Interactive Keras captioning Documentation

Release 0.2

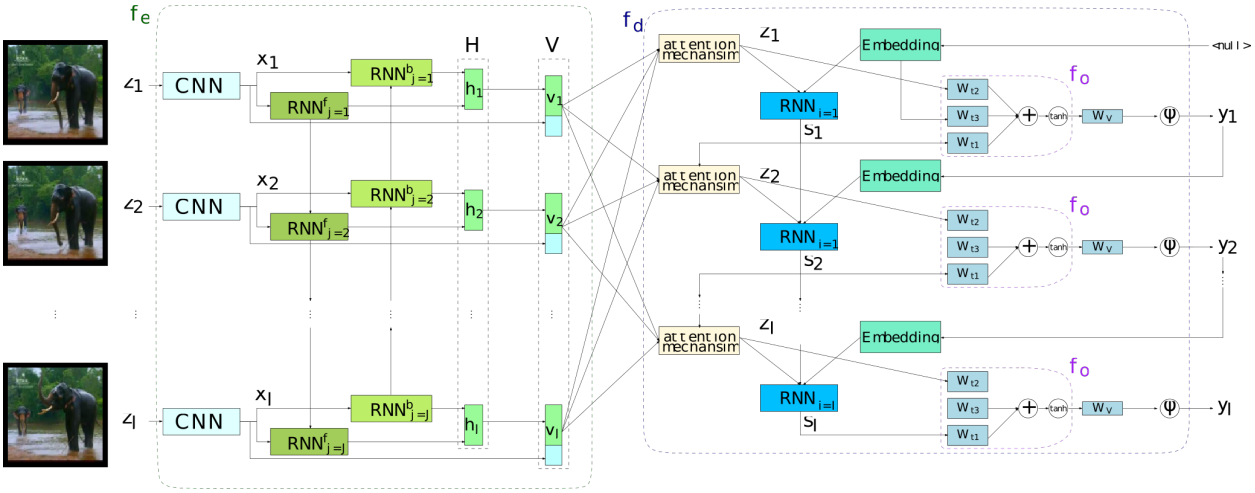
Álvaro Peris

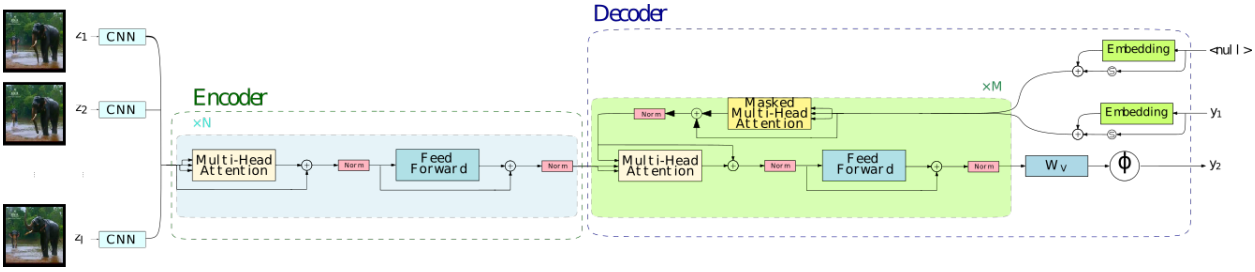
Aug 02, 2019

Contents

Interactive multimedia captioning with Keras (Theano and Tensorflow). Given an input image or video, we describe its content.

Checkout the live demo!





CHAPTER 1

Interactive captioning

Interactive-predictive pattern recognition is a collaborative human-machine framework for obtaining high-quality predictions while minimizing the human effort spent during the process.

It consists in an iterative prediction-correction process: each time the user introduces a correction to a hypothesis, the system reacts offering an alternative, considering the user feedback.

For further reading about this framework, please refer to Interactive Neural Machine Translation, Online Learning for Effort Reduction in Interactive Neural Machine Translation and Interactive-predictive neural multimodal systems.

1.1 Features

- Attention-based RNN and Transformer models.
- Support for GRU/LSTM networks: - Regular GRU/LSTM units. - Conditional GRU/LSTM units in the decoder. - Multilayered residual GRU/LSTM networks.
- 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.
- Caption scoring.
- N-best list generation (as byproduct of the beam search process).
- 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.

1.2 Guide

1.2.1 Installation

Assuming that you have pip installed, run:

```
git clone https://github.com/lvapeab/interactive-keras-captioning
cd interactive-keras-captioning
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.

Requirements

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

1.2.2 Usage

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
```

Decoding

Once we have our model trained, we can translate new text using the caption.py script. If we want to use the models from the first three epochs to translate the *test* split from our dataset, just run:

```
python caption.py --models trained_models/tutorial_model/epoch_1 \
                                trained_models/tutorial_model/epoch_2 \
                          --dataset datasets/Dataset_tutorial_dataset.pkl \
                          --split test
```

1.2.3 Configuration options

This document describes the available hyperparameters used for training Interactive Keras Captioning.

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

Warning! This document may be out-dated. Please, check config.py for new hyperparameters.

Naming and experiment setup

- **TASK_NAME**: Task name. Used for naming and for indexing files.
- **DATASET_NAME**: Dataset name. Used for naming.
- **DATA_ROOT_PATH**: Path to the data
- **TRG_LAN**: Language of the target text. Used for naming and for coputing language-dependent metrics (e.g. Meteor)
- **INPUT_DATA_TYPE**: Type of the input data. Supported: 'image-features', 'raw-image', 'video-features', 'raw-video'. raw-* somewhat untested.

Options for input data type '*-features'

- **FEATURE_NAMES**: Feature names.
- **FEATURE_DIMENSION**: Dimension of the features. List of shapes or integer specifying the last dimension.
- **NUM_FRAMES**: Number of frames per sample (must be set to -1 in the case of image-features).
- **FRAMES_LIST_FILES**: Path to the image and feature files (the chars {} will be replaced by each type of features)
- **FRAMES_COUNTS_FILES**: Frame counts for each sample (leave blank for image-features).

Options for input data type 'raw-image'

- **IMG_SIZE**: Size of the input images (will be resized to the desired size).
- **IMG_CROP_SIZE**: Size of the image crops inputted to the model
- **FEATURE_NAMES**: Name of the features computed by the model.
- **NORMALIZE**: Apply image normalization.
- **NORMALIZATION_TYPE**: Type of normalization.
- **DATA_AUGMENTATION**: Apply data augmentation on input data .
- **FRAMES_LIST_FILES**: Path to the image and feature files (the chars {} will be replaced by each type of features)

Output data

- **DESCRIPTION_FILES**: Path to the files containing the captions
- **LABELS_PER_SAMPLE**: Number of references per sample. Set to 0 for using a variable number of references per sample.
- **DESCRIPTION_COUNTS_FILES**: If LABELS_PER_SAMPLE == 0, counters of the number of labels for each sample.

Input/output mappings

- **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.

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.

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).

Search normalization

- **SEARCH_PRUNING**: Apply pruning strategies to the beam search method. It will likely increase decoding speed, but decrease quality.
- **LENGTH_PENALTY**: Apply length penalty (Wu et al. (2016)).
- **LENGTH_NORM_FACTOR**: Length 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)).

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)

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.

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.

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.

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.

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.

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.

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.

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.
- **TRAINABLE_ENCODER**: Whether the encoder's weights should be modified during training.
- **TRAINABLE_DECODER**: Whether the decoder's weights should be modified during training.

Embedding options

- **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).
- **SCALE_SOURCE_WORD_EMBEDDINGS**: Scale source word embeddings by Sqrt(SOURCE_TEXT_EMBEDDING_SIZE).

Deepness

- **N_LAYERS_ENCODER**: Stack this number of encoding layers.
- **N_LAYERS_DECODER**: Stack this number of decoding layers.
- **IMG_EMBEDDING_LAYERS**: Fully-conected layers for visual embedding.
- **DEEP_OUTPUT_LAYERS**: Additional Fully-Connected layers applied before softmax.

AttentionRNNEncoderDecoder model

- **ENCODER_RNN_TYPE**: Encoder's RNN unit type ('LSTM' and 'GRU' supported).
- **USE_CUDNN**: Use CuDNN's implementation of GRU and LSTM (only for Tensorflow backend).
- **DECODER_RNN_TYPE**: Decoder's RNN unit type ('LSTM', 'GRU', 'ConditionalLSTM' and 'Conditional-GRU' supported).
- **ATTENTION_MODE**: Attention mode. 'add' (Bahdanau-style) or 'dot' (Luong-style).

Encoder configuration

- **RNN_ENCODER_HIDDEN_SIZE**: Encoder RNN size.
- **BIDIRECTIONAL_ENCODER**: Use a bidirectional encoder.

- **BIDIRECTIONAL_DEEP_ENCODER**: Use bidirectional encoder in all stacked encoding layers
- **BIDIRECTIONAL_MERGE_MODE**: Merge function for bidirectional layers.

Decoder configuration

- **RNN_DECODER_HIDDEN_SIZE**: Decoder RNN size.
- **ATTENTION_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.

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.

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).
- **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.

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.

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.

1.2.4 Modules

captioner package

Submodules

model_zoo

class captioner.model_zoo.**Captioning_Model**(*params*, *model_type='Captioning_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*) – split 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*)

Video captioning with:

- Attention mechansim on video frames
- Conditional LSTM for processing the video
- **Feed forward layers:**
  - Context projected to output
  - Last word projected to output

Parameters **params** – Dictionary of hyperparmeters.

Returns

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** (*dict*) – 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:

captioner.model_zoo.**VideoDesc_Model**
alias of *captioner.model_zoo.Captioning_Model*

captioner.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

captioner.training.**train_model**(*params*)
Training function. Sets the training parameters from params. Build or loads the model and launches the training. :param params: Dictionary of network hyperparameters. :return: None

apply_model

captioner.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.

build_callbacks

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

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

- PrintPerformanceMetricOnEpochEndOrEachNUpdates: Evaluates the model in the validation split 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

data_engine package

Submodules

prepare_data module

data_engine.prepare_data.**build_dataset**(*params*)

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.**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** – New inputs.
- **params** – Parameters for building the dataset
- **splits** – Splits to sample
- **output_text_filename** – New output 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

utils package

Submodules

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

1.2.5 Contact

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

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:

- NMT Keras: Neural Machine Translation.
- 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.

1.2.6 Indices and tables

- genindex
- modindex
- search

Python Module Index

c

d

u

Index