the OOV behavior of the model.

Parameters

- **train_trg_file** (`str`) – path to word-tokenized training target file
- **report_path** (`str`) – Path of directory to write text files to

create_sent_report(*output*, *ref_file*, ***kwargs*)

Create the report.

The reporter should specify the arguments it needs explicitly, and should specify `kwargs` in addition to handle extra (unused) arguments without crashing.

Parameters ****kwargs** – additional arguments

Return type `None`

5.11 Settings

Global settings that control the overall behavior of XNMT.

Currently, settings control the following:

- `OVERWRITE_LOG`: whether logs should be overwritten (not overwriting helps when copy-pasting config files and forgetting to change the output location)
- `IMMEDIATE_COMPUTE`: whether to execute DyNet in eager mode
- `CHECK_VALIDITY`: configure xnmt and DyNet to perform checks of validity
- `RESOURCE_WARNINGS`: whether to show resource warnings
- `LOG_LEVEL_CONSOLE`: verbosity of console output (`DEBUG` | `INFO` | `WARNING` | `ERROR` | `CRITICAL`)
- `LOG_LEVEL_FILE`: verbosity of file output (`DEBUG` | `INFO` | `WARNING` | `ERROR` | `CRITICAL`)
- `DEFAULT_MOD_PATH`: default location to write models to
- `DEFAULT_LOG_PATH`: default location to write out logs

There are several predefined configurations (`Standard`, `Debug`, `Unittest`), with `Standard` being used by default. Settings are specified from the command line using `--settings={standard|debug|unittest}` and should not be changed during execution.

It is possible to control individual settings by setting an environment variable of the same name, e.g. like this: `OVERWRITE_LOG=1 python -m xnmt.xnmt_run_experiments my_config.yaml`

To specify a custom configuration, subclass `settings.Standard` accordinly and add an alias to `settings._aliases`.

class xnmt.settings.**Standard**

Bases: `object`

Standard configuration, used by default.

class xnmt.settings.**Debug**

Bases: *xnmt.settings.Standard*

Adds checks and verbosity to help debugging code or configuration files.

class xnmt.settings.**Unittest**

Bases: *xnmt.settings.Standard*

More checks and less verbosity, activated automatically when running the unit tests from the "test" package.

CHAPTER 6

Programming Conventions

6.1 Philosphy

The over-arching goal of *xnmt* is that it be easy to use for research. When implementing a new method, it should require only minimal changes (e.g. ideally the changes will be limited to a single file, over-riding an existing class). Obviously this ideal will not be realizable all the time, but when designing new functionality, try to think of this goal. If there are tradeoffs, the following is the order of priority (of course getting all is great!):

1. Code Correctness
2. Extensibility and Readability
3. Accuracy and Effectiveness of the Models
4. Efficiency

6.2 Style

There are some minimal coding style conventions:

- Functions should be snake_case, classes should be UpperCamelCase.
- Indentation should be two whitespace characters.
- In variable names, common words should be abbreviated as:
  - source -> src
  - target -> trg
  - sentence -> sent
  - hypothesis -> hyp
  - reference -> ref

6.3 Documentation

- Docstrings should be made according to the Google style guide.
- Types should be annotated consistently, see corresponding Python docs. As a rule of thumb, function arguments should be given a general type (e.g. `numbers.Real`, `numbers.Integral`, `typing.Sequence[str]`), whereas return types may be more specific (`float`, `int`, `typing.List[str]`).
- Ideally, documentation should be added at module-level (giving a summary of the most relevant contents of the module), the class level (including arguments for `__init__()`), and method level. Documentation for methods/classes etc. that do not need to be accessed from outside may be omitted and these should ideally marked as private by adding a single underscore as prefix.
- Note: some of these conventions are currently not followed consistently; PRs welcome!

6.4 Testing

A collection of unit tests exists to make sure things don't break. When writing new code:

- The minimum recommendation is to add a config file to `test/config` and add a corresponding entry to `test/test_run.py` which will ensure that future commits will not cause this to crash. This "crash test config" should run as fast as possible.
- Even better would be correctness tests, several examples for which can be found in the test package.

6.5 Logging

For printing output in a consistent and controllable way, a few conventions should be followed (see _official documentation: https://docs.python.org/3/howto/logging.html#when-to-use-logging for more details):

- `logger.info()` should be used for most outputs. Such outputs are assumed to be usually shown but can be turned off if needed.
- `print()` for regular output without which the execution would be incomplete. The main use case is to print final results, etc.
- `logger.debug()` for detailed information that isn't needed in normal operation
- `logger.warning()`, logger.error() or logger.critical() for problematic situations
- `yaml_logger(dict)` for structured logging of information that should be easily automatically parseable and might be too bulky to print to the console.

These loggers can be requested as follows:

```
from xnmt import logger
from xnmt import yaml_logger
```

6.6 Contributing

Go ahead and send a pull request! If you're not sure whether something will be useful and want to ask beforehand, feel free to open an issue on the github.

CHAPTER 7

Writing XNMT classes

In order to write new components that can be created both from YAML config files as well as programmatically, support sharing of DyNet parameters, etc., one must adhere to the Serializable interface including a few simple conventions:

Note: XNMT will perform automatic checks and raise an informative error in case these conventions are violated, so there is no need to worry about these too much.

7.1 Marking classes as serializable

Classes are marked as serializable by specifying `xnmt.persistence.Serializable` as super class. They must specify a unique yaml_tag class attribute, set to `!ClassName` with ClassName replaced by the class name. It follows that class names must be unique, even across different XNMT modules. (Note: Serializable should be explicitly specified even if another super class already does the same)

7.2 Specifying init arguments

The arguments accepted in the YAML config file correspond directly to the arguments of the class's `__init__()` method. The `__init__` is required to be decorated with `@xnmt.persistence.serializable_init`. Note that sub-objects are initialized before being passed to `__init__`, and in the order in which they are specified in `__init__`.

7.3 Using DyNet parameters

If the component uses DyNet parameters, the calls to `dynet_model.add_parameters()` etc. must take place in `__init__` (or a helper called from within `__init__`). It is not possible to allocate parameters after `__init__` has returned. The component will get assigned its own unique DyNet parameter collection, which can be requested using xnmt.param_collection.ParamManager.my_params(self). Subcollections should never be passed to sub-objects that are Serializable. Behind the scenes, components will get assigned a unique subcollection id which ensures that they can be loaded later along with their pretrained weights, and even combined with components trained from a different config file.

7.4 Using Serializable subcomponents

If a class uses helper objects that are also Serializable, this must occur in a certain way:

- the Serializable object must be accepted as argument in __init__.
- It can be set to None by default, in which case it must be constructed manually within __init__. This should take place using the Serializable.add_serializable_component() helper, e.g. with the following idiom:

```
@serializable_init
def __init__(self, ..., vocab_projector=None, ...):
  ...
  self.vocab_projector = \
       self.add_serializable_component(\
               "vocab_projector",
               vocab_projector,
               lambda: xnmt.linear.Linear(input_dim=mlp_hidden_dim,
                                          output_dim=vocab_size,
                                          param_init=param_init_output,
                                          bias_init=bias_init_output))
  ...
```

CHAPTER 8

Save File Format

8.1 Overview

When saving a (partly) trained model to disk, the resulting model file is in YAML format and looks very similar to the configuration files (see *Experiment configuration file format*) with a few exceptions:

- Saved model files hold only one experiment (in contrast, config files contain dictionaries of several named experiments).
- Saved models are accompanied by a `.data` directory holding trained DyNet weights.
- Some components replace the originally specified arguments with updated contents. For instance, the vocabulary is usually stored as an explicit list in saved model files, whereas config files typically refer to an external vocab file.

8.2 .data sub-directory

This directory contains a list of DyNet subcollections with names such as `Linear.98dc700f` or `UniLSTMSeqTransducer.519cfb41`. Every `Serializable` class that allocates DyNet parameters using `xnmt.param_collection.ParamManager.my_params(self)` (see *Writing XNMT classes*) will have one such subcollection written to disk. The file names correspond to the component's `xnmt_subcol_name`, consisting of the component name and a unique identifier. The `xnmt_subcol_name` is also stored in the saved model's YAML file to establish the correspondence. Each subcollection is stored using DyNet's serialization format which is a readable text file.

In case several checkpoints are saved, there will be additional `.data.1`, `.data.2` etc. files. It is worth mentioning that `xnmt_subcol_name` does not change between checkpoints, and only one YAML file is written out. Also note that the additional checkpoints are generally ignored when loading a saved model, but can be substituted manually by renaming them, or be processed by the below utilities.

8.3 Command-line utilities

- `script/code/avg_checkpoints.py`: Perform checkpoint-averaging by taking the elementwise arithmetic average of parameters from all saved checkpoints.
- `script/code/conv_checkpoints_to_model.py`: Convert a checkpoint to its own model. This is for example useful to enable checkpoint ensembling. Under the hood, this draw new random `xnmt_subcol_name` identifiers and in order to enable loading all checkpoints as separate models into XNMT.

CHAPTER 9

Visualization

XNMT comes with several visualization tools.

9.1 Visualization of training progress

The training progress can be monitored via Tensorboard. XNMT uses the `tensorboardX` package to write logs that can be read and visualized via Tensorboard. These logs are written out by default, no configuration is required. To run Tensorboard, Tensorflow must be installed first (see Tensorflow home page for further instructions):

```
pip install tensorflow
tensorboard --logdir <path/to/base/xnmt/log/dir>
```

9.2 Visualization of translation outputs

Translation outputs can be analyzed via reporters as defined in `xnmt/reports.py`. To use reporters, just simply define any reporter class inside the inference. Reports can only be used for inference-only experiments, i.e. experiments that load a pretrained model and only perform inference but no training. The following reporters are available (see API doc for more details):

- `AttentionReporter`: print attention matrices
- `ReferenceDiffReporter`: HTML-visualization of diffs between reference and actual output
- `CompareMtReporter`: perform detailed analysis, including computing over- and undergenerated n-grams.
- `OOVStatisticsReporter`: compute OOV statistics, useful when using character- or subword models.
- `SegmentationReporter`: Used only for the SegmentationSeqTransducer encoder, to print the segmentation of the input.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

x

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z