PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 107 APRIL 2017 5–17

Neural Monkey: An Open-source Tool for Sequence Learning

Jindřich Helcl,[a b] Jindřich Libovický[a]

[a] Charles University, Faculty of Mathematics and Physics, Institute of Formal and Applied Linguistics
[b] German Research Center for Artificial Intelligence (DFKI), Language Technology Lab

Abstract

In this paper, we announce the development of Neural Monkey – an open-source neural machine translation (NMT) and general sequence-to-sequence learning system built over the TensorFlow machine learning library. The system provides a high-level API tailored for fast prototyping of complex architectures with multiple sequence encoders and decoders. Models' overall architecture is specified in easy-to-read configuration files. The long-term goal of the Neural Monkey project is to create and maintain a growing collection of implementations of recently proposed components or methods, and therefore it is designed to be easily extensible. Trained models can be deployed either for batch data processing or as a web service. In the presented paper, we describe the design of the system and introduce the reader to running experiments using Neural Monkey.

1. Introduction

Neural machine translation (NMT) recently became a new successful paradigm in machine translation (Sutskever et al., 2014; Bahdanau et al., 2014). Besides NMT, sequence-to-sequence (S2S) learning in general proved its usefulness in various other tasks including building a chatbot (Vinyals and Le, 2015), image captioning (Vinyals et al., 2015; Xu et al., 2015), or text segmentation and entity recognition (Gillick et al., 2016).

Neural Monkey is an open-source toolkit written using the TensorFlow machine learning library (Abadi et al., 2016). It provides a higher level API, such that it should be enough for the users to be familiar with the models on the equation level, without delving into implementation details. For that reason, we try to use as big abstract

 Corresponding author: helcl@ufal.mff.cuni.cz
Cite as: Jindřich Helcl, Jindřich Libovický. Neural Monkey: An Open-source Tool for Sequence Learning. The Prague Bulletin of Mathematical Linguistics No. 107, 2017, pp. 5–17. doi: 10.1515/pralin-2017-0001.

building blocks as possible. Unlike *tfLearn*[1] or *Lasagne*,[2] our building blocks are not individual network layers, but more abstract objects like encoders or classifiers. These objects are parametrized so that their properties (e.g. number and sizes of hidden layers or dropout probability) can be set from a farther perspective. This design decision allows us to place the configuration of the experiments away from the actual code into a separate comprehensive configuration file. This way, users are prevented from interleaving the configuration with other program logic.

Neural Monkey is still under development and has an ambition to become an ever-growing collection of recent innovations in NMT and S2S learning in general, which would allow its users to easily try out the model for their specific tasks and datasets. With its simple experiment management, we hope that it will be used as a ready-made easily-extensible toolkit for experiments with machine translation, image captioning, text classification tasks or scene text recognition. It has already been used for the submission for the WMT16 automatic post-editing and multimodal translation tasks (Libovický et al., 2016), linguistic analysis of MT systems (Avramidis et al., 2016).

The current version of the software is available at `https://github.com/ufal/neuralmonkey` under the BSD license.

2. Related Work

Most of the deep learning frameworks split the computation into two stages. In the first stage, the programmer designs the computation symbolically. The symbolic code which describes a computation graph is then optimized and compiled to efficiently perform the numeric computation. In the second stage, the program performs the numerical computation on the compiled computation graph using a supplied input data.

In Theano (Bergstra et al., 2010), C code is generated from the original Python code and then it is compiled. One of the problems with this approach is that it is difficult to track the computation once the code is compiled.

Chainer (Tokui et al., 2015), on the other hand, is written in Python, and therefore, debugging of the code is easier. In order to match the speed of compiled C code, Chainer uses fast libraries for numerical computation both on CPUs and GPUs.

TensorFlow (Abadi et al., 2016) stays in between of these approaches. Its building blocks are implemented in C and are manipulated using Python code. Additionally, TensorFlow provides tools for greater amount of transparency and offers more convenient ways of debugging the computation graphs.

Torch (Collobert et al., 2011) also uses a similar approach, employing a scripting language that operates over a highly optimized C backend. Unlike the other frameworks which use Python, Torch uses Lua for the scripts.

[1] `https://github.com/tflearn/tflearn`

[2] `https://github.com/Lasagne/Lasagne`

These libraries are basically general purpose tools for linear algebra computations (although they bring more and more functionality tailored for deep learning) and writing the models requires typing a lot of service code that deals with preparing the inputs and outputs of the models. All of these three basic libraries are distributed under free and commercial-friendly licenses.

For the reasons above, various libraries provide higher levels of abstractions over the basic libraries. Most of them aim for general-purpose use and do not provide many features tailored for natural language processing. Among these, there are the previously mentioned *tfLearn* and *Lasagne*, built on top of TensorFlow and Theano, respectively. Another popular library for Theano, similar to *Lasagne*, is *Blocks* (van Merriënboer et al., 2015). All of these libraries offer a lower level abstraction, operating directly with layers. *Keras* (Chollet, 2015) operates with the layers similarly as the other frameworks. Currently, it provides probably the richest set features both over Theano and TensorFlow.

A software package similar to Neural Monkey is *Nematus* (Sennrich et al., 2017). It is a state-of-the-art research software for NMT. Unlike Neural Monkey, it is written using Theano and focuses only on single-input and single-output setup and does not by default support input of non-textual modalities.

Another software package for NMT is *OpenNMT* (Klein et al., 2017), which is built on Torch. Similarly to Nematus, it focuses on single-source and single-output models. Besides that, there is an extension for image-to-text models. Compared to Nematus, it does not support so many features for NMT such as sub-word units or direct optimization towards BLEU score (Shen et al., 2016).

Unlike Neural Monkey, none of the S2S learning packages above provides basic experiment management functionality.

3. Problem Conceptualization

This section provides the basic overview on how Neural Monkey conceptualizes the problem of S2S learning and how the data flows during training and how running the models looks like.

3.1. Loading and Processing Datasets

Before the models are applied to the data, there is a short pipeline of steps preparing the data. For that, we introduce the notion of dataset and data series.

A *dataset* is a collection of named data *series*. Each data series is a list of data instances of the same type that represents a single kind of input or desired output of a model. In the simple case of machine translation, there are two data series: a list of sentences in the source language and a list of sentences in the target language. Figure 1 captures how a dataset is created from the input data.

The dataset is created in the following steps:

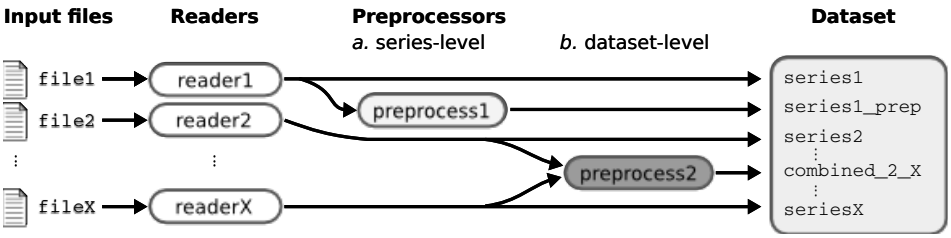

Figure 1. Creating a dataset in Neural Monkey

1. An input file is read using a reader. Reader can e.g., load a file containing paths to JPEG images and load them as NumPy arrays, or read tokenized text as a list of lists (sentences) of string tokens.
2. Series created by the readers can be preprocessed by some series-level preprocessors. An example of such preprocessing is byte-pair encoding (Sennrich et al., 2016) which loads a list of merges and segments the text accordingly.
3. The final step before creating a dataset is applying dataset-level preprocessors which can take more series and output a new series.

Currently, there are two implementations of a dataset. An in-memory dataset which stores all data in the memory, and a lazy dataset which gradually reads the input files step by step and only stores the batches necessary for the computation in the memory.

3.2. Training and Running a Model

This section describes the training and running workflow. In general, we try to separate model parts which provide a declarative description of the model (e.g., equations for an RNN decoder) and the way the models are run (e.g., run it using a beam search, sample a random sentence, or get the error derivatives). The main concepts and their interconnection can be seen in the scheme in Figure 2. It shows how the model is executed on the data using what we call *runners*.

Dataset series can be used to create a *vocabulary*. A vocabulary represents an indexed set of tokens and provides functionality for converting lists of tokenized sentences into matrices of token indices and vice versa. Vocabularies are used by encoders and decoders for feeding the provided series into the neural network.

The model itself is defined by model parts which categorize into encoders and decoders. This is where most of the TensorFlow code is located. Encoders are parts of the model which take some input and compute a representation of it. Decoders are model parts that produce some outputs. Our definition of encoders and decoders is more general than in the classical S2S learning. An encoder can be for example a convolutional network processing an image. The RNN decoder is for us only a special

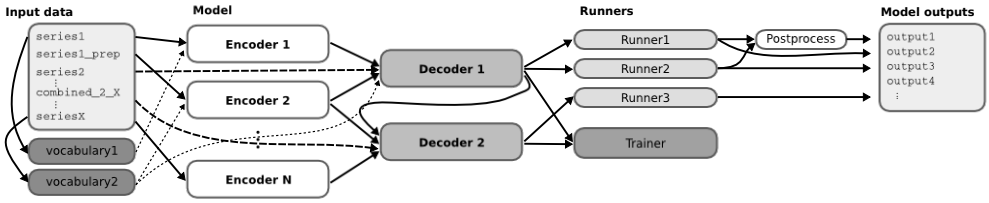

Figure 2. Model workflow in Neural Monkey

type of decoder, it can be also a sequence labeler or a simple multilayer perceptron classifier or regressor.

Decoders are executed using so-called *runners*. Different runners represent different ways of running the model. We might want to get a single best estimation, get an n-best list or a sample from the model. We might want to use an RNN decoder to get the decoded sequences or we might be interested in the word alignment obtained by its attention model. This is all done by employing different runners over the decoders. The outputs of the runners can be subject to further postprocessing.

In addition to runners, each experiment has to have its *trainer*. A trainer is a special case of a runner that actually modifies the parameters of the model. It collects the objective functions and uses them in an optimizer.

Neural Monkey manages TensorFlow sessions using an object called TensorFlow manager. Its basic capability is to execute runners on provided datasets. It encapsulates all logic concerning the TensorFlow sessions.

We can demonstrate this rather abstract description on an example of automatic post-editing of MT output. The model is illustrated in Figure 3. In this case, the model uses two encoders which are implemented using a bi-directional GRU (Cho et al., 2014) network. The first encoder is fed with sentences in the source language and the second one is fed with the machine translation output. A decoder combines outputs of both the encoders and attends to their hidden states during decoding. A trainer that follows the decoders computes the error of the decoder outputs given the desired target text from the train dataset.

4. Configuration

The experiments are described using configuration files. They contain a complete specification of the network architecture, preprocessing and postprocessing of the data, the training and validation data, all meta-parameters of the training, and metrics used for model evaluation. The configuration files are the main tool of Neural Monkey's experiment management. The same configuration can be used after training to run the trained model.

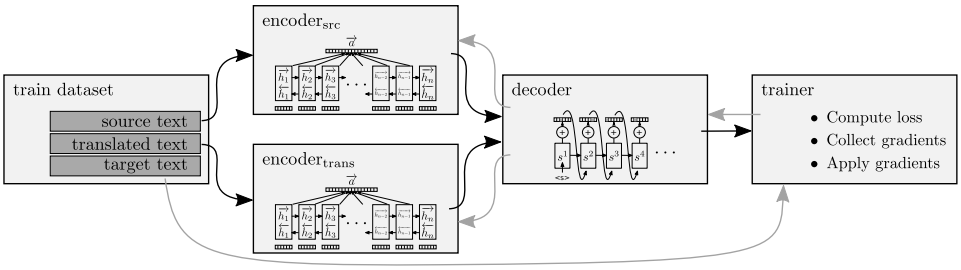

Figure 3. Scheme of the architecture for the training process of an automatic post-editing task

The following sections explain the main ideas how the configuration works.

4.1. Syntax

The configuration files are based on the syntax of INI files.[3] Neural Monkey INI files contain key-value pairs, delimited by an equal sign (=) with no spaces around. The key-value pairs are grouped into sections. (Neural Monkey requires all pairs to belong to a section.)

Every section starts with its header which consists of the section name in square brackets. Everything below the header is considered to be a part of the section. Comments can appear on their own line, prefixed either with a hash sign (#) or a semicolon (;) and possibly indented. The configuration introduces several additional constructs for the values. These can be atomic values, and compound values.

The supported atomic values are:

- booleans: literals `True` and `False`;
- integers: strings that could be interpreted as integers by Python (e.g., `1`, `002`);
- floats: strings that could be interpreted as floats by Python (e.g., `1.0`, `.123`, `2.`, `2.34e-12`);
- strings: string literals in quotes (e.g., `"walrus"`, `"5"`);
- section references: string literals in angle brackets (e.g., `<encoder>`), sections are later interpreted as Python objects;
- Python names: strings without quotes which are neither booleans, integers and floats, nor section references (e.g., `neuralmonkey.encoders.SentenceEncoder`).

On top of that, there are two compound types syntax from Python:

- lists: comma-separated in squared brackets (e.g., `[1, 2, 3]`);

[3] `https://en.wikipedia.org/wiki/INI_file`

- tuples: comma-separated in round brackets (e.g., `("target", <ter>)`).

4.2. Interpretation

Each configuration file contains a `[main]` section which is interpreted as a dictionary having keys specified in the section and values which are results of interpretation of the right-hand sides.

Both the atomic and compound types taken from Python (i.e., everything except the section references) are interpreted as their Python counterparts.

Section references are interpreted as references to objects constructed while interpreting the referenced section. (So, if `<session_manager>` is on the right-hand side of a variable assignment and section `[session_manager]` is located elsewhere in the file, Neural Monkey will construct a Python object based on the key-value pairs in section `[session_manager]` and use it as a value for the variable.)

Each section, except for the `[main]` section, needs to contain the key `class` with a value of Python name which is a callable (e.g., a class constructor or a function). The other keys are used as named arguments of the callable.

4.3. Content of the Configuration

The configuration needs to specify what is needed during the model runtime (the model architecture, preprocessing and postprocessing of the data) and the configuration for training (loss computation, mini-batch sizes etc.). Since the configuration refers directly to the Python code, there is no need for separate user and programmer documentation. The documentation of the configuration elements can be found in the API documentation of the respective modules of the package. A few self-explanatory examples of the configuration can be seen in Figures 4, 5, and 6.

5. Usage

Neural Monkey is written in Python 3.5. Because of the abstraction over computing devices that TensorFlow offers, the models can be run both on CPU and GPU. Neural Monkey has only a few requirements (besides TensorFlow), all of them can be easily installed with pip.

There are four main scripts that are used to run Neural Monkey. They are located in the `bin` directory of the source repository:

- `neuralmonkey-train` – The main script used for training a model. It takes one argument, which is the location of the configuration file for the training. Each experiment has its own directory which contains a copy of the configuration file, the current git diff and commit ID, all experiment logs, TensorBoard visualization files, and the saved model.

```
[encoder]
class=(...).SentenceEncoder
name="encoder-1"
rnn_size=256
max_input_len=50
embedding_size=200
dropout_keep_prob=0.5
attention_type=(...).Attention
data_id="source"
vocabulary=<encoder_vocabulary>

[trainer]
class=(...).CrossEntropyTrainer
decoders=[<decoder>]
l2_regularization=1.0e-8
```

```
[decoder]
class=(...).Decoder
name="decoder"
encoders=[<encoder>]
rnn_size=256
max_output_len=50
embedding_size=256
use_attention=True
dropout_keep_prob=0.5
data_id="target"
vocabulary=<decoder_vocabulary>

[runner]
class=(...).GreedyRunner
decoder=<decoder>
output_series="target"
```

Figure 4. An example of construction of the encoder, decoder, trainer and runner objects by direct calls of their class constructors.

```
[train_data]
class=dataset.load_dataset_from_files
s_source="tests/data/train.tc.en"
s_target="tests/data/train.tc.de"
s_target_out="train.translated.de"
```

The `dataset.load_dataset_from_files` function is called as the dataset initializer. All arguments starting with `s_` correspond to named data series (`source` and `target`) and provide paths to files containing the data. If a data series argument ends with `_out`, the path is interpreted as the output file for the series.

Figure 5. Example of a dataset. The <val_data> *object is created analogically.*

```
[encoder_vocabulary]
class=vocabulary.from_dataset
datasets=[<train_data>]
series_ids=[source]
max_size=25000
save_file="enc_vocab.pkl"
```

This snippet shows how a vocabulary is created from a dataset. The function simply iterates through the dataset series `source`, computes the tokens frequency and keeps 25,000 most frequent tokens. After the vocabulary is created, it is stored in the specified file.

Figure 6. Example of creating vocabulary. The <decoder_vocabulary> *object is created analogically.*

- `neuralmonkey-run` – This script is used for loading a trained model and its execution on a dataset. It requires the location of the original configuration file and another configuration file that specifies the location of the dataset files.
- `neuralmonkey-server` – A trained model can be run as a web service. It accepts dictionaries of data series for encoders and returns the series produced by decoders in a JSON format with a REST API.
- `neuralmonkey-logbook` – The experiment logs and configurations can be browsed using a small web service Neural Monkey LogBook. It is called with a `--logdir` argument, which points to a directory with experiments. When the directory has sub-directories, they are regarded as separate experiments and are all shown in the LogBook.

6. Implemented Features

This section provides a short overview on features that have been already implemented in Neural Monkey:

- standard attentive S2S learning with a decoder capable of working with multiple encoders with GRU or LSTM networks;
- sequence labeling and sequence classification and regression;
- pre-trained ImageNet network for experiments with image captioning and multimodal machine translation (with optional model fine-tuning);
- custom deep convolutional networks for image processing;
- beam search and model ensembling;
- conditional gated recurrent units for decoder (Firat and Cho, 2016);
- byte-pair encoding for translation with sub-word units (Sennrich et al., 2016).

The possibility to use multiple decoders at one moment allows to easily build architectures for multi-task learning, like joint training of POS tagging on the source language together with machine translation.

Other advanced features like layer normalization or various learning methods optimizing the outputs directly towards BLEU score (Shen et al., 2016; Rennie et al., 2016) are expected to be added in the future.

7. Benchmarking

In order to validate the Neural Monkey's performance, we a sanity check evaluation on the architecture introduced by Bahdanau et al. (2014) which became the standard baseline model in NMT research.[4] Following Bahdanau et al. (2014), we train and evaluate our models using data provided for the WMT14 news task[5] from

[4]Reference implementation using Theano can be found here: `https://github.com/lisa-groundhog/GroundHog`

[5]Task details can be found here: `http://www.statmt.org/wmt14/translation-task.html`

model	BLEU score	epochs
Bahdanau et al. (2014) – neural	26.75	2.2
Bahdanau et al. (2014) – neural	28.45	6.0
SMT	33.30	—
Neural Monkey – greedy decoding	25.08	1.2
Neural Monkey + CGRU – greedy decoding	27.35	1.6

Table 1. Results achieved by Neural Monkey on the WMT14 News Task French to English dataset with the number epochs the training was running.

English to French. We use the same dataset as Bahdanau et al. (2014) for both training and evaluation of the models.

The encoder is a bi-directional GRU network (Cho et al., 2014) with 500 hidden units in each direction, the decoder is an RNN decoder with 1,000 units in the hidden layer. Unlike the original model, we do not use the max-out projection, but a simple 'tanh' projection. Whereas the original paper used Adadelta (Zeiler, 2012) optimizer, we used Adam (Kingma and Ba, 2014) in our experiments. The comparison of the models is shown in Table 1.

We have also experimented with conditional GRU (Firat and Cho, 2016) units which show the expected improvement on top of the baseline model. Another easily achievable improvement using Neural Monkey could be done with sub-word units (Sennrich et al., 2016) to deal with the out-of-vocabulary tokens.

8. Conclusions

We presented *Neural Monkey* – a new toolkit for sequence learning built using TensorFlow. It is aimed at gathering implementations of recent deep learning methods in various fields, primarily focused on NMT. Compared to other toolkits, it provides a higher level of abstraction, along with a simple configuration mechanism that allows for fast prototyping and reusing trained models and experiment management.

In the future, more features will be added. We also hope for community feedback from using the toolkit in practice that will help us to reevaluate the design decisions we have made. For the information on the recent development, we refer the reader to the online documentation. [6]

Acknowledgments

This research was supported by the Charles University grant no. GAUK 52315/2015, SVV 260 224, EU grants no. FP7-ICT-2013-10-610516 (QTLeap), H2020-ICT-2014-

[6] `https://readthedocs.org/neuralmonkey`

1-645452 (QT21), and H2020-ICT-2014-1-644753 (KConnect), and the Czech Science Foundation grant no. P103/12/G084 (CEMI).

Bibliography

Abadi, Martın, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Avramidis, Eleftherios, Vivien Macketanz, Aljoscha Burchardt, Jindrich Helcl, and Hans Uszkoreit. Deeper Machine Translation and Evaluation for German. In Hajič, Jan, Gertjan van Noord, and António Branco, editors, *Proceedings of the 2nd Deep Machine Translation Workshop*, pages 29–38, Praha, Czechia, 2016. ÚFAL MFF UK, ÚFAL MFF UK. ISBN 978-80-88132-02-8. URL `http://www.aclweb.org/anthology/W16-6404`.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1410.0473`.

Bergstra, James, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

Cho, Kyunghyun, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W14-4012`.

Chollet, François. Keras. `https://github.com/fchollet/keras`, 2015.

Collobert, Ronan, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Firat, Orhan and Kyunghyun Cho. Conditional Gated Recurrent Unit with Attention Mechanism. https://github.com/nyu-dl/dl4mt-tutorial/blob/master/docs/cgru.pdf, May 2016. Published online, version `adbaeea`.

Gillick, Dan, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. Multilingual Language Processing From Bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1296–1306, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/N16-1155`.

Kingma, Diederik P. and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Klein, Guillaume, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *CoRR*, abs/1701.02810, 2017. URL `http://arxiv.org/abs/1701.02810`.

Libovický, Jindřich, Jindřich Helcl, Marek Tlustý, Pavel Pecina, and Ondřej Bojar. CUNI System for WMT16 Automatic Post-Editing and Multimodal Translation Tasks. *CoRR*, abs/1606.07481, 2016. URL http://arxiv.org/abs/1606.07481.

Rennie, Steven J., Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical Sequence Training for Image Captioning. *CoRR*, abs/1612.00563, 2016. URL http://arxiv.org/abs/1612.00563.

Sennrich, Rico, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1162.

Sennrich, Rico, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. Nematus: a Toolkit for Neural Machine Translation. In *Proceedings of the Demonstrations at the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Valencia, Spain, 2017.

Shen, Shiqi, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum Risk Training for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1683–1692, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1159.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In Ghahramani, Z., M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

Tokui, Seiya, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: A next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

van Merriënboer, Bart, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and Fuel: Frameworks for deep learning. *CoRR*, abs/1506.00619, 2015. URL http://arxiv.org/abs/1506.00619.

Vinyals, Oriol and Quoc V. Le. A Neural Conversational Model. In *ICML Deep Learning Workshop*, 2015. URL http://arxiv.org/pdf/1506.05869v3.pdf.

Vinyals, Oriol, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3156–3164, 2015. doi: 10.1109/CVPR.2015.7298935. URL http://dx.doi.org/10.1109/CVPR.2015.7298935.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image

Caption Generation with Visual Attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, pages 2048–2057, Lille, France, 2015. URL http://jmlr.org/proceedings/papers/v37/xuc15.html.

Zeiler, Matthew D. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

Address for correspondence:
Jindřich Helcl
helcl@ufal.mff.cuni.cz
Institute of Formal and Applied Linguistics,
Faculty of Mathematics and Physics,
Charles University
Malostranské náměstí 25
118 00 Praha 1, Czech Republic