PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 109 OCTOBER 2017 15–28

NMTPY: A Flexible Toolkit for Advanced Neural Machine Translation Systems

Ozan Caglayan, Mercedes García-Martínez, Adrien Bardet, Walid Aransa, Fethi Bougares, Loïc Barrault

Laboratoire d'Informatique de l'Université du Maine (LIUM)

Abstract

In this paper, we present *nmtpy*, a flexible Python toolkit based on Theano for training Neural Machine Translation and other neural sequence-to-sequence architectures. *nmtpy* decouples the specification of a network from the training and inference utilities to simplify the addition of a new architecture and reduce the amount of boilerplate code to be written. *nmtpy* has been used for LIUM's top-ranked submissions to WMT Multimodal Machine Translation and News Translation tasks in 2016 and 2017.

1. Introduction

nmtpy is a refactored, extended and Python 3 only version of *dl4mt-tutorial*[1], a Theano (Theano Development Team, 2016) implementation of attentive Neural Machine Translation (NMT) (Bahdanau et al., 2014). The development of *nmtpy* project which has been open-sourced[2] under MIT license in March 2017, started in March 2016 as an effort to adapt *dl4mt-tutorial* to multimodal translation models. *nmtpy* has now become a powerful toolkit where adding a new model is as simple as deriving from an abstract base class, implementing a set of its methods and writing a custom data iterator if necessary. The training and inference utilities are as model-agnostic

[1] https://github.com/nyu-dl/dl4mt-tutorial

[2] https://github.com/lium-lst/nmtpy

 Corresponding author: ozancag@gmail.com
Cite as: Ozan Caglayan, Mercedes García-Martínez, Adrien Bardet, Walid Aransa, Fethi Bougares, Loïc Barrault. NMTPY: A Flexible Toolkit for Advanced Neural Machine Translation Systems. The Prague Bulletin of Mathematical Linguistics No. 109, 2017, pp. 15–28. doi: 10.1515/pralin-2017-0035.

as possible allowing one to use them for different sequence generation networks such as multimodal NMT and image captioning to name a few.

Other prominent toolkits in the field are OpenNMT (Klein et al., 2017), Neural Monkey (Helcl and Libovický, 2017) and Nematus (Sennrich et al., 2017). While *nmtpy* and Nematus share the same *dl4mt-tutorial* codebase, the flexibility and the rich set of architectures (Section 3) are what differentiate our toolkit from Nematus. Both OpenNMT and Nematus are solely focused on translation by providing feature-rich but monolithic NMT implementations. Neural Monkey which is based on TensorFlow (Abadi et al., 2016), provides a more generic sequence-to-sequence learning framework similar to *nmtpy*.

2. Design

In this section we first give an overview of a typical NMT training session in *nmtpy* and the design of the translation utility *nmt-translate*. We then describe the configuration file format, explain how to define new architectures and finally introduce the basic deep learning elements and techniques provided by *nmtpy*. A more detailed tutorial about training an NMT model is available on Github [3].

2.1. Training

A training experiment (Figure 1) is launched by providing an INI-style experiment configuration file to *nmt-train* (Listing 1). *nmt-train* then automatically selects a free GPU, sets the seed for NumPy and Theano random number generators, constructs an informative filename for log files and model checkpoints and finally instantiates a Python object of type `"model_type"` given through the configuration file. The tasks of data loading, weight initialization and graph construction are all delegated to this model instance.

```
$ nmt-train -c en-de.conf                                    # Launch an experiment
$ nmt-train -c en-de.conf 'model_type:new_nmt'               # Override model_type
$ nmt-train -c en-de.conf 'rnn_dim:500' 'embedding_dim:300'  # Change dimensions
$ nmt-train -c en-de.conf 'device_id:gpu5'                   # Force specific GPU device
```

Listing 1. Example usages of nmt-train.

During training, *nmt-train* consumes mini-batches of data from the model's iterator and performs forward/backward passes along with the weight updates. Translation performance on a held-out corpus is periodically evaluated in order to early-stop the training process to avoid overfitting. These periodic evaluations are realized by calling *nmt-translate* which performs beam-search, computes metrics and returns them back to *nmt-train*.

[3] `https://github.com/lium-lst/wmt17-mmt`

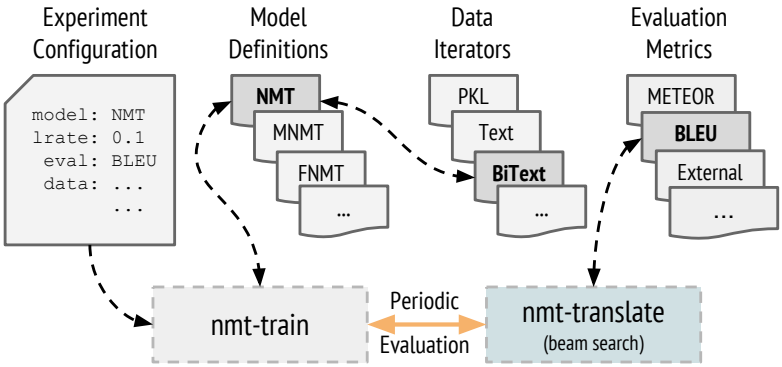

Figure 1. The workflow of a training experiment.

2.2. Translation

nmt-translate performs translation decoding using a beam-search implementation that supports single and ensemble decoding for both monomodal and multimodal translation models (Listing 2).

Since the number of CPUs in a single machine is 2x-4x higher than the number of GPUs and we mainly reserve the GPUs for training, *nmt-translate* makes use of CPU workers for maximum efficiency. More specifically, each worker receives a model instance (or instances when ensembling) and performs the beam-search on samples that it continuously fetches from a shared queue filled by the master process. One thing to note for parallel CPU decoding is that if the installed NumPy is linked against a BLAS implementation with threading support enabled (as in the case with Anaconda & Intel MKL), each spawned process attempts to use all available threads in the machine leading to a resource conflict. In order for *nmt-translate* to benefit correctly from parallelism, the number of threads per process should thus be limited to one [4]. The impact of this setting and the overall decoding speed in terms of words/sec (wps) are reported in Table 1 for a medium-sized En→Tr NMT with ~10M parameters.

```
# Decode on 30 CPUs with beam size 10, compute BLEU/METEOR
$ nmt-translate -j 30 -b 10 -M bleu meteor -m model.npz -S val.bpe.en -R val.de -o out.de
# Generate 50-best list with an ensemble of checkpoints
$ nmt-translate -b 50 -N 50 -m model*npz -S val.tok.de -o out.tok.50best.de
```

Listing 2. Example usages of nmt-translate.

[4] This is achieved by setting `X_NUM_THREADS=1` environment variable where `X` is one of `OPENBLAS,OMP,MKL` depending on the NumPy installation.

# BLAS Threads	Tesla K40	4 CPU	8 CPU	16 CPU
Default	185 wps	25 wps	25 wps	25 wps
Set to 1	185 wps	109 wps	198 wps	332 wps

Table 1. Median beam-search speed over 3 runs with beam size 12: decoding on a single Tesla K40 GPU is roughly equivalent to using 8 CPUs (Intel Xeon E5-2687v3).

2.3. Configuration

Each *nmtpy* experiment is defined with an INI-style configuration file that has four mandatory sections, namely `[training]`, `[model]`, `[model.dicts]` and `[model.data]`. Each section may contain a number of options in `key:value` format where the value can be built-in Python data types like integer, float, boolean, string, list, etc. Paths starting with a tilde are automatically expanded to `$HOME` folder.

The options defined in the `[training]` section are consumed by *nmt-train* while the ones in the `[model.*]` sections are automatically passed to the model instance (specifically, to its `__init__()` method) created by *nmt-train*. This allows one to add a new `key:value` option to the configuration file and access it automatically from the model instance.

Any option defined in the configuration file can be overridden through the command line by passing new `key:value` pair as the last argument to *nmt-train* (Listing 1). The common defaults defined in `nmtpy/defaults.py` are shortly described in Table 2. A complete configuration example is provided in Appendix A.

2.4. Defining New Architectures

A new architecture can be defined by creating a new file (i.e. `my_amazing_nmt.py`) under `nmtpy/models`, defining a new `Model` class derived from `BaseModel` and implementing [5] the set of methods detailed below:

- `__init__()`: Instantiates a model. Keyword arguments can be used to gather model specific options from the configuration file.
- `init_params()`: Initializes the layers and their weights.
- `build()`: Defines the computation graph for training.
- `build_sampler()`: Defines the computation graph for beam-search. This is similar to `build()` except two additional Theano functions.
- `load_valid_data()`: Loads the validation data for perplexity computation.
- `load_data()`: Loads the training data.

[5] The NMT architecture defined in `attention.py` can generally be used as a skeleton code when developing new architectures.

2.5. Building Blocks

Initialization Weight initialization is governed by the `weight_init` option and supports Xavier (Glorot and Bengio, 2010), He (He et al., 2015), orthogonal (Saxe et al., 2013) and random normal initializations.

Regularization An inverse-mode (the magnitudes are scaled during training instead of testing) dropout (Srivastava et al., 2014) can be applied over any tensor. L_2 weight regularization with a scalar factor given by `decay_c` option is also provided.

Option	Value	Description
`[training]` options		
`init`	**`None`**`/<.npz file>`	Pretrained checkpoint to initialize the weights.
`device_id`	**`auto`**`/cpu/gpu<int>`	Select training device automatically or manually.
`seed`	`1234`	The seed for Theano and NumPy RNGs.
`clip_c`	`5.0`	Gradient norm clipping threshold.
`decay_c`	`0.0`	L_2 regularization factor.
`patience`	`10`	Early-stopping patience.
`patience_delta`	`0.0`	Absolute difference of early-stopping metric that will be taken into account as an improvement.
`max_epochs`	`100`	Maximum number of epochs for training.
`max_iteration`	`1e6`	Maximum number of updates for training.
`valid_metric`	**`bleu`**`/meteor/px`	Validation metric(s) (separated by comma) to be printed, first being the early-stopping metric.
`valid_start`	`1`	Start validation after this number of epochs finished.
`valid_freq`	`0`	`0` means validations occur at end of epochs while an explicit `<int>` defines the period in terms of updates.
`valid_njobs`	`16`	Number of CPUs to use during validation beam-search.
`valid_beam`	`12`	The size of the beam during validation beam-search.
`valid_save_hyp`	**`False`**`/True`	Dumps validation hypotheses to separate text files.
`disp_freq`	`10`	The frequency of logging in terms of updates.
`save_best_n`	`4`	Save 4 best models on-disk based on validation metric for further ensembling.
`[model]` options		
`weight_init`	**`xavier`**`/he/<float>`	Weight initialization method or a `<float>` to define the scale of random normal distribution.
`batch_size`	`32`	Mini-batch size for training.
`optimizer`	**`adam`**`/adadelta/sgd/rmsprop`	Stochastic optimizer to use for training.
`lrate`	**`None`**`/<float>`	If given, overrides the optimizer default defined in `nmtpy/optimizers.py`.

*Table 2. Description of options and their default values: when the number of possible values is finite, the default is written in **bold**.*

Layers Feed-forward layer, highway layer (Srivastava et al., 2015), Gated Recurrent Unit (GRU) (Chung et al., 2014) Conditional GRU (CGRU) (Firat and Cho, 2016) and Multimodal CGRU (Caglayan et al., 2016a,b) are currently available for architecture design. Layer normalization (Ba et al., 2016), a method that adaptively learns to scale and shift the incoming activations of a neuron is available for GRU and CGRU blocks.

Iteration Parallel and monolingual text iterators with compressed (.gz, .bz2, .xz) file support are available under the names `TextIterator` and `BiTextIterator`. Additionally, the multimodal `WMTIterator` allows using image features and source/target sentences at the same time for multimodal NMT (Section 3.3). An efficient target length based batch sorting is available with the option `shuffle_mode:trglen`.

Training *nmtpy* provides Theano implementations of stochastic gradient descent (SGD) and its adaptive variants RMSProp (Tieleman and Hinton, 2012), Adadelta (Zeiler, 2012) and Adam (Kingma and Ba, 2014) to optimize the weights of the trained network. A preliminary support for gradient noise (Neelakantan et al., 2015) is available for Adam. Gradient norm clipping (Pascanu et al., 2013) is enabled by default with a threshold of 5 to avoid exploding gradients. Although the provided architectures all use the cross-entropy objective by their nature, any arbitrary differentiable objective function can be used since the training loop is agnostic to the architecture being trained.

Post-processing All decoded translations will be post-processed if `filter` option is given in the configuration file. This is useful in the case where one would like to compute automatic metrics on surface forms instead of segmented. Currently available filters are *bpe* and *compound* for cleaning subword BPE (Sennrich et al., 2016) and German compound-splitting (Sennrich and Haddow, 2015) respectively.

Metrics *nmt-train* performs a patience based early-stopping using either validation perplexity or one of the automatic metric wrappers i.e. BLEU (Papineni et al., 2002) or METEOR (Lavie and Agarwal, 2007). These metrics are also available for *nmt-translate* to immediately score the produced hypotheses. Other metrics can be easily added and made available as early-stopping metrics.

3. Architectures

3.1. Neural Machine Translation (NMT)

The NMT architecture (**attention**) is based on *dl4mt-tutorial* which differs from Bahdanau et al. (2014) in the following major aspects:

- The decoder is CGRU (Firat and Cho, 2016) which consists of two GRU interleaved with attention mechanism,
- The hidden state of the decoder is initialized with a non-linear transformation applied to *mean* bi-directional encoder state instead of *last* one,
- Maxout (Goodfellow et al., 2013) layer before the softmax operation is removed.

Option	Value(s) (default)	Description
`init_cgru`	zero (text)	Initializes CGRU with zero instead of mean encoder state (García-Martínez et al., 2017).
`tied_emb`	2way/3way (False)	Allows 2way and 3way sharing of embeddings in the network (Inan et al., 2016; Press and Wolf, 2016).
`shuffle_mode`	simple (trglen)	Switch between simple and target-length ordered shuffling.
`layer_norm`	bool (False)	Enable/disable layer normalization for GRU encoder.
`simple_output`	bool (False)	Condition target probability only on decoder's hidden state (García-Martínez et al., 2017).
`n_enc_layers`	int (1)	Number of unidirectional encoders to stack on top of the bi-directional encoder.
`emb_dropout`	float (0)	Rate of dropout applied on source embeddings.
`ctx_dropout`	float (0)	Rate of dropout applied on source encoder states.
`out_dropout`	float (0)	Rate of dropout applied on pre-softmax activations.

Table 3. Description of configuration options for the NMT architecture.

The final NMT architecture offers many new options which are shortly explained in Table 3. We also provide a set of auxiliary tools which are useful for pre-processing and post-training tasks (Table 4).

Tool	Description
`nmt-bpe-*`	Clone of subword utilities for BPE processing (Sennrich et al., 2016).
`nmt-build-dict`	Generates `.pkl` vocabulary files from corpora prior to training.
`nmt-rescore`	Rescores n-best hypotheses with single/ensemble of models on GPU.
`nmt-coco-metrics`	Computes several metrics using MSCOCO evaluation tools (Chen et al., 2015).
`nmt-extract`	Extracts and saves weights from a trained model instance.

Table 4. Brief descriptions of helper NMT tools.

3.2. Factored NMT (FNMT)

Factored NMT (FNMT) is an extension of NMT which generates two output symbols (García-Martínez et al., 2016). In contrast to multi-task architectures, FNMT outputs share the same recurrence and output symbols are generated in a synchronous

fashion. Two variants which differ in how they handle the output layer are currently available: (`attention_factors`) where the lemma and factor embeddings are concatenated to form a single feedback embedding and (`attention_factors_seplogits`) where the output path for lemmas and factors are kept separate with different pre-softmax transformations applied for specialization.

3.3. Multimodal NMT (MNMT)

We provide several multimodal architectures where the probability of a target word is estimated given source sentence representations and visual features: (1) Fusion architectures (Caglayan et al., 2016a,b) extend monomodal CGRU into a multimodal one where a multimodal attention is applied over textual and visual features, (2) MNMT architectures based on global features make use of fixed-width visual features to ground NMT with visual informations (Caglayan et al., 2017).

3.4. Other

- A GRU-based reimplementation (**img2txt**) of *Show, Attend and Tell* image captioning architecture (Xu et al., 2015),
- A GRU-based language model architecture (**rnnlm**) to train recurrent language models. *nmt-test-lm* is the inference utility for perplexity computation of a corpus using a trained checkpoint.

4. Results

System	MMT	Test2017 Meteor (Rank)
NMT	En→De	53.8 (#3)
MNMT	En→De	54.0 (#1)
NMT	En→Fr	70.1 (#4)
MNMT	En→Fr	72.1 (#1)
System	**News**	**Test2017 BLEU**
NMT-UEDIN (Winner)	En→Tr	16.5
NMT-Ours (Post-deadline)	En→Tr	18.1
FNMT	En→Lv	16.2
FNMT	En→Cs	19.9

Table 5. Ensembling scores for LIUM's WMT17 MMT and News Translation submissions.

System	Test2017 BLEU	Test2017 METEOR
Nmtpy	30.8 ± 1.0	51.6 ± 0.5
Nematus	31.6	50.6

Table 6. Mean/std. deviation of 5 Nmtpy runs vs 1 Nematus run for WMT17 MMT En→De.

We present our submitted *nmtpy* systems for Multimodal Translation (MMT) and News Translation tasks of WMT17 (Table 5). For MMT, state-of-the-art results are obtained by our systems (Caglayan et al., 2017)[6] in both En→De and En→Fr tracks (Elliott et al., 2017). In the context of news translation task, our post-deadline En→Tr NMT system (García-Martínez et al., 2017) surpassed the official winner by 1.6 BLEU.

We also trained a monomodal NMT for WMT17 MMT En→De track with Nematus using hyper-parameters very similar to our submitted NMT architecture and found that the results are comparable for BLEU and slightly better for *nmtpy* in terms of METEOR (Table 6).

5. Conclusion

We have presented *nmtpy*, an open-source sequence-to-sequence framework based on *dl4mt-tutorial* and refined in many ways to ease the task of integrating new architectures. The toolkit has been internally used in our team for tasks ranging from monomodal, multimodal and factored NMT to image captioning and language modeling to achieve top-ranked campaign results and state-of-the-art performance.

Acknowledgements

This work was supported by the French National Research Agency (ANR) through the CHIST-ERA M2CR project, under the contract number ANR-15-CHR2-0006-01[7].

Bibliography

Abadi, Martín, 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.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL `http://arxiv.org/abs/1607.06450`.

[6] `http://github.com/lium-lst/wmt17-mmt`

[7] `http://m2cr.univ-lemans.fr`

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/1409.0473.

Caglayan, Ozan, Walid Aransa, Yaxing Wang, Marc Masana, Mercedes García-Martínez, Fethi Bougares, Loïc Barrault, and Joost van de Weijer. Does Multimodality Help Human and Machine for Translation and Image Captioning? In *Proceedings of the First Conference on Machine Translation*, pages 627–633, Berlin, Germany, August 2016a. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W16/W16-2358.pdf.

Caglayan, Ozan, Loïc Barrault, and Fethi Bougares. Multimodal Attention for Neural Machine Translation. *arXiv preprint arXiv:1609.03976*, 2016b. URL http://arxiv.org/abs/1609.03976.

Caglayan, Ozan, Walid Aransa, Adrien Bardet, Mercedes García-Martínez, Fethi Bougares, Loïc Barrault, Marc Masana, Luis Herranz, and Joost van de Weijer. LIUM-CVC Submissions for WMT17 Multimodal Translation Task. In *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark, September 2017.

Chen, Xinlei, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

Chung, Junyoung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Elliott, Desmond, Stella Frank, Loïc Barrault, Fethi Bougares, and Lucia Specia. Findings of the Second Shared Task on Multimodal Machine Translation and Multilingual Image Description. In *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark, September 2017.

Firat, Orhan and Kyunghyun Cho. Conditional Gated Recurrent Unit with Attention Mechanism. github.com/nyu-dl/dl4mt-tutorial/blob/master/docs/cgru.pdf, 2016.

García-Martínez, Mercedes, Loïc Barrault, and Fethi Bougares. Factored Neural Machine Translation Architectures. In *Proceedings of the International Workshop on Spoken Language Translation*, IWSLT'16, Seattle, USA, 2016. URL http://workshop2016.iwslt.org/downloads/IWSLT_2016_paper_2.pdf.

García-Martínez, Mercedes, Ozan Caglayan, Walid Aransa, Adrien Bardet, Fethi Bougares, and Loïc Barrault. LIUM Machine Translation Systems for WMT17 News Translation Task. In *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark, September 2017.

Glorot, Xavier and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 13–15 May 2010. URL http://proceedings.mlr.press/v9/glorot10a.html.

Goodfellow, Ian, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout Networks. In Dasgupta, Sanjoy and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine*

Learning Research, pages 1319–1327, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/goodfellow13.html.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 1026–1034. IEEE, 2015.

Helcl, Jindřich and Jindřich Libovický. Neural Monkey: An Open-source Tool for Sequence Learning. *The Prague Bulletin of Mathematical Linguistics*, (107):5–17, 2017. ISSN 0032-6585. doi: 10.1515/pralin-2017-0001. URL http://ufal.mff.cuni.cz/pbml/107/art-helcl-libovicky.pdf.

Inan, Hakan, Khashayar Khosravi, and Richard Socher. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Kingma, Diederik and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL http://arxiv.org/abs/1412.6980.

Klein, G., Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*, 2017.

Lavie, Alon and Abhaya Agarwal. Meteor: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 228–231, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1626355.1626389.

Neelakantan, Arvind, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015. URL http://arxiv.org/abs/1511.06807.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL http://dx.doi.org/10.3115/1073083.1073135.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043083.

Press, Ofir and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Saxe, Andrew M, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Sennrich, Rico and Barry Haddow. A Joint Dependency Model of Morphological and Syntactic Structure for Statistical Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 114–121. Association for Computational Linguistics, 2015.

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-Mayne, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Miceli Barone, Jozef Mokry, and Maria Nadejde. *Nematus: a Toolkit for Neural Machine Translation*, pages 65–68. Association for Computational Linguistics (ACL), 4 2017. ISBN 978-1-945626-34-0.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016. URL http://arxiv.org/abs/1605.02688.

Tieleman, Tijmen and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich 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-15)*, pages 2048–2057. JMLR Workshop and Conference Proceedings, 2015. URL http://jmlr.org/proceedings/papers/v37/xuc15.pdf.

Zeiler, Matthew D. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Appendix A: Example NMT Configuration

```
# Options in this section are consumed by nmt-train
[training]
model_type: attention   # Model type without .py
patience: 20            # early-stopping patience
valid_freq: 1000        # Compute metrics each 1000 updates
valid_metric: meteor    # Use meteor during validations
valid_start: 2          # Start validations after 2nd epoch
valid_beam: 3           # Decode with beam size 3
valid_njobs: 16         # Use 16 processes for beam-search
valid_save_hyp: True    # Save validation hypotheses
decay_c: 1e-5           # L2 regularization factor
clip_c: 5               # Gradient clip threshold
seed: 1235              # Seed for numpy and Theano RNG
save_best_n: 2          # Keep 2 best models on-disk
device_id: auto         # Pick 1st available GPU
max_epochs: 100

# Options in this section are passed to model instance
[model]
tied_emb: 2way          # weight-tying mode (False,2way,3way)
layer_norm: True        # layer norm in GRU encoder
shuffle_mode: trglen    # Shuffled/length-ordered batches
filter: bpe             # post-processing filter(s)
n_words_src: 0          # limit src vocab if > 0
n_words_trg: 0          # limit trg vocab if > 0
save_path: ~/models     # Where to store checkpoints
rnn_dim: 100            # Encoder and decoder RNN dim
embedding_dim: 100      # All embedding dim
weight_init: xavier
batch_size: 32
optimizer: adam
lrate: 0.0004
emb_dropout: 0.2        # Set dropout rates
ctx_dropout: 0.4
out_dropout: 0.4

# Vocabulary paths produced by nmt-build-dict
[model.dicts]
src: ~/data/train.norm.max50.tok.lc.bpe.en.pkl
trg: ~/data/train.norm.max50.tok.lc.bpe.de.pkl

# Training and validation data
[model.data]
train_src     : ~/data/train.norm.max50.tok.lc.bpe.en
train_trg     : ~/data/train.norm.max50.tok.lc.bpe.de
valid_src     : ~/data/val.norm.tok.lc.bpe.en
valid_trg     : ~/data/val.norm.tok.lc.bpe.de # BPE refs for validation perplexity
valid_trg_orig: ~/data/val.norm.tok.lc.de     # non-BPE refs for correct metric computation
```

Appendix B: Installation

nmtpy requires a Python 3 environment with NumPy and Theano v0.9 installed. A Java runtime (`java` should be in the `PATH`) is also needed by the METEOR implementation. You can run the below commands in the order they are given to install *nmtpy* into your Python environment:

```
# 1. Clone the repository
$ git clone https://github.com/lium-lst/nmtpy.git

# 2. Download METEOR paraphrase data files
$ cd nmtpy; scripts/get-meteor-data.sh

# 3. Install nmtpy
$ python setup.py install
```

Note that once you installed *nmtpy* with `python setup.py install`, any modifications to the source tree will not be visible until *nmtpy* is reinstalled. If you would like to avoid this because you are constantly modifying the source code (for adding new architectures, iterators, features), you can replace the last command above by `python setup.py develop`. This tells the Python interpreter to directly use *nmtpy* from the GIT folder. The final alternative is to copy `scripts/snaprun` into your `$PATH`, modify it to point to your GIT folder and launch training using it as in below:

```
$ which snaprun
/usr/local/bin/snaprun

# Creates a snapshot of nmtpy under /tmp and uses it
$ snaprun nmt-train -c wmt17-en-de.conf
```

Performance In order to get the best speed in terms of training and beam-search, we recommend using a recent version of CUDA, CuDNN and a NumPy linked against Intel MKL[8] or OpenBLAS.

Address for correspondence:
Ozan Caglayan
ozancag@gmail.com
Laboratoire d'Informatique de l'Université du Maine (LIUM)
Avenue Laënnec 72085
Le Mans, France

[8] Anaconda Python distribution is a good option which already ships an MKL-enabled NumPy.