NMT-Keras Documentation

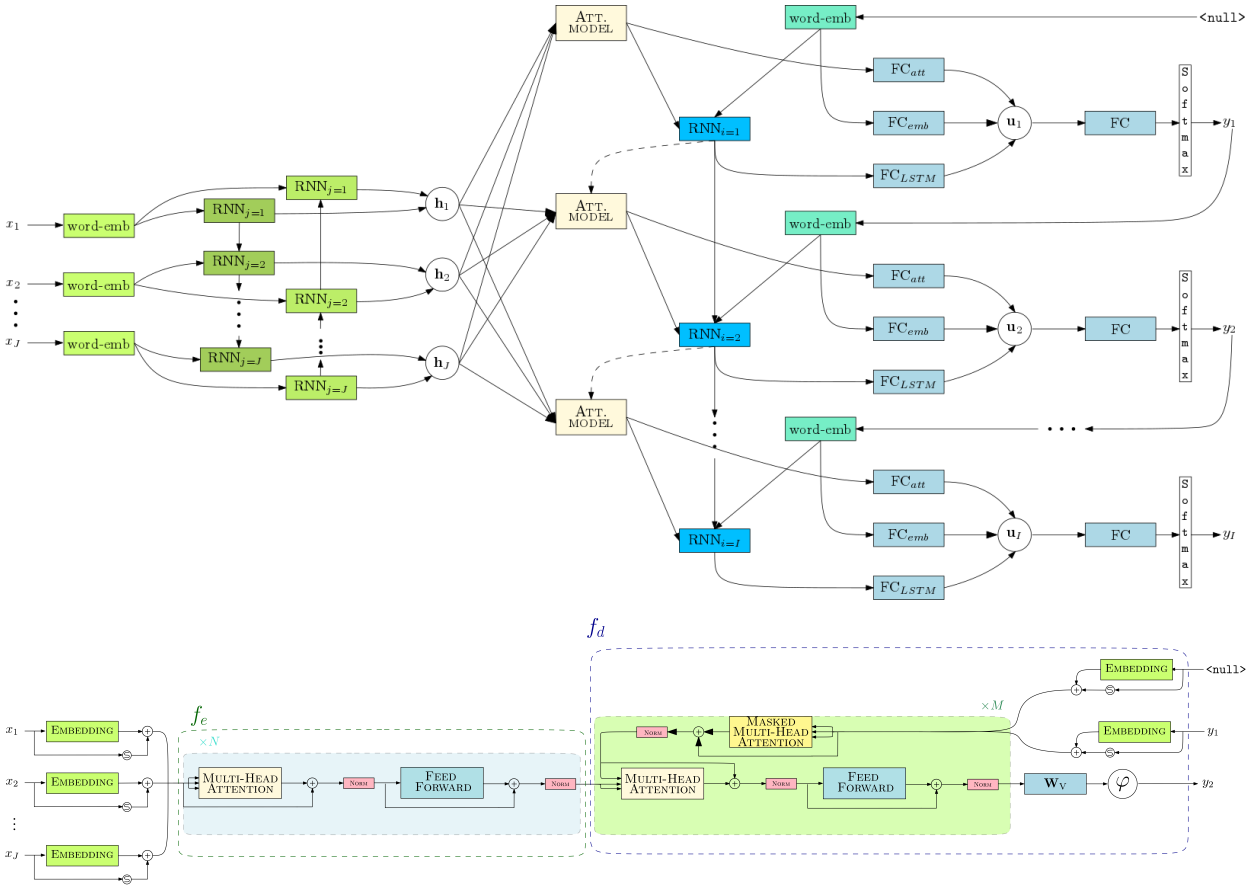
Release 0.2

Álvaro Peris

Jul 30, 2021

Contents

Neural Machine Translation with Keras.



CHAPTER 1

Features

- Attention RNN and Transformer models.
- Online learning and Interactive neural machine translation (INMT). See the interactive NMT branch.
- Tensorboard integration. Training process, models and word embeddings visualization.
- Attention model over the input sequence of annotations. - Supporting Bahdanau (Add) and Luong (Dot) attention mechanisms. - Also supports double stochastic attention.
- Peeked decoder: The previously generated word is an input of the current timestep.
- Beam search decoding. - Featuring length and source coverage normalization.
- Ensemble decoding.
- Translation scoring.
- N-best list generation (as byproduct of the beam search process).
- Support for GRU/LSTM networks: - Regular GRU/LSTM units. - Conditional GRU/LSTM units in the decoder. - Multilayered residual GRU/LSTM networks.
- Unknown words replacement.
- Use of pretrained (Glove or Word2Vec) word embedding vectors.
- MLPs for initializing the RNN hidden and memory state.
- Spearmint wrapper for hyperparameter optimization.
- Client-server architecture for web demos.

CHAPTER 2

Guide

2.1 Installation

Assuming that you have pip installed, run:

```
git clone https://github.com/lvapeab/nmt-keras
cd nmt-keras
pip install -r requirements.txt
```

for obtaining the required packages for running this library.

Nevertheless, it is highly recommended to install and configure Theano or Tensorflow with the GPU and speed optimizations enabled.

Alternatively, you can run the install.sh script, which will also downloads Python:

```
git clone https://github.com/lvapeab/nmt-keras
cd nmt-keras
bash ./install.sh
```

2.1.1 Requirements

- Our version of Keras.
- Multimodal Keras Wrapper. See the documentation and tutorial.
- Coco-caption evaluation package (Only required to perform evaluation).

2.2 Usage

2.2.1 Training

1) Set a training configuration in the config.py script. Each parameter is commented. See the documentation file for further info about each specific hyperparameter. You can also specify the parameters when calling the main.py script following the syntax *Key=Value*

2) Train!:

```
python main.py
```

2.2.2 Decoding

Once we have our model trained, we can translate new text using the sample_ensemble.py script. Please refer to the ensembling tutorial for more details about this script. In short, if we want to use the models from the first three epochs to translate the *examples/EuTrans/test.en* file, just run:

```
python sample_ensemble.py --models trained_models/tutorial_model/epoch_1 \
                                   trained_models/tutorial_model/epoch_2 \
                          --dataset datasets/Dataset_tutorial_dataset.pkl \
                          --text examples/EuTrans/test.en
```

2.2.3 Scoring

The score.py script can be used to obtain the (-log)probabilities of a parallel corpus. Its syntax is the following:

```
python score.py --help
usage: Use several translation models for scoring source--target pairs
   [-h] -ds DATASET [-src SOURCE] [-trg TARGET] [-s SPLITS [SPLITS ...]]
   [-d DEST] [-v] [-c CONFIG] --models MODELS [MODELS ...]
optional arguments:
    -h, --help            show this help message and exit
    -ds DATASET, --dataset DATASET
                          Dataset instance with data
    -src SOURCE, --source SOURCE
                          Text file with source sentences
    -trg TARGET, --target TARGET
                          Text file with target sentences
    -s SPLITS [SPLITS ...], --splits SPLITS [SPLITS ...]
                          Splits to sample. Should be already included into the
                          dataset object.
    -d DEST, --dest DEST  File to save scores in
    -v, --verbose         Be verbose
    -c CONFIG, --config CONFIG
                          Config pkl for loading the model configuration. If not
                          specified, hyperparameters are read from config.py
    --models MODELS [MODELS ...]
                          path to the models
```

2.3 Configuration options

This document describes the available hyperparameters used for training NMT-Keras.

These hyperparameters are set in the config.py script or via command-line-interface.

2.3.1 Naming and experiment setup

- **DATASET_NAME**: Task name. Used for naming and for indexing files.
- **SRC_LAN**: Language of the source text. Used for naming.
- **TRG_LAN**: Language of the target text. Used for naming and for coputing language-dependent metrics (e.g. Meteor)
- **DATA_ROOT_PATH**: Path to the data
- **TEXT_FILES**: Dictionary containing the splits ('train/val/test) and the files corresponding to each one. The source/target languages will be appended to these files.

2.3.2 Input/output

- **INPUTS_IDS_DATASET**: Name of the inputs of the Dataset class.
- **OUTPUTS_IDS_DATASET**: Name of the outputs of the Dataset class.
- **INPUTS_IDS_MODEL**: Name of the inputs of the Model.
- **OUTPUTS_IDS_MODEL**: Name of the outputs of the Model.

2.3.3 Evaluation

- **METRICS**: List of metric used for evaluating the model. The coco package is recommended.
- **EVAL_ON_SETS**: List of splits ('train', 'val', 'test') to evaluate with the metrics from METRICS. Typically: 'val'
- **EVAL_ON_SETS_KERAS**: List of splits ('train', 'val', 'test') to evaluate with the Keras metrics.
- **START_EVAL_ON_EPOCH**: The evaluation starts at this epoch.
- **EVAL_EACH_EPOCHS**: Whether the evaluation frequency units are epochs or updates.
- **EVAL_EACH**: Evaluation frequency.

2.3.4 Decoding

- **SAMPLING**: Decoding mode. Only 'max_likelihood' tested.
- **TEMPERATURE**: Multinomial sampling temerature.
- **BEAM_SEARCH**: Switches on-off the beam search.
- **BEAM_SIZE**: Beam size.
- **OPTIMIZED_SEARCH**: Encode the source only once per sample (recommended).

2.3.5 Search normalization

- **SEARCH_PRUNING**: Apply pruning strategies to the beam search method. It will likely increase decoding speed, but decrease quality.
- **MAXLEN_GIVEN_X**: Generate translations of similar length to the source sentences.
- **MAXLEN_GIVEN_X_FACTOR**: The hypotheses will have (as maximum) the number of words of the source sentence * LENGTH_Y_GIVEN_X_FACTOR.
- **MINLEN_GIVEN_X**: Generate translations of similar length to the source sentences.
- **MINLEN_GIVEN_X_FACTOR**: The hypotheses will have (as minimum) the number of words of the source sentence / LENGTH_Y_GIVEN_X_FACTOR.
- **LENGTH_PENALTY**: Apply length penalty (Wu et al. (2016)).
- **LENGTH_NORM_FACTOR**: Length penalty factor (Wu et al. (2016)).
- **COVERAGE_PENALTY**: Apply source coverage penalty (Wu et al. (2016)).
- **COVERAGE_NORM_FACTOR**: Coverage penalty factor (Wu et al. (2016)).
- **NORMALIZE_SAMPLING**: Alternative (simple) length normalization. Normalize hypotheses scores according to their length.
- **ALPHA_FACTOR**: Normalization according to |h|**ALPHA_FACTOR (Wu et al. (2016)).

2.3.6 Sampling

- **SAMPLE_ON_SETS**: Splits from where we'll sample.
- **N_SAMPLES**: Number of samples generated
- **START_SAMPLING_ON_EPOCH**: First epoch where to start the sampling counter
- **SAMPLE_EACH_UPDATES**: Sampling frequency (always in #updates)

2.3.7 Unknown words treatment

- **POS_UNK**: Enable unknown words replacement strategy.
- **HEURISTIC**: Heuristic followed for replacing unks:
- 0: Replace the UNK by the correspondingly aligned source.
- 1: Replace the UNK by the translation (given by an external dictionary of the aligned source.
- 2: Replace the UNK by the translation (given by an external dictionary of the aligned source only if it starts with a lowercase. Otherwise, copies the source word.
- **ALIGN_FROM_RAW**: Align using the source files or the short-list model vocabulary.
- **MAPPING**: Mapping dictionary path (for heuristics 1 and 2). Obtained with the build_mapping_file script.

2.3.8 Word representation

- **TOKENIZATION_METHOD**: Tokenization applied to the input and output text.
- **DETOKENIZATION_METHOD**: Detokenization applied to the input and output text.

- **APPLY_DETOKENIZATION**: Wheter we apply the detokenization method
- **TOKENIZE_HYPOTHESES**: Whether we tokenize the hypotheses (for computing metrics).
- **TOKENIZE_REFERENCES**: Whether we tokenize the references (for computing metrics).
- **BPE_CODES_PATH**: If *TOKENIZATION_METHOD == 'tokenize_bpe'*, sets the path to the learned BPE codes.

2.3.9 Text representation

- **FILL**: Padding mode: Insert zeroes at the 'start', 'center' or 'end'.
- **PAD_ON_BATCH**: Make batches of a fixed number of timesteps or pad to the maximum length of the mini-batch.

2.3.10 Input text

- **INPUT_VOCABULARY_SIZE**: Input vocabulary size. Set to 0 for using all, otherwise it will be truncated to these most frequent words.
- **MIN_OCCURRENCES_INPUT_VOCAB**: Discard all input words with a frequency below this threshold.
- **MAX_INPUT_TEXT_LEN**: Maximum length of the input sentences.

2.3.11 Output text

- **INPUT_VOCABULARY_SIZE**: Output vocabulary size. Set to 0 for using all, otherwise it will be truncated to these most frequent words.
- **MIN_OCCURRENCES_OUTPUT_VOCAB**: Discard all output words with a frequency below this threshold.
- **MAX_INPUT_TEXT_LEN**: Maximum length of the output sentences.
- **MAX_OUTPUT_TEXT_LEN_TEST**: Maximum length of the output sequence during test time.

2.3.12 Optimization

- **LOSS**: Loss function to optimize.
- **CLASSIFIER_ACTIVATION**: Last layer activation function.
- **SAMPLE_WEIGHTS**: Apply a mask to the output sequence. Should be set to True.
- **LR_DECAY**: Reduce the learning rate each this number of epochs. Set to None if don't want to decay the learning rate
- **LR_GAMMA**: Decay rate.
- **LABEL_SMOOTHING**: Epsilon value for label smoothing. Only valid for 'categorical_crossentropy' loss. See [1512.00567](arxiv.org/abs/1512.00567).

Optimizer setup

- **OPTIMIZER**: Optimizer to use. See the available Keras optimizers.
- **LR**: Learning rate.
- **CLIP_C**: During training, clip L2 norm of gradients to this value.
- **CLIP_V**: During training, clip absolute value of gradients to this value.
- **USE_TF_OPTIMIZER**: Use native Tensorflow's optimizer (only for the Tensorflow backend).

Advanced parameters for optimizers

- **MOMENTUM**: Momentum value (for SGD optimizer).
- **NESTEROV_MOMENTUM**: Use Nesterov momentum (for SGD optimizer).
- **RHO**: Rho value (for Adadelta and RMSprop optimizers).
- **BETA_1**: Beta 1 value (for Adam, Adamax Nadam optimizers).
- **BETA_2**: Beta 2 value (for Adam, Adamax Nadam optimizers).
- **EPSILON**: Oprimizers epsilon value.

2.3.13 Learning rate schedule

- **LR_DECAY**: Frequency (number of epochs or updates) between LR annealings. Set to None for not decay the learning rate.
- **LR_GAMMA**: Multiplier used for decreasing the LR.
- **LR_REDUCE_EACH_EPOCHS**: Reduce each LR_DECAY number of epochs or updates.
- **LR_START_REDUCTION_ON_EPOCH**: Epoch to start the reduction.
- **LR_REDUCER_TYPE**: Function to reduce. 'linear' and 'exponential' implemented.
- **LR_REDUCER_EXP_BASE**: Base for the exponential decay.
- **LR_HALF_LIFE**: Factor/warmup steps for exponenital/noam decay.
- **WARMUP_EXP**: Warmup steps for noam decay.

2.3.14 Training options

- **MAX_EPOCH**: Stop when computed this number of epochs.
- **BATCH_SIZE**: Size of each minibatch.
- **HOMOGENEOUS_BATCHES**: If activated, use batches with similar output lengths, in order to better profit parallel computations.
- **JOINT_BATCHES**: When using homogeneous batches, size of the maxibatch.
- **PARALLEL_LOADERS**: Parallel CPU data batch loaders.
- **EPOCHS_FOR_SAVE**: Save model each this number of epochs.
- **WRITE_VALID_SAMPLES**: Write validation samples in file.
- **SAVE_EACH_EVALUATION**: Save the model each time we evaluate.

2.3.15 Early stop

- **EARLY_STOP** = Turns on/off the early stop regularizer.
- **PATIENCE**: We'll stop if we don't improve after this number of evaluations
- **STOP_METRIC**: Stopping metric.

2.3.16 Model main hyperparameters

- **MODEL_TYPE**: Model to train. See the model zoo for the supported architectures.
- **RNN_TYPE**: RNN unit type ('LSTM' and 'GRU' supported).
- **INIT_FUNCTION**: Initialization function for matrices (see keras/initializations).
- **INNER_INIT**: Initialization function for inner RNN matrices.
- **INIT_ATT**: Initialization function for attention mechism matrices.

Source word embeddings

- **SOURCE_TEXT_EMBEDDING_SIZE**: Source language word embedding size.
- **SRC_PRETRAINED_VECTORS**: Path to source pretrained vectors. See the utils folder for preprocessing scripts. Set to None if you don't want to use source pretrained vectors. When using pretrained word embeddings. this parameter must match with the source word embeddings size
- **SRC_PRETRAINED_VECTORS_TRAINABLE**: Finetune or not the target word embedding vectors.
- **SCALE_SOURCE_WORD_EMBEDDINGS**: Scale source word embeddings by Sqrt(SOURCE_TEXT_EMBEDDING_SIZE).

Target word embedding

- **TARGET_TEXT_EMBEDDING_SIZE**: Source language word embedding size.
- **TRG_PRETRAINED_VECTORS**: Path to target pretrained vectors. See the utils folder for preprocessing scripts. Set to None if you don't want to use source pretrained vectors. When using pretrained word embeddings. this parameter must match with the target word embeddings size
- **TRG_PRETRAINED_VECTORS_TRAINABLE**: Finetune or not the target word embedding vectors.
- **SCALE_TARGET_WORD_EMBEDDINGS**: Scale target word embeddings by Sqrt(TARGET_TEXT_EMBEDDING_SIZE).

Deepness

- **N_LAYERS_DECODER**: Stack this number of decoding layers.
- **DEEP_OUTPUT_LAYERS**: Additional Fully-Connected layers applied before softmax.

2.3.17 AttentionRNNEncoderDecoder model

- **ENCODER_RNN_TYPE**: Encoder's RNN unit type ('LSTM' and 'GRU' supported).
- **DECODER_RNN_TYPE**: Decoder's RNN unit type ('LSTM', 'GRU', 'ConditionalLSTM' and 'ConditionalGRU' supported).
- **ATTENTION_MODE**: Attention mode. 'add' (Bahdanau-style) or 'dot' (Luong-style).

Encoder configuration

- **ENCODER_HIDDEN_SIZE**: Encoder RNN size.
- **BIDIRECTIONAL_ENCODER**: Use a bidirectional encoder.
- **BIDIRECTIONAL_DEEP_ENCODER**: Use bidirectional encoder in all stacked encoding layers

Decoder configuration

- **DECODER_HIDDEN_SIZE**: Decoder RNN size.
- **ADDITIONAL_OUTPUT_MERGE_MODE**: Merge mode for the deep output layer.
- **SKIP_VECTORS_HIDDEN_SIZE**: Deep output layer size
- **INIT_LAYERS**: Initialize the first decoder state with these layers (from the encoder).
- **SKIP_VECTORS_SHARED_ACTIVATION**: Activation for the skip vectors.

2.3.18 Transformer model

- **MODEL_SIZE**: Transformer model size (dmodel in de paper).
- **MULTIHEAD_ATTENTION_ACTIVATION**: Activation the input projections in the Multi-Head Attention blocks.
- **FF_SIZE**: Size of the feed-forward layers of the Transformer model.
- **N_HEADS**: Number of parallel attention layers of the Transformer model.

2.3.19 Regularizers

Regularization functions

- **REGULARIZATION_FN**: Regularization function. 'L1', 'L2' and 'L1_L2' supported.
- **WEIGHT_DECAY**: L2 regularization in non-recurrent layers.
- **RECURRENT_WEIGHT_DECAY**: L2 regularization in recurrent layers
- **DOUBLE_STOCHASTIC_ATTENTION_REG**: Doubly stochastic attention (Eq. 14 from arXiv:1502.03044).

Dropout

- **DROPOUT_P**: Percentage of units to drop in non-recurrent layers (0 means no dropout).
- **RECURRENT_DROPOUT_P**: Percentage of units to drop in recurrent layers(0 means no dropout).
- **ATTENTION_DROPOUT_P**: Percentage of units to drop in attention layers (0 means no dropout).

Gaussian noise

- **USE_NOISE**: Apply gaussian noise during training.
- **NOISE_AMOUNT**: Amount of noise.

Batch normalization

- **USE_BATCH_NORMALIZATION**: Batch normalization regularization in non-recurrent layers and recurrent inputs. If True it is recommended to deactivate Dropout.
- **BATCH_NORMALIZATION_MODE**: Sample-wise or feature-wise BN mode.

Additional normalization layers

- **USE_PRELU**: Apply use PReLU activations as regularizer.
- **USE_L1**: L1 normalization on the features.
- **USE_L2**: Apply L2 function on the features.

2.3.20 Tensorboard

- **TENSORBOARD**: Switches On/Off the tensorboard callback.
- **LOG_DIR**: irectory to store teh model. Will be created inside STORE_PATH.
- **EMBEDDINGS_FREQ**: Frequency (in epochs) at which selected embedding layers will be saved.
- **EMBEDDINGS_LAYER_NAMES**: A list of names of layers to keep eye on. If None or empty list all the embedding layer will be watched.
- **EMBEDDINGS_METADATA**: Dictionary which maps layer name to a file name in which metadata for this embedding layer is saved.
- **LABEL_WORD_EMBEDDINGS_WITH_VOCAB**: Whether to use vocabularies as word embeddings labels (will overwrite EMBEDDINGS_METADATA).
- **WORD_EMBEDDINGS_LABELS**: Vocabularies for labeling. Must match EMBEDDINGS_LAYER_NAMES.

2.3.21 Storage and plotting

- **MODEL_NAME**: Name for the model.
- **EXTRA_NAME**: MODEL_NAME suffix
- **STORE_PATH**: Models and evaluation results will be stored here.
- **DATASET_STORE_PATH**: Dataset instance will be stored here.

- **SAMPLING_SAVE_MODE**: Save evaluation outputs in this format. Set to 'list' for a raw file.
- **VERBOSE**: Verbosity level.
- **RELOAD**: Reload a stored model. If 0 start training from scratch, otherwise use the model from this epoch/update.
- **REBUILD_DATASET**: Build dataset again or use a stored instance.
- **MODE**: 'training' or 'sampling' (if 'sampling' then RELOAD must be greater than 0 and EVAL_ON_SETS will be used). For 'sampling' mode, is recommended to use the sample_ensemble script.

2.4 Resources

2.4.1 Theoretical NMT

Before using an NMT, you should read and understand the theoretical basis of attentional NMT systems.

2.4.2 NMT-Keras Step-by-step

- NMT-Keras step-by-step guide (iPython and html versions): Tutorials for running this library. They are expected to be followed in order:
  1. Dataset setup: Shows how to invoke and configure a Dataset instance for a translation problem.
  2. Training tutorial: Shows how to call a translation model, link it with the dataset object and build callbacks for monitorizing the training.
  3. Decoding tutorial: Shows how to call a trained translation model and use it to translate new text.
  4. NMT model tutorial: Shows how to build a state-of-the-art NMT model with Keras in few (~50) lines.

2.4.3 NMT-Keras Output

This is a brief explanation about the typical output produced by the training pipeline of NMT-Keras. Assuming that we launched NMT-Keras for the example from tutorials, we'll have the following tree of folders (after 1 epoch):

```
├── trained_models
│   ├── EuTrans_GroundHogModel_src_emb_420_bidir_True_enc_600_dec_600_deepout_maxout_
↪trg_emb_420_Adam_0.001
│   │   ├── config.pkl
│   │   ├── epoch_1_Model_Wrapper.pkl
│   │   ├── epoch_1_structure_init.json
│   │   ├── epoch_1_structure.json
│   │   ├── epoch_1_structure_next.json
│   │   ├── epoch_1_weights.h5
│   │   ├── epoch_1_weights_init.h5
│   │   ├── epoch_1_weights_next.h5
│   │   ├── val.coco
│   │   ├── val_epoch_1.pred
```

Let's have a look to these files.

- *config.pkl*: Pickle containing the training parameters.
- *epoch_1_Model_Wrapper.pkl*: Pickle containing the Model_Wrapper object that we have trained.

- *epoch_1_structure.json*: Keras json specifying the layer and connections of the model.
- *epoch_1_structure_init.json*: Keras json specifying the layer and connections of the model_init (see tutorial 4 for more info about the model).
- *epoch_1_structure_next.json*: Keras json specifying the layer and connections of the model_next (see tutorial 4 for more info about the model).
- *epoch_1_weights.h5*: Model parameters (weight matrices).
- *epoch_1_weights_init.h5*: Model init parameters (weight matrices).
- *epoch_1_weights_next.h5*: Model next parameters (weight matrices).
- *val.coco*: Metrics dump. This file is name as [tested_split].[metrics_name]. It contains a header with the metrics name and the value of all evaluations (epoch/updates). For instance:

```
epoch,Bleu_1, Bleu_2, Bleu_3, Bleu_4, CIDEr, METEOR, ROUGE_L,
1,0.906982874122, 0.875873151361, 0.850383597611, 0.824070996966, 8.084477458, 0.
↪550547408997, 0.931523374569,
2,0.932937494321, 0.90923787501, 0.889965151506, 0.871819102335, 8.53565391657, 0.
↪586377788443, 0.947634196936,
3,0.965579088172, 0.947927460597, 0.934090548706, 0.920166838768, 9.0864109399, 0.
↪63234570058, 0.971618921459,
```

- *val_epoch_1.pred*: Raw file with the output of the NMT system at the evaluation performed at the end of epoch 1.

We can modify the save and evaluation frequencies from the config file.

2.4.4 Tensorboard integration

TensorBoard is a visualization tool provided with TensorFlow.

It can be accessed by NMT-Keras and provide visualization of the learning process, dynamic graphs of our training and metrics, as well representation of different layers (such as word embeddings). Of course, this tool is only available with the Tensorflow backend.

In this document, we'll set some parameters and explore some of the options that Tensorboard provides. We'll:

- Configure Tensorboard and NMT-Keras.
- Visualize the learning process (loss curves).
- Visualize the computation graphs built by NMT Keras.
- Visualize the words embeddings obtained during the training stage.

In the [configuration file] we have available the following tensorboard-related options:

```
TENSORBOARD = True                          # Switches On/Off the tensorboard callback
LOG_DIR = 'tensorboard_logs'                # Directory to store teh model. Will be␣
↪created inside STORE_PATH
EMBEDDINGS_FREQ = 1                         # Frequency (in epochs) at which selected␣
↪embedding layers will be saved.
EMBEDDINGS_LAYER_NAMES = [                  # A list of names of layers to keep eye on.␣
↪If None or empty list all the embedding layer will be watched.
'source_word_embedding',
'target_word_embedding']
EMBEDDINGS_METADATA = None                  # Dictionary which maps layer name to a file␣
↪name in which metadata for this embedding layer is saved.
```

(continues on next page)

(continued from previous page)

```
LABEL_WORD_EMBEDDINGS_WITH_VOCAB = True  # Whether to use vocabularies as word␣
↪embeddings labels (will overwrite EMBEDDINGS_METADATA)
WORD_EMBEDDINGS_LABELS = [               # Vocabularies for labeling. Must match␣
↪EMBEDDINGS_LAYER_NAMES
                         'source_text',
                         'target_text']
```

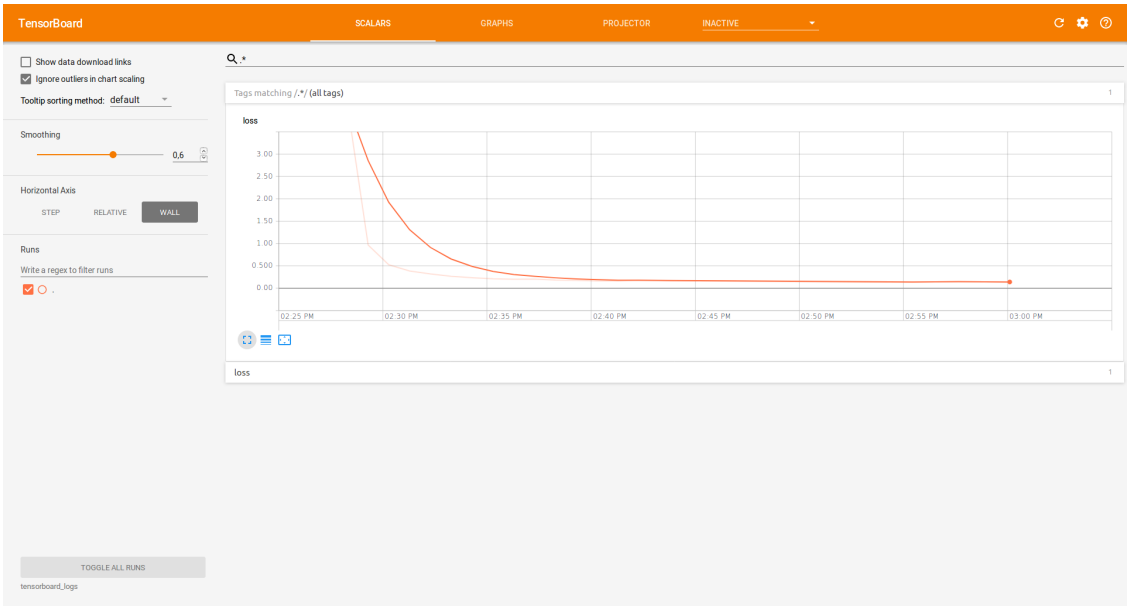
With these options, we are telling Tensorboard where to store the data we want to visualize: loss curve, computation graph and word embeddings. Moreover, we are specifying the word embedding layers that we want to visualize. By setting the *WORD_EMBEDDINGS_LABELS* to the corresponding *Dataset* ids, we can print labels in the word embedding visualization.

Now, we run a regular training: *python main.py*. If we *cd* to the model directory, we'll see a directiory named *tensorboard_logs*. Now, we launch Tensorboard on this directory:

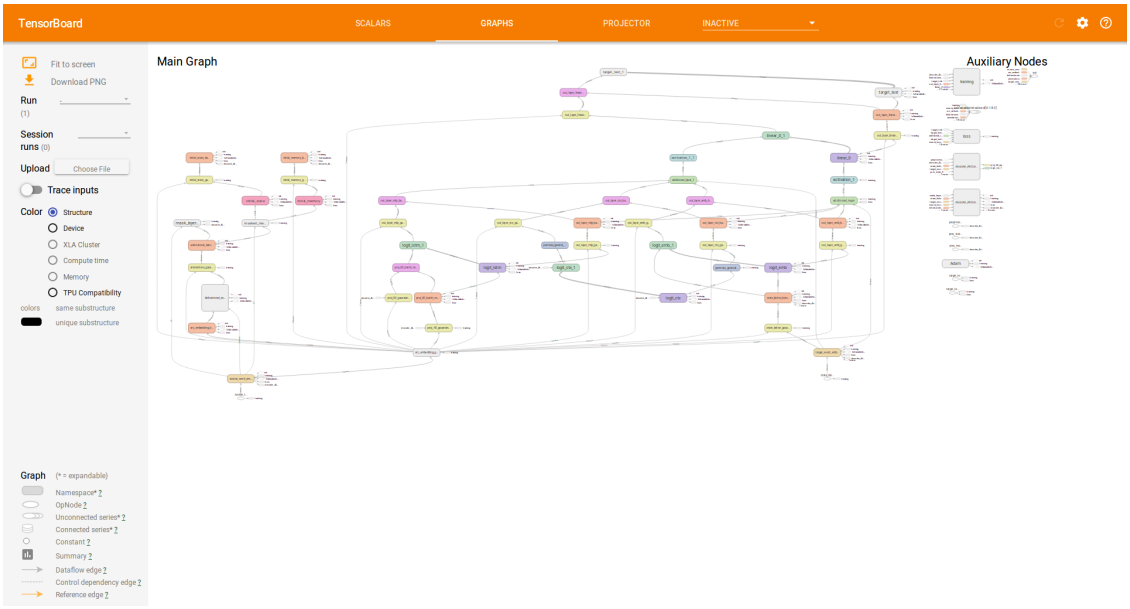
```
$ tensorboard --logdir=tensorboard_logs
TensorBoard 0.1.5 at http://localhost:6006 (Press CTRL+C to quit)
```

We can open Tensorboard in our browser (http://localhost:6006) with the NMT-Keras information:

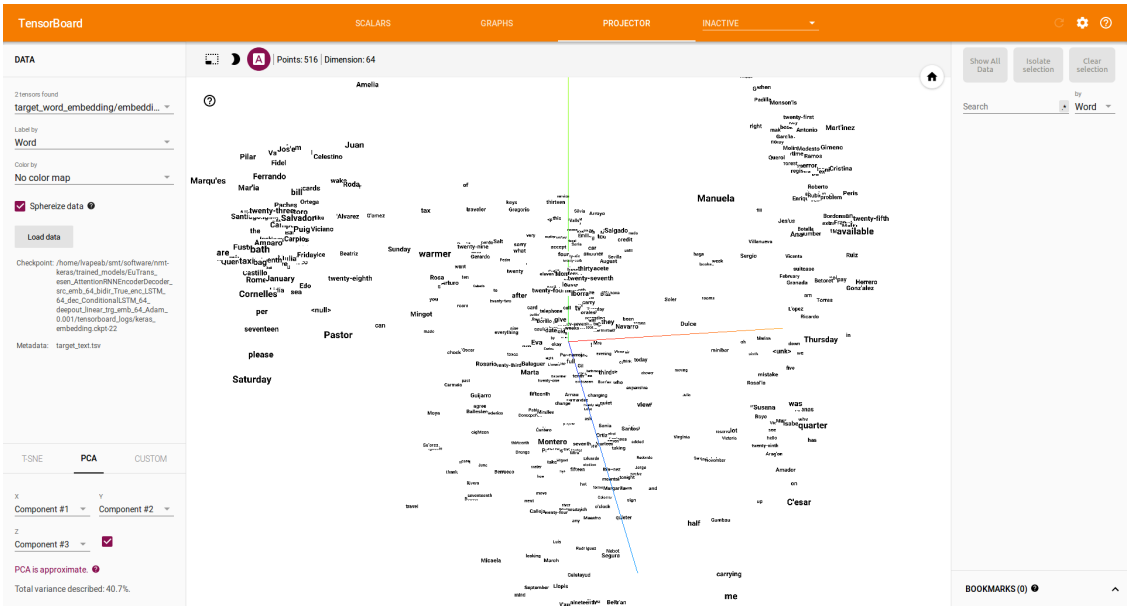
Loss curve



Model graphs



Embedding visualization



2.5 Tutorials

There are 2 tutorials in colab notebooks:

- An introduction to a complete NMT experiment.
- A dissected NMT model.

Follow the links to open them in Colab.

2.6 Modules

2.6.1 nmt_keras package

Submodules

model_zoo

class nmt_keras.model_zoo.**TranslationModel**(*params*, *model_type='Translation_Model'*, *verbose=1*, *structure_path=None*, *weights_path=None*, *model_name=None*, *vocabularies=None*, *store_path=None*, *set_optimizer=True*, *clear_dirs=True*)

Bases: keras_wrapper.cnn_model.Model_Wrapper

Translation model class. Instance of the Model_Wrapper class (see staged_keras_wrapper).

Parameters

- **params** (*dict*) – all hyperparameters of the model.
- **model_type** (*str*) – network name type (corresponds to any method defined in the section 'MODELS' of this class). Only valid if 'structure_path' == None.
- **verbose** (*int*) – set to 0 if you don't want the model to output informative messages
- **structure_path** (*str*) – path to a Keras' model json file. If we speficy this parameter then 'type' will be only an informative parameter.
- **weights_path** (*str*) – path to the pre-trained weights file (if None, then it will be initialized according to params)
- **model_name** (*str*) – optional name given to the network (if None, then it will be assigned to current time as its name)
- **vocabularies** (*dict*) – vocabularies used for word embedding
- **store_path** (*str*) – path to the folder where the temporal model packups will be stored
- **set_optimizer** (*bool*) – Compile optimizer or not.
- **clear_dirs** (*bool*) – Clean model directories or not.

AttentionRNNEncoderDecoder(*params*)

Neural machine translation with:

- BRNN encoder
- Attention mechansim on input sequence of annotations
- Conditional RNN for decoding
- Deep output layers:
- Context projected to output
- Last word projected to output
- Possibly deep encoder/decoder

See:

- Neural Machine Translation by Jointly Learning to Align and Translate.

- **'Nematus: a Toolkit for Neural Machine Translation'_**.

Parameters **params** (*int*) – Dictionary of hyper-params (see config.py)

Returns None

GroundHogModel (*params*)

Neural machine translation with:

- BRNN encoder
- Attention mechansim on input sequence of annotations
- Conditional RNN for decoding
- Deep output layers:
- Context projected to output
- Last word projected to output
- Possibly deep encoder/decoder

See:

- Neural Machine Translation by Jointly Learning to Align and Translate.
- **'Nematus: a Toolkit for Neural Machine Translation'_**.

Parameters **params** (*int*) – Dictionary of hyper-params (see config.py)

Returns None

Transformer (*params*)

Neural machine translation consisting in stacking blocks of:

- Multi-head attention.
- Dropout.
- Residual connection.
- Normalization.
- Position-wise feed-forward networks.

Positional information is injected to the model via embeddings with positional encoding.

See:

- Attention Is All You Need.

Parameters **params** (*int*) – Dictionary of params (see config.py)

Returns None

setOptimizer (***kwargs*)
Sets and compiles a new optimizer for the Translation_Model. The configuration is read from Translation_Model.params. :return: None

setParams (*params*)
Set self.params as params. :param params: :return:

nmt_keras.model_zoo.**getPositionalEncodingWeights**(*input_dim*, *output_dim*, *name=''*, *verbose=True*)

Obtains fixed sinusoidal embeddings for obtaining the positional encoding.

Parameters

- **input_dim** (*int*) – Input dimension of the embeddings (i.e. vocabulary size).
- **output_dim** (*int*) – Embeddings dimension.
- **name** (*str*) – Name of the layer
- **verbose** (*int*) – Be verbose

Returns A list with sinusoidal embeddings.

training

nmt_keras.training.**train_model**(*params*, *load_dataset=None*)

Training function.

Sets the training parameters from params.

Build or loads the model and launches the training.

Parameters

- **params** (*dict*) – Dictionary of network hyperparameters.
- **load_dataset** (*str*) – Load dataset from file or build it from the parameters.

Returns None

apply_model

nmt_keras.apply_model.**sample_ensemble**(*args*, *params*)

Use several translation models for obtaining predictions from a source text file.

Parameters

- **args** (*argparse.Namespace*) – Arguments given to the method:
  - dataset: Dataset instance with data.
  - text: Text file with source sentences.
  - splits: Splits to sample. Should be already included in the dataset object.
  - dest: Output file to save scores.
  - weights: Weight given to each model in the ensemble. You should provide the same number of weights than models. By default, it applies the same weight to each model (1/N).
  - n_best: Write n-best list (n = beam size).
  - config: Config .pkl for loading the model configuration. If not specified, hyperparameters are read from config.py.
  - models: Path to the models.
  - verbose: Be verbose or not.
- **params** – parameters of the translation model.

nmt_keras.apply_model.**score_corpus**(*args*, *params*)
Use one or several translation models for scoring source–target pairs-

Parameters

- **args** (*argparse.Namespace*) – Arguments given to the method:
  - dataset: Dataset instance with data.
  - source: Text file with source sentences.
  - target: Text file with target sentences.
  - splits: Splits to sample. Should be already included in the dataset object.
  - dest: Output file to save scores.
  - weights: Weight given to each model in the ensemble. You should provide the same number of weights than models. By default, it applies the same weight to each model (1/N).
  - verbose: Be verbose or not.
  - config: Config .pkl for loading the model configuration. If not specified, hyperparameters are read from config.py.
  - models: Path to the models.
- **params** (*dict*) – parameters of the translation model.

build_callbacks

nmt_keras.build_callbacks.**buildCallbacks**(*params*, *model*, *dataset*)

Builds the selected set of callbacks run during the training of the model:

- EvalPerformance: Evaluates the model in the validation set given a number of epochs/updates.
- SampleEachNUpdates: Shows several translation samples during training.

Parameters

- **params** (*dict*) – Dictionary of network hyperparameters.
- **model** (*Model_Wrapper*) – Model instance on which to apply the callback.
- **dataset** (*Dataset*) – Dataset instance on which to apply the callback.

Returns list of callbacks to pass to the Keras' training.

Module contents

2.6.2 data_engine package

Submodules

prepare_data module

data_engine.prepare_data.**build_dataset**(*params*)
Builds (or loads) a Dataset instance. :param params: Parameters specifying Dataset options :return: Dataset object

data_engine.prepare_data.**keep_n_captions**(*ds*, *repeat*, *n=1*, *set_names=None*)
Keeps only n captions per image and stores the rest in dictionaries for a later evaluation :param ds: Dataset object :param repeat: Number of input samples per output :param n: Number of outputs to keep. :param set_names: Set name. :return:

data_engine.prepare_data.**prepare_references**(*ds*, *repeat*, *n=1*, *set_names=None*)
Keeps only n captions per image and stores the rest in dictionaries for a later evaluation :param ds: Dataset object :param repeat: Number of input samples per output :param n: Number of outputs to keep. :param set_names: Set name. :return:

data_engine.prepare_data.**update_dataset_from_file**(*ds*, *input_text_filename*, *params*, *splits=None*, *output_text_filename=None*, *remove_outputs=False*, *compute_state_below=False*, *recompute_references=False*)
Updates the dataset instance from a text file according to the given params. Used for sampling

Parameters

- **ds** – Dataset instance
- **input_text_filename** – Source language sentences
- **params** – Parameters for building the dataset
- **splits** – Splits to sample
- **output_text_filename** – Target language sentences
- **remove_outputs** – Remove outputs from dataset (if True, will ignore the output_text_filename parameter)
- **compute_state_below** – Compute state below input (shifted target text for professor teaching)
- **recompute_references** – Whether we should rebuild the references of the dataset or not.

Returns Dataset object with the processed data

Module contents

2.6.3 utils package

Submodules

evaluate_from_file module

preprocess_binary_word_vectors module

utils.preprocess_binary_word_vectors.**parse_args**()

utils.preprocess_binary_word_vectors.**word2vec2npy**(*v_path*, *base_path_save*, *dest_filename*)
Preprocess pretrained binary vectors and stores them in a suitable format. :param v_path: Path to the binary vectors file. :param base_path_save: Path where the formatted vectors will be stored. :param dest_filename: Filename of the formatted vectors.

preprocess_text_word_vectors module

utils.preprocess_text_word_vectors.**parse_args**()

utils.preprocess_text_word_vectors.**txtvec2npy**(*v_path*, *base_path_save*, *dest_filename*)
Preprocess pretrained text vectors and stores them in a suitable format :param v_path: Path to the text vectors file. :param base_path_save: Path where the formatted vectors will be stored. :param dest_filename: Filename of the formatted vectors.

utils module

utils.utils.**update_parameters**(*params*, *updates*, *restrict=False*)
Updates the parameters from params with the ones specified in updates :param params: Parameters dictionary to update :param updates: Updater dictionary :param restrict: If True, parameters from the original dict are not overwritten. :return:

Module contents

2.7 Contact

If you have any trouble using NMT-Keras, please post a GitHub issue or drop an email to: lvapeab@prhlt.upv.es

2.7.1 Acknowledgement

Much of this library has been developed together with Marc Bolaños for other multimodal projects.

Related projects

To see other projects following the philosophy of NMT-Keras, take a look to:

- TMA: for egocentric captioning based on temporally-linked sequences.
- VIBIKNet: for visual question answering.
- ABiViRNet: for video description.
- Sentence SelectioNN for sentence classification and selection.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

n

u

Index