Sockeye Documentation

Release 1.18.25

amazon

Jun 21, 2018

Contents

This is the documentation for sockeye, a framework for sequence-to-sequence modeling with MXNet. To get started, please read through the *README*.

The individual modules and functions of the project are documented under *Python Modules*.

CHAPTER 1

For Contributors

If you want to contribute or develop for sockeye, please see the *Developer Guide*.

CHAPTER 2

Table of Contents

2.1 Sockeye

This package contains the Sockeye project, a sequence-to-sequence framework for Neural Machine Translation based on Apache MXNet Incubating. It implements state-of-the-art encoder-decoder architectures, such as

- Deep Recurrent Neural Networks with Attention [Bahdanau, '14]
- Transformer Models with self-attention [Vaswani et al, '17]
- Fully convolutional sequence-to-sequence models [Gehring et al, '17]

If you use Sockeye, please cite:

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton and Matt Post (2017): Sockeye: A Toolkit for Neural Machine Translation. In eprint arXiv:cs-CL/1712.05690.

```
@article{Sockeye:17,
   author = {Hieber, Felix and Domhan, Tobias and Denkowski, Michael
             and Vilar, David and Sokolov, Artem and Clifton, Ann and Post, Matt},
    title = "{Sockeye: A Toolkit for Neural Machine Translation}",
  journal = {arXiv preprint arXiv:1712.05690},
archivePrefix = "arXiv",
   eprint = {1712.05690},
 primaryClass = "cs.CL",
 keywords = {Computer Science - Computation and Language,
              Computer Science - Learning,
              Statistics - Machine Learning},
     year = 2017,
    month = dec,
      url = {https://arxiv.org/abs/1712.05690}
}
```

If you are interested in collaborating or have any questions, please submit a pull request or issue. Click to find our developer guidelines. You can also send questions to *sockeye-dev-at-amazon-dot-com*.

Recent developments and changes are tracked in our changelog.

2.1.1 Dependencies

Sockeye requires:

- **Python3**
- MXNet-1.2.0
- numpy

2.1.2 Installation

There are several options for installing Sockeye and it's dependencies. Below we list several alternatives and the corresponding instructions.

2.1.2.1 Either: AWS DeepLearning AMI

AWS DeepLearning AMI users only need to run the following line to install sockeye:

```
> sudo pip3 install sockeye --no-deps
```

For other environments, you can choose between installing via pip or directly from source. Note that for the remaining instructions to work you will need to use `python3` instead of `python` and `pip3` instead of `pip`.

2.1.2.2 Or: pip package

CPU

```
> pip install sockeye
```

GPU

If you want to run sockeye on a GPU you need to make sure your version of Apache MXNet Incubating contains the GPU bindings. Depending on your version of CUDA, you can do this by running the following:

```
> wget https://raw.githubusercontent.com/awslabs/sockeye/master/requirements/
↪requirements.gpu-cu${CUDA_VERSION}.txt
> pip install sockeye --no-deps -r requirements.gpu-cu${CUDA_VERSION}.txt
> rm requirements.gpu-cu${CUDA_VERSION}.txt
```

where `${CUDA_VERSION}` can be `75` (7.5), `80` (8.0), `90` (9.0), or `91` (9.1).

2.1.2.3 Or: From Source

CPU

If you want to just use sockeye without extending it, simply install it via

```
> pip install -r requirements/requirements.txt
> pip install .
```

after cloning the repository from git.

GPU

If you want to run sockeye on a GPU you need to make sure your version of Apache MXNet Incubating contains the GPU bindings. Depending on your version of CUDA you can do this by running the following:

```
> pip install -r requirements/requirements.gpu-cu${CUDA_VERSION}.txt
> pip install .
```

where `${CUDA_VERSION}` can be `75` (7.5), `80` (8.0), `90` (9.0), or `91` (9.1).

2.1.2.4 Optional dependencies

In order to write training statistics to a Tensorboard event file for visualization, you can optionally install mxboard (`pip install mxboard`). To visualize these, run the Tensorboard tool (`pip install tensorboard tensorflow`) with the logging directory pointed to the training output folder: `tensorboard --logdir <model>`

If you want to create alignment plots you will need to install matplotlib (`pip install matplotlib`).

In general you can install all optional dependencies from the Sockeye source folder using:

```
> pip install '.[optional]'
```

2.1.2.5 Running sockeye

After installation, command line tools such as *sockeye-train, sockeye-translate, sockeye-average* and *sockeye-embeddings* are available. Alternatively, if the sockeye directory is on your`$PYTHONPATH` you can run the modules directly. For example *sockeye-train* can also be invoked as

```
> python -m sockeye.train <args>
```

2.1.3 First Steps

For easily training popular model types on known data sets, see the Sockeye Autopilot documentation. For manually training and running translation models on your data, read on.

2.1.3.1 Train

In order to train your first Neural Machine Translation model you will need two sets of parallel files: one for training and one for validation. The latter will be used for computing various metrics during training. Each set should consist of two files: one with source sentences and one with target sentences (translations). Both files should have the same number of lines, each line containing a single sentence. Each sentence should be a whitespace delimited list of tokens.

Say you wanted to train a RNN German-to-English translation model, then you would call sockeye like this:

```
> python -m sockeye.train --source sentences.de \
                          --target sentences.en \
                          --validation-source sentences.dev.de \
                          --validation-target sentences.dev.en \
                          --use-cpu \
                          --output <model_dir>
```

After training the directory *<model_dir>* will contain all model artifacts such as parameters and model configuration. The default setting is to train a 1-layer LSTM model with attention.

2.1.3.2 Translate

Input data for translation should be in the same format as the training data (tokenization, preprocessing scheme). You can translate as follows:

```
> python -m sockeye.translate --models <model_dir> --use-cpu
```

This will take the best set of parameters found during training and then translate strings from STDIN and write translations to STDOUT.

For more detailed examples check out our user documentation.

2.1.4 Step-by-step tutorial

More detailed step-by-step tutorials can be found in the tutorials directory.

2.2 User documentation

2.2.1 Training

Training is carried out by the `sockeye.train` module. Basic usage is given by

```
> python -m sockeye.train
usage: train.py [-h] --source SOURCE --target TARGET --validation-source
                VALIDATION_SOURCE --validation-target VALIDATION_TARGET
                --output OUTPUT [...]
```

Training requires 5 arguments:

- `--source`, `--target`: give the training data files. Gzipped files are supported, provided that their filenames end with .gz.
- `--validation-source`, `--validation-target`: give the validation data files, gzip supported as above.
- `--output`: gives the output directory where the intermediate and final results will be written to. Intermediate directories will be created if needed. Logging will be written to `<model_dir>/log` as well as being echoed on the console.

For a complete list of supported options use the `--help` option.

2.2.1.1 Data format

All input files files should be UTF-8 encoded, tokenized with standard whitespaces. Each line should contain a single sentence and the source and target files should have the same number of lines. Vocabularies will automatically be created from the training data and vocabulary coverage on the validation set during initialization will be reported.

2.2.1.2 Checkpointing and early-stopping

Training is governed by the concept of "checkpoints", rather than epochs. You can specify the checkpoint frequency in terms of updates/batches with `--checkpoint-frequency`. Training performs early-stopping to prevent overfitting, i.e. training is stopped once a defined evaluation metric computed on the held-out validation data does not improve for a number of checkpoints given by the parameter `--max-num-checkpoint-not-improved`. You can specify a maximum number of updates/batches using `--max-updates`.

Perplexity is the default metric to be considered for early-stopping, but you can also choose to optimize accuracy or BLEU using the `--optimized-metric` argument. In case of optimizing with respect to BLEU, you will need to specify `--monitor-bleu`. For efficiency reasons, sockeye spawns a sub-processes after each checkpoint to decode the validation data and compute BLEU. This may introduce some delay in the reporting of results, i.e. there may be checkpoints with no BLEU results reported or with results corresponding to older checkpoints. This is expected behaviour and sockeye internally keeps track of the results in the correct order.

Note that evaluation metrics for training data and held-out validation data are written in a tab-separated file called `metrics`.

At each checkpoint, the internal state of the training process is stored to disk. If the training is interrupted (e.g. due to a hardware failure), you can start sockeye again, with the same parameters as for the initial call, and training will resume from the last checkpoint. Note that this is different to using the `--params` argument. This argument is used only to initialize the training with pre-computed values for the parameters of the model, but the parameters of the optimizer and other parts of the system are initialized from scratch.

2.2.1.3 Monitoring training progress with Tensorboard

Sockeye can write all evaluation metrics in a Tensorboard compatible format. This way you can monitor the training progress in the browser. To enable this feature, install the MXNet-compatible interface, mxboard:

```
> pip install mxboard
```

For visualization, you still need the official tensorboard release (i.e. `pip install tensorboard`). Start tensorboard and point it to the model directory (or any parent directory):

```
> tensorboard --logdir model_dir
```

2.2.1.4 CPU/GPU training

By default, training is carried out on the first GPU device of your machine. You can specify alternative GPU devices with the `--device-ids` option, with which you can also activate multi-GPU training (see below). If `--device-ids -1`, sockeye will try to find a free GPU on your machine and block until one is available. The locking mechanism is based on files and therefore assumes all processes are running on the same machine with the same file system. If this is not the case there is a chance that two processes will be using the same GPU and you run out of GPU memory. If you do not have or do not want to use a GPU, specify `--use-cpu`. In this case a drop in performance is expected.

Multi-GPU training

Training can be carried out on multiple GPUs by either specifying multiple GPU device ids: `--device-ids 0 1 2 3`, or specifying the number GPUs required: `--device-ids -n` attempts to acquire `n` GPUs through the locking mechanism described above. This will train using Data Parallelism. MXNet will divide the data in each batch and send it to the different devices. Note that you should increase the batch size: for `k` GPUs use `--batch-size k*<original_batch_size>`. Also note that this will likely linearly increase your throughput in terms of sentences/second, but not necessarily increase the model's convergence speed.

2.2.1.5 Checkpoint averaging

A common technique for improving model performance is to average the weights for the last checkpoints. This can be done as follows:

```
> python -m sockeye.average <model_dir> -o <model_dir>/model.best.avg.params
```

2.2.2 Translation

Translating is handled by the `sockeye.translate` module:

```
> python -m sockeye.translate
```

The only required argument is `--models`, which should point to an `<model_dir>` folder of trained models. By default, sockeye chooses the parameters from the best checkpoint and uses these for translation. You can specify parameters from a specific checkpoint by using `--checkpoints X`.

You can control the size of the beam using `--beam-size` and the maximum input length by `--max-input-length`. Sentences that are longer than `max-input-length` are stripped.

Input is read from the standard input and the output is written to the standard output. The CLI will log translation speed once the input is consumed. Like in the training module, the first GPU device is used by default. Note however that multi-GPU translation is not currently supported. For CPU decoding use `--use-cpu`.

Use the `--help` option to see a full list of options for translation.

2.2.2.1 Ensemble Decoding

Sockeye supports ensemble decoding by specifying multiple model directories and multiple checkpoints. The given lists must have the same length, such that the first given checkpoint will be taken from the first model directory, the second specified checkpoint from the second directory, etc.

```
> python -m sockeye.translate --models [<m1prefix> <m2prefix>] --checkpoints [<cp1>
↪<cp2>]
```

2.2.2.2 Visualization

The default mode of the translate CLI is to output translations to STDOUT. You can also print out an ASCII matrix of the alignments using `--output-type align_text`, or save the alignment matrix as a PNG plot using `--output-type align_plot`. The PNG files will be written to files beginning with the prefix given by the `--align-plot-prefix` option, one for each input sentence, indexed by the sentence id.

2.3 Developer Documentation

2.3.1 Requirements

There are three types of dependencies: core dependencies, development dependencies and dependencies for generating the documentation.

Install them via

```
> pip install -r requirements/requirements.txt
> pip install -r requirements/requirements.dev.txt
> pip install -r requirements/requirements.docs.txt
```

2.3.2 Developer Guidelines

We welcome contributions to sockeye in form of pull requests on Github. If you want to develop sockeye, please adhere to the following development guidelines.

- Write Python 3.5, PEP8 compatible code.
- Functions should be documented with Sphinx-style docstrings and should include type hints for static code analyzers.

```
def foo(bar: <type of bar>) -> <returnType>:
    """
    <Docstring for foo method, followed by a period>.

    :param bar: <Description of bar argument followed by a period>.
    :return: <Description of the return value followed by a period>.
    """
```

- When using MXNet operators, preceding symbolic statements in the code with the resulting, expected shape of the tensor greatly improves readability of the code:

```
# (batch_size, num_hidden)
data = mx.sym.Variable('data')
# (batch_size * num_hidden,)
data = mx.sym.reshape(data=data, shape=(-1))
```

- The desired line length of Python modules should not exceed 120 characters.
- When writing symbol-generating classes (such as encoders/decoders), initialize variables in the constructor of the class and re-use them in the class methods.
- Make sure to pass unit tests before submitting a pull request.
- Whenever reasonable, write py.test unit tests covering your contribution.
- When importing other sockeye modules import the entire module instead of individual functions and classes using relative imports:

```
from . import attention
```

2.3.3 Building the Documentation

Full documentation, including a code reference, can be generated using Sphinx with the following command:

```
> python setup.py docs
```

The results are written to `docs/_build/html/index.html`.

2.3.4 Unit & Integration Tests

Unit & integration tests are written using py.test. They can be run with:

```
> python setup.py test
```

or:

```
> pytest
```

Integration tests run Sockeye CLI tools on small, synthetic data to test for functional correctness.

2.3.5 System Tests

System tests test Sockeye CLI tools on synthetic tasks (digit sequence copying & sorting) for functional correctness and successful learning. They assert on validation metrics (perplexity) and BLEU scores from decoding. A subset of the system tests are run on Travis for every commit. The full set of system tests is run as a nightly Travis Cron job. You can manually run the system tests with:

```
> pytest test/system
```

2.3.6 Submitting a New Version to PyPI

Before starting make sure you have the TestPyPI and PyPI accounts and the corresponding `~/.pypirc` set up.

1. Build source distribution:

   ```
   > python setup.py sdist bdist_wheel
   ```

2. Upload to PyPITest:

   ```
   > twine upload dist/sockeye-${VERSION}.tar.gz dist/sockeye-${VERSION}-py3-none-
   ↪any.whl -r pypitest
   ```

3. In a new python environment check that the package is installable

   ```
   > pip install -i https://testpypi.python.org/pypi sockeye
   ```

4. Upload to PyPI

   ```
   > twine upload dist/sockeye-${VERSION}.tar.gz dist/sockeye-${VERSION}-py3-none-
   ↪any.whl
   ```

When pushing a new git tag to the repository, it is automatically built and deployed to PyPI as a new version via Travis.

2.3.7 Code of Conduct

This project has adopted the Amazon Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opensource-codeofconduct@amazon.com with any additional questions or comments.

2.3.8 Licensing

See the LICENSE file for our project's licensing. We will ask you confirm the licensing of your contribution.

We may ask you to sign a Contributor License Agreement (CLA) for larger changes.

2.4 Changelog

All notable changes to the project are documented in this file.

Version numbers are of the form `1.0.0`. Any version bump in the last digit is backwards-compatible, in that a model trained with the previous version can still be used for translation with the new version. Any bump in the second digit indicates a backwards-incompatible change, e.g. due to changing the architecture or simply modifying model parameter names. Note that Sockeye has checks in place to not translate with an old model that was trained with an incompatible version.

Each version section may have have subsections for: *Added*, *Changed*, *Removed*, *Deprecated*, and *Fixed*.

2.4.1 [1.18.25]

2.4.1.1 Changed

- Update requirements to use MKL versions of MXNet for fast CPU operation.

2.4.2 [1.18.24]

2.4.2.1 Added

- Dockerfiles and convenience scripts for running `fast_align` to generate lexical tables. These tables can be used to create top-K lexicons for faster decoding via vocabulary selection (documentation).

2.4.2.2 Changed

- Updated default top-K lexicon size from 20 to 200.

2.4.3 [1.18.23]

2.4.3.1 Fixed

- Correctly create the convolutional embedding layers when the encoder is set to `transformer-with-conv-embed`. Previously no convolutional layers were added so that a standard Transformer model was trained instead.

2.4.4 [1.18.22]

2.4.4.1 Fixed

- Make sure the default bucket is large enough with word based batching when the source is longer than the target (Previously there was an edge case where the memory usage was sub-optimal with word based batching and longer source than target sentences).

2.4.5 [1.18.21]

2.4.5.1 Fixed

- Constrained decoding was missed a crucial cast
- Fixed test cases that should have caught this

2.4.6 [1.18.20]

2.4.6.1 Changed

- Transformer parametrization flags (model size, # of attention heads, feed-forward layer size) can now optionally defined separately for encoder & decoder. For example, to use a different transformer model size for the encoder, pass `--transformer-model-size 1024:512`.

2.4.7 [1.18.19]

2.4.7.1 Added

- LHUC is now supported in transformer models

2.4.8 [1.18.18]

2.4.8.1 Added

- [Experimental] Introducing the image captioning module. Type of models supported: ConvNet encoder - Sockeye NMT decoders. This includes also a feature extraction script, an image-text iterator that loads features, training and inference pipelines and a visualization script that loads images and captions. See this tutorial for its usage. This module is experimental therefore its maintenance is not fully guaranteed.

2.4.9 [1.18.17]

2.4.9.1 Changed

- Updated to MXNet 1.2
- Use of the new LayerNormalization operator to save GPU memory.

2.4.10 [1.18.16]

2.4.10.1 Fixed

- Removed summation of gradient arrays when logging gradients. This clogged the memory on the primary GPU device over time when many checkpoints were done. Gradient histograms are now logged to Tensorboard separated by device.

2.4.11 [1.18.15]

2.4.11.1 Added

- Added decoding with target-side lexical constraints (documentation in `tutorials/constraints`).

2.4.12 [1.18.14]

2.4.12.1 Added

- Introduced Sockeye Autopilot for single-command end-to-end system building. See the Autopilot documentation and run with: `sockeye-autopilot`. Autopilot is a `contrib` module with its own tests that are run periodically. It is not included in the comprehensive tests run for every commit.

2.4.13 [1.18.13]

2.4.13.1 Fixed

- Fixed two bugs with training resumption:
  1. removed overly strict assertion in the data iterator for model states before the first checkpoint.
  2. removed deletion of Tensorboard log directory.

2.4.13.2 Added

- Added support for config files. Command line parameters have precedence over the values read from the config file. Minimal working example: `python -m sockeye.train --config config.yaml` with contents of `config.yaml` as follows:

```
source: source.txt
target: target.txt
output: out
validation_source: valid.source.txt
validation_target: valid.target.txt
```

2.4.13.3 Changed

The full set of arguments is serialized to `out/args.yaml` at the beginning of training (before json was used).

2.4.14 [1.18.12]

2.4.14.1 Changed

- All source side sequences now get appended an additional end-of-sentence (EOS) symbol. This change is backwards compatible meaning that inference with older models will still work without the EOS symbol.

2.4.15 [1.18.11]

2.4.15.1 Changed

- Default training parameters have been changed to reflect the setup used in our arXiv paper. Specifically, the default is now to train a 6 layer Transformer model with word based batching. The only difference to the paper is that weight tying is still turned off by default, as there may be use cases in which tying the source and target vocabularies is not appropriate. Turn it on using `--weight-tying --weight-tying-type=src_trg_softmax`. Additionally, BLEU scores from a checkpoint decoder are now monitored by default.

2.4.16 [1.18.10]

2.4.16.1 Fixed

- Re-allow early stopping w.r.t BLEU

2.4.17 [1.18.9]

2.4.17.1 Fixed

- Fixed a problem with lhuc boolean flags passed as None.

2.4.17.2 Added

- Reorganized beam search. Normalization is applied only to completed hypotheses, and pruning of hypotheses (logprob against highest-scoring completed hypothesis) can be specified with `--beam-prune X`
- Enabled stopping at first completed hypothesis with `--beam-search-stop first` (default is 'all')

2.4.18 [1.18.8]

2.4.18.1 Removed

- Removed tensorboard logging of embedding & output parameters at every checkpoint. This used a lot of disk space.

2.4.19 [1.18.7]

2.4.19.1 Added

- Added support for LHUC in RNN models (David Vilar, "Learning Hidden Unit Contribution for Adapting Neural Machine Translation Models" NAACL 2018)

2.4.19.2 Fixed

- Word based batching with very small batch sizes.

2.4.20 [1.18.6]

2.4.20.1 Fixed

- Fixed a problem with learning rate scheduler not properly being loaded when resuming training.

2.4.21 [1.18.5]

2.4.21.1 Fixed

- Fixed a problem with trainer not waiting for the last checkpoint decoder (#367).

2.4.22 [1.18.4]

2.4.22.1 Added

- Added options to control training length w.r.t number of updates/batches or number of samples: `--min-updates`, `--max-updates`, `--min-samples`, `--max-samples`.

2.4.23 [1.18.3]

2.4.23.1 Changed

- Training now supports training and validation data that contains empty segments. If a segment is empty, it is skipped during loading and a warning message including the number of empty segments is printed.

2.4.24 [1.18.2]

2.4.24.1 Changed

- Removed combined linear projection of keys & values in source attention transformer layers for performance improvements.
- The topk operator is performed in a single operation during batch decoding instead of running in a loop over each sentence, bringing speed benefits in batch decoding.

2.4.25 [1.18.1]

2.4.25.1 Added

- Added Tensorboard logging for all parameter values and gradients as histograms/distributions. The logged values correspond to the current batch at checkpoint time.

2.4.25.2 Changed

- Tensorboard logging now is done with the MXNet compatible 'mxboard' that supports logging of all kinds of events (scalars, histograms, embeddings, etc.). If installed, training events are written out to Tensorboard compatible even files automatically.

2.4.25.3 Removed

- Removed the `--use-tensorboard` argument from `sockeye.train`. Tensorboard logging is now enabled by default if `mxboard` is installed.

2.4.26 [1.18.0]

2.4.26.1 Changed

- Change default target vocab name in model folder to `vocab.trg.0.json`
- Changed serialization format of top-k lexica to pickle/Numpy instead of JSON.
- `sockeye-lexicon` now supports two subcommands: create & inspect. The former provides the same functionality as the previous CLI. The latter allows users to pass source words to the top-k lexicon to inspect the set of allowed target words.

2.4.26.2 Added

- Added ability to choose a smaller `k` at decoding runtime for lexicon restriction.

2.4.27 [1.17.5]

2.4.27.1 Added

- Added a flag `--strip-unknown-words` to `sockeye.translate` to remove any `<unk>` symbols from the output strings.

2.4.28 [1.17.4]

2.4.28.1 Added

- Added a flag `--fixed-param-names` to prevent certain parameters from being optimized during training. This is useful if you want to keep pre-trained embeddings fixed during training.
- Added a flag `--dry-run` to `sockeye.train` to not perform any actual training, but print statistics about the model and mode of operation.

2.4.29 [1.17.3]

2.4.29.1 Changed

- `sockeye.evaluate` can now handle multiple hypotheses files by simply specifying `--hypotheses file1 file2...`. For each metric the mean and standard deviation will be reported across files.

2.4.30 [1.17.2]

2.4.30.1 Added

- Optionally store the beam search history to a `json` output using the `beam_store` output handler.

2.4.30.2 Changed

- Use stack operator instead of expand_dims + concat in RNN decoder. Reduces memory usage.

2.4.31 [1.17.1]

2.4.31.1 Changed

- Updated to MXNet 1.1.0

2.4.32 [1.17.0]

2.4.32.1 Added

- Source factors, as described in

  Linguistic Input Features Improve Neural Machine Translation (Sennrich & Haddow, WMT 2016) PDF bibtex

  Additional source factors are enabled by passing `--source-factors file1 [file2 ...]` (`-sf`), where file1, etc. are token-parallel to the source (`-s`). An analogous parameter, `--validation-source-factors`, is used to pass factors for validation data. The flag `--source-factors-num-embed D1 [D2 ...]` denotes the embedding dimensions and is required if source factor files are given. Factor embeddings are concatenated to the source embeddings dimension (`--num-embed`).

  At test time, the input sentence and its factors can be passed in via STDIN or command-line arguments.

  - For STDIN, the input and factors should be in a token-based factored format, e.g., `word1|factor1|factor2|... w2|f1|f2|... ...1`.
  - You can also use file arguments, which mirrors training: `--input` takes the path to a file containing the source, and `--input-factors` a list of files containing token-parallel factors. At test time, an exception is raised if the number of expected factors does not match the factors passed along with the input.

- Removed bias parameters from multi-head attention layers of the transformer.

2.4.33 [1.16.6]

2.4.33.1 Changed

- Loading/Saving auxiliary parameters of the models. Before aux parameters were not saved or used for initialization. Therefore the parameters of certain layers were ignored (e.g., BatchNorm) and randomly initialized. This change enables to properly load, save and initialize the layers which use auxiliary parameters.

2.4.34 [1.16.5]

2.4.34.1 Changed

- Device locking: Only one process will be acquiring GPUs at a time. This will lead to consecutive device ids whenever possible.

2.4.35 [1.16.4]

2.4.35.1 Changed

- Internal change: Standardized all data to be batch-major both at training and at inference time.

2.4.36 [1.16.3]

2.4.36.1 Changed

- When a device lock file exists and the process has no write permissions for the lock file we assume that the device is locked. Previously this lead to an permission denied exception. Please note that in this scenario we an not detect if the original Sockeye process did not shut down gracefully. This is not an issue when the sockeye process has write permissions on existing lock files as in that case locking is based on file system locks, which cease to exist when a process exits.

2.4.37 [1.16.2]

2.4.37.1 Changed

- Changed to a custom speedometer that tracks samples/sec AND words/sec. The original MXNet speedometer did not take variable batch sizes due to word-based batching into account.

2.4.38 [1.16.1]

2.4.38.1 Fixed

- Fixed entry points in `setup.py`.

2.4.39 [1.16.0]

2.4.39.1 Changed

- Update to MXNet 1.0.0 which adds more advanced indexing features, benefitting the beam search implementation.
- `--kvstore` now accepts 'nccl' value. Only works if MXNet was compiled with `USE_NCCL=1`.

2.4.39.2 Added

- `--gradient-compression-type` and `--gradient-compression-threshold` flags to use gradient compression. See MXNet FAQ on Gradient Compression.

2.4.40 [1.15.8]

2.4.40.1 Fixed

- Taking the BOS and EOS tag into account when calculating the maximum input length at inference.

2.4.41 [1.15.7]

2.4.41.1 Fixed

- fixed a problem with `--num-samples-per-shard` flag not being parsed as int.

2.4.42 [1.15.6]

2.4.42.1 Added

- New CLI `sockeye.prepare_data` for preprocessing the training data only once before training, potentially splitting large datasets into shards. At training time only one shard is loaded into memory at a time, limiting the maximum memory usage.

2.4.42.2 Changed

- Instead of using the `--source` and `--target` arguments `sockeye.train` now accepts a `--prepared-data` argument pointing to the folder containing the preprocessed and sharded data. Using the raw training data is still possible and now consumes less memory.

2.4.43 [1.15.5]

2.4.43.1 Added

- Optionally apply query, key and value projections to the source and target hidden vectors in the CNN model before applying the attention mechanism. CLI parameter: `--cnn-project-qkv`.

2.4.44 [1.15.4]

2.4.44.1 Added

- A warning will be printed if the checkpoint decoder slows down training.

2.4.45 [1.15.3]

2.4.45.1 Added

- Exposing the xavier random number generator through `--weight-init-xavier-rand-type`.

2.4.46 [1.15.2]

2.4.46.1 Added

- Exposing MXNet's Nesterov Accelerated Gradient, Adadelta and Adadelta optimizers.

2.4.47 [1.15.1]

2.4.47.1 Added

- A tool that initializes embedding weights with pretrained word representations, `sockeye.init_embedding`.

2.4.48 [1.15.0]

2.4.48.1 Added

- Added support for Swish-1 (SiLU) activation to transformer models (Ramachandran et al. 2017: Searching for Activation Functions, Elfwing et al. 2017: Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning). Use `--transformer-activation-type swish1`.
- Added support for GELU activation to transformer models (Hendrycks and Gimpel 2016: Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. Use `--transformer-activation-type gelu`.

2.4.49 [1.14.3]

2.4.49.1 Changed

- Fast decoding for transformer models. Caches keys and values of self-attention before softmax. Changed decoding flag `--bucket-width` to apply only to source length.

2.4.50 [1.14.2]

2.4.50.1 Added

- Gradient norm clipping (`--gradient-clipping-type`) and monitoring.

2.4.50.2 Changed

- Changed `--clip-gradient` to `--gradient-clipping-threshold` for consistency.

2.4.51 [1.14.1]

2.4.51.1 Changed

- Sorting sentences during decoding before splitting them into batches.
- Default chunk size: The default chunk size when batching is enabled is now batch_size * 500 during decoding to avoid users accidentally forgetting to increase the chunk size.

2.4.52 [1.14.0]

2.4.52.1 Changed

- Downscaled fixed positional embeddings for CNN models.
- Renamed `--monitor-bleu` flag to `--decode-and-evaluate` to illustrate that it computes other metrics in addition to BLEU.

2.4.52.2 Added

- `--decode-and-evaluate-use-cpu` flag to use CPU for decoding validation data.
- `--decode-and-evaluate-device-id` flag to use a separate GPU device for validation decoding. If not specified, the existing and still default behavior is to use the last acquired GPU for training.

2.4.53 [1.13.2]

2.4.53.1 Added

- A tool that extracts specified parameters from params.x into a .npz file for downstream applications or analysis.

2.4.54 [1.13.1]

2.4.54.1 Added

- Added chrF metric (Popovic 2015: chrF: character n-gram F-score for automatic MT evaluation) to Sockeye. sockeye.evaluate now accepts `bleu` and `chrf` as values for `--metrics`

2.4.55 [1.13.0]

2.4.55.1 Fixed

- Transformer models do not ignore `--num-embed` anymore as they did silently before. As a result there is an error thrown if `--num-embed` != `--transformer-model-size`.
- Fixed the attention in upper layers (`--rnn-attention-in-upper-layers`), which was previously not passed correctly to the decoder.

2.4.55.2 Removed

- Removed RNN parameter (un-)packing and support for FusedRNNCells (removed `--use-fused-rnns` flag). These were not used, not correctly initialized, and performed worse than regular RNN cells. Moreover, they made the code much more complex. RNN models trained with previous versions are no longer compatible.
- Removed the lexical biasing functionality (Arthur ETAL'16) (removed arguments `--lexical-bias` and `--learn-lexical-bias`).

2.4.56 [1.12.2]

2.4.56.1 Changed

- Updated to MXNet 0.12.1, which includes an important bug fix for CPU decoding.

2.4.57 [1.12.1]

2.4.57.1 Changed

- Removed dependency on sacrebleu pip package. Now imports directly from `contrib/`.

2.4.58 [1.12.0]

2.4.58.1 Changed

- Transformers now always use the linear output transformation after combining attention heads, even if input & output depth do not differ.

2.4.59 [1.11.2]

2.4.59.1 Fixed

- Fixed a bug where vocabulary slice padding was defaulting to CPU context. This was affecting decoding on GPUs with very small vocabularies.

2.4.60 [1.11.1]

2.4.60.1 Fixed

- Fixed an issue with the use of `ignore` in `CrossEntropyMetric::cross_entropy_smoothed`. This was affecting runs with Eve optimizer and label smoothing. Thanks @kobenaxie for reporting.

2.4.61 [1.11.0]

2.4.61.1 Added

- Lexicon-based target vocabulary restriction for faster decoding. New CLI for top-k lexicon creation, sockeye.lexicon. New translate CLI argument `--restrict-lexicon`.

2.4.61.2 Changed

- Bleu computation based on Sacrebleu.

2.4.62 [1.10.5]

2.4.62.1 Fixed

- Fixed yet another bug with the data iterator.

2.4.63 [1.10.4]

2.4.63.1 Fixed

- Fixed a bug with the revised data iterator not correctly appending EOS symbols for variable-length batches. This reverts part of the commit added in 1.10.1 but is now correct again.

2.4.64 [1.10.3]

2.4.64.1 Changed

- Fixed a bug with max_observed_{source,target}_len being computed on the complete data set, not only on the sentences actually added to the buckets based on `--max_seq_len`.

2.4.65 [1.10.2]

2.4.65.1 Added

- `--max-num-epochs` flag to train for a maximum number of passes through the training data.

2.4.66 [1.10.1]

2.4.66.1 Changed

- Reduced memory footprint when creating data iterators: integer sequences are streamed from disk when being assigned to buckets.

2.4.67 [1.10.0]

2.4.67.1 Changed

- Updated MXNet dependency to 0.12 (w/ MKL support by default).
- Changed `--smoothed-cross-entropy-alpha` to `--label-smoothing`. Label smoothing should now require significantly less memory due to its addition to MXNet's `SoftmaxOutput` operator.
- `--weight-normalization` now applies not only to convolutional weight matrices, but to output layers of all decoders. It is also independent of weight tying.
- Transformers now use `--embed-dropout`. Before they were using `--transformer-dropout-prepost` for this.
- Transformers now scale their embedding vectors before adding fixed positional embeddings. This turns out to be crucial for effective learning.
- `.param` files now use 5 digit identifiers to reduce risk of overflowing with many checkpoints.

2.4.67.2 Added

- Added CUDA 9.0 requirements file.
- `--loss-normalization-type`. Added a new flag to control loss normalization. New default is to normalize by the number of valid, non-PAD tokens instead of the batch size.
- `--weight-init-xavier-factor-type`. Added new flag to control Xavier factor type when `--weight-init=xavier`.
- `--embed-weight-init`. Added new flag for initialization of embeddings matrices.

2.4.67.3 Removed

- `--smoothed-cross-entropy-alpha` argument. See above.
- `--normalize-loss` argument. See above.

2.4.68 [1.9.0]

2.4.68.1 Added

- Batch decoding. New options for the translate CLI: `--batch-size` and `--chunk-size`. Translator.translate() now accepts and returns lists of inputs and outputs.

2.4.69 [1.8.4]

2.4.69.1 Added

- Exposing the MXNet KVStore through the `--kvstore` argument, potentially enabling distributed training.

2.4.70 [1.8.3]

2.4.70.1 Added

- Optional smart rollback of parameters and optimizer states after updating the learning rate if not improved for x checkpoints. New flags: `--learning-rate-decay-param-reset`, `--learning-rate-decay-optimizer-states-reset`

2.4.71 [1.8.2]

2.4.71.1 Fixed

- The RNN variational dropout mask is now independent of the input (previously any zero initial state led to the first state being canceled).
- Correctly pass `self.dropout_inputs` float to `mx.sym.Dropout` in `VariationalDropoutCell`.

2.4.72 [1.8.1]

2.4.72.1 Changed

- Instead of truncating sentences exceeding the maximum input length they are now translated in chunks.

2.4.73 [1.8.0]

2.4.73.1 Added

- Convolutional decoder.
- Weight normalization (for CNN only so far).
- Learned positional embeddings for the transformer.

2.4.73.2 Changed

- `--attention-*` CLI params renamed to `--rnn-attention-*`.
- `--transformer-no-positional-encodings` generalized to `--transformer-positional-embedding-type`.

2.5 Frequently Asked Questions

2.5.1 What does Sockeye mean?

Sockeye is a salmon found in the Northern Pacific Ocean.

2.5.2 What does 'Operator _zeros cannot be run; requires at least one of FCompute' mean?

If you get the following or a similar error message:

```
mxnet.base.MXNetError: [16:16:21] src/c_api/c_api_ndarray.cc:392: Operator _zeros␣
↪cannot be run; requires at least one of FCompute<xpu>, NDArrayFunction,␣
↪FCreateOperator be registered
```

this means you are running a version of MXNet that does not include GPU instructions. Try installing a version with GPU instructions such as `mxnet-cu80mkl` or `mxnet-cu75mkl`.

2.6 Python Modules

2.6.1 sockeye.arguments module

Defines commandline arguments for the main CLIs with reasonable defaults.

class sockeye.arguments.**ConfigArgumentParser**(**args*, ***kwargs*)

Bases: argparse.ArgumentParser

Extension of argparse.ArgumentParser supporting config files.

The option –config is added automatically and expects a YAML serialized dictionary, similar to the return value of parse_args(). Command line parameters have precendence over config file values. Usage should be transparent, just substitute argparse.ArgumentParser with this class.

Extended from https://stackoverflow.com/questions/28579661/getting-required-option-from-namespace-in-python

sockeye.arguments.**file_or_stdin**()

Returns a file descriptor from stdin or opening a file from a given path.

Return type Callable

sockeye.arguments.**int_greater_or_equal**(*threshold*)

Returns a method that can be used in argument parsing to check that the argument is greater or equal to *threshold*.

Parameters **threshold** (int) – The threshold that we assume the cli argument value is greater or equal to.

Return type Callable

Returns A method that can be used as a type in argparse.

sockeye.arguments.**learning_schedule**()

Returns a method that can be used in argument parsing to check that the argument is a valid learning rate schedule string.

Return type Callable

Returns A method that can be used as a type in argparse.

sockeye.arguments.**multiple_values**(*num_values=0*, *greater_or_equal=None*, *data_type=<class 'int'>*)

Returns a method to be used in argument parsing to parse a string of the form "<val>:<val>[:<val>...]" into a tuple of values of type data_type.

Parameters

- **num_values** (int) – Optional number of ints required.
- **greater_or_equal** (Optional[float]) – Optional constraint that all values should be greater or equal to this value.
- **data_type** (Callable) – Type of values. Default: int.

Return type Callable

Returns Method for parsing.

sockeye.arguments.**regular_file**()

Returns a method that can be used in argument parsing to check the argument is a regular file or a symbolic link, but not, e.g., a process substitution.

Return type Callable

Returns A method that can be used as a type in argparse.

sockeye.arguments.**regular_folder**()

Returns a method that can be used in argument parsing to check the argument is a directory.

Return type Callable

Returns A method that can be used as a type in argparse.

sockeye.arguments.**simple_dict**()

A simple dictionary format that does not require spaces or quoting.

Supported types: bool, int, float

Return type Callable

Returns A method that can be used as a type in argparse.

2.6.2 sockeye.average module

Average parameters from multiple model checkpoints. Checkpoints can be either specified manually or automatically chosen according to one of several strategies. The default strategy of simply selecting the top-scoring N points works well in practice.

sockeye.average.**average**(*param_paths*)

Averages parameters from a list of .params file paths.

Parameters **param_paths** (Iterable[str]) – List of paths to parameter files.

Return type Dict[str, NDArray]

Returns Averaged parameter dictionary.

sockeye.average.**find_checkpoints**(*model_path*, *size=4*, *strategy='best'*, *metric='perplexity'*)

Finds N best points from .metrics file according to strategy.

Parameters

- **model_path** (str) – Path to model.
- **size** – Number of checkpoints to combine.

- **strategy** – Combination strategy.
- **metric** (str) – Metric according to which checkpoints are selected. Corresponds to columns in model/metrics file.

Return type List[str]

Returns List of paths corresponding to chosen checkpoints.

sockeye.average.**main**()
Commandline interface to average parameters.

2.6.3 sockeye.checkpoint_decoder module

Implements a thin wrapper around Translator to compute BLEU scores on (a sample of) validation data during training.

class sockeye.checkpoint_decoder.**CheckpointDecoder**(*context*, *inputs*, *references*, *model*, *max_input_len=None*, *batch_size=16*, *beam_size=5*, *bucket_width_source=10*, *length_penalty_alpha=1.0*, *length_penalty_beta=0.0*, *softmax_temperature=None*, *max_output_length_num_stds=2*, *ensemble_mode='linear'*, *sample_size=-1*, *random_seed=42*)

Bases: object

Decodes a (random sample of a) dataset using parameters at given checkpoint and computes BLEU against references.

Parameters

- **context** (Context) – MXNet context to bind the model to.
- **inputs** (List[str]) – Path(s) to file containing input sentences (and their factors).
- **references** (str) – Path to file containing references.
- **model** (str) – Model to load.
- **max_input_len** (Optional[int]) – Maximum input length.
- **batch_size** (int) – Batch size.
- **beam_size** (int) – Size of the beam.
- **bucket_width_source** (int) – Source bucket width.
- **length_penalty_alpha** (float) – Alpha factor for the length penalty
- **length_penalty_beta** (float) – Beta factor for the length penalty
- **softmax_temperature** (Optional[float]) – Optional parameter to control steepness of softmax distribution.
- **max_output_length_num_stds** (int) – Number of standard deviations as safety margin for maximum output length.
- **ensemble_mode** (str) – Ensemble mode: linear or log_linear combination.

- **sample_size** (int) – Maximum number of sentences to sample and decode. If <=0, all sentences are used.
- **random_seed** (int) – Random seed for sampling. Default: 42.

decode_and_evaluate(*checkpoint=None*, *output_name='/dev/null'*)
Decodes data set and evaluates given a checkpoint.

Parameters

- **checkpoint** (Optional[int]) – Checkpoint to load parameters from.
- **output_name** (str) – Filename to write translations to. Defaults to /dev/null.

Return type Dict[str, float]

Returns Mapping of metric names to scores.

2.6.4 sockeye.convolution module

Convolutional layers.

class sockeye.convolution.**ConvolutionBlock**(*config*, *pad_type*, *prefix*)
Bases: object

A Convolution-GLU block consists of the 2 following sublayers: 1. Dropout (optional) 1. A Convolution (padded either both to the left and to the right or just to the left). 2. An activation: Either a Gated Linear Unit or any other activation supported by MXNet.

Parameters

- **config** (*ConvolutionConfig*) – Configuration for Convolution block.
- **pad_type** (str) – 'left' or 'centered'. 'left' only pads to the left (for decoding the target sequence). 'centered' pads on both sides (for encoding the source sequence).
- **prefix** (str) – Name prefix for symbols of this block.

step(*data*)
Run convolution over a single position. The data must be exactly as wide as the convolution filters.

Parameters **data** – Shape: (batch_size, kernel_width, num_hidden).

Returns Single result of a convolution. Shape: (batch_size, 1, num_hidden).

class sockeye.convolution.**ConvolutionConfig**(*kernel_width*, *num_hidden*, *act_type='glu'*, *weight_normalization=False*)
Bases: sockeye.config.Config

Configuration for a stack of convolutions with Gated Linear Units between layers, similar to Gehring et al. 2017.

Parameters

- **kernel_width** (int) – Kernel size for 1D convolution.
- **num_hidden** (int) – Size of hidden representation after convolution.
- **act_type** (str) – The type of activation to use.

2.6.5 sockeye.coverage module

Defines the dynamic source encodings ('coverage' mechanisms) for encoder/decoder networks as used in Tu et al. (2016).

class sockeye.coverage.**ActivationCoverage**(*coverage_num_hidden*, *activation*, *layer_normalization*)

Bases: *sockeye.coverage.Coverage*

Implements a coverage mechanism whose updates are performed by a Perceptron with configurable activation function.

Parameters

- **coverage_num_hidden** (int) – Number of hidden units for coverage vectors.
- **activation** (str) – Type of activation for Perceptron.
- **layer_normalization** (bool) – If true, applies layer normalization before non-linear activation.

on(*source*, *source_length*, *source_seq_len*)

Returns callable to be used for updating coverage vectors in a sequence decoder.

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Coverage callable.

class sockeye.coverage.**CountCoverage**

Bases: *sockeye.coverage.Coverage*

Coverage class that accumulates the attention weights for each source word.

on(*source*, *source_length*, *source_seq_len*)

Returns callable to be used for updating coverage vectors in a sequence decoder.

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Coverage callable.

class sockeye.coverage.**Coverage**(*prefix='cov_'*)

Bases: object

Generic coverage class. Similar to Attention classes, a coverage instance returns a callable, update_coverage(), function when self.on() is called.

on(*source*, *source_length*, *source_seq_len*)

Returns callable to be used for updating coverage vectors in a sequence decoder.

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Coverage callable.

class sockeye.coverage.**CoverageConfig**(*type*, *num_hidden*, *layer_normalization*)
Bases: sockeye.config.Config

Coverage configuration.

Parameters

- **type** (str) – Coverage name.
- **num_hidden** (int) – Number of hidden units for coverage networks.
- **layer_normalization** (bool) – Apply layer normalization to coverage networks.

class sockeye.coverage.**GRUCoverage**(*coverage_num_hidden*, *layer_normalization*)
Bases: *sockeye.coverage.Coverage*

Implements a GRU whose state is the coverage vector.

TODO: This implementation is slightly inefficient since the source is fed in at every step. It would be better to pre-compute the mapping of the source but this will likely mean opening up the GRU.

Parameters

- **coverage_num_hidden** (int) – Number of hidden units for coverage vectors.
- **layer_normalization** (bool) – If true, applies layer normalization for each gate in the GRU cell.

on(*source*, *source_length*, *source_seq_len*)
Returns callable to be used for updating coverage vectors in a sequence decoder.

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Coverage callable.

sockeye.coverage.**get_coverage**(*config*)
Returns a Coverage instance.

Parameters **config** (*CoverageConfig*) – Coverage configuration.

Return type *Coverage*

Returns Instance of Coverage.

sockeye.coverage.**mask_coverage**(*coverage*, *source_length*)
Masks all coverage scores that are outside the actual sequence.

Parameters

- **coverage** (Symbol) – Input coverage vector. Shape: (batch_size, seq_len, coverage_num_hidden).
- **source_length** (Symbol) – Source length. Shape: (batch_size,).

Return type Symbol

Returns Masked coverage vector. Shape: (batch_size, seq_len, coverage_num_hidden).

2.6.6 sockeye.data_io module

Implements data iterators and I/O related functions for sequence-to-sequence models.

class sockeye.data_io.**BucketBatchSize**(*bucket*, *batch_size*, *average_words_per_batch*)
Bases: object

Parameters

- **bucket** (Tuple[int, int]) – The corresponding bucket.
- **batch_size** (int) – Number of sequences in each batch.
- **average_words_per_batch** (float) – Approximate number of non-padding tokens in each batch.

class sockeye.data_io.**DataConfig**(*data_statistics*, *max_seq_len_source*, *max_seq_len_target*, *num_source_factors*, *source_with_eos=False*)
Bases: sockeye.config.Config

Stores data statistics relevant for inference.

class sockeye.data_io.**DataInfo**(*sources*, *target*, *source_vocabs*, *target_vocab*, *shared_vocab*, *num_shards*)
Bases: sockeye.config.Config

Stores training data information that is not relevant for inference.

class sockeye.data_io.**DataStatistics**(*num_sents*, *num_discarded*, *num_tokens_source*, *num_tokens_target*, *num_unks_source*, *num_unks_target*, *max_observed_len_source*, *max_observed_len_target*, *size_vocab_source*, *size_vocab_target*, *length_ratio_mean*, *length_ratio_std*, *buckets*, *num_sents_per_bucket*, *mean_len_target_per_bucket*)
Bases: sockeye.config.Config

class sockeye.data_io.**FileListReader**(*fname*, *path*)
Bases: typing.Iterator

Reads sequence samples from path provided in a file.

Parameters

- **fname** (str) – File name containing a list of relative paths.
- **path** (str) – Path to read data from, which is prefixed to the relative paths of fname.

class sockeye.data_io.**LengthStatistics**(*num_sents*, *length_ratio_mean*, *length_ratio_std*)
Bases: sockeye.config.Config

class sockeye.data_io.**MetaBaseParallelSampleIter**
Bases: abc.ABC

class sockeye.data_io.**ParallelDataSet**(*source*, *target*, *label*)
Bases: collections.abc.Sized

Bucketed parallel data set with labels

fill_up(*bucket_batch_sizes*, *fill_up*, *seed=42*)
Returns a new dataset with buckets filled up using the specified fill-up strategy.

Parameters

- **bucket_batch_sizes** (List[*BucketBatchSize*]) – Bucket batch sizes.

- **fill_up** (str) – Fill-up strategy.
- **seed** (int) – The random seed used for sampling sentences to fill up.

Return type *ParallelDataSet*

Returns New dataset with buckets filled up to the next multiple of batch size

static load(*fname*)
Loads a dataset from a binary .npy file.

Return type *ParallelDataSet*

save(*fname*)
Saves the dataset to a binary .npy file.

class sockeye.data_io.**RawParallelDatasetLoader**(*buckets*, *eos_id*, *pad_id*, *dtype='float32'*)

Bases: object

Loads a data set of variable-length parallel source/target sequences into buckets of NDArrays.

Parameters

- **buckets** (List[Tuple[int, int]]) – Bucket list.
- **eos_id** (int) – End-of-sentence id.
- **pad_id** (int) – Padding id.
- **eos_id** – Unknown id.
- **dtype** (str) – Data type.

class sockeye.data_io.**SequenceReader**(*path*, *vocabulary=None*, *add_bos=False*, *add_eos=False*, *limit=None*)

Bases: typing.Iterable

Reads sequence samples from path and (optionally) creates integer id sequences. Streams from disk, instead of loading all samples into memory. If vocab is None, the sequences in path are assumed to be integers coded as strings. Empty sequences are yielded as None.

Parameters

- **path** (str) – Path to read data from.
- **vocab** – Optional mapping from strings to integer ids.
- **add_bos** (bool) – Whether to add Beginning-Of-Sentence (BOS) symbol.
- **limit** (Optional[int]) – Read limit.

sockeye.data_io.**are_token_parallel**(*sequences*)
Returns True if all sequences in the list have the same length.

Return type bool

sockeye.data_io.**calculate_length_statistics**(*source_iterables*, *target_iterable*, *max_seq_len_source*, *max_seq_len_target*)
Returns mean and standard deviation of target-to-source length ratios of parallel corpus.

Parameters

- **source_iterables** (Sequence[Iterable[Any]]) – Source sequence readers.
- **target_iterable** (Iterable[Any]) – Target sequence reader.
- **max_seq_len_source** (int) – Maximum source sequence length.

- **max_seq_len_target** (int) – Maximum target sequence length.

Return type *LengthStatistics*

Returns The number of sentences as well as the mean and standard deviation of target to source length ratios.

sockeye.data_io.**create_sequence_readers**(*sources*, *target*, *vocab_sources*, *vocab_target*)

Create source readers with EOS and target readers with BOS.

Parameters

- **sources** (List[str]) – The file names of source data and factors.
- **target** (str) – The file name of the target data.
- **vocab_sources** (List[Dict[str, int]]) – The source vocabularies.
- **vocab_target** (Dict[str, int]) – The target vocabularies.

Return type Tuple[List[*SequenceReader*[]], *SequenceReader*[]]

Returns The source sequence readers and the target reader.

sockeye.data_io.**define_bucket_batch_sizes**(*buckets*, *batch_size*, *batch_by_words*, *batch_num_devices*, *data_target_average_len*)

Computes bucket-specific batch sizes (sentences, average_words).

If sentence-based batching: number of sentences is the same for each batch, determines the number of words. Hence all batch sizes for each bucket are equal.

If word-based batching: number of sentences for each batch is set to the multiple of number of devices that produces the number of words closest to the target batch size. Average target sentence length (non-padding symbols) is used for word number calculations.

Parameters

- **buckets** (List[Tuple[int, int]]) – Bucket list.
- **batch_size** (int) – Batch size.
- **batch_by_words** (bool) – Batch by words.
- **batch_num_devices** (int) – Number of devices.
- **data_target_average_len** (List[Optional[float]]) – Optional average target length for each bucket.

Return type List[*BucketBatchSize*]

sockeye.data_io.**define_buckets**(*max_seq_len*, *step=10*)

Returns a list of integers defining bucket boundaries. Bucket boundaries are created according to the following policy: We generate buckets with a step size of step until the final bucket fits max_seq_len. We then limit that bucket to max_seq_len (difference between semi-final and final bucket may be less than step).

Parameters

- **max_seq_len** (int) – Maximum bucket size.
- **step** – Distance between buckets.

Return type List[int]

Returns List of bucket sizes.

sockeye.data_io.**define_empty_source_parallel_buckets**(*max_seq_len_target*, *bucket_width=10*)

Returns (source, target) buckets up to (None, max_seq_len_target). The source is empty since it is supposed to not contain data that can be bucketized. The target is used as reference to create the buckets.

Parameters

- **max_seq_len_target** (int) – Maximum target bucket size.
- **bucket_width** (int) – Width of buckets on longer side.

Return type List[Tuple[int, int]]

sockeye.data_io.**define_parallel_buckets**(*max_seq_len_source*, *max_seq_len_target*, *bucket_width=10*, *length_ratio=1.0*)

Returns (source, target) buckets up to (max_seq_len_source, max_seq_len_target). The longer side of the data uses steps of bucket_width while the shorter side uses steps scaled down by the average target/source length ratio. If one side reaches its max_seq_len before the other, width of extra buckets on that side is fixed to that max_seq_len.

Parameters

- **max_seq_len_source** (int) – Maximum source bucket size.
- **max_seq_len_target** (int) – Maximum target bucket size.
- **bucket_width** (int) – Width of buckets on longer side.
- **length_ratio** (float) – Length ratio of data (target/source).

Return type List[Tuple[int, int]]

sockeye.data_io.**describe_data_and_buckets**(*data_statistics*, *bucket_batch_sizes*)

Describes statistics across buckets

sockeye.data_io.**get_batch_indices**(*data*, *bucket_batch_sizes*)

Returns a list of index tuples that index into the bucket and the start index inside a bucket given the batch size for a bucket. These indices are valid for the given dataset.

Parameters

- **data** (*ParallelDataSet*) – Data to create indices for.
- **bucket_batch_sizes** (List[*BucketBatchSize*]) – Bucket batch sizes.

Return type List[Tuple[int, int]]

Returns List of 2d indices.

sockeye.data_io.**get_bucket**(*seq_len*, *buckets*)

Given sequence length and a list of buckets, return corresponding bucket.

Parameters

- **seq_len** (int) – Sequence length.
- **buckets** (List[int]) – List of buckets.

Return type Optional[int]

Returns Chosen bucket.

sockeye.data_io.**get_default_bucket_key**(*buckets*)

Returns the default bucket from a list of buckets, i.e. the largest bucket.

Parameters **buckets** (List[Tuple[int, int]]) – List of buckets.

Return type Tuple[int, int]

Returns The largest bucket in the list.

sockeye.data_io.**get_num_shards**(*num_samples*, *samples_per_shard*, *min_num_shards*)
Returns the number of shards.

Parameters

- **num_samples** (int) – Number of training data samples.
- **samples_per_shard** (int) – Samples per shard.
- **min_num_shards** (int) – Minimum number of shards.

Return type int

Returns Number of shards.

sockeye.data_io.**get_parallel_bucket**(*buckets*, *length_source*, *length_target*)
Returns bucket index and bucket from a list of buckets, given source and target length. Returns (None, None) if no bucket fits.

Parameters

- **buckets** (List[Tuple[int, int]]) – List of buckets.
- **length_source** (int) – Length of source sequence.
- **length_target** (int) – Length of target sequence.

Return type Tuple[Optional[int], Optional[Tuple[int, int]]]

Returns Tuple of (bucket index, bucket), or (None, None) if not fitting.

sockeye.data_io.**get_permutations**(*bucket_counts*)
Returns the indices of a random permutation for each bucket and the corresponding inverse permutations that can restore the original order of the data if applied to the permuted data.

Parameters **bucket_counts** (List[int]) – The number of elements per bucket.

Return type Tuple[List[NDArray], List[NDArray]]

Returns For each bucket a permutation and inverse permutation is returned.

sockeye.data_io.**get_target_bucket**(*buckets*, *length_target*)
Returns bucket index and bucket from a list of buckets, given source and target length. Returns (None, None) if no bucket fits.

Parameters

- **buckets** (List[Tuple[int, int]]) – List of buckets.
- **length_target** (int) – Length of target sequence.

Return type Optional[Tuple[int, Tuple[int, int]]]

Returns Tuple of (bucket index, bucket), or (None, None) if not fitting.

sockeye.data_io.**get_training_data_iters**(*sources*, *target*, *validation_sources*, *validation_target*, *source_vocabs*, *target_vocab*, *source_vocab_paths*, *target_vocab_path*, *shared_vocab*, *batch_size*, *batch_by_words*, *batch_num_devices*, *fill_up*, *max_seq_len_source*, *max_seq_len_target*, *bucketing*, *bucket_width*)
Returns data iterators for training and validation data.

Parameters

- **sources** (List[str]) – Path to source training data (with optional factor data paths).

- **target** (str) – Path to target training data.
- **validation_sources** (List[str]) – Path to source validation data (with optional factor data paths).
- **validation_target** (str) – Path to target validation data.
- **source_vocabs** (List[Dict[str, int]]) – Source vocabulary and optional factor vocabularies.
- **target_vocab** (Dict[str, int]) – Target vocabulary.
- **source_vocab_paths** (List[Optional[str]]) – Path to source vocabulary.
- **target_vocab_path** (Optional[str]) – Path to target vocabulary.
- **shared_vocab** (bool) – Whether the vocabularies are shared.
- **batch_size** (int) – Batch size.
- **batch_by_words** (bool) – Size batches by words rather than sentences.
- **batch_num_devices** (int) – Number of devices batches will be parallelized across.
- **fill_up** (str) – Fill-up strategy for buckets.
- **max_seq_len_source** (int) – Maximum source sequence length.
- **max_seq_len_target** (int) – Maximum target sequence length.
- **bucketing** (bool) – Whether to use bucketing.
- **bucket_width** (int) – Size of buckets.

Return type Tuple[DataIter, DataIter, *DataConfig*, *DataInfo*]

Returns Tuple of (training data iterator, validation data iterator, data config).

sockeye.data_io.**get_validation_data_iter**(*data_loader*, *validation_sources*, *validation_target*, *buckets*, *bucket_batch_sizes*, *source_vocabs*, *target_vocab*, *max_seq_len_source*, *max_seq_len_target*, *batch_size*, *fill_up*)

Returns a ParallelSampleIter for the validation data.

Return type DataIter

sockeye.data_io.**ids2strids**(*ids*)

Returns a string representation of a sequence of integers.

Parameters **ids** (Iterable[int]) – Sequence of integers.

Return type str

Returns String sequence

sockeye.data_io.**parallel_iter**(*source_iters*, *target_iterable*)

Yields parallel source(s), target sequences from iterables. Checks for token parallelism in source sequences. Skips pairs where element in at least one iterable is None. Checks that all iterables have the same number of elements.

sockeye.data_io.**read_content**(*path*, *limit=None*)

Returns a list of tokens for each line in path up to a limit.

Parameters

- **path** (str) – Path to files containing sentences.

- **limit** (Optional[int]) – How many lines to read from path.

Return type Iterator[List[str]]

Returns Iterator over lists of words.

sockeye.data_io.**shard_data**(*source_fnames*, *target_fname*, *source_vocabs*, *target_vocab*, *num_shards*, *buckets*, *length_ratio_mean*, *length_ratio_std*, *output_prefix*)

Assign int-coded source/target sentence pairs to shards at random.

Parameters

- **source_fnames** (List[str]) – The path to the source text (and optional token-parallel factor files).
- **target_fname** (str) – The file name of the target file.
- **source_vocabs** (List[Dict[str, int]]) – Source vocabulary (and optional source factor vocabularies).
- **target_vocab** (Dict[str, int]) – Target vocabulary.
- **num_shards** (int) – The total number of shards.
- **buckets** (List[Tuple[int, int]]) – Bucket list.
- **length_ratio_mean** (float) – Mean length ratio.
- **length_ratio_std** (float) – Standard deviation of length ratios.
- **output_prefix** (str) – The prefix under which the shard files will be created.

Return type Tuple[List[Tuple[List[str], str, Datastatistics]], *DataStatistics*]

Returns Tuple of source (and source factor) file names, target file names and statistics for each shard, as well as global statistics.

sockeye.data_io.**strids2ids**(*tokens*)

Returns sequence of integer ids given a sequence of string ids.

Parameters **tokens** (Iterable[str]) – List of integer tokens.

Return type List[int]

Returns List of word ids.

sockeye.data_io.**tokens2ids**(*tokens*, *vocab*)

Returns sequence of integer ids given a sequence of tokens and vocab.

Parameters

- **tokens** (Iterable[str]) – List of string tokens.
- **vocab** (Dict[str, int]) – Vocabulary (containing UNK symbol).

Return type List[int]

Returns List of word ids.

2.6.7 sockeye.decoder module

Decoders for sequence-to-sequence models.

class sockeye.decoder.**ConvolutionalDecoder**(*config*, *prefix='decoder_'*)

Bases: *sockeye.decoder.Decoder*

Convolutional decoder similar to Gehring et al. 2017.

The decoder consists of an embedding layer, positional embeddings, and layers of convolutional blocks with residual connections.

Notable differences to Gehring et al. 2017:

- Here the context vectors are created from the last encoder state (instead of using the last encoder state as the key and the sum of the encoder state and the source embedding as the value)
- The encoder gradients are not scaled down by 1/(2 * num_attention_layers).
- Residual connections are not scaled down by math.sqrt(0.5).
- Attention is computed in the hidden dimension instead of the embedding dimension (removes need for training several projection matrices)

Parameters

- **config** (*ConvolutionalDecoderConfig*) – Configuration for convolutional decoder.
- **prefix** (str) – Name prefix for symbols of this decoder.

decode_sequence(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*, *target_embed*, *target_embed_lengths*, *target_embed_max_length*)

Decodes a sequence of embedded target words and returns sequence of last decoder representations for each time step.

Parameters

- **source_encoded** (Symbol) – Encoded source: (batch_size, source_encoded_max_length, encoder_depth).
- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.
- **target_embed** (Symbol) – Embedded target sequence. Shape: (batch_size, target_embed_max_length, target_num_embed).
- **target_embed_lengths** (Symbol) – Lengths of embedded target sequences. Shape: (batch_size,).
- **target_embed_max_length** (int) – Dimension of the embedded target sequence.

Return type Symbol

Returns Decoder data. Shape: (batch_size, target_embed_max_length, decoder_depth).

decode_step(*step*, *target_embed_prev*, *source_encoded_max_length*, **states*)

Decodes a single time step given the current step, the previous embedded target word, and previous decoder states. Returns decoder representation for the next prediction, attention probabilities, and next decoder states. Implementations can maintain an arbitrary number of states.

Parameters

- **step** (int) – Global step of inference procedure, starts with 1.
- **target_embed_prev** (Symbol) – Previous target word embedding. Shape: (batch_size, target_num_embed).

- **source_encoded_max_length** (int) – Length of encoded source time dimension.
- **states** (Symbol) – Arbitrary list of decoder states.

Return type Tuple[Symbol, Symbol, List[Symbol]]

Returns logit inputs, attention probabilities, next decoder states.

get_max_seq_len()

Return type Optional[int]

Returns The maximum length supported by the decoder if such a restriction exists.

get_num_hidden()

Return type int

Returns The representation size of this decoder.

init_states(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*)

Returns a list of symbolic states that represent the initial states of this decoder. Used for inference.

Parameters

- **source_encoded** (Symbol) – Encoded source. Shape: (batch_size, source_encoded_max_length, encoder_depth).
- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.

Return type List[Symbol]

Returns List of symbolic initial states.

reset()

Reset decoder method. Used for inference.

state_shapes(*batch_size*, *target_max_length*, *source_encoded_max_length*, *source_encoded_depth*)

Returns a list of shape descriptions given batch size, encoded source max length and encoded source depth. Used for inference.

Parameters

- **batch_size** (int) – Batch size during inference.
- **target_max_length** (int) – Current target sequence length.
- **source_encoded_max_length** (int) – Size of encoder time dimension.
- **source_encoded_depth** (int) – Depth of encoded source.

Return type List[DataDesc]

Returns List of shape descriptions.

state_variables(*target_max_length*)

Returns the list of symbolic variables for this decoder to be used during inference.

Parameters **target_max_length** (int) – Current target sequence lengths.

Return type List[Symbol]

Returns List of symbolic variables.

class sockeye.decoder.**ConvolutionalDecoderConfig**(*cnn_config*, *max_seq_len_target*, *num_embed*, *encoder_num_hidden*, *num_layers*, *positional_embedding_type*, *project_qkv=False*, *hidden_dropout=0.0*, *dtype='float32'*)

Bases: sockeye.config.Config

Convolutional decoder configuration.

Parameters

- **cnn_config** (*ConvolutionConfig*) – Configuration for the convolution block.
- **max_seq_len_target** (int) – Maximum target sequence length.
- **num_embed** (int) – Target word embedding size.
- **encoder_num_hidden** (int) – Number of hidden units of the encoder.
- **num_layers** (int) – The number of convolutional layers.
- **positional_embedding_type** (str) – The type of positional embedding.
- **hidden_dropout** (float) – Dropout probability on next decoder hidden state.
- **dtype** (str) – Data type.

class sockeye.decoder.**Decoder**(*dtype*)

Bases: abc.ABC

Generic decoder interface. A decoder needs to implement code to decode a target sequence known in advance (decode_sequence), and code to decode a single word given its decoder state (decode_step). The latter is typically used for inference graphs in beam search. For the inference module to be able to keep track of decoder's states a decoder provides methods to return initial states (init_states), state variables and their shapes.

Parameters **dtype** – Data type.

decode_sequence(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*, *target_embed*, *target_embed_lengths*, *target_embed_max_length*)

Decodes a sequence of embedded target words and returns sequence of last decoder representations for each time step.

Parameters

- **source_encoded** (Symbol) – Encoded source: (batch_size, source_encoded_max_length, encoder_depth).
- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.
- **target_embed** (Symbol) – Embedded target sequence. Shape: (batch_size, target_embed_max_length, target_num_embed).
- **target_embed_lengths** (Symbol) – Lengths of embedded target sequences. Shape: (batch_size,).
- **target_embed_max_length** (int) – Dimension of the embedded target sequence.

Return type Symbol

Returns Decoder data. Shape: (batch_size, target_embed_max_length, decoder_depth).

decode_step(*step*, *target_embed_prev*, *source_encoded_max_length*, **states*)
Decodes a single time step given the current step, the previous embedded target word, and previous decoder states. Returns decoder representation for the next prediction, attention probabilities, and next decoder states. Implementations can maintain an arbitrary number of states.

Parameters

- **step** (int) – Global step of inference procedure, starts with 1.
- **target_embed_prev** (Symbol) – Previous target word embedding. Shape: (batch_size, target_num_embed).
- **source_encoded_max_length** (int) – Length of encoded source time dimension.
- **states** (Symbol) – Arbitrary list of decoder states.

Return type Tuple[Symbol, Symbol, List[Symbol]]

Returns logit inputs, attention probabilities, next decoder states.

classmethod get_decoder(*config*, *prefix*)
Creates decoder based on config type.

Parameters

- **config** (Union[*RecurrentDecoderConfig*, TransformerConfig, *ConvolutionalDecoderConfig*]) – Decoder config.
- **prefix** (str) – Prefix to prepend for decoder.

Return type *Decoder*

Returns Decoder instance.

get_max_seq_len()

Return type Optional[int]

Returns The maximum length supported by the decoder if such a restriction exists.

get_num_hidden()

Return type int

Returns The representation size of this decoder.

init_states(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*)
Returns a list of symbolic states that represent the initial states of this decoder. Used for inference.

Parameters

- **source_encoded** (Symbol) – Encoded source. Shape: (batch_size, source_encoded_max_length, encoder_depth).
- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.

Return type List[Symbol]

Returns List of symbolic initial states.

classmethod register(*config_type*, *suffix*)
Registers decoder type for configuration. Suffix is appended to decoder prefix.

Parameters

- **config_type** (Type[Union[Recurrentdecoderconfig, TransformerConfig, Convolutionaldecoderconfig]]) – Configuration type for decoder.
- **suffix** (str) – String to append to decoder prefix.

Returns Class decorator.

reset()
Reset decoder method. Used for inference.

state_shapes(*batch_size*, *target_max_length*, *source_encoded_max_length*, *source_encoded_depth*)
Returns a list of shape descriptions given batch size, encoded source max length and encoded source depth. Used for inference.

Parameters

- **batch_size** (int) – Batch size during inference.
- **target_max_length** (int) – Current target sequence length.
- **source_encoded_max_length** (int) – Size of encoder time dimension.
- **source_encoded_depth** (int) – Depth of encoded source.

Return type List[DataDesc]

Returns List of shape descriptions.

state_variables(*target_max_length*)
Returns the list of symbolic variables for this decoder to be used during inference.

Parameters **target_max_length** (int) – Current target sequence lengths.

Return type List[Symbol]

Returns List of symbolic variables.

class sockeye.decoder.**RecurrentDecoder**(*config*, *prefix='decoder_rnn_'*)
Bases: *sockeye.decoder.Decoder*

RNN Decoder with attention. The architecture is based on Luong et al, 2015: Effective Approaches to Attention-based Neural Machine Translation.

Parameters

- **config** (*RecurrentDecoderConfig*) – Configuration for recurrent decoder.
- **prefix** (str) – Decoder symbol prefix.

decode_sequence(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*, *target_embed*, *target_embed_lengths*, *target_embed_max_length*)
Decodes a sequence of embedded target words and returns sequence of last decoder representations for each time step.

Parameters

- **source_encoded** (Symbol) – Encoded source: (batch_size, source_encoded_max_length, encoder_depth).
- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.

- **target_embed** (Symbol) – Embedded target sequence. Shape: (batch_size, target_embed_max_length, target_num_embed).
- **target_embed_lengths** (Symbol) – Lengths of embedded target sequences. Shape: (batch_size,).
- **target_embed_max_length** (int) – Dimension of the embedded target sequence.

Return type Symbol

Returns Decoder data. Shape: (batch_size, target_embed_max_length, decoder_depth).

decode_step(*step*, *target_embed_prev*, *source_encoded_max_length*, **states*)

Decodes a single time step given the current step, the previous embedded target word, and previous decoder states. Returns decoder representation for the next prediction, attention probabilities, and next decoder states. Implementations can maintain an arbitrary number of states.

Parameters

- **step** (int) – Global step of inference procedure, starts with 1.
- **target_embed_prev** (Symbol) – Previous target word embedding. Shape: (batch_size, target_num_embed).
- **source_encoded_max_length** (int) – Length of encoded source time dimension.
- **states** (Symbol) – Arbitrary list of decoder states.

Return type Tuple[Symbol, Symbol, List[Symbol]]

Returns logit inputs, attention probabilities, next decoder states.

get_initial_state(*source_encoded*, *source_encoded_length*)

Computes initial states of the decoder, hidden state, and one for each RNN layer. Optionally, init states for RNN layers are computed using 1 non-linear FC with the last state of the encoder as input.

Parameters

- **source_encoded** (Symbol) – Concatenated encoder states. Shape: (batch_size, source_seq_len, encoder_num_hidden).
- **source_encoded_length** (Symbol) – Lengths of source sequences. Shape: (batch_size,).

Return type *RecurrentDecoderState*

Returns Decoder state.

get_num_hidden()

Return type int

Returns The representation size of this decoder.

get_rnn_cells()

Returns a list of RNNCells used by this decoder.

Return type List[BaseRNNCell]

init_states(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*)

Returns a list of symbolic states that represent the initial states of this decoder. Used for inference.

Parameters

- **source_encoded** (Symbol) – Encoded source. Shape: (batch_size, source_encoded_max_length, encoder_depth).

- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.

Return type List[Symbol]

Returns List of symbolic initial states.

reset()
Calls reset on the RNN cell.

state_shapes(*batch_size*, *target_max_length*, *source_encoded_max_length*, *source_encoded_depth*)
Returns a list of shape descriptions given batch size, encoded source max length and encoded source depth. Used for inference.

Parameters

- **batch_size** (int) – Batch size during inference.
- **target_max_length** (int) – Current target sequence length.
- **source_encoded_max_length** (int) – Size of encoder time dimension.
- **source_encoded_depth** (int) – Depth of encoded source.

Return type List[DataDesc]

Returns List of shape descriptions.

state_variables(*target_max_length*)
Returns the list of symbolic variables for this decoder to be used during inference.

Parameters **target_max_length** (int) – Current target sequence lengths.

Return type List[Symbol]

Returns List of symbolic variables.

class sockeye.decoder.**RecurrentDecoderConfig**(*max_seq_len_source*, *rnn_config*, *attention_config*, *hidden_dropout=0.0*, *state_init='last'*, *state_init_lhuc=False*, *context_gating=False*, *layer_normalization=False*, *attention_in_upper_layers=False*, *dtype='float32'*, *enc_last_hidden_concat_to_embedding=False*)
Bases: sockeye.config.Config

Recurrent decoder configuration.

Parameters

- **max_seq_len_source** (int) – Maximum source sequence length
- **rnn_config** (*RNNConfig*) – RNN configuration.
- **attention_config** (*AttentionConfig*) – Attention configuration.
- **hidden_dropout** (float) – Dropout probability on next decoder hidden state.
- **state_init** (str) – Type of RNN decoder state initialization: zero, last, average.
- **state_init_lhuc** (bool) – Apply LHUC for encoder to decoder initialization.
- **context_gating** (bool) – Whether to use context gating.

- **layer_normalization** (bool) – Apply layer normalization.
- **attention_in_upper_layers** (bool) – Pass the attention value to all layers in the decoder.
- **enc_last_hidden_concat_to_embedding** (bool) – Concatenate the last hidden representation of the encoder to the input of the decoder (e.g., context + current embedding).
- **dtype** (str) – Data type.

class sockeye.decoder.**RecurrentDecoderState**(*hidden*, *layer_states*)

Bases: tuple

RecurrentDecoder state.

Parameters

- **hidden** – Hidden state after attention mechanism. Shape: (batch_size, num_hidden).
- **layer_states** – Hidden states for RNN layers of RecurrentDecoder. Shape: List[(batch_size, rnn_num_hidden)]

hidden

Alias for field number 0

layer_states

Alias for field number 1

class sockeye.decoder.**TransformerDecoder**(*config*, *prefix='decoder_transformer_'*)

Bases: *sockeye.decoder.Decoder*

Transformer decoder as in Vaswani et al, 2017: Attention is all you need. In training, computation scores for each position of the known target sequence are compouted in parallel, yielding most of the speedup. At inference time, the decoder block is evaluated again and again over a maximum length input sequence that is initially filled with zeros and grows during beam search with predicted tokens. Appropriate masking at every time-step ensures correct self-attention scores and is updated with every step.

Parameters

- **config** (TransformerConfig) – Transformer configuration.
- **prefix** (str) – Name prefix for symbols of this decoder.

decode_sequence(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*, *target_embed*, *target_embed_lengths*, *target_embed_max_length*)

Decodes a sequence of embedded target words and returns sequence of last decoder representations for each time step.

Parameters

- **source_encoded** (Symbol) – Encoded source: (batch_size, source_encoded_max_length, encoder_depth).
- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.
- **target_embed** (Symbol) – Embedded target sequence. Shape: (batch_size, target_embed_max_length, target_num_embed).
- **target_embed_lengths** (Symbol) – Lengths of embedded target sequences. Shape: (batch_size,).
- **target_embed_max_length** (int) – Dimension of the embedded target sequence.

Return type Symbol

Returns Decoder data. Shape: (batch_size, target_embed_max_length, decoder_depth).

decode_step(*step*, *target_embed_prev*, *source_encoded_max_length*, **states*)
Decodes a single time step given the current step, the previous embedded target word, and previous decoder states. Returns decoder representation for the next prediction, attention probabilities, and next decoder states. Implementations can maintain an arbitrary number of states.

Parameters

- **step** (int) – Global step of inference procedure, starts with 1.
- **target_embed_prev** (Symbol) – Previous target word embedding. Shape: (batch_size, target_num_embed).
- **source_encoded_max_length** (int) – Length of encoded source time dimension.
- **states** (Symbol) – Arbitrary list of decoder states.

Return type Tuple[Symbol, Symbol, List[Symbol]]

Returns logit inputs, attention probabilities, next decoder states.

get_max_seq_len()

Return type Optional[int]

Returns The maximum length supported by the decoder if such a restriction exists.

get_num_hidden()

Return type int

Returns The representation size of this decoder.

init_states(*source_encoded*, *source_encoded_lengths*, *source_encoded_max_length*)
Returns a list of symbolic states that represent the initial states of this decoder. Used for inference.

Parameters

- **source_encoded** (Symbol) – Encoded source. Shape: (batch_size, source_encoded_max_length, encoder_depth).
- **source_encoded_lengths** (Symbol) – Lengths of encoded source sequences. Shape: (batch_size,).
- **source_encoded_max_length** (int) – Size of encoder time dimension.

Return type List[Symbol]

Returns List of symbolic initial states.

reset()
Reset decoder method. Used for inference.

state_shapes(*batch_size*, *target_max_length*, *source_encoded_max_length*, *source_encoded_depth*)
Returns a list of shape descriptions given batch size, encoded source max length and encoded source depth. Used for inference.

Parameters

- **batch_size** (int) – Batch size during inference.
- **target_max_length** (int) – Current target sequence length.
- **source_encoded_max_length** (int) – Size of encoder time dimension.

- **source_encoded_depth** (int) – Depth of encoded source.

Return type List[DataDesc]

Returns List of shape descriptions.

state_variables(*target_max_length*)
Returns the list of symbolic variables for this decoder to be used during inference.

Parameters **target_max_length** (int) – Current target sequence length.

Return type List[Symbol]

Returns List of symbolic variables.

2.6.8 sockeye.embeddings module

Command-line tool to inspect model embeddings.

sockeye.embeddings.**compute_sims**(*inputs*, *normalize*)
Returns a matrix with pair-wise similarity scores between inputs. Similarity score is (normalized) Euclidean distance. 'Similarity with self' is masked to large negative value.

Parameters

- **inputs** (NDArray) – NDArray of inputs.
- **normalize** (bool) – Whether to normalize to unit-length.

Return type NDArray

Returns NDArray with pairwise similarities of same shape as inputs.

sockeye.embeddings.**main**()
Command-line tool to inspect model embeddings.

sockeye.embeddings.**nearest_k**(*similarity_matrix*, *query_word_id*, *k*, *gamma=1.0*)
Returns values and indices of k items with largest similarity.

Parameters

- **similarity_matrix** (NDArray) – Similarity matrix.
- **query_word_id** (int) – Query word id.
- **k** (int) – Number of closest items to retrieve.
- **gamma** (float) – Parameter to control distribution steepness.

Return type Iterable[Tuple[int, float]]

Returns List of indices and values of k nearest elements.

2.6.9 sockeye.encoder module

Encoders for sequence-to-sequence models.

class sockeye.encoder.**AddLearnedPositionalEmbeddings**(*num_embed*, *max_seq_len*, *prefix*, *embed_weight=None*, *dtype='float32'*)
Bases: *sockeye.encoder.PositionalEncoder*

Takes an encoded sequence and adds positional embeddings to it, which are learned jointly. Note that this will limited the maximum sentence length during decoding.

Parameters

- **num_embed** (int) – Embedding size.
- **max_seq_len** (int) – Maximum sequence length.
- **prefix** (str) – Name prefix for symbols of this encoder.
- **embed_weight** (Optional[Symbol]) – Optionally use an existing embedding matrix instead of creating a new one.
- **dtype** (str) – Data type.

encode(*data*, *data_length*, *seq_len*)

Parameters

- **data** (Symbol) – (batch_size, source_seq_len, num_embed)
- **data_length** (Optional[Symbol]) – (batch_size,)
- **seq_len** (int) – sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns (batch_size, source_seq_len, num_embed)

encode_positions(*positions*, *data*)

Parameters

- **positions** (Symbol) – (batch_size,)
- **data** (Symbol) – (batch_size, num_embed)

Return type Symbol

Returns (batch_size, num_embed)

get_max_seq_len()

Return type Optional[int]

Returns The maximum length supported by the encoder if such a restriction exists.

get_num_hidden()

Return type int

Returns The representation size of this encoder.

class sockeye.encoder.**AddSinCosPositionalEmbeddings**(*num_embed*, *prefix*, *scale_up_input*, *scale_down_positions*, *dtype='float32'*)

Bases: *sockeye.encoder.PositionalEncoder*

Takes an encoded sequence and adds fixed positional embeddings as in Vaswani et al, 2017 to it.

Parameters

- **num_embed** (int) – Embedding size.
- **prefix** (str) – Name prefix for symbols of this encoder.
- **scale_up_input** (bool) – If True, scales input data up by num_embed ** 0.5.
- **scale_down_positions** (bool) – If True, scales positional embeddings down by num_embed ** -0.5.

- **dtype** (str) – Data type.

encode(*data*, *data_length*, *seq_len*)

Parameters

- **data** (Symbol) – (batch_size, source_seq_len, num_embed)
- **data_length** (Optional[Symbol]) – (batch_size,)
- **seq_len** (int) – sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns (batch_size, source_seq_len, num_embed)

encode_positions(*positions*, *data*)

Parameters

- **positions** (Symbol) – (batch_size,)
- **data** (Symbol) – (batch_size, num_embed)

Return type Symbol

Returns (batch_size, num_embed)

get_num_hidden()

Return type int

Returns The representation size of this encoder.

class sockeye.encoder.**BiDirectionalRNNEncoder**(*rnn_config*, *prefix='encoder_birnn_'*, *layout='TNC'*, *encoder_class=<class 'sockeye.encoder.RecurrentEncoder'>*)

Bases: *sockeye.encoder.Encoder*

An encoder that runs a forward and a reverse RNN over input data. States from both RNNs are concatenated together.

Parameters

- **rnn_config** (*RNNConfig*) – RNN configuration.
- **prefix** – Prefix for variable names.
- **layout** – Data layout.
- **encoder_class** (Callable) – Recurrent encoder class to use.

encode(*data*, *data_length*, *seq_len*)

Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Symbol) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_num_hidden()

Return the representation size of this encoder.

Return type int

get_rnn_cells()
Returns a list of RNNCells used by this encoder.

Return type List[BaseRNNCell]

class sockeye.encoder.**ConvertLayout**(*target_layout*, *num_hidden*, *dtype='float32'*)
Bases: *sockeye.encoder.Encoder*

Converts batch major data to time major by swapping the first dimension and setting the __layout__ attribute.

Parameters

- **target_layout** (str) – The target layout to convert to (C.BATCH_MAJOR or C.TIMEMAJOR).
- **num_hidden** (int) – The number of hidden units of the previous encoder.
- **dtype** (str) – Data type.

encode(*data*, *data_length*, *seq_len*)
Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Optional[Symbol]) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_num_hidden()

Return type int

Returns The representation size of this encoder.

class sockeye.encoder.**ConvolutionalEmbeddingConfig**(*num_embed*, *output_dim=None*, *max_filter_width=8*, *num_filters=(200, 200, 250, 250, 300, 300, 300, 300)*, *pool_stride=5*, *num_highway_layers=4*, *dropout=0.0*, *add_positional_encoding=False*, *dtype='float32'*)
Bases: sockeye.config.Config

Convolutional embedding encoder configuration.

Parameters

- **num_embed** (int) – Input embedding size.
- **output_dim** (Optional[int]) – Output segment embedding size.
- **max_filter_width** (int) – Maximum filter width for convolutions.
- **num_filters** (Tuple[int, ...]) – Number of filters of each width.
- **pool_stride** (int) – Stride for pooling layer after convolutions.
- **num_highway_layers** (int) – Number of highway layers for segment embeddings.

- **dropout** (float) – Dropout probability.
- **add_positional_encoding** (bool) – Dropout probability.
- **dtype** (str) – Data type.

class sockeye.encoder.**ConvolutionalEmbeddingEncoder**(*config*, *prefix='encoder_char_'*)

Bases: *sockeye.encoder.Encoder*

An encoder developed to map a sequence of character embeddings to a shorter sequence of segment embeddings using convolutional, pooling, and highway layers. More generally, it maps a sequence of input embeddings to a sequence of span embeddings.

- "Fully Character-Level Neural Machine Translation without Explicit Segmentation" Jason Lee; Kyunghyun Cho; Thomas Hofmann (https://arxiv.org/pdf/1610.03017.pdf)

Parameters

- **config** (*ConvolutionalEmbeddingConfig*) – Convolutional embedding config.
- **prefix** (str) – Name prefix for symbols of this encoder.

encode(*data*, *data_length*, *seq_len*)

Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Symbol) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data data, data_length, seq_len.

get_encoded_seq_len(*seq_len*)

Returns the size of the encoded sequence.

Return type int

get_num_hidden()

Return the representation size of this encoder.

Return type int

class sockeye.encoder.**ConvolutionalEncoder**(*config*, *prefix='encoder_cnn_'*)

Bases: *sockeye.encoder.Encoder*

Encoder that uses convolution instead of recurrent connections, similar to Gehring et al. 2017.

Parameters

- **config** (*ConvolutionalEncoderConfig*) – Configuration for convolutional encoder.
- **prefix** (str) – Name prefix for operations in this encoder.

encode(*data*, *data_length*, *seq_len*)

Encodes data with a stack of Convolution+GLU blocks given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data. Shape: (batch_size, seq_len, input_num_hidden).

- **data_length** (Symbol) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded version of the data.

get_num_hidden()

Return type int

Returns The representation size of this encoder.

class sockeye.encoder.**ConvolutionalEncoderConfig**(*num_embed*, *max_seq_len_source*, *cnn_config*, *num_layers*, *positional_embedding_type*, *dtype='float32'*)

Bases: sockeye.config.Config

Convolutional encoder configuration.

Parameters

- **cnn_config** (*ConvolutionConfig*) – CNN configuration.
- **num_layers** (int) – The number of convolutional layers on top of the embeddings.
- **positional_embedding_type** (str) – The type of positional embedding.
- **dtype** (str) – Data type.

class sockeye.encoder.**Embedding**(*config*, *prefix*, *embed_weight=None*, *is_source=False*)

Bases: *sockeye.encoder.Encoder*

Thin wrapper around MXNet's Embedding symbol. Works with both time- and batch-major data layouts.

Parameters

- **config** (*EmbeddingConfig*) – Embedding config.
- **prefix** (str) – Name prefix for symbols of this encoder.
- **embed_weight** (Optional[Symbol]) – Optionally use an existing embedding matrix instead of creating a new one.
- **is_source** (bool) – Whether this is the source embedding instance. Default: False.

encode(*data*, *data_length*, *seq_len*)

Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Optional[Symbol]) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_num_hidden()

Return the representation size of this encoder.

Return type int

class sockeye.encoder.**EmbeddingConfig**(*vocab_size*, *num_embed*, *dropout*, *factor_configs=None*, *dtype='float32'*)

Bases: sockeye.config.Config

class sockeye.encoder.**Encoder**(*dtype*)

Bases: abc.ABC

Generic encoder interface.

Parameters **dtype** – Data type.

encode(*data*, *data_length*, *seq_len*)

Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Optional[Symbol]) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_encoded_seq_len(*seq_len*)

Return type int

Returns The size of the encoded sequence.

get_max_seq_len()

Return type Optional[int]

Returns The maximum length supported by the encoder if such a restriction exists.

get_num_hidden()

Return type int

Returns The representation size of this encoder.

class sockeye.encoder.**EncoderSequence**(*encoders*, *dtype='float32'*)

Bases: *sockeye.encoder.Encoder*

A sequence of encoders is itself an encoder.

Parameters

- **encoders** (List[*Encoder*]) – List of encoders.
- **dtype** (str) – Data type.

append(*cls*, *infer_hidden=False*, ***kwargs*)

Extends sequence with new Encoder. 'dtype' gets passed into Encoder instance if not present in parameters and supported by specific Encoder type.

Parameters

- **cls** – Encoder type.
- **infer_hidden** (bool) – If number of hidden should be inferred from previous encoder.
- **kwargs** – Named arbitrary parameters for Encoder.

Return type *Encoder*

Returns Instance of Encoder.

encode(*data*, *data_length*, *seq_len*)
Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Symbol) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_encoded_seq_len(*seq_len*)
Returns the size of the encoded sequence.

Return type int

get_max_seq_len()

Return type Optional[int]

Returns The maximum length supported by the encoder if such a restriction exists.

get_num_hidden()
Return the representation size of this encoder.

Return type int

class sockeye.encoder.**FactorConfig**(*vocab_size*, *num_embed*)
Bases: sockeye.config.Config

class sockeye.encoder.**NoOpPositionalEmbeddings**(*num_embed*, *dtype='float32'*)
Bases: *sockeye.encoder.PositionalEncoder*

Simple NoOp pos embedding. It does not modify the data, but avoids lots of if statements.

Parameters **dtype** (str) – Data type.

encode(*data*, *data_length*, *seq_len*)
Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Optional[Symbol]) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

encode_positions(*positions*, *data*)
Add positional encodings to the data using the provided positions. :type positions: Symbol :param positions: (batch_size,) :type data: Symbol :param data: (batch_size, num_embed) :rtype: Symbol :return: (batch_size, num_embed)

get_num_hidden()

Return type int

Returns The representation size of this encoder.

class sockeye.encoder.**PassThroughEmbedding**(*config*)
Bases: *sockeye.encoder.Encoder*

This is an embedding which passes through an input symbol without doing any operation.

Parameters **config** (*PassThroughEmbeddingConfig*) – PassThroughEmbeddingConfig config.

encode(*data*, *data_length*, *seq_len=0*)
Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Optional[Symbol]) – Vector with sequence lengths.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_num_hidden()
Return the representation size of this encoder.

Return type int

class sockeye.encoder.**PassThroughEmbeddingConfig**
Bases: sockeye.config.Config

class sockeye.encoder.**PositionalEncoder**(*dtype*)
Bases: *sockeye.encoder.Encoder*

encode_positions(*positions*, *data*)
Add positional encodings to the data using the provided positions. :type positions: Symbol :param positions: (batch_size,) :type data: Symbol :param data: (batch_size, num_embed) :rtype: Symbol :return: (batch_size, num_embed)

class sockeye.encoder.**RecurrentEncoder**(*rnn_config*, *prefix='encoder_rnn_'*, *layout='TNC'*)
Bases: *sockeye.encoder.Encoder*

Uni-directional (multi-layered) recurrent encoder.

Parameters

- **rnn_config** (*RNNConfig*) – RNN configuration.
- **prefix** (str) – Prefix for variable names.
- **layout** (str) – Data layout.

encode(*data*, *data_length*, *seq_len*)
Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Optional[Symbol]) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_num_hidden()
Return the representation size of this encoder.

get_rnn_cells()
Returns RNNCells used in this encoder.

class sockeye.encoder.**RecurrentEncoderConfig**(*rnn_config*, *conv_config=None*, *reverse_input=False*, *dtype='float32'*)
Bases: sockeye.config.Config

Recurrent encoder configuration.

Parameters

- **rnn_config** (*RNNConfig*) – RNN configuration.
- **conv_config** (Optional[*ConvolutionalEmbeddingConfig*]) – Optional configuration for convolutional embedding.
- **reverse_input** (bool) – Reverse embedding sequence before feeding into RNN.
- **dtype** (str) – Data type.

class sockeye.encoder.**ReverseSequence**(*num_hidden*, *dtype='float32'*)
Bases: *sockeye.encoder.Encoder*

Reverses the input sequence. Requires time-major layout.

Parameters **dtype** (str) – Data type.

encode(*data*, *data_length*, *seq_len*)
Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Symbol) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data (data, data_length, seq_len).

get_num_hidden()

Returns The representation size of this encoder.

class sockeye.encoder.**TransformerEncoder**(*config*, *prefix='encoder_transformer_'*)
Bases: *sockeye.encoder.Encoder*

Non-recurrent encoder based on the transformer architecture in:

Attention Is All You Need, Figure 1 (left) Vaswani et al. (https://arxiv.org/pdf/1706.03762.pdf).

Parameters

- **config** (TransformerConfig) – Configuration for transformer encoder.
- **prefix** (str) – Name prefix for operations in this encoder.

encode(*data*, *data_length*, *seq_len*)
Encodes data given sequence lengths of individual examples and maximum sequence length.

Parameters

- **data** (Symbol) – Input data.
- **data_length** (Symbol) – Vector with sequence lengths.
- **seq_len** (int) – Maximum sequence length.

Return type Tuple[Symbol, Symbol, int]

Returns Encoded versions of input data data, data_length, seq_len.

get_num_hidden()
Return the representation size of this encoder.

Return type int

sockeye.encoder.**get_convolutional_encoder**(*config*, *prefix*)
Creates a convolutional encoder.

Parameters

- **config** (*ConvolutionalEncoderConfig*) – Configuration for convolutional encoder.
- **prefix** (str) – Prefix for variable names.

Return type *Encoder*

Returns Encoder instance.

sockeye.encoder.**get_recurrent_encoder**(*config*, *prefix*)
Returns an encoder stack with a bi-directional RNN, and a variable number of uni-directional forward RNNs.

Parameters

- **config** (*RecurrentEncoderConfig*) – Configuration for recurrent encoder.
- **prefix** (str) – Prefix for variable names.

Return type *Encoder*

Returns Encoder instance.

sockeye.encoder.**get_transformer_encoder**(*config*, *prefix*)
Returns a Transformer encoder, consisting of an embedding layer with positional encodings and a TransformerEncoder instance.

Parameters

- **config** (TransformerConfig) – Configuration for transformer encoder.
- **prefix** (str) – Prefix for variable names.

Return type *Encoder*

Returns Encoder instance.

2.6.10 sockeye.evaluate module

Evaluation CLI. Prints corpus BLEU

sockeye.evaluate.**raw_corpus_bleu**(*hypotheses*, *references*, *offset=0.01*)
Simple wrapper around sacreBLEU's BLEU without tokenization and smoothing.

Parameters

- **hypotheses** (Iterable[str]) – Hypotheses stream.
- **references** (Iterable[str]) – Reference stream.
- **offset** (Optional[float]) – Smoothing constant.

Return type float

Returns BLEU score as float between 0 and 1.

sockeye.evaluate.**raw_corpus_chrf**(*hypotheses*, *references*)
Simple wrapper around sacreBLEU's chrF implementation, without tokenization.

Parameters

- **hypotheses** (Iterable[str]) – Hypotheses stream.
- **references** (Iterable[str]) – Reference stream.

Return type float

Returns chrF score as float between 0 and 1.

2.6.11 sockeye.extract_parameters module

Extract specific parameters.

sockeye.extract_parameters.**extract**(*param_path*, *param_names*, *list_all*)
Extract specific parameters given their names.

Parameters

- **param_path** (str) – Path to the parameter file.
- **param_names** (List[str]) – Names of parameters to be extracted.
- **list_all** (bool) – List names of all available parameters.

Return type Dict[str, ndarray]

Returns Extracted parameter dictionary.

sockeye.extract_parameters.**main**()
Commandline interface to extract parameters.

2.6.12 sockeye.inference module

Code for inference/translation

class sockeye.inference.**BadTranslatorInput**(*sentence_id*, *tokens*)
Bases: *sockeye.inference.TranslatorInput*

class sockeye.inference.**InferenceModel**(*config*, *params_fname*, *context*, *beam_size*, *batch_size*, *softmax_temperature=None*, *max_output_length_num_stds=2*, *decoder_return_logit_inputs=False*, *cache_output_layer_w_b=False*, *forced_max_output_len=None*)
Bases: *sockeye.model.SockeyeModel*

InferenceModel is a SockeyeModel that supports three operations used for inference/decoding:

1. Encoder forward call: encode source sentence and return initial decoder states.
2. Decoder forward call: single decoder step: predict next word.

Parameters

- **config** (*ModelConfig*) – Configuration object holding details about the model.
- **params_fname** (str) – File with model parameters.

- **context** (Context) – MXNet context to bind modules to.
- **beam_size** (int) – Beam size.
- **batch_size** (int) – Batch size.
- **softmax_temperature** (Optional[float]) – Optional parameter to control steepness of softmax distribution.
- **max_output_length_num_stds** (int) – Number of standard deviations as safety margin for maximum output length.
- **decoder_return_logit_inputs** (bool) – Decoder returns inputs to logit computation instead of softmax over target vocabulary. Used when logits/softmax are handled separately.
- **cache_output_layer_w_b** (bool) – Cache weights and biases for logit computation.

initialize(*max_input_length*, *get_max_output_length_function*)
Delayed construction of modules to ensure multiple Inference models can agree on computing a common maximum output length.

Parameters

- **max_input_length** (int) – Maximum input length.
- **get_max_output_length_function** (Callable) – Callable to compute maximum output length.

max_supported_seq_len_source
If not None this is the maximally supported source length during inference (hard constraint).

Return type Optional[int]

max_supported_seq_len_target
If not None this is the maximally supported target length during inference (hard constraint).

Return type Optional[int]

num_source_factors
Returns the number of source factors of this InferenceModel (at least 1).

Return type int

run_decoder(*prev_word*, *bucket_key*, *model_state*)
Runs forward pass of the single-step decoder.

Return type Tuple[NDArray, NDArray, *ModelState*]

Returns Decoder stack output (logit inputs or probability distribution), attention scores, updated model state.

run_encoder(*source*, *source_max_length*)
Runs forward pass of the encoder. Encodes source given source length and bucket key. Returns encoder representation of the source, source_length, initial hidden state of decoder RNN, and initial decoder states tiled to beam size.

Parameters

- **source** (NDArray) – Integer-coded input tokens. Shape (batch_size, source length, num_source_factors).
- **source_max_length** (int) – Bucket key.

Return type *ModelState*

Returns Initial model state.

training_max_seq_len_source
The maximum sequence length on the source side during training.

Return type int

training_max_seq_len_target
The maximum sequence length on the target side during training.

Return type int

class sockeye.inference.**ModelState**(*states*)
Bases: object

A ModelState encapsulates information about the decoder states of an InferenceModel.

sort_state(*best_hyp_indices*)
Sorts states according to k-best order from last step in beam search.

class sockeye.inference.**TranslatedChunk**(*id*, *chunk_id*, *translation*)
Bases: tuple

Translation of a chunk of a sentence.

Parameters

- **id** – Id of the sentence.
- **chunk_id** – Id of the chunk.
- **translation** – The translation of the input chunk.

chunk_id
Alias for field number 1

id
Alias for field number 0

translation
Alias for field number 2

class sockeye.inference.**Translator**(*context*, *ensemble_mode*, *bucket_source_width*, *length_penalty*, *beam_prune*, *beam_search_stop*, *models*, *source_vocabs*, *target_vocab*, *restrict_lexicon=None*, *store_beam=False*, *strip_unknown_words=False*)
Bases: object

Translator uses one or several models to translate input. The translator holds a reference to vocabularies to convert between word ids and text tokens for input and translation strings.

Parameters

- **context** (Context) – MXNet context to bind modules to.
- **ensemble_mode** (str) – Ensemble mode: linear or log_linear combination.
- **length_penalty** (HybridBlock) – Length penalty instance.
- **beam_prune** (float) – Beam pruning difference threshold.
- **beam_search_stop** (str) – The stopping criterium.
- **models** (List[*InferenceModel*]) – List of models.
- **source_vocabs** (List[Dict[str, int]]) – Source vocabularies.
- **target_vocab** (Dict[str, int]) – Target vocabulary.

- **restrict_lexicon** (Optional[*TopKLexicon*]) – Top-k lexicon to use for target vocabulary restriction.
- **store_beam** (bool) – If True, store the beam search history and return it in the TranslatorOutput.
- **strip_unknown_words** (bool) – If True, removes any <unk> symbols from outputs.

translate(*trans_inputs*)

Batch-translates a list of TranslatorInputs, returns a list of TranslatorOutputs. Splits oversized sentences to sentence chunks of size less than max_input_length.

Parameters **trans_inputs** (List[*TranslatorInput*]) – List of TranslatorInputs as returned by make_input().

Return type List[*TranslatorOutput*]

Returns List of translation results.

class sockeye.inference.**TranslatorInput**(*sentence_id*, *tokens*, *factors=None*, *constraints=None*, *chunk_id=-1*)

Bases: object

Object required by Translator.translate().

Parameters

- **sentence_id** (int) – Sentence id.
- **tokens** (List[str]) – List of input tokens.
- **factors** (Optional[List[List[str]]]) – Optional list of additional factor sequences.
- **constraints** (Optional[List[List[str]]]) – Optional list of target-side constraints.
- **chunk_id** (int) – Chunk id. Defaults to -1.

chunks(*chunk_size*)

Takes a TranslatorInput (itself) and yields TranslatorInputs for chunks of size chunk_size.

Parameters **chunk_size** (int) – The maximum size of a chunk.

Return type Generator[Translatorinput, None, None]

Returns A generator of TranslatorInputs, one for each chunk created.

num_factors

Returns the number of factors of this instance.

Return type int

with_eos()

Return type *TranslatorInput*

Returns A new translator input with EOS appended to the tokens and factors.

class sockeye.inference.**TranslatorOutput**(*id*, *translation*, *tokens*, *attention_matrix*, *score*, *beam_histories=None*)

Bases: object

Output structure from Translator.

Parameters

- **id** (int) – Id of input sentence.

- **translation** (str) – Translation string without sentence boundary tokens.
- **tokens** (List[str]) – List of translated tokens.
- **attention_matrix** (ndarray) – Attention matrix. Shape: (target_length, source_length).
- **score** (float) – Negative log probability of generated translation.
- **beam_histories** (Optional[List[Dict[str, List[~T]]]]) – List of beam histories. The list will contain more than one history if it was split due to exceeding max_length.

sockeye.inference.**get_max_input_output_length**(*supported_max_seq_len_source*, *supported_max_seq_len_target*, *training_max_seq_len_source*, *length_ratio_mean*, *length_ratio_std*, *num_stds*, *forced_max_input_len=None*, *forced_max_output_len=None*)

Returns a function to compute maximum output length given a fixed number of standard deviations as a safety margin, and the current input length. It takes into account optional maximum source and target lengths.

Parameters

- **supported_max_seq_len_source** (Optional[int]) – The maximum source length supported by the models.
- **supported_max_seq_len_target** (Optional[int]) – The maximum target length supported by the models.
- **training_max_seq_len_source** (Optional[int]) – The maximum source length observed during training.
- **length_ratio_mean** (float) – The mean of the length ratio that was calculated on the raw sequences with special symbols such as EOS or BOS.
- **length_ratio_std** (float) – The standard deviation of the length ratio.
- **num_stds** (int) – The number of standard deviations the target length may exceed the mean target length (as long as the supported maximum length allows for this).
- **forced_max_input_len** (Optional[int]) – An optional overwrite of the maximum input length.
- **forced_max_output_len** (Optional[int]) – An optional overwrite of the maximum out length.

Return type Tuple[int, Callable]

Returns The maximum input length and a function to get the output length given the input length.

sockeye.inference.**load_models**(*context*, *max_input_len*, *beam_size*, *batch_size*, *model_folders*, *checkpoints=None*, *softmax_temperature=None*, *max_output_length_num_stds=2*, *decoder_return_logit_inputs=False*, *cache_output_layer_w_b=False*, *forced_max_output_len=None*, *override_dtype=None*)

Loads a list of models for inference.

Parameters

- **context** (Context) – MXNet context to bind modules to.
- **max_input_len** (Optional[int]) – Maximum input length.
- **beam_size** (int) – Beam size.

- **batch_size** (int) – Batch size.
- **model_folders** (List[str]) – List of model folders to load models from.
- **checkpoints** (Optional[List[int]]) – List of checkpoints to use for each model in model_folders. Use None to load best checkpoint.
- **softmax_temperature** (Optional[float]) – Optional parameter to control steepness of softmax distribution.
- **max_output_length_num_stds** (int) – Number of standard deviations to add to mean target-source length ratio to compute maximum output length.
- **decoder_return_logit_inputs** (bool) – Model decoders return inputs to logit computation instead of softmax over target vocabulary. Used when logits/softmax are handled separately.
- **cache_output_layer_w_b** (bool) – Models cache weights and biases for logit computation as NumPy arrays (used with restrict lexicon).
- **forced_max_output_len** (Optional[int]) – An optional overwrite of the maximum output length.
- **override_dtype** (Optional[str]) – Overrides dtype of encoder and decoder defined at training time to a different one.

Return type Tuple[List[*InferenceModel*], List[Dict[str, int]], Dict[str, int]]

Returns List of models, source vocabulary, target vocabulary, source factor vocabularies.

sockeye.inference.**make_input_from_factored_string**(*sentence_id*, *factored_string*, *translator*, *delimiter='|'*)

Returns a TranslatorInput object from a string with factor annotations on a token level, separated by delimiter. If translator does not require any source factors, the string is parsed as a plain token string.

Parameters

- **sentence_id** (int) – An integer id.
- **factored_string** (str) – An input string with additional factors per token, separated by delimiter.
- **translator** (*Translator*) – A translator object.
- **delimiter** (str) – A factor delimiter. Default: '|'.

Return type *TranslatorInput*

Returns A TranslatorInput.

sockeye.inference.**make_input_from_json_string**(*sentence_id*, *json_string*)

Returns a TranslatorInput object from a JSON object, serialized as a string.

Parameters

- **sentence_id** (int) – An integer id.
- **json_string** (str) – A JSON object serialized as a string that must contain a key "text", mapping to the input text, and optionally a key "factors" that maps to a list of strings, each of which representing a factor sequence for the input text.

Return type *TranslatorInput*

Returns A TranslatorInput.

sockeye.inference.**make_input_from_multiple_strings**(*sentence_id*, *strings*)

Returns a TranslatorInput object from multiple strings, where the first element corresponds to the surface tokens and the remaining elements to additional factors. All strings must parse into token sequences of the same length.

Parameters

- **sentence_id** (int) – An integer id.
- **strings** (List[str]) – A list of strings representing a factored input sequence.

Return type *TranslatorInput*

Returns A TranslatorInput.

sockeye.inference.**make_input_from_plain_string**(*sentence_id*, *string*)

Returns a TranslatorInput object from a plain string.

Parameters

- **sentence_id** (int) – An integer id.
- **string** (str) – An input string.

Return type *TranslatorInput*

Returns A TranslatorInput.

sockeye.inference.**models_max_input_output_length**(*models*, *num_stds*, *forced_max_input_len=None*, *forced_max_output_len=None*)

Returns a function to compute maximum output length given a fixed number of standard deviations as a safety margin, and the current input length. Mean and std are taken from the model with the largest values to allow proper ensembling of models trained on different data sets.

Parameters

- **models** (List[*InferenceModel*]) – List of models.
- **num_stds** (int) – Number of standard deviations to add as a safety margin. If -1, returned maximum output lengths will always be 2 * input_length.
- **forced_max_input_len** (Optional[int]) – An optional overwrite of the maximum input length.
- **forced_max_output_len** (Optional[int]) – An optional overwrite of the maximum output length.

Return type Tuple[int, Callable]

Returns The maximum input length and a function to get the output length given the input length.

2.6.13 sockeye.init_embedding module

Initializing Sockeye embedding weights with pretrained word representations. It also supports updating vocabulary-sized weights for a new vocabulary.

Quick usage:

```
python3 -m sockeye.init_embedding -w embed-in-src.npy embed-in-tgt.npy -i vocab-in-src.json vocab-in-tgt.json -o vocab-out-src.json vocab-out-tgt.json -n source_embed_weight target_embed_weight -f params.init
```

Optional arguments:

--weight-files, -w list of input weight files in .npy, .npz or Sockeye parameter format .npy: a single array with shape=(vocab-in-size, embedding-size/hidden-size) .npz: a dictionary of {parameter_name: array}

parameter_name is given by "–names"

Sockeye parameter: the parameter name is given by "–names"

—vocabularies-in, -i list of input vocabularies as token-index dictionaries in .json format

--vocabularies-out, -o list of output vocabularies as token-index dictionaries in .json format They can be generated using sockeye.vocab before actual Sockeye training.

—names, -n list of Sockeye parameter names for embedding weights (or other vocabulary-sized weights) Most common ones are source_embed_weight, target_embed_weight, source_target_embed_weight, target_output_weight and target_output_bias.

Sizes of above 4 lists should be exactly the same - they are vertically aligned.

—file, -f file to write initialized parameters

--encoding, -c open input vocabularies with specified encoding (default: utf-8)

sockeye.init_embedding.**init_weight**(*weight*, *vocab_in*, *vocab_out*, *initializer=:class:'~'*)

Initialize vocabulary-sized weight by existing values given input and output vocabularies.

Parameters

- **weight** (ndarray) – Input weight.
- **vocab_in** (Dict[str, int]) – Input vocabulary.
- **vocab_out** (Dict[str, int]) – Output vocabulary.
- **initializer** (Initializer) – MXNet initializer.

Return type NDArray

Returns Initialized output weight.

sockeye.init_embedding.**load_weight**(*weight_file*, *weight_name*, *weight_file_cache*)

Load wight fron a file or the cache if it was loaded before.

Parameters

- **weight_file** (str) – Weight file.
- **weight_name** (str) – Weight name.
- **weight_file_cache** (Dict[str, Dict[~KT, ~VT]]) – Cache of loaded files.

Return type NDArray

Returns Loaded weight.

sockeye.init_embedding.**main**()

Commandline interface to initialize Sockeye embedding weights with pretrained word representations.

2.6.14 sockeye.initializer module

sockeye.initializer.**get_initializer**(*default_init_type*, *default_init_scale*, *default_init_xavier_rand_type*, *default_init_xavier_factor_type*, *embed_init_type*, *embed_init_sigma*, *rnn_init_type*, *extra_initializers=None*)

Returns a mixed MXNet initializer.

Parameters

- **default_init_type** (str) – The default weight initializer type.
- **default_init_scale** (float) – The scale used for default weight initialization (only used with uniform initialization).
- **default_init_xavier_rand_type** (str) – Xavier random number generator type.
- **default_init_xavier_factor_type** (str) – Xavier factor type.
- **embed_init_type** (str) – Embedding matrix initialization type.
- **embed_init_sigma** (float) – Sigma for normal initialization of embedding matrix.
- **rnn_init_type** (str) – Initialization type for RNN h2h matrices.
- **extra_initializers** (Optional[List[Tuple[str, Initializer]]]) – Optional initializers provided from other sources.

Return type Initializer

Returns Mixed initializer.

2.6.15 sockeye.layers module

class sockeye.layers.**LHUC**(*num_hidden*, *weight=None*, *prefix=''*)

Bases: object

Learning Hidden Unit Contribution

David Vilar. "Learning Hidden Unit Contribution for Adapting Neural Machine Translation Models" NAACL 2018

Parameters

- **num_hidden** (int) – Number of hidden units of the layer to be modified.
- **weight** (Optional[Symbol]) – Optional parameter vector.
- **prefix** (str) – Optional prefix for created parameters (if not given as weight).

class sockeye.layers.**LayerNormalization**(*prefix='layernorm'*, *scale=None*, *shift=None*, *scale_init=1.0*, *shift_init=0.0*)

Bases: object

Implements Ba et al, Layer Normalization (https://arxiv.org/abs/1607.06450).

Parameters

- **prefix** (str) – Optional prefix of layer name.
- **scale** (Optional[Symbol]) – Optional variable for scaling of shape (num_hidden,). Will be created if None.
- **shift** (Optional[Symbol]) – Optional variable for shifting of shape (num_hidden,). Will be created if None.

- **scale_init** (float) – Initial value of scale variable if scale is None. Default 1.0.
- **shift_init** (float) – Initial value of shift variable if shift is None. Default 0.0.

class sockeye.layers.**MultiHeadAttention**(*prefix*, *depth_att=512*, *heads=8*, *depth_out=512*, *dropout=0.0*)

Bases: *sockeye.layers.MultiHeadAttentionBase*

Multi-head attention layer for queries independent from keys/values.

Parameters

- **prefix** (str) – Attention prefix.
- **depth_att** (int) – Attention depth / number of hidden units.
- **heads** (int) – Number of attention heads.
- **depth_out** (int) – Output depth / number of output units.
- **dropout** (float) – Dropout probability on attention scores

class sockeye.layers.**MultiHeadAttentionBase**(*prefix*, *depth_att=512*, *heads=8*, *depth_out=512*, *dropout=0.0*)

Bases: object

Base class for Multi-head attention.

Parameters

- **prefix** (str) – Attention prefix.
- **depth_att** (int) – Attention depth / number of hidden units.
- **heads** (int) – Number of attention heads.
- **depth_out** (int) – Output depth / number of output units.
- **dropout** (float) – Dropout probability on attention scores

class sockeye.layers.**MultiHeadSelfAttention**(*prefix*, *depth_att=512*, *heads=8*, *depth_out=512*, *dropout=0.0*)

Bases: *sockeye.layers.MultiHeadAttentionBase*

Multi-head self-attention. Independent linear projections of inputs serve as queries, keys, and values for the attention.

Parameters

- **prefix** (str) – Attention prefix.
- **depth_att** (int) – Attention depth / number of hidden units.
- **heads** (int) – Number of attention heads.
- **depth_out** (int) – Output depth / number of output units.
- **dropout** (float) – Dropout probability on attention scores

class sockeye.layers.**OutputLayer**(*hidden_size*, *vocab_size*, *weight*, *weight_normalization*, *prefix='target_output_'*)

Bases: object

Defines the output layer of Sockeye decoders. Supports weight tying and weight normalization.

Parameters

- **hidden_size** (int) – Decoder hidden size.
- **vocab_size** (int) – Target vocabulary size.

- **weight_normalization** (bool) – Whether to apply weight normalization.
- **prefix** (str) – Prefix used for naming.

class sockeye.layers.**PlainDotAttention**
Bases: object

Dot attention layer for queries independent from keys/values.

class sockeye.layers.**ProjectedDotAttention**(*prefix*, *num_hidden*)
Bases: object

Dot attention layer for queries independent from keys/values.

Parameters

- **prefix** (str) – Attention prefix.
- **num_hidden** – Attention depth / number of hidden units.

class sockeye.layers.**WeightNormalization**(*weight*, *num_hidden*, *ndim=2*, *prefix=''*)
Bases: object

Implements Weight Normalization, see Salimans & Kingma 2016 (https://arxiv.org/abs/1602.07868). For a given tensor the normalization is done per hidden dimension.

Parameters

- **weight** – Weight tensor of shape: (num_hidden, d1, d2, ...).
- **num_hidden** – Size of the first dimension.
- **ndim** – The total number of dimensions of the weight tensor.
- **prefix** (str) – The prefix used for naming.

sockeye.layers.**activation**(*data*, *act_type*)
Apply custom or standard activation.

Custom activation types include:

- Swish-1, also called Sigmoid-Weighted Linear Unit (SiLU): Ramachandran et al. (https://arxiv.org/pdf/1710.05941.pdf), Elfwing et al. (https://arxiv.org/pdf/1702.03118.pdf)
- Gaussian Error Linear Unit (GELU): Hendrycks and Gimpel (https://arxiv.org/pdf/1606.08415.pdf)

Parameters

- **data** (Symbol) – input Symbol of any shape.
- **act_type** (str) – Type of activation.

Return type Symbol

Returns output Symbol with same shape as input.

sockeye.layers.**broadcast_to_heads**(*x*, *num_heads*, *ndim*, *fold_heads=True*)
Broadcasts batch-major input of shape (batch, d1 ... dn-1) to (batch*heads, d1 ... dn-1).

Parameters

- **x** (Symbol) – Batch-major input. Shape: (batch, d1 ... dn-1).
- **num_heads** (int) – Number of heads.
- **ndim** (int) – Number of dimensions in x.
- **fold_heads** (bool) – Whether to fold heads dimension into batch dimension.

Return type Symbol

Returns Tensor with each sample repeated heads-many times. Shape: (batch * heads, d1 ... dn-1) if fold_heads == True, (batch, heads, d1 ... dn-1) else.

sockeye.layers.**combine_heads**(*x*, *depth_per_head*, *heads*)

Returns a symbol with both batch & length, and head & depth dimensions combined.

Parameters

- **x** (Symbol) – Symbol of shape (batch * heads, length, depth_per_head).
- **depth_per_head** (int) – Depth per head.
- **heads** (int) – Number of heads.

Return type Symbol

Returns Symbol of shape (batch, length, depth).

sockeye.layers.**dot_attention**(*queries*, *keys*, *values*, *lengths=None*, *dropout=0.0*, *bias=None*, *prefix=''*)

Computes dot attention for a set of queries, keys, and values.

Parameters

- **queries** (Symbol) – Attention queries. Shape: (n, lq, d).
- **keys** (Symbol) – Attention keys. Shape: (n, lk, d).
- **values** (Symbol) – Attention values. Shape: (n, lk, dv).
- **lengths** (Optional[Symbol]) – Optional sequence lengths of the keys. Shape: (n,).
- **dropout** (float) – Dropout probability.
- **bias** (Optional[Symbol]) – Optional 3d bias tensor.
- **prefix** (Optional[str]) – Optional prefix

Returns 'Context' vectors for each query. Shape: (n, lq, dv).

sockeye.layers.**split_heads**(*x*, *depth_per_head*, *heads*)

Returns a symbol with head dimension folded into batch and depth divided by the number of heads.

Parameters

- **x** (Symbol) – Symbol of shape (batch, length, depth).
- **depth_per_head** (int) – Depth per head.
- **heads** (int) – Number of heads.

Return type Symbol

Returns Symbol of shape (batch * heads, length, depth_per_heads).

2.6.16 sockeye.lexical_constraints module

class sockeye.lexical_constraints.**ConstrainedCandidate**(*row*, *col*, *score*, *hypothesis*)

Bases: object

Object used to hold candidates for the beam in topk().

Parameters

- **row** (int) – The row in the scores matrix.

- **col** (int) – The column (word ID) in the scores matrix.
- **score** (float) – the associated accumulated score.
- **hypothesis** (*ConstrainedHypothesis*) – The ConstrainedHypothesis containing information about met constraints.

class sockeye.lexical_constraints.**ConstrainedHypothesis**(*constraint_list*, *eos_id*)
Bases: object

Represents a set of words and phrases that must appear in the output. A constraint is of two types: sequence or non-sequence. A non-sequence constraint is a single word and can therefore be followed by anything, whereas a sequence constraint must be followed by a particular word (the next word in the sequence). This class also records which constraints have been met.

A list of raw constraints is maintained internally as two parallel arrays. The following raw constraint represents two phrases that must appear in the output: 14 and 19 35 14.

raw constraint: [[14], [19, 35, 14]]

This is represented internally as:

constraints: [14 19 35 14] is_sequence: [1 1 0 0

Parameters

- **constraint_list** (List[List[int]]) – A list of zero or raw constraints (each represented as a list of integers).
- **eos_id** (int) – The end-of-sentence ID.

advance(*word_id*)
Updates the constraints object based on advancing on word_id. There is a complication, in that we may have started but not yet completed a multi-word constraint. We need to allow constraints to be added as unconstrained words, so if the next word is invalid, we must "back out" of the current (incomplete) phrase, re-setting all of its words as unmet.

Parameters **word_id** (int) – The word ID to advance on.

Return type *ConstrainedHypothesis*

Returns A deep copy of the object, advanced on word_id.

allowed()
Returns the set of constrained words that could follow this one. For unfinished phrasal constraints, it is the next word in the phrase. In other cases, it is the list of all unmet constraints. If all constraints are met, an empty set is returned.

Return type Set[int]

Returns The ID of the next required word, or -1 if any word can follow

finished()
Return true if all the constraints have been met.

Return type bool

Returns True if all the constraints are met.

is_valid(*wordid*)
Ensures </s> is only generated when the hypothesis is completed.

Parameters **wordid** – The wordid to validate.

Return type bool

Returns True if all constraints are already met or the word ID is not the EOS id.

num_met()

Return type int

Returns the number of constraints that have been met.

num_needed()

Return type int

Returns the number of un-met constraints.

size()

Return type int

Returns the number of constraints

sockeye.lexical_constraints.**get_bank_sizes**(*num_constraints*, *beam_size*, *candidate_counts*)

Evenly distributes the beam across the banks, where each bank is a portion of the beam devoted to hypotheses having met the same number of constraints, 0..num_constraints. After the assignment, banks with more slots than candidates are adjusted.

Parameters

- **num_constraints** (int) – The number of constraints.
- **beam_size** (int) – The beam size.
- **candidate_counts** (List[int]) – The empirical counts of number of candidates in each bank.

Return type List[int]

Returns A distribution over banks.

sockeye.lexical_constraints.**init_batch**(*raw_constraints*, *beam_size*, *start_id*, *eos_id*)

Parameters

- **raw_constraints** (List[Optional[List[List[int]]]]) – The list of raw constraints (list of list of IDs).
- **beam_size** (int) – The beam size.
- **start_id** (int) – The target-language vocabulary ID of the SOS symbol.
- **eos_id** (int) – The target-language vocabulary ID of the EOS symbol.

Return type List[Optional[*ConstrainedHypothesis*]]

Returns A list of ConstrainedHypothesis objects (shape: (batch_size * beam_size,)).

sockeye.lexical_constraints.**main**()

Usage: python3 -m sockeye.lexical_constraints [–bpe BPE_MODEL]

Reads sentences and constraints on STDIN (tab-delimited) and generates the JSON format that can be used when passing *–json-input* to sockeye.translate.

e.g.,

echo -e “Dies ist ein Test . This is test” | python3 -m sockeye.lexical_constraints

will produce the following JSON object:

{ “text”: “Dies ist ein Test .”, “constraints”: [“This is”, “test”] }

Make sure you apply all preprocessing (tokenization, BPE, etc.) to both the source and the target-side constraints. You can then translate this object by passing it to Sockeye on STDIN as follows:

python3 -m sockeye.translate -m /path/to/model –json-input –beam-size 20 –beam-prune 20

(Note the recommended Sockeye parameters).

sockeye.lexical_constraints.**topk**(*batch_size*, *beam_size*, *inactive*, *scores*, *hypotheses*, *best_ids*, *best_word_ids*, *seq_scores*, *context*)

Builds a new topk list such that the beam contains hypotheses having completed different numbers of constraints. These items are built from three different types: (1) the best items across the whole scores matrix, (2) the set of words that must follow existing constraints, and (3) k-best items from each row.

Parameters

- **batch_size** (int) – The number of segments in the batch.
- **beam_size** (int) – The length of the beam for each segment.
- **inactive** (ndarray) – Array listing inactive rows (shape: (beam_size,)).
- **scores** (ndarray) – The scores array (shape: (beam_size, target_vocab_size)).
- **hypotheses** (List[*ConstrainedHypothesis*]) – The list of hypothesis objects.
- **best_ids** (ndarray) – The current list of best hypotheses (shape: (beam_size,)).
- **best_word_ids** (ndarray) – The parallel list of best word IDs (shape: (beam_size,)).
- **seq_scores** (ndarray) – (shape: (beam_size, 1)).
- **context** (Context) – The MXNet device context.

Return type Tuple[array, array, array, List[*ConstrainedHypothesis*], array]

Returns A tuple containing the best hypothesis rows, the best hypothesis words, the scores, the updated constrained hypotheses, and the updated set of inactive hypotheses.

2.6.17 sockeye.lexicon module

class sockeye.lexicon.**Lexicon**(*source_vocab_size*, *target_vocab_size*, *learn=False*)

Bases: object

Lexicon model component. Stores lexicon and supports two operations:

1. Given source batch, lookup translation distributions in the lexicon
2. Given attention score vector and lexicon lookups, compute the lexical bias for the decoder

Parameters

- **source_vocab_size** (int) – Source vocabulary size.
- **target_vocab_size** (int) – Target vocabulary size.
- **learn** (bool) – Whether to adapt lexical biases during training.

static calculate_lex_bias(*source_lexicon*, *attention_prob_score*)

Given attention/alignment scores, calculates a weighted sum over lexical distributions that serve as a bias for the decoder softmax. * https://arxiv.org/pdf/1606.02006.pdf * http://www.aclweb.org/anthology/W/W16/W16-4610.pdf

Parameters

- **source_lexicon** (Symbol) – Lexical biases for sentence Shape: (batch_size, target_vocab_size, source_seq_len).
- **attention_prob_score** (Symbol) – Attention score. Shape: (batch_size, source_seq_len).

Return type Symbol

Returns Lexical bias. Shape: (batch_size, 1, target_vocab_size).

lookup(*source*)
Lookup lexicon distributions for source.

Parameters **source** (Symbol) – Input. Shape: (batch_size, source_seq_len).

Return type Symbol

Returns Lexicon distributions for input. Shape: (batch_size, target_vocab_size, source_seq_len).

class sockeye.lexicon.**TopKLexicon**(*vocab_source*, *vocab_target*)
Bases: object

Lexicon component that stores the k most likely target words for each source word. Used during decoding to restrict target vocabulary for each source sequence.

Parameters

- **vocab_source** (Dict[str, int]) – Trained model source vocabulary.
- **vocab_target** (Dict[str, int]) – Trained mode target vocabulary.

create(*path*, *k=20*)
Create from a scored lexicon file (fast_align format) using vocab from a trained Sockeye model.

Parameters

- **path** (str) – Path to lexicon file.
- **k** (int) – Number of target entries per source to keep.

get_trg_ids(*src_ids*)
Lookup possible target ids for input sequence of source ids.

Parameters **src_ids** (ndarray) – Sequence(s) of source ids (any shape).

Return type ndarray

Returns Possible target ids for source (unique sorted, always includes special symbols).

load(*path*, *k=None*)
Load lexicon from Numpy array file. The top-k target ids will be sorted by increasing target id.

Parameters

- **path** (str) – Path to Numpy array file.
- **k** (Optional[int]) – Optionally load less items than stored in path.

save(*path*)
Save lexicon in Numpy array format. Lexicon will be specific to Sockeye model.

Parameters **path** (str) – Path to Numpy array output file.

sockeye.lexicon.**initialize_lexicon**(*cmdline_arg*, *vocab_source*, *vocab_target*)
Reads a probabilistic word lexicon as given by the commandline argument and converts to log probabilities. If specified, smooths with custom value, uses 0.001 otherwise.

Parameters

- **cmdline_arg** (str) – Commandline argument.
- **vocab_source** (Dict[str, int]) – Source vocabulary.
- **vocab_target** (Dict[str, int]) – Target vocabulary.

Return type NDArray

Returns Lexicon array. Shape: (vocab_source_size, vocab_target_size).

sockeye.lexicon.**lexicon_iterator**(*path*, *vocab_source*, *vocab_target*)
Yields lines from a translation table of format: src, trg, logprob.

Parameters

- **path** (str) – Path to lexicon file.
- **vocab_source** (Dict[str, int]) – Source vocabulary.
- **vocab_target** (Dict[str, int]) – Target vocabulary.

Return type Generator[Tuple[int, int, float], None, None]

Returns Generator returning tuples (src_id, trg_id, prob).

sockeye.lexicon.**main**()
Commandline interface for building/inspecting top-k lexicons using during decoding.

sockeye.lexicon.**read_lexicon**(*path*, *vocab_source*, *vocab_target*)
Loads lexical translation probabilities from a translation table of format: src, trg, logprob. Source words unknown to vocab_source are discarded. Target words unknown to vocab_target contribute to p(unk|source_word). See Incorporating Discrete Translation Lexicons into Neural Machine Translation, Section 3.1 & Equation 5 (https://arxiv.org/pdf/1606.02006.pdf))

Parameters

- **path** (str) – Path to lexicon file.
- **vocab_source** (Dict[str, int]) – Source vocabulary.
- **vocab_target** (Dict[str, int]) – Target vocabulary.

Return type ndarray

Returns Lexicon array. Shape: (vocab_source_size, vocab_target_size).

2.6.18 sockeye.log module

sockeye.log.**setup_main_logger**(*name*, *file_logging=True*, *console=True*, *path=None*)
Return a logger that configures logging for the main application.

Parameters

- **name** (str) – Name of the returned logger.
- **file_logging** – Whether to log to a file.
- **console** – Whether to log to the console.
- **path** (Optional[str]) – Optional path to write logfile to.

Return type Logger

2.6.19 sockeye.loss module

Functions to generate loss symbols for sequence-to-sequence models.

class sockeye.loss.**CrossEntropyLoss**(*loss_config*)
Bases: *sockeye.loss.Loss*

Computes the cross-entropy loss.

Parameters **loss_config** (*LossConfig*) – Loss configuration.

create_metric()
Create an instance of the EvalMetric that corresponds to this Loss function.

Return type EvalMetric

get_loss(*logits*, *labels*)
Returns loss and softmax output symbols given logits and integer-coded labels.

Parameters

- **logits** (Symbol) – Shape: (batch_size * target_seq_len, target_vocab_size).
- **labels** (Symbol) – Shape: (batch_size * target_seq_len,).

Return type List[Symbol]

Returns List of loss symbol.

class sockeye.loss.**Loss**
Bases: abc.ABC

Generic Loss interface. get_loss() method should return a loss symbol and the softmax outputs. The softmax outputs (named C.SOFTMAX_NAME) are used by EvalMetrics to compute various metrics, e.g. perplexity, accuracy. In the special case of cross_entropy, the SoftmaxOutput symbol provides softmax outputs for forward() AND cross_entropy gradients for backward().

create_metric()
Create an instance of the EvalMetric that corresponds to this Loss function.

Return type EvalMetric

get_loss(*logits*, *labels*)
Returns loss and softmax output symbols given logits and integer-coded labels.

Parameters

- **logits** (Symbol) – Shape: (batch_size * target_seq_len, target_vocab_size).
- **labels** (Symbol) – Shape: (batch_size * target_seq_len,).

Return type List[Symbol]

Returns List of loss and softmax output symbols.

class sockeye.loss.**LossConfig**(*name*, *vocab_size*, *normalization_type*, *label_smoothing=0.0*)
Bases: sockeye.config.Config

Loss configuration.

Parameters

- **name** (str) – Loss name.
- **vocab_size** (int) – Target vocab size.
- **normalization_type** (str) – How to normalize the loss.

- **label_smoothing** (float) – Optional smoothing constant for label smoothing.

sockeye.loss.**get_loss**(*loss_config*)
Returns Loss instance.

Parameters **loss_config** (*LossConfig*) – Loss configuration.

Return type *Loss*

2.6.20 sockeye.lr_scheduler module

class sockeye.lr_scheduler.**AdaptiveLearningRateScheduler**(*warmup=0*)
Bases: sockeye.lr_scheduler.LearningRateScheduler

Learning rate scheduler that implements *new_evaluation_result* and accordingly adaptively adjust the learning rate.

new_evaluation_result(*has_improved*)
Returns true if the parameters should be reset to the ones with the best validation score.

Parameters **has_improved** (bool) – Whether the model improved on held-out validation data.

Return type bool

Returns True if parameters should be reset to the ones with best validation score.

class sockeye.lr_scheduler.**LearningRateSchedulerFixedStep**(*schedule*, *updates_per_checkpoint*)
Bases: *sockeye.lr_scheduler.AdaptiveLearningRateScheduler*

Use a fixed schedule of learning rate steps: lr_1 for N steps, lr_2 for M steps, etc.

Parameters

- **schedule** (List[Tuple[float, int]]) – List of learning rate step tuples in the form (rate, num_updates).
- **updates_per_checkpoint** (int) – Updates per checkpoint.

new_evaluation_result(*has_improved*)
Returns true if the parameters should be reset to the ones with the best validation score.

Parameters **has_improved** (bool) – Whether the model improved on held-out validation data.

Return type bool

Returns True if parameters should be reset to the ones with best validation score.

static parse_schedule_str(*schedule_str*)
Parse learning schedule string.

Parameters **schedule_str** (str) – String in form rate1:num_updates1[,rate2:num_updates2,...]

Return type List[Tuple[float, int]]

Returns List of tuples (learning_rate, num_updates).

class sockeye.lr_scheduler.**LearningRateSchedulerInvSqrtT**(*updates_per_checkpoint*, *half_life*, *warmup=0*)
Bases: sockeye.lr_scheduler.LearningRateScheduler

Learning rate schedule: lr / sqrt(1 + factor * t). Note: The factor is calculated from the half life of the learning rate.

Parameters

- **updates_per_checkpoint** (int) – Number of batches between checkpoints.
- **half_life** (int) – Half life of the learning rate in number of checkpoints.
- **warmup** (int) – Number of (linear) learning rate increases to warm-up.

class sockeye.lr_scheduler.**LearningRateSchedulerInvT**(*updates_per_checkpoint*, *half_life*, *warmup=0*)

Bases: sockeye.lr_scheduler.LearningRateScheduler

Learning rate schedule: lr / (1 + factor * t). Note: The factor is calculated from the half life of the learning rate.

Parameters

- **updates_per_checkpoint** (int) – Number of batches between checkpoints.
- **half_life** (int) – Half life of the learning rate in number of checkpoints.

class sockeye.lr_scheduler.**LearningRateSchedulerPlateauReduce**(*reduce_factor*, *reduce_num_not_improved*, *warmup=0*)

Bases: *sockeye.lr_scheduler.AdaptiveLearningRateScheduler*

Lower the learning rate as soon as the validation score plateaus.

Parameters

- **reduce_factor** (float) – Factor to reduce learning rate with.
- **reduce_num_not_improved** (int) – Number of checkpoints with no improvement after which learning rate is reduced.

new_evaluation_result(*has_improved*)

Returns true if the parameters should be reset to the ones with the best validation score.

Parameters **has_improved** (bool) – Whether the model improved on held-out validation data.

Return type bool

Returns True if parameters should be reset to the ones with best validation score.

sockeye.lr_scheduler.**get_lr_scheduler**(*scheduler_type*, *updates_per_checkpoint*, *learning_rate_half_life*, *learning_rate_reduce_factor*, *learning_rate_reduce_num_not_improved*, *learning_rate_schedule=None*, *learning_rate_warmup=0*)

Returns a learning rate scheduler.

Parameters

- **scheduler_type** (str) – Scheduler type.
- **updates_per_checkpoint** (int) – Number of batches between checkpoints.
- **learning_rate_half_life** (int) – Half life of the learning rate in number of checkpoints.
- **learning_rate_reduce_factor** (float) – Factor to reduce learning rate with.
- **learning_rate_reduce_num_not_improved** (int) – Number of checkpoints with no improvement after which learning rate is reduced.
- **learning_rate_schedule** (Optional[List[Tuple[float, int]]]) – Optional fixed learning rate schedule.

- **learning_rate_warmup** (Optional[int]) – Number of batches that the learning rate is linearly increased.

Raises ValueError if unknown scheduler_type

Return type Optional[LearningRateScheduler]

Returns Learning rate scheduler.

2.6.21 sockeye.model module

class sockeye.model.**ModelConfig**(*config_data*, *vocab_source_size*, *vocab_target_size*, *config_embed_source*, *config_embed_target*, *config_encoder*, *config_decoder*, *config_loss*, *weight_tying=False*, *weight_tying_type='trg_softmax'*, *weight_normalization=False*, *lhuc=False*)

Bases: sockeye.config.Config

ModelConfig defines model parameters defined at training time which are relevant to model inference. Add new model parameters here. If you want backwards compatibility for models trained with code that did not contain these parameters, provide a reasonable default under default_values.

Parameters

- **config_data** (*DataConfig*) – Used training data.
- **vocab_source_size** (int) – Source vocabulary size.
- **vocab_target_size** (int) – Target vocabulary size.
- **config_embed_source** (*EmbeddingConfig*) – Embedding config for source.
- **config_embed_target** (*EmbeddingConfig*) – Embedding config for target.
- **config_encoder** (Union[*RecurrentEncoderConfig*, TransformerConfig, *ConvolutionalEncoderConfig*]) – Encoder configuration.
- **config_decoder** (Union[*RecurrentDecoderConfig*, TransformerConfig, *ConvolutionalDecoderConfig*]) – Decoder configuration.
- **config_loss** (*LossConfig*) – Loss configuration.
- **weight_tying** (bool) – Enables weight tying if True.
- **weight_tying_type** (Optional[str]) – Determines which weights get tied. Must be set if weight_tying is enabled.
- **lhuc** (bool) – LHUC (Vilar 2018) is applied at some part of the model.

class sockeye.model.**SockeyeModel**(*config*, *prefix=''*)

Bases: object

SockeyeModel shares components needed for both training and inference. The main components of a Sockeye model are 1) Source embedding 2) Target embedding 3) Encoder 4) Decoder 5) Output Layer

ModelConfig contains parameters and their values that are fixed at training time and must be re-used at inference time.

Parameters

- **config** (*ModelConfig*) – Model configuration.
- **prefix** (str) – Name prefix for all parameters of this model.

static load_config(*fname*)
Loads model configuration.

Parameters **fname** (str) – Path to load model configuration from.

Return type *ModelConfig*

Returns Model configuration.

load_params_from_file(*fname*)
Loads and sets model parameters from file.

Parameters **fname** (str) – Path to load parameters from.

save_config(*folder*)
Saves model configuration to <folder>/config

Parameters **folder** (str) – Destination folder.

save_params_to_file(*fname*)
Saves model parameters to file.

Parameters **fname** (str) – Path to save parameters to.

static save_version(*folder*)
Saves version to <folder>/version.

Parameters **folder** (str) – Destination folder.

2.6.22 sockeye.output_handler module

class sockeye.output_handler.**AlignPlotHandler**(*plot_prefix*)
Bases: *sockeye.output_handler.OutputHandler*

Output handler to plot alignment matrices to PNG files.

Parameters **plot_prefix** (str) – Prefix for generated PNG files.

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total wall-clock time for translation.

class sockeye.output_handler.**AlignTextHandler**(*threshold*)
Bases: *sockeye.output_handler.OutputHandler*

Output handler to write alignment matrices as ASCII art.

Parameters **threshold** (float) – Threshold for considering alignment links as sure.

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total wall-clock time for translation.

class sockeye.output_handler.**BeamStoringHandler**(*stream*)
Bases: *sockeye.output_handler.OutputHandler*

Output handler to store beam histories in JSON format.

Parameters **stream** – Stream to write translations to (e.g. sys.stdout).

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total wall-clock time for translation.

class sockeye.output_handler.**BenchmarkOutputHandler**(*stream*)
Bases: *sockeye.output_handler.StringOutputHandler*

Output handler to write detailed benchmark information to a stream.

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total walltime for translation.

class sockeye.output_handler.**OutputHandler**
Bases: abc.ABC

Abstract output handler interface

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total wall-clock time for translation.

class sockeye.output_handler.**StringOutputHandler**(*stream*)
Bases: *sockeye.output_handler.OutputHandler*

Output handler to write translation to a stream

Parameters **stream** – Stream to write translations to (e.g. sys.stdout).

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total walltime for translation.

class sockeye.output_handler.**StringWithAlignmentMatrixOutputHandler**(*stream*)
Bases: *sockeye.output_handler.StringOutputHandler*

Output handler to write translations and an alignment matrix to a stream. Note that unlike other output handlers each input sentence will result in an output consisting of multiple lines. More concretely the format is:

` sentence id ||| target words ||| score ||| source words ||| number of source words ||| number of target words ALIGNMENT FOR T_1 ALIGNMENT FOR T_2 ... ALIGNMENT FOR T_n `

where the alignment is a list of probabilities of alignment to the source words.

Parameters **stream** – Stream to write translations and alignments to.

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total wall-clock time for translation.

class sockeye.output_handler.**StringWithAlignmentsOutputHandler**(*stream*, *threshold*)

Bases: *sockeye.output_handler.StringOutputHandler*

Output handler to write translations and alignments to a stream. Translation and alignment string are separated by a tab. Alignments are written in the format: <src_index>-<trg_index> ... An alignment link is included if its probability is above the threshold.

Parameters

- **stream** – Stream to write translations and alignments to.
- **threshold** (float) – Threshold for including alignment links.

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total wall-clock time for translation.

class sockeye.output_handler.**StringWithScoreOutputHandler**(*stream*)

Bases: *sockeye.output_handler.OutputHandler*

Output handler to write translation score and translation to a stream. The score and translation string are tab-delimited.

Parameters **stream** – Stream to write translations to (e.g. sys.stdout).

handle(*t_input*, *t_output*, *t_walltime=0.0*)

Parameters

- **t_input** (*TranslatorInput*) – Translator input.
- **t_output** (*TranslatorOutput*) – Translator output.
- **t_walltime** (float) – Total walltime for translation.

sockeye.output_handler.**get_output_handler**(*output_type*, *output_fname*, *sure_align_threshold*)

Parameters

- **output_type** (str) – Type of output handler.
- **output_fname** (Optional[str]) – Output filename. If none sys.stdout is used.

- **sure_align_threshold** (float) – Threshold to consider an alignment link as 'sure'.

Raises ValueError for unknown output_type.

Return type *OutputHandler*

Returns Output handler.

2.6.23 sockeye.prepare_data module

2.6.24 sockeye.rnn module

class sockeye.rnn.**RNNConfig**(*cell_type*, *num_hidden*, *num_layers*, *dropout_inputs*, *dropout_states*, *dropout_recurrent=0*, *residual=False*, *first_residual_layer=2*, *forget_bias=0.0*, *lhuc=False*, *dtype='float32'*)

Bases: sockeye.config.Config

RNN configuration.

Parameters

- **cell_type** (str) – RNN cell type.
- **num_hidden** (int) – Number of RNN hidden units.
- **num_layers** (int) – Number of RNN layers.
- **dropout_inputs** (float) – Dropout probability on RNN inputs (Gal, 2015).
- **dropout_states** (float) – Dropout probability on RNN states (Gal, 2015).
- **dropout_recurrent** (float) – Dropout probability on cell update (Semeniuta, 2016).
- **residual** (bool) – Whether to add residual connections between multi-layered RNNs.
- **first_residual_layer** (int) – First layer with a residual connection (1-based indexes). Default is to start at the second layer.
- **forget_bias** (float) – Initial value of forget biases.
- **lhuc** (bool) – Apply LHUC (Vilar 2018) to the hidden units of the RNN.
- **dtype** (str) – Data type.

sockeye.rnn.**get_stacked_rnn**(*config*, *prefix*, *parallel_inputs=False*, *layers=None*)

Returns (stacked) RNN cell given parameters.

Parameters

- **config** (*RNNConfig*) – rnn configuration.
- **prefix** (str) – Symbol prefix for RNN.
- **parallel_inputs** (bool) – Support parallel inputs for the stacked RNN cells.
- **layers** (Optional[Iterable[int]]) – Specify which layers to create as a list of layer indexes.

Return type SequentialRNNCell

Returns RNN cell.

2.6.25 sockeye.rnn_attention module

Implementations of different attention mechanisms in sequence-to-sequence models.

class sockeye.rnn_attention.**Attention**(*input_previous_word*, *dynamic_source_num_hidden=1*, *prefix='att_'*, *dtype='float32'*)

Bases: object

Generic attention interface that returns a callable for attending to source states.

Parameters

- **input_previous_word** (bool) – Feed the previous target embedding into the attention mechanism.
- **dynamic_source_num_hidden** (int) – Number of hidden units of dynamic source encoding update mechanism.
- **dtype** (str) – Data type.

get_initial_state(*source_length*, *source_seq_len*)

Returns initial attention state. Dynamic source encoding is initialized with zeros.

Parameters

- **source_length** (Symbol) – Source length. Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type *AttentionState*

make_input(*seq_idx*, *word_vec_prev*, *decoder_state*)

Returns AttentionInput to be fed into the attend callable returned by the on() method.

Parameters

- **seq_idx** (int) – Decoder time step.
- **word_vec_prev** (Symbol) – Embedding of previously predicted ord
- **decoder_state** (Symbol) – Current decoder state

Return type *AttentionInput*

Returns Attention input.

on(*source*, *source_length*, *source_seq_len*)

Returns callable to be used for recurrent attention in a sequence decoder. The callable is a recurrent function of the form: AttentionState = attend(AttentionInput, AttentionState).

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Attention callable.

class sockeye.rnn_attention.**AttentionConfig**(*type*, *num_hidden*, *input_previous_word*, *source_num_hidden*, *query_num_hidden*, *layer_normalization*, *config_coverage=None*, *num_heads=None*, *is_scaled=False*, *dtype='float32'*)

Bases: sockeye.config.Config

Attention configuration.

Parameters

- **type** (str) – Attention name.
- **num_hidden** (int) – Number of hidden units for attention networks.
- **input_previous_word** (bool) – Feeds the previous target embedding into the attention mechanism.
- **source_num_hidden** (int) – Number of hidden units of the source.
- **query_num_hidden** (int) – Number of hidden units of the query.
- **layer_normalization** (bool) – Apply layer normalization to MLP attention.
- **config_coverage** (Optional[*CoverageConfig*]) – Optional coverage configuration.
- **num_heads** (Optional[int]) – Number of attention heads. Only used for Multi-head dot attention.
- **is_scaled** (Optional[bool]) – If 'dot' attentions should be scaled.
- **dtype** (str) – Data type.

class sockeye.rnn_attention.**AttentionInput**(*seq_idx*, *query*)

Bases: tuple

Input to attention callables.

Parameters

- **seq_idx** – Decoder time step / sequence index.
- **query** – Query input to attention mechanism, e.g. decoder hidden state (plus previous word).

query
Alias for field number 1

seq_idx
Alias for field number 0

class sockeye.rnn_attention.**AttentionState**(*context*, *probs*, *dynamic_source*)

Bases: tuple

Results returned from attention callables.

Parameters

- **context** – Context vector (Bahdanau et al, 15). Shape: (batch_size, encoder_num_hidden)
- **probs** – Attention distribution over source encoder states. Shape: (batch_size, source_seq_len).
- **dynamic_source** – Dynamically updated source encoding. Shape: (batch_size, source_seq_len, dynamic_source_num_hidden)

context
Alias for field number 0

dynamic_source
Alias for field number 2

probs
Alias for field number 1

class sockeye.rnn_attention.**BilinearAttention**(*query_num_hidden*, *dtype='float32'*, *prefix='att_'*)

Bases: *sockeye.rnn_attention.Attention*

Bilinear attention based on Luong et al. 2015.

$score(h_t, h_s) = h_t^T \mathbf{W} h_s$

For implementation reasons we modify to:

$score(h_t, h_s) = h_s^T \mathbf{W} h_t$

Parameters

- **query_num_hidden** (int) – Number of hidden units the source will be projected to.
- **dtype** (str) – data type.
- **prefix** (str) – Name prefix.

on(*source*, *source_length*, *source_seq_len*)
Returns callable to be used for recurrent attention in a sequence decoder. The callable is a recurrent function of the form: AttentionState = attend(AttentionInput, AttentionState).

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Attention callable.

class sockeye.rnn_attention.**DotAttention**(*input_previous_word*, *source_num_hidden*, *query_num_hidden*, *num_hidden*, *is_scaled=False*, *prefix='att_'*, *dtype='float32'*)

Bases: *sockeye.rnn_attention.Attention*

Attention mechanism with dot product between encoder and decoder hidden states [Luong et al. 2015].

$score(h_t, h_s) = \langle h_t, h_s \rangle$

$a = softmax(score(*, h_s))$

If rnn_num_hidden != num_hidden, states are projected with additional parameters to num_hidden.

$score(h_t, h_s) = \langle \mathbf{W}_t h_t, \mathbf{W}_s h_s \rangle$

Parameters

- **input_previous_word** (bool) – Feed the previous target embedding into the attention mechanism.
- **source_num_hidden** (int) – Number of hidden units in source.
- **query_num_hidden** (int) – Number of hidden units in query.

- **num_hidden** (int) – Number of hidden units.
- **is_scaled** (bool) – Optionally scale query before dot product [Vaswani et al, 2017].
- **prefix** (str) – Name prefix.
- **dtype** (str) – data type.

on(*source*, *source_length*, *source_seq_len*)

Returns callable to be used for recurrent attention in a sequence decoder. The callable is a recurrent function of the form: AttentionState = attend(AttentionInput, AttentionState).

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Attention callable.

class sockeye.rnn_attention.**EncoderLastStateAttention**(*input_previous_word*, *dynamic_source_num_hidden=1*, *prefix='att_'*, *dtype='float32'*)

Bases: *sockeye.rnn_attention.Attention*

Always returns the last encoder state independent of the query vector. Equivalent to no attention.

on(*source*, *source_length*, *source_seq_len*)

Returns callable to be used for recurrent attention in a sequence decoder. The callable is a recurrent function of the form: AttentionState = attend(AttentionInput, AttentionState).

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Attention callable.

class sockeye.rnn_attention.**LocationAttention**(*input_previous_word*, *max_seq_len*, *prefix='att_'*, *dtype='float32'*)

Bases: *sockeye.rnn_attention.Attention*

Attends to locations in the source [Luong et al, 2015]

$a_t = softmax(\mathbf{W}_a h_t)$ for decoder hidden state at time t.

Note $\mathbf{W}_a$ is of shape (max_source_seq_len, decoder_num_hidden).

Parameters

- **input_previous_word** (bool) – Feed the previous target embedding into the attention mechanism.
- **max_seq_len** (int) – Maximum length of source sequences.
- **prefix** (str) – Name prefix.
- **dtype** (str) – data type.

on(*source*, *source_length*, *source_seq_len*)
Returns callable to be used for recurrent attention in a sequence decoder. The callable is a recurrent function of the form: AttentionState = attend(AttentionInput, AttentionState).

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Attention callable.

class sockeye.rnn_attention.**MlpAttention**(*input_previous_word*, *num_hidden*, *layer_normalization=False*, *prefix='att_'*, *dtype='float32'*)

Bases: *sockeye.rnn_attention.Attention*

Attention computed through a one-layer MLP with num_hidden units [Luong et al, 2015].

$score(h_t, h_s) = \mathbf{W}_a tanh(\mathbf{W}_c[h_t, h_s] + b)$

$a = softmax(score(*, h_s))$

Optionally, if attention_coverage_type is not None, attention uses dynamic source encoding ('coverage' mechanism) as in Tu et al. (2016): Modeling Coverage for Neural Machine Translation.

$score(h_t, h_s) = \mathbf{W}_a tanh(\mathbf{W}_c[h_t, h_s, c_s] + b)$

c_s is the decoder time-step dependent source encoding which is updated using the current decoder state.

Parameters

- **input_previous_word** (bool) – Feed the previous target embedding into the attention mechanism.
- **num_hidden** (int) – Number of hidden units.
- **layer_normalization** (bool) – If true, normalizes hidden layer outputs before tanh activation.
- **prefix** (str) – Name prefix
- **dtype** (str) – data type.

on(*source*, *source_length*, *source_seq_len*)
Returns callable to be used for recurrent attention in a sequence decoder. The callable is a recurrent function of the form: AttentionState = attend(AttentionInput, AttentionState).

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Attention callable.

class sockeye.rnn_attention.**MlpCovAttention**(*input_previous_word*, *num_hidden*, *layer_normalization=False*, *config_coverage=None*, *prefix='att_'*, *dtype='float32'*)

Bases: *sockeye.rnn_attention.MlpAttention*

MlpAttention with optional coverage config.

Parameters

- **input_previous_word** (bool) – Feed the previous target embedding into the attention mechanism.
- **num_hidden** (int) – Number of hidden units.
- **layer_normalization** (bool) – If true, normalizes hidden layer outputs before tanh activation.
- **config_coverage** (Optional[*CoverageConfig*]) – coverage config.
- **prefix** (str) – Name prefix.
- **dtype** (str) – data type.

class sockeye.rnn_attention.**MultiHeadDotAttention**(*input_previous_word*, *source_num_hidden*, *num_heads*, *prefix='att_'*, *dtype='float32'*)

Bases: *sockeye.rnn_attention.Attention*

Dot product attention with multiple heads as proposed in Vaswani et al, Attention is all you need. Can be used with a RecurrentDecoder.

Parameters

- **input_previous_word** (bool) – Feed the previous target embedding into the attention mechanism.
- **source_num_hidden** (int) – Number of hidden units.
- **num_heads** (int) – Number of attention heads / independently computed attention scores.
- **prefix** (str) – Name prefix.
- **dtype** (str) – data type.

on(*source*, *source_length*, *source_seq_len*)

Returns callable to be used for recurrent attention in a sequence decoder. The callable is a recurrent function of the form: AttentionState = attend(AttentionInput, AttentionState).

Parameters

- **source** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **source_length** (Symbol) – Shape: (batch_size,).
- **source_seq_len** (int) – Maximum length of source sequences.

Return type Callable

Returns Attention callable.

sockeye.rnn_attention.**get_attention**(*config*, *max_seq_len*, *prefix='att_'*)

Returns an Attention instance based on attention_type.

Parameters

- **config** (*AttentionConfig*) – Attention configuration.

- **max_seq_len** (int) – Maximum length of source sequences.
- **prefix** (str) – Name prefix.

Return type *Attention*

Returns Instance of Attention.

sockeye.rnn_attention.**get_context_and_attention_probs**(*values*, *length*, *logits*, *dtype*)
Returns context vector and attention probabilities via a weighted sum over values.

Parameters

- **values** (Symbol) – Shape: (batch_size, seq_len, encoder_num_hidden).
- **length** (Symbol) – Shape: (batch_size,).
- **logits** (Symbol) – Shape: (batch_size, seq_len, 1).
- **dtype** (str) – data type.

Return type Tuple[Symbol, Symbol]

Returns context: (batch_size, encoder_num_hidden), attention_probs: (batch_size, seq_len).

2.6.26 sockeye.train module

Simple Training CLI.

sockeye.train.**check_arg_compatibility**(*args*)
Check if some arguments are incompatible with each other.

Parameters **args** (Namespace) – Arguments as returned by argparse.

sockeye.train.**check_encoder_decoder_args**(*args*)
Check possible encoder-decoder argument conflicts.

Parameters **args** – Arguments as returned by argparse.

Return type None

sockeye.train.**check_resume**(*args*, *output_folder*)
Check if we should resume a broken training run.

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **output_folder** (str) – Main output folder for the model.

Return type bool

Returns Flag signaling if we are resuming training and the directory with the training status.

sockeye.train.**create_checkpoint_decoder**(*args*, *exit_stack*, *train_context*)
Returns a checkpoint decoder or None.

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **exit_stack** (ExitStack) – An ExitStack from contextlib.
- **train_context** (List[Context]) – Context for training.

Return type Optional[*CheckpointDecoder*]

Returns A CheckpointDecoder if –decode-and-evaluate != 0, else None.

sockeye.train.**create_data_iters_and_vocabs**(*args*, *max_seq_len_source*, *max_seq_len_target*, *shared_vocab*, *resume_training*, *output_folder*)

Create the data iterators and the vocabularies.

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **max_seq_len_source** (int) – Source maximum sequence length.
- **max_seq_len_target** (int) – Target maximum sequence length.
- **shared_vocab** (bool) – Whether to create a shared vocabulary.
- **resume_training** (bool) – Whether to resume training.
- **output_folder** (str) – Output folder.

Return type Tuple[DataIter, DataIter, *DataConfig*, List[Dict[str, int]], Dict[str, int]]

Returns The data iterators (train, validation, config_data) as well as the source and target vocabularies.

sockeye.train.**create_decoder_config**(*args*, *encoder_num_hidden*, *max_seq_len_source*, *max_seq_len_target*)

Create the config for the decoder.

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **encoder_num_hidden** (int) – Number of hidden units of the Encoder.
- **max_seq_len_source** (int) – Maximum source sequence length.
- **max_seq_len_target** (int) – Maximum target sequence length.

Return type Union[*RecurrentDecoderConfig*, TransformerConfig, *ConvolutionalDecoderConfig*]

Returns The config for the decoder.

sockeye.train.**create_encoder_config**(*args*, *max_seq_len_source*, *max_seq_len_target*, *config_conv*)

Create the encoder config.

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **max_seq_len_source** (int) – Maximum source sequence length.
- **max_seq_len_target** (int) – Maximum target sequence length.
- **config_conv** (Optional[*ConvolutionalEmbeddingConfig*]) – The config for the convolutional encoder (optional).

Return type Tuple[Union[*RecurrentEncoderConfig*, TransformerConfig, *ConvolutionalEncoderConfig*], int]

Returns The encoder config and the number of hidden units of the encoder.

sockeye.train.**create_model_config**(*args*, *source_vocab_sizes*, *target_vocab_size*, *max_seq_len_source*, *max_seq_len_target*, *config_data*)

Create a ModelConfig from the argument given in the command line.

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **source_vocab_sizes** (List[int]) – The size of the source vocabulary (and source factors).
- **target_vocab_size** (int) – The size of the target vocabulary.
- **max_seq_len_source** (int) – Maximum source sequence length.
- **max_seq_len_target** (int) – Maximum target sequence length.
- **config_data** (*DataConfig*) – Data config.

Return type *ModelConfig*

Returns The model configuration.

sockeye.train.**create_optimizer_config**(*args*, *source_vocab_sizes*, *extra_initializers=None*)
Returns an OptimizerConfig.

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **source_vocab_sizes** (List[int]) – Source vocabulary sizes.
- **extra_initializers** (Optional[List[Tuple[str, Initializer]]]) – extra initializer to pass to *get_initializer*.

Return type OptimizerConfig

Returns The optimizer type and its parameters as well as the kvstore.

sockeye.train.**create_training_model**(*config*, *context*, *output_dir*, *train_iter*, *args*)
Create a training model and load the parameters from disk if needed.

Parameters

- **config** (*ModelConfig*) – The configuration for the model.
- **context** (List[Context]) – The context(s) to run on.
- **output_dir** (str) – Output folder.
- **train_iter** (DataIter) – The training data iterator.
- **args** (Namespace) – Arguments as returned by argparse.

Return type *TrainingModel*

Returns The training model.

sockeye.train.**determine_context**(*args*, *exit_stack*)
Determine the context we should run on (CPU or GPU).

Parameters

- **args** (Namespace) – Arguments as returned by argparse.
- **exit_stack** (ExitStack) – An ExitStack from contextlib.

Return type List[Context]

Returns A list with the context(s) to run on.

sockeye.train.**gradient_compression_params**(*args*)

Parameters **args** (Namespace) – Arguments as returned by argparse.

Return type Optional[Dict[str, Any]]

Returns Gradient compression parameters or None.

sockeye.train.**use_shared_vocab**(*args*)
True if arguments entail a shared source and target vocabulary.

Param args: Arguments as returned by argparse.

Return type bool

2.6.27 sockeye.training module

Code for training

class sockeye.training.**DecoderProcessManager**(*output_folder*, *decoder*)
Bases: object

Thin wrapper around a CheckpointDecoder instance to start non-blocking decodes and collect the results.

Parameters

- **output_folder** (str) – Folder where decoder outputs are written to.
- **decoder** (*CheckpointDecoder*) – CheckpointDecoder instance.

collect_results()
Returns the decoded checkpoint and the decoder metrics or None if the queue is empty.

Return type Optional[Tuple[int, Dict[str, float]]]

start_decoder(*checkpoint*)
Starts a new CheckpointDecoder process and returns. No other process may exist.

Parameters **checkpoint** (int) – The checkpoint to decode.

class sockeye.training.**EarlyStoppingTrainer**(*model*, *optimizer_config*, *max_params_files_to_keep*, *source_vocabs*, *target_vocab*)
Bases: object

Trainer class that fits a TrainingModel using early stopping on held-out validation data.

Parameters

- **model** (*TrainingModel*) – TrainingModel instance.
- **optimizer_config** (OptimizerConfig) – The optimizer configuration.
- **max_params_files_to_keep** (int) – Maximum number of params files to keep in the output folder (last n are kept).
- **source_vocabs** (List[Dict[str, int]]) – Source vocabulary (and optional source factor vocabularies).
- **target_vocab** (Dict[str, int]) – Target vocabulary.

fit(*train_iter*, *validation_iter*, *early_stopping_metric*, *metrics*, *checkpoint_frequency*, *max_num_not_improved*, *min_samples=None*, *max_samples=None*, *min_updates=None*, *max_updates=None*, *min_epochs=None*, *max_epochs=None*, *lr_decay_param_reset=False*, *lr_decay_opt_states_reset='off'*, *decoder=None*, *mxmonitor_pattern=None*, *mxmonitor_stat_func=None*, *allow_missing_parameters=False*, *existing_parameters=None*)
Fits model to data given by train_iter using early-stopping w.r.t data given by val_iter. Saves all intermediate and final output to output_folder.

Parameters

- **train_iter** (DataIter) – The training data iterator.
- **validation_iter** (DataIter) – The data iterator for held-out data.
- **early_stopping_metric** – The metric that is evaluated on held-out data and optimized.
- **metrics** (List[str]) – List of metrics that will be tracked during training.
- **checkpoint_frequency** (int) – Frequency of checkpoints in number of update steps.
- **max_num_not_improved** (int) – Stop training if early_stopping_metric did not improve for this many checkpoints. Use -1 to disable stopping based on early_stopping_metric.
- **min_samples** (Optional[int]) – Optional minimum number of samples.
- **max_samples** (Optional[int]) – Optional maximum number of samples.
- **min_updates** (Optional[int]) – Optional minimum number of update steps.
- **max_updates** (Optional[int]) – Optional maximum number of update steps.
- **min_epochs** (Optional[int]) – Optional minimum number of epochs to train, overrides early stopping.
- **max_epochs** (Optional[int]) – Optional maximum number of epochs to train, overrides early stopping.
- **lr_decay_param_reset** (bool) – Reset parameters to previous best after a learning rate decay.
- **lr_decay_opt_states_reset** (str) – How to reset optimizer states after a learning rate decay.
- **decoder** (Optional[*CheckpointDecoder*]) – Optional CheckpointDecoder instance to decode and compute evaluation metrics.
- **mxmonitor_pattern** (Optional[str]) – Optional pattern to match to monitor weights/gradients/outputs with MXNet's monitor. Default is None which means no monitoring.
- **mxmonitor_stat_func** (Optional[str]) – Choice of statistics function to run on monitored weights/gradients/outputs when using MXNEt's monitor.
- **allow_missing_parameters** (bool) – Allow missing parameters when initializing model parameters from file.
- **existing_parameters** (Optional[str]) – Optional filename of existing/pretrained parameters to initialize from.

Returns Best score on validation data observed during training.

class sockeye.training.**Speedometer**(*frequency=50*, *auto_reset=True*)

Bases: object

Custom Speedometer to log samples and words per second.

class sockeye.training.**TensorboardLogger**(*logdir*, *source_vocab=None*, *target_vocab=None*)

Bases: object

Thin wrapper for MXBoard API to log training events. Flushes logging events to disk every 60 seconds.

Parameters

- **logdir** (str) – Directory to write Tensorboard event files to.
- **source_vocab** (Optional[Dict[str, int]]) – Optional source vocabulary to log source embeddings.
- **target_vocab** (Optional[Dict[str, int]]) – Optional target vocabulary to log target and output embeddings.

class sockeye.training.**TrainState**(*early_stopping_metric*)

Bases: object

Stores the state an EarlyStoppingTrainer instance.

static load(*fname*)

Loads a training state from fname.

Return type *TrainState*

save(*fname*)

Saves this training state to fname.

class sockeye.training.**TrainingModel**(*config*, *context*, *output_dir*, *provide_data*, *provide_label*, *default_bucket_key*, *bucketing*, *gradient_compression_params=None*, *fixed_param_names=None*)

Bases: *sockeye.model.SockeyeModel*

TrainingModel is a SockeyeModel that fully unrolls over source and target sequences.

Parameters

- **config** (*ModelConfig*) – Configuration object holding details about the model.
- **context** (List[Context]) – The context(s) that MXNet will be run in (GPU(s)/CPU).
- **output_dir** (str) – Directory where this model is stored.
- **provide_data** (List[DataDesc]) – List of input data descriptions.
- **provide_label** (List[DataDesc]) – List of label descriptions.
- **default_bucket_key** (Tuple[int, int]) – Default bucket key.
- **bucketing** (bool) – If True bucketing will be used, if False the computation graph will always be unrolled to the full length.
- **gradient_compression_params** (Optional[Dict[str, Any]]) – Optional dictionary of gradient compression parameters.
- **fixed_param_names** (Optional[List[str]]) – Optional list of params to fix during training (i.e. their values will not be trained).

evaluate(*eval_iter*, *eval_metric*)

Resets and recomputes evaluation metric on given data iterator.

get_global_gradient_norm()

Returns global gradient norm.

Return type float

get_gradients()

Returns a mapping of parameters names to gradient arrays. Parameter names are prefixed with the device.

Return type Dict[str, List[NDArray]]

initialize_optimizer(*config*)
Initializes the optimizer of the underlying module with an optimizer config.

initialize_parameters(*initializer*, *allow_missing_params*)
Initializes the parameters of the underlying module.

Parameters

- **initializer** (Initializer) – Parameter initializer.
- **allow_missing_params** (bool) – Whether to allow missing parameters.

install_monitor(*monitor_pattern*, *monitor_stat_func_name*)
Installs an MXNet monitor onto the underlying module.

Parameters

- **monitor_pattern** (str) – Pattern string.
- **monitor_stat_func_name** (str) – Name of monitor statistics function.

load_optimizer_states(*fname*)
Loads optimizer states from file.

Parameters **fname** (str) – File name to load optimizer states from.

load_params_from_file(*fname*, *allow_missing_params=False*)
Loads parameters from a file and sets the parameters of the underlying module and this model instance.

Parameters

- **fname** (str) – File name to load parameters from.
- **allow_missing_params** (bool) – If set, the given parameters are allowed to be a subset of the Module parameters.

log_parameters()
Logs information about model parameters.

optimizer
Returns the optimizer of the underlying module.

Return type Union[Optimizer, SockeyeOptimizer]

prepare_batch(*batch*)
Pre-fetches the next mini-batch.

Parameters **batch** (DataBatch) – The mini-batch to prepare.

rescale_gradients(*scale*)
Rescales gradient arrays of executors by scale.

run_forward_backward(*batch*, *metric*)
Runs forward/backward pass and updates training metric(s).

save_optimizer_states(*fname*)
Saves optimizer states to a file.

Parameters **fname** (str) – File name to save optimizer states to.

save_params_to_file(*fname*)
Synchronizes parameters across devices, saves the parameters to disk, and updates self.params and self.aux_params.

Parameters **fname** (str) – Filename to write parameters to.

update()
Updates parameters of the module.

2.6.28 sockeye.transformer module

class sockeye.transformer.**TransformerDecoderBlock**(*config*, *prefix*)
Bases: object

A transformer encoder block consists self-attention, encoder attention, and a feed-forward layer with pre/post process blocks in between.

class sockeye.transformer.**TransformerEncoderBlock**(*config*, *prefix*)
Bases: object

A transformer encoder block consists self-attention and a feed-forward layer with pre/post process blocks in between.

class sockeye.transformer.**TransformerFeedForward**(*num_hidden*, *num_model*, *act_type*, *dropout*, *prefix*)
Bases: object

Position-wise feed-forward network with activation.

class sockeye.transformer.**TransformerProcessBlock**(*sequence*, *dropout*, *prefix*)
Bases: object

Block to perform pre/post processing on layer inputs. The processing steps are determined by the sequence argument, which can contain one of the three operations: n: layer normalization r: residual connection d: dropout

sockeye.transformer.**get_autoregressive_bias**(*max_length*, *name*)
Returns bias/mask to ensure position i can only attend to positions <i.

Parameters

- **max_length** (int) – Sequence length.
- **name** (str) – Name of symbol.

Return type Symbol

Returns Bias symbol of shape (1, max_length, max_length).

sockeye.transformer.**get_variable_length_bias**(*lengths*, *max_length*, *num_heads=None*, *fold_heads=True*, *name=''*)
Returns bias/mask for variable sequence lengths.

Parameters

- **lengths** (Symbol) – Sequence lengths. Shape: (batch,).
- **max_length** (int) – Maximum sequence length.
- **num_heads** (Optional[int]) – Number of attention heads.
- **fold_heads** (bool) – Whether to fold heads dimension into batch dimension.
- **name** (str) – Name of symbol.

Return type Symbol

Returns Bias symbol.

2.6.29 sockeye.translate module

Translation CLI.

sockeye.translate.**make_inputs**(*input_file*, *translator*, *input_is_json*, *input_factors=None*)
Generates TranslatorInput instances from input. If input is None, reads from stdin. If num_input_factors > 1, the function will look for factors attached to each token, separated by '|'. If source is not None, reads from the source file. If num_source_factors > 1, num_source_factors source factor filenames are required.

Parameters

- **input_file** (Optional[str]) – The source file (possibly None).
- **translator** (*Translator*) – Translator that will translate each line of input.
- **input_is_json** (bool) – Whether the input is in json format.
- **input_factors** (Optional[List[str]]) – Source factor files.

Return type Generator[*TranslatorInput*, None, None]

Returns TranslatorInput objects.

sockeye.translate.**read_and_translate**(*translator*, *output_handler*, *chunk_size*, *input_file=None*, *input_factors=None*, *input_is_json=False*)
Reads from either a file or stdin and translates each line, calling the output_handler with the result.

Parameters

- **output_handler** (*OutputHandler*) – Handler that will write output to a stream.
- **translator** (*Translator*) – Translator that will translate each line of input.
- **chunk_size** (Optional[int]) – The size of the portion to read at a time from the input.
- **input_file** (Optional[str]) – Optional path to file which will be translated line-by-line if included, if none use stdin.
- **input_factors** (Optional[List[str]]) – Optional list of paths to files that contain source factors.
- **input_is_json** (bool) – Whether the input is in json format.

Return type None

sockeye.translate.**translate**(*output_handler*, *trans_inputs*, *translator*)
Translates each line from source_data, calling output handler after translating a batch.

Parameters

- **output_handler** (*OutputHandler*) – A handler that will be called once with the output of each translation.
- **trans_inputs** (List[*TranslatorInput*]) – A enumerable list of translator inputs.
- **translator** (*Translator*) – The translator that will be used for each line of input.

Return type float

Returns Total time taken.

2.6.30 sockeye.utils module

A set of utility methods.

class sockeye.utils.**GpuFileLock**(*candidates*, *lock_dir*)
Bases: object

Acquires a single GPU by locking a file (therefore this assumes that everyone using GPUs calls this method and shares the lock directory). Sets target to a GPU id or None if none is available.

Parameters

- **candidates** (List[~GpuDeviceType]) – List of candidate device ids to try to acquire.
- **lock_dir** (str) – The directory for storing the lock file.

exception sockeye.utils.**SockeyeError**
Bases: Exception

sockeye.utils.**acquire_gpus**(*requested_device_ids*, *lock_dir='/tmp'*, *retry_wait_min=10*, *retry_wait_rand=60*, *num_gpus_available=None*)
Acquire a number of GPUs in a transactional way. This method should be used inside a *with* statement. Will try to acquire all the requested number of GPUs. If currently not enough GPUs are available all locks will be released and we wait until we retry. Will retry until enough GPUs become available.

Parameters

- **requested_device_ids** (List[int]) – The requested device ids, each number is either negative indicating the number of GPUs that will be allocated, or positive indicating we want to acquire a specific device id.
- **lock_dir** (str) – The directory for storing the lock file.
- **retry_wait_min** (int) – The minimum number of seconds to wait between retries.
- **retry_wait_rand** (int) – Randomly add between 0 and *retry_wait_rand* seconds to the wait time.
- **num_gpus_available** (Optional[int]) – The number of GPUs available, if None we will call get_num_gpus().

Returns yields a list of GPU ids.

sockeye.utils.**average_arrays**(*arrays*)
Take a list of arrays of the same shape and take the element wise average.

Parameters **arrays** (List[NDArray]) – A list of NDArrays with the same shape that will be averaged.

Return type NDArray

Returns The average of the NDArrays in the same context as arrays[0].

sockeye.utils.**cast_conditionally**(*data*, *dtype*)
Workaround until no-op cast will be fixed in MXNet codebase. Creates cast symbol only if dtype is different from default one, i.e. float32.

Parameters

- **data** (Symbol) – Input symbol.
- **dtype** (str) – Target dtype.

Return type Symbol

Returns Cast symbol or just data symbol.

sockeye.utils.**check_condition**(*condition*, *error_message*)
Check the condition and if it is not met, exit with the given error message and error_code, similar to assertions.

Parameters

- **condition** (bool) – Condition to check.
- **error_message** (str) – Error message to show to the user.

sockeye.utils.**check_version**(*version*)
Checks given version against code version and determines compatibility. Throws if versions are incompatible.

Parameters **version** (str) – Given version.

sockeye.utils.**chunks**(*some_list*, *n*)
Yield successive n-sized chunks from l.

Return type Iterable[List[~T]]

sockeye.utils.**cleanup_params_files**(*output_folder*, *max_to_keep*, *checkpoint*, *best_checkpoint*)
Deletes oldest parameter files from a model folder.

Parameters

- **output_folder** (str) – Folder where param files are located.
- **max_to_keep** (int) – Maximum number of files to keep, negative to keep all.
- **checkpoint** (int) – Current checkpoint (i.e. index of last params file created).
- **best_checkpoint** (int) – Best checkpoint. The parameter file corresponding to this checkpoint will not be deleted.

sockeye.utils.**compute_lengths**(*sequence_data*)
Computes sequence lengths of PAD_ID-padded data in sequence_data.

Parameters **sequence_data** (Symbol) – Input data. Shape: (batch_size, seq_len).

Return type Symbol

Returns Length data. Shape: (batch_size,).

sockeye.utils.**expand_requested_device_ids**(*requested_device_ids*)
Transform a list of device id requests to concrete device ids. For example on a host with 8 GPUs when requesting [-4, 3, 5] you will get [0, 1, 2, 3, 4, 5]. Namely you will get device 3 and 5, as well as 3 other available device ids (starting to fill up from low to high device ids).

Parameters **requested_device_ids** (List[int]) – The requested device ids, each number is either negative indicating the number of GPUs that will be allocated, or positive indicating we want to acquire a specific device id.

Return type List[int]

Returns A list of device ids.

sockeye.utils.**get_alignments**(*attention_matrix*, *threshold=0.9*)
Yields hard alignments from an attention_matrix (target_length, source_length) given a threshold.

Parameters

- **attention_matrix** (ndarray) – The attention matrix.
- **threshold** (float) – The threshold for including an alignment link in the result.

Return type Iterator[Tuple[int, int]]

Returns Generator yielding strings of the form 0-0, 0-1, 2-1, 2-2, 3-4...

sockeye.utils.**get_gpu_memory_usage**(*ctx*)
Returns used and total memory for GPUs identified by the given context list.

Parameters **ctx** (List[Context]) – List of MXNet context devices.

Return type Dict[int, Tuple[int, int]]

Returns Dictionary of device id mapping to a tuple of (memory used, memory total).

sockeye.utils.**get_num_gpus**()
Gets the number of GPUs available on the host (depends on nvidia-smi).

Return type int

Returns The number of GPUs on the system.

sockeye.utils.**get_tokens**(*line*)
Yields tokens from input string.

Parameters **line** (str) – Input string.

Return type Iterator[str]

Returns Iterator over tokens.

sockeye.utils.**get_validation_metric_points**(*model_path*, *metric*)
Returns tuples of value and checkpoint for given metric from metrics file at model_path. :type model_path: str :param model_path: Model path containing .metrics file. :type metric: str :param metric: Metric values to extract. :return: List of tuples (value, checkpoint).

sockeye.utils.**grouper**(*iterable*, *size*)
Collect data into fixed-length chunks or blocks without discarding underfilled chunks or padding them.

Parameters

- **iterable** (Iterable[+T_co]) – A sequence of inputs.
- **size** (int) – Chunk size.

Return type Iterable[+T_co]

Returns Sequence of chunks.

sockeye.utils.**load_params**(*fname*)
Loads parameters from a file.

Parameters **fname** (str) – The file containing the parameters.

Return type Tuple[Dict[str, NDArray], Dict[str, NDArray]]

Returns Mapping from parameter names to the actual parameters for both the arg parameters and the aux parameters.

sockeye.utils.**load_version**(*fname*)
Loads version from file.

Parameters **fname** (str) – Name of file to load version from.

Return type str

Returns Version string.

sockeye.utils.**log_basic_info**(*args*)
Log basic information like version number, arguments, etc.

Parameters **args** – Arguments as returned by argparse.

Return type None

sockeye.utils.**metric_value_is_better**(*new*, *old*, *metric*)
Returns true if new value is strictly better than old for given metric.

Return type bool

sockeye.utils.**parse_version**(*version_string*)
Parse version string into release, major, minor version.

Parameters **version_string** (str) – Version string.

Return type Tuple[str, str, str]

Returns Tuple of strings.

sockeye.utils.**plot_attention**(*attention_matrix*, *source_tokens*, *target_tokens*, *filename*)
Uses matplotlib for creating a visualization of the attention matrix.

Parameters

- **attention_matrix** (ndarray) – The attention matrix.
- **source_tokens** (List[str]) – A list of source tokens.
- **target_tokens** (List[str]) – A list of target tokens.
- **filename** (str) – The file to which the attention visualization will be written to.

sockeye.utils.**print_attention_text**(*attention_matrix*, *source_tokens*, *target_tokens*, *threshold*)
Prints the attention matrix to standard out.

Parameters

- **attention_matrix** (ndarray) – The attention matrix.
- **source_tokens** (List[str]) – A list of source tokens.
- **target_tokens** (List[str]) – A list of target tokens.
- **threshold** (float) – The threshold for including an alignment link in the result.

sockeye.utils.**read_metrics_file**(*path*)
Reads lines metrics file and returns list of mappings of key and values.

Parameters **path** (str) – File to read metric values from.

Return type List[Dict[str, Any]]

Returns Dictionary of metric names (e.g. perplexity-train) mapping to a list of values.

sockeye.utils.**save_graph**(*symbol*, *filename*, *hide_weights=True*)
Dumps computation graph visualization to .pdf and .dot file.

Parameters

- **symbol** (Symbol) – The symbol representing the computation graph.
- **filename** (str) – The filename to save the graphic to.
- **hide_weights** (bool) – If true the weights will not be shown.

sockeye.utils.**save_params**(*arg_params*, *fname*, *aux_params=None*)
Saves the parameters to a file.

Parameters

- **arg_params** (Mapping[str, NDArray]) – Mapping from parameter names to the actual parameters.
- **fname** (str) – The file name to store the parameters in.

- **aux_params** (Optional[Mapping[str, NDArray]]) – Optional mapping from parameter names to the auxiliary parameters.

sockeye.utils.**seedRNGs**(*seed*)
Seed the random number generators (Python, Numpy and MXNet)

Parameters **seed** (int) – The random seed.

Return type None

sockeye.utils.**smart_open**(*filename*, *mode='rt'*, *ftype='auto'*, *errors='replace'*)
Returns a file descriptor for filename with UTF-8 encoding. If mode is "rt", file is opened read-only. If ftype is "auto", uses gzip iff filename endswith .gz. If ftype is {"gzip","gz"}, uses gzip.

Note: encoding error handling defaults to "replace"

Parameters

- **filename** (str) – The filename to open.
- **mode** (str) – Reader mode.
- **ftype** (str) – File type. If 'auto' checks filename suffix for gz to try gzip.open
- **errors** (str) – Encoding error handling during reading. Defaults to 'replace'

Returns File descriptor

sockeye.utils.**topk**(*scores*, *k*, *batch_size*, *offset*, *use_mxnet_topk*)
Get the lowest k elements per sentence from a *scores* matrix.

Parameters

- **scores** (NDArray) – Vocabulary scores for the next beam step. (batch_size * beam_size, target_vocabulary_size)
- **k** (int) – The number of smallest scores to return.
- **batch_size** (int) – Number of sentences being decoded at once.
- **offset** (NDArray) – Array to add to the hypothesis indices for offsetting in batch decoding.
- **use_mxnet_topk** (bool) – True to use the mxnet implementation or False to use the numpy one.

Return type Tuple[NDArray, NDArray, NDArray]

Returns The row indices, column indices and values of the k smallest items in matrix.

sockeye.utils.**uncast_conditionally**(*data*, *dtype*)
Workaround until no-op cast will be fixed in MXNet codebase. Creates cast to float32 symbol only if dtype is different from default one, i.e. float32.

Parameters

- **data** (Symbol) – Input symbol.
- **dtype** (str) – Input symbol dtype.

Return type Symbol

Returns Cast symbol or just data symbol.

sockeye.utils.**write_metrics_file**(*metrics*, *path*)
Write metrics data to tab-separated file.

Parameters

- **metrics** (List[Dict[str, Any]]) – metrics data.
- **path** (str) – Path to write to.

2.6.31 sockeye.vocab module

sockeye.vocab.**build_from_paths**(*paths*, *num_words=50000*, *min_count=1*)
Creates vocabulary from paths to a file in sentence-per-line format. A sentence is just a whitespace delimited list of tokens. Note that special symbols like the beginning of sentence (BOS) symbol will be added to the vocabulary.

Parameters

- **paths** (List[str]) – List of paths to files with one sentence per line.
- **num_words** (int) – Maximum number of words in the vocabulary.
- **min_count** (int) – Minimum occurrences of words to be included in the vocabulary.

Return type Dict[str, int]

Returns Word-to-id mapping.

sockeye.vocab.**build_vocab**(*data*, *num_words=50000*, *min_count=1*)
Creates a vocabulary mapping from words to ids. Increasing integer ids are assigned by word frequency, using lexical sorting as a tie breaker. The only exception to this are special symbols such as the padding symbol (PAD).

Parameters

- **data** (Iterable[str]) – Sequence of sentences containing whitespace delimited tokens.
- **num_words** (int) – Maximum number of words in the vocabulary.
- **min_count** (int) – Minimum occurrences of words to be included in the vocabulary.

Return type Dict[str, int]

Returns Word-to-id mapping.

sockeye.vocab.**get_ordered_tokens_from_vocab**(*vocab*)
Returns the list of tokens in a vocabulary, ordered by increasing vocabulary id.

Parameters **vocab** (Dict[str, int]) – Input vocabulary.

Return type List[str]

Returns List of tokens.

sockeye.vocab.**load_or_create_vocab**(*data*, *vocab_path*, *num_words*, *word_min_count*)
If the vocabulary path is defined, the vocabulary is loaded from the path. Otherwise, it is built from the data file. No writing to disk occurs.

Return type Dict[str, int]

sockeye.vocab.**load_or_create_vocabs**(*source_paths*, *target_path*, *source_vocab_paths*, *target_vocab_path*, *shared_vocab*, *num_words_source*, *word_min_count_source*, *num_words_target*, *word_min_count_target*)
Returns vocabularies for source files (including factors) and target. If the respective vocabulary paths are not None, the vocabulary is read from the path and returned. Otherwise, it is built from the support and saved to the path.

Parameters

- **source_paths** (List[str]) – The path to the source text (and optional token-parallel factor files).
- **target_path** (str) – The target text.
- **source_vocab_paths** (List[Optional[str]]) – The source vocabulary path (and optional factor vocabulary paths).
- **target_vocab_path** (Optional[str]) – The target vocabulary path.
- **shared_vocab** (bool) – Whether the source and target vocabularies are shared.
- **num_words_source** (int) – Number of words in the source vocabulary.
- **word_min_count_source** (int) – Minimum frequency of words in the source vocabulary.
- **num_words_target** (int) – Number of words in the target vocabulary.
- **word_min_count_target** (int) – Minimum frequency of words in the target vocabulary.

Return type Tuple[List[Dict[str, int]], Dict[str, int]]

Returns List of source vocabularies (for source and factors), and target vocabulary.

sockeye.vocab.**load_source_vocabs**(*folder*)

Loads source vocabularies from folder. The first element in the list is the primary source vocabulary. Other elements correspond to optional additional source factor vocabularies found in folder.

Parameters **folder** (str) – Source folder.

Return type List[Dict[str, int]]

Returns List of vocabularies.

sockeye.vocab.**load_target_vocab**(*folder*)

Loads target vocabulary from folder.

Parameters **folder** (str) – Source folder.

Return type Dict[str, int]

Returns Target vocabulary

sockeye.vocab.**reverse_vocab**(*vocab*)

Returns value-to-key mapping from key-to-value-mapping.

Parameters **vocab** (Dict[str, int]) – Key to value mapping.

Return type Dict[int, str]

Returns A mapping from values to keys.

sockeye.vocab.**save_source_vocabs**(*source_vocabs*, *folder*)

Saves source vocabularies (primary surface form vocabulary) and optional factor vocabularies to folder.

Parameters

- **source_vocabs** (List[Dict[str, int]]) – List of source vocabularies.
- **folder** (str) – Destination folder.

sockeye.vocab.**save_target_vocab**(*target_vocab*, *folder*)

Saves target vocabulary to folder.

Parameters

- **target_vocab** (Dict[str, int]) – Target vocabulary.
- **folder** (str) – Destination folder.

sockeye.vocab.**vocab_from_json**(*path*, *encoding='utf-8'*)
Saves vocabulary in json format.

Parameters

- **path** (str) – Path to json file containing the vocabulary.
- **encoding** (str) – Vocabulary encoding.

Return type Dict[str, int]

Returns The loaded vocabulary.

sockeye.vocab.**vocab_to_json**(*vocab*, *path*)
Saves vocabulary in human-readable json.

Parameters

- **vocab** (Dict[str, int]) – Vocabulary mapping.
- **path** (str) – Output file path.

CHAPTER 3

Resources

- genindex
- modindex

Python Module Index

S

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W