Sockeye Documentation

Release 1.16.6

amazon

Feb 15, 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 e-prints},
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. 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-1.0.0
- 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.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), or `90` (9.0).

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

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

2.1.2.4 Optional dependencies

In order to track learning curves during training you can optionally install dmlc's fork of tensorboard (`pip install tensorboard==1.0.0a6`).

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

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. If you have not installed dmlc's fork of tensorboard do so as follows:

```
> pip install tensorboard==1.0.0a6
```

Now when training specify the additional command line parameter `--use-tensorboard` to `sockeye.train`. Then 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.txt
> pip install -r requirements.dev.txt
> pip install -r 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 tests

Unit tests are written using py.test. They can be run like this:

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

2.3.5 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
   ```

2.3.6 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.

2.3.7 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 [1.16.6]

2.4.1.1 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.

2.4.2 [1.16.5]

2.4.2.1 Changed

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

2.4.3 [1.16.4]

2.4.3.1 Changed

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

2.4.4 [1.16.3]

2.4.4.1 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.

2.4.5 [1.16.2]

2.4.5.1 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.

2.4.6 [1.16.1]

2.4.6.1 Fixed

- Fixed entry points in `setup.py`.

2.4.7 [1.16.0]

2.4.7.1 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`.

2.4.7.2 Added

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

2.4.8 [1.15.8]

2.4.8.1 Fixed

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

2.4.9 [1.15.7]

2.4.9.1 Fixed

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

2.4.10 [1.15.6]

2.4.10.1 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.

2.4.10.2 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.

2.4.11 [1.15.5]

2.4.11.1 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`.

2.4.12 [1.15.4]

2.4.12.1 Added

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

2.4.13 [1.15.3]

2.4.13.1 Added

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

2.4.14 [1.15.2]

2.4.14.1 Added

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

2.4.15 [1.15.1]

2.4.15.1 Added

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

2.4.16 [1.15.0]

2.4.16.1 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`.

2.4.17 [1.14.3]

2.4.17.1 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.

2.4.18 [1.14.2]

2.4.18.1 Added

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

2.4.18.2 Changed

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

2.4.19 [1.14.1]

2.4.19.1 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.

2.4.20 [1.14.0]

2.4.20.1 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.

2.4.20.2 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.

2.4.21 [1.13.2]

2.4.21.1 Added

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

2.4.22 [1.13.1]

2.4.22.1 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`

2.4.23 [1.13.0]

2.4.23.1 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.

2.4.23.2 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`).

2.4.24 [1.12.2]

2.4.24.1 Changed

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

2.4.25 [1.12.1]

2.4.25.1 Changed

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

2.4.26 [1.12.0]

2.4.26.1 Changed

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

2.4.27 [1.11.2]

2.4.27.1 Fixed

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

2.4.28 [1.11.1]

2.4.28.1 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.

2.4.29 [1.11.0]

2.4.29.1 Added

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

2.4.29.2 Changed

- Bleu computation based on Sacrebleu.

2.4.30 [1.10.5]

2.4.30.1 Fixed

- Fixed yet another bug with the data iterator.

2.4.31 [1.10.4]

2.4.31.1 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.

2.4.32 [1.10.3]

2.4.32.1 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`.

2.4.33 [1.10.2]

2.4.33.1 Added

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

2.4.34 [1.10.1]

2.4.34.1 Changed

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

2.4.35 [1.10.0]

2.4.35.1 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.

2.4.35.2 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.

2.4.35.3 Removed

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

2.4.36 [1.9.0]

2.4.36.1 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.

2.4.37 [1.8.4]

2.4.37.1 Added

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

2.4.38 [1.8.3]

2.4.38.1 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`

2.4.39 [1.8.2]

2.4.39.1 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`.

2.4.40 [1.8.1]

2.4.40.1 Changed

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

2.4.41 [1.8.0]

2.4.41.1 Added

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

2.4.41.2 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

2.6.2 sockeye.average module

2.6.3 sockeye.callback module

2.6.4 sockeye.checkpoint_decoder module

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

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_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.

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*)
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.

class sockeye.decoder.**Decoder**

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.

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.**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'*, *context_gating=False*, *layer_normalization=False*, *attention_in_upper_layers=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.
- **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.

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_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.

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

2.6.9 sockeye.encoder module

class sockeye.encoder.**AddLearnedPositionalEmbeddings**(*num_embed*, *max_seq_len*, *prefix*, *embed_weight=None*)

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.

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)

class sockeye.encoder.**AddSinCosPositionalEmbeddings**(*num_embed*, *prefix*, *scale_up_input*, *scale_down_positions*)

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.

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)

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.
- **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*)
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.

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).

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*)
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.

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.

class sockeye.encoder.**ConvolutionalEncoderConfig**(*num_embed*, *max_seq_len_source*, *cnn_config*, *num_layers*, *positional_embedding_type*)

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.

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

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.

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.**Encoder**

Bases: abc.ABC

Generic encoder interface.

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*)
Bases: *sockeye.encoder.Encoder*

A sequence of encoders is itself an encoder.

Parameters **encoders** (List[*Encoder*]) – List of encoders.

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.**NoOpPositionalEmbeddings**(*num_embed*)
Bases: sockeye.encoder.PositionalEncoder

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

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.
- **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*)
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.

class sockeye.encoder.**ReverseSequence**(*num_hidden*)
Bases: *sockeye.encoder.Encoder*

Reverses the input sequence. Requires time-major layout.

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*)
Creates a convolutional encoder.

Parameters

- **config** (*ConvolutionalEncoderConfig*) – Configuration for convolutional encoder.
- **embed_weight** – Optionally use an existing embedding matrix instead of creating a new one.

Return type *Encoder*

Returns Encoder instance.

sockeye.encoder.**get_recurrent_encoder**(*config*)
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.

Return type *Encoder*

Returns Encoder instance.

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

Parameters **config** (TransformerConfig) – Configuration for transformer encoder.

Return type *Encoder*

Returns Encoder instance.

2.6.10 sockeye.inference module

2.6.11 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*)
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.

Return type Initializer

Returns Mixed initializer.

2.6.12 sockeye.layers module

class sockeye.layers.**LayerNormalization**(*num_hidden*, *prefix=None*, *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

- **num_hidden** (int) – Number of hidden units of layer to be normalized.
- **prefix** (Optional[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.

static moments(*inputs*)

Computes mean and variance of the last dimension of a Symbol.

Parameters **inputs** (Symbol) – Shape: (d0, . . . , dn, hidden).

Return type Tuple[Symbol, Symbol]

Returns mean, var: Shape: (d0, . . . , dn, 1).

normalize(*inputs*, *eps=1e-06*)

Normalizes hidden units of inputs as follows:

inputs = scale * (inputs - mean) / sqrt(var + eps) + shift

Normalization is performed over the last dimension of the input data.

Parameters

- **inputs** (Symbol) – Inputs to normalize. Shape: (d0, . . . , dn, num_hidden).
- **eps** (float) – Variance epsilon.

Return type Symbol

Returns inputs_norm: Normalized inputs. Shape: (d0, . . . , dn, num_hidden).

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).

2.6.13 sockeye.lexicon module

2.6.14 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

2.6.15 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.

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*

2.6.16 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_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.

2.6.17 sockeye.model module

2.6.18 sockeye.output_handler module

2.6.19 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*)

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.

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.

2.6.20 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_'*)

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.

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*)

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.

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**(*num_hidden*)
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 **num_hidden** (int) – Number of hidden units the source will be projected to.

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*, *scale=None*)

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.
- **scale** (Optional[float]) – Optionally scale query before dot product [Vaswani et al, 2017].

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_'*)

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_source_seq_len*)

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_source_seq_len** (int) – Maximum length of source sequences.

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*, *attention_num_hidden*, *layer_normalization=False*, *config_coverage=None*)

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.
- **attention_num_hidden** (int) – Number of hidden units.
- **layer_normalization** (bool) – If true, normalizes hidden layer outputs before tanh activation.
- **config_coverage** (Optional[*CoverageConfig*]) – Optional coverage config.

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.**MultiHeadDotAttention**(*input_previous_word*, *num_hidden*, *heads*)
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.
- **num_hidden** (int) – Number of hidden units.
- **heads** (int) – Number of attention heads / independently computed attention scores.

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*)
Returns an Attention instance based on attention_type.

Parameters
- **config** (*AttentionConfig*) – Attention configuration.
- **max_seq_len** (int) – Maximum length of source sequences.

Return type *Attention*

Returns Instance of Attention.

sockeye.rnn_attention.**get_context_and_attention_probs**(*values*, *length*, *logits*)
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).

Return type Tuple[Symbol, Symbol]

Returns context: (batch_size, encoder_num_hidden), attention_probs: (batch_size, seq_len).

sockeye.rnn_attention.**mask_attention_scores**(*logits*, *length*)
Masks attention scores according to sequence length.

Parameters

- **logits** (Symbol) – Shape: (batch_size, seq_len, 1).
- **length** (Symbol) – Shape: (batch_size,).

Return type Symbol

Returns Masked logits: (batch_size, seq_len, 1).

2.6.21 sockeye.training module

2.6.22 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*, *num_hidden*, *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.

2.6.23 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.

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.**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.**compute_lengths**(*sequence_data*)
Computes sequence lenghts 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 Optional[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.

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.**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.**smallest_k**(*matrix*, *k*, *only_first_row=False*)
Find the smallest elements in a numpy matrix.

Parameters

- **matrix** (ndarray) – Any matrix.
- **k** (int) – The number of smallest elements to return.
- **only_first_row** (bool) – If true the search is constrained to the first row of the matrix.

Return type Tuple[Tuple[ndarray, ndarray], ndarray]

Returns The row indices, column indices and values of the k smallest items in matrix.

sockeye.utils.**smallest_k_mx**(*matrix*, *k*, *only_first_row=False*)
Find the smallest elements in a NDarray.

Parameters

- **matrix** (NDArray) – Any matrix.
- **k** (int) – The number of smallest elements to return.
- **only_first_row** (bool) – If True the search is constrained to the first row of the matrix.

Return type Tuple[Tuple[ndarray, ndarray], ndarray]

Returns The row indices, column indices and values of the k smallest items in matrix.

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.**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.

2.6.24 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.**reverse_vocab**(*vocab*)
Returns value-to-key mapping from key-to-value-mapping.

Parameters **vocab** (Mapping[~KT, +VT_co]) – Key to value mapping.

Return type Dict[int, str]

Returns A mapping from values to keys.

sockeye.vocab.**vocab_from_json**(*path*, *encoding='utf-8'*)
Saves vocabulary in json format.

Parameters **path** (str) – Path to json file containing the vocabulary.

Return type Dict[str, int]

Returns The loaded vocabulary.

sockeye.vocab.**vocab_from_json_or_pickle**(*path*)
Try loading the json version of the vocab and fall back to pickle for backwards compatibility.

Parameters **path** – Path to vocab without the json suffix. If it exists the *path* + '.json' will be loaded as a JSON object and otherwise *path* is loaded as a pickle object.

Return type Dict[str, int]

Returns The loaded vocabulary.

sockeye.vocab.**vocab_from_pickle**(*path*)
Saves vocabulary in pickle format.

Parameters **path** (str) – Path to pickle file containing the vocabulary.

Return type Dict[str, int]

Returns The loaded vocabulary.

sockeye.vocab.**vocab_to_json**(*vocab*, *path*)
Saves vocabulary in human-readable json.

Parameters

- **vocab** (Mapping[~KT, +VT_co]) – Vocabulary mapping.
- **path** (str) – Output file path.

sockeye.vocab.**vocab_to_pickle**(*vocab*, *path*)
Saves vocabulary in pickle format.

Parameters

- **vocab** (Mapping[~KT, +VT_co]) – Vocabulary mapping.
- **path** (str) – Output file path.

CHAPTER 3

Resources

- genindex
- modindex

Python Module Index

S

Index

G

S

T

V

W