Nematus: a Toolkit for Neural Machine Translation

Rico Sennrich[†] **Orhan Firat**[⋆] **Kyunghyun Cho**[‡] **Alexandra Birch**[†]
Barry Haddow[†] **Julian Hitschler**[¶] **Marcin Junczys-Dowmunt**[†] **Samuel Läubli**[§]
Antonio Valerio Miceli Barone[†] **Jozef Mokry**[†] **Maria Nădejde**[†]
[†]University of Edinburgh [⋆]Middle East Technical University
[‡]New York University [¶]Heidelberg University [§]University of Zurich

Abstract

We present Nematus, a toolkit for Neural Machine Translation. The toolkit prioritizes high translation accuracy, usability, and extensibility. Nematus has been used to build top-performing submissions to shared translation tasks at WMT and IWSLT, and has been used to train systems for production environments.

1 Introduction

Neural Machine Translation (NMT) (Bahdanau et al., 2015; Sutskever et al., 2014) has recently established itself as a new state-of-the art in machine translation. We present Nematus[1], a new toolkit for **Ne**ural **Ma**chine **T**ranslation.

Nematus has its roots in the dl4mt-tutorial.[2] We found the codebase of the tutorial to be compact, simple and easy to extend, while also producing high translation quality. These characteristics make it a good starting point for research in NMT. Nematus has been extended to include new functionality based on recent research, and has been used to build top-performing systems to last year's shared translation tasks at WMT (Sennrich et al., 2016) and IWSLT (Junczys-Dowmunt and Birch, 2016).

Nematus is implemented in Python, and based on the Theano framework (Theano Development Team, 2016). It implements an attentional encoder–decoder architecture similar to Bahdanau et al. (2015). Our neural network architecture differs in some aspect from theirs, and we will discuss differences in more detail. We will also describe additional functionality, aimed to enhance usability and performance, which has been implemented in Nematus.

[1] available at `https://github.com/rsennrich/nematus`

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

2 Neural Network Architecture

Nematus implements an attentional encoder–decoder architecture similar to the one described by Bahdanau et al. (2015), but with several implementation differences. The main differences are as follows:

- We initialize the decoder hidden state with the mean of the source annotation, rather than the annotation at the last position of the encoder backward RNN.
- We implement a novel conditional GRU with attention.
- In the decoder, we use a feedforward hidden layer with $\tanh$ non-linearity rather than a maxout before the softmax layer.
- In both encoder and decoder word embedding layers, we do not use additional biases.
- Compared to *Look*, *Generate*, *Update* decoder phases in Bahdanau et al. (2015), we implement *Look*, *Update*, *Generate* which drastically simplifies the decoder implementation (see Table 1).
- Optionally, we perform recurrent Bayesian dropout (Gal, 2015).
- Instead of a single word embedding at each source position, our input representations allows multiple features (or "factors") at each time step, with the final embedding being the concatenation of the embeddings of each feature (Sennrich and Haddow, 2016).

- We allow tying of embedding matrices (Press and Wolf, 2017; Inan et al., 2016).

Table 1: Decoder phase differences

RNNSearch (Bahdanau et al., 2015)		Nematus (DL4MT)	
Phase	Output - Input	Phase	Output - Input
Look	$\mathbf{c}_j \leftarrow \mathbf{s}_{j-1}, \mathrm{C}$	Look	$\mathbf{c}_j \leftarrow \mathbf{s}_{j-1}, y_{j-1}, \mathrm{C}$
Generate	$y_j \leftarrow \mathbf{s}_{j-1}, y_{j-1}, \mathbf{c}_j$	Update	$\mathbf{s}_j \leftarrow \mathbf{s}_{j-1}, y_{j-1}, \mathbf{c}_j$
Update	$\mathbf{s}_j \leftarrow \mathbf{s}_{j-1}, y_j, \mathbf{c}_j$	Generate	$y_j \leftarrow \mathbf{s}_j, y_{j-1}, \mathbf{c}_j$

We will here describe some differences in more detail:

Given a source sequence $(x_1, \ldots, x_{T_x})$ of length T_x and a target sequence $(y_1, \ldots, y_{T_y})$ of length T_y, let $\mathbf{h}_i$ be the annotation of the source symbol at position i, obtained by concatenating the forward and backward encoder RNN hidden states, $\mathbf{h}_i = [\overrightarrow{\mathbf{h}}_i; \overleftarrow{\mathbf{h}}_i]$, and $\mathbf{s}_j$ be the decoder hidden state at position j.

decoder initialization Bahdanau et al. (2015) initialize the decoder hidden state $\mathbf{s}$ with the last backward encoder state.

$$\mathbf{s}_0 = \tanh\left(\mathbf{W}_{init}\overleftarrow{\mathbf{h}}_1\right)$$

with $\mathbf{W}_{init}$ as trained parameters.[3] We use the average annotation instead:

$$\mathbf{s}_0 = \tanh\left(\mathbf{W}_{init}\frac{\sum_{i=1}^{T_x}\mathbf{h}_i}{T_x}\right)$$

conditional GRU with attention Nematus implements a novel conditional GRU with attention, $\mathrm{cGRU_{att}}$. A $\mathrm{cGRU_{att}}$ uses its previous hidden state $\mathbf{s}_{j-1}$, the whole set of source annotations $\mathrm{C} = \{\mathbf{h}_1, \ldots, \mathbf{h}_{T_x}\}$ and the previously decoded symbol y_{j-1} in order to update its hidden state $\mathbf{s}_j$, which is further used to decode symbol y_j at position j,

$$\mathbf{s}_j = \mathrm{cGRU_{att}}\left(\mathbf{s}_{j-1}, y_{j-1}, \mathrm{C}\right)$$

Our conditional GRU layer with attention mechanism, $\mathrm{cGRU_{att}}$, consists of three components: two GRU state transition blocks and an attention mechanism ATT in between. The first transition block, GRU_1, combines the previous decoded symbol y_{j-1} and previous hidden state $\mathbf{s}_{j-1}$ in order to generate an intermediate representation $\mathbf{s}'_j$ with the following formulations:

$$\begin{aligned}
\mathbf{s}'_j &= \mathrm{GRU}_1\left(y_{j-1}, \mathbf{s}_{j-1}\right) = (1 - \mathbf{z}'_j) \odot \underline{\mathbf{s}}'_j + \mathbf{z}'_j \odot \mathbf{s}_{j-1}, \\
\underline{\mathbf{s}}'_j &= \tanh\left(\mathbf{W}'\mathbf{E}[y_{j-1}] + \mathbf{r}'_j \odot (\mathbf{U}'\mathbf{s}_{j-1})\right), \\
\mathbf{r}'_j &= \sigma\left(\mathbf{W}'_r\mathbf{E}[y_{j-1}] + \mathbf{U}'_r\mathbf{s}_{j-1}\right), \\
\mathbf{z}'_j &= \sigma\left(\mathbf{W}'_z\mathbf{E}[y_{j-1}] + \mathbf{U}'_z\mathbf{s}_{j-1}\right),
\end{aligned}$$

where $\mathbf{E}$ is the target word embedding matrix, $\underline{\mathbf{s}}'_j$ is the proposal intermediate representation, $\mathbf{r}'_j$ and $\mathbf{z}'_j$ being the reset and update gate activations. In this formulation, $\mathbf{W}'$, $\mathbf{U}'$, $\mathbf{W}'_r$, $\mathbf{U}'_r$, $\mathbf{W}'_z$, $\mathbf{U}'_z$ are trained model parameters; σ is the logistic sigmoid activation function.

The attention mechanism, ATT, inputs the entire context set C along with intermediate hidden state $\mathbf{s}'_j$ in order to compute the context vector $\mathbf{c}_j$ as follows:

$$\begin{aligned}
\mathbf{c}_j &= \mathrm{ATT}\left(\mathrm{C}, \mathbf{s}'_j\right) = \sum_i^{T_x}\alpha_{ij}\mathbf{h}_i, \\
\alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{k=1}^{Tx}\exp(e_{kj})}, \\
e_{ij} &= \mathbf{v}_a^\intercal \tanh\left(\mathbf{U}_a\mathbf{s}'_j + \mathbf{W}_a\mathbf{h}_i\right),
\end{aligned}$$

where α_{ij} is the normalized alignment weight between source symbol at position i and target symbol at position j and $\mathbf{v}_a, \mathbf{U}_a, \mathbf{W}_a$ are the trained model parameters.

Finally, the second transition block, GRU_2, generates $\mathbf{s}_j$, the hidden state of the $\mathrm{cGRU_{att}}$, by looking at intermediate representation $\mathbf{s}'_j$ and context vector $\mathbf{c}_j$ with the following formulations:

$$\begin{aligned}
\mathbf{s}_j &= \mathrm{GRU}_2\left(\mathbf{s}'_j, \mathbf{c}_j\right) = (1 - \mathbf{z}_j) \odot \underline{\mathbf{s}}_j + \mathbf{z}_j \odot \mathbf{s}'_j, \\
\underline{\mathbf{s}}_j &= \tanh\left(\mathbf{W}\mathbf{c}_j + \mathbf{r}_j \odot (\mathbf{U}\mathbf{s}'_j)\right), \\
\mathbf{r}_j &= \sigma\left(\mathbf{W}_r\mathbf{c}_j + \mathbf{U}_r\mathbf{s}'_j\right), \\
\mathbf{z}_j &= \sigma\left(\mathbf{W}_z\mathbf{c}_j + \mathbf{U}_z\mathbf{s}'_j\right),
\end{aligned}$$

similarly, $\underline{\mathbf{s}}_j$ being the proposal hidden state, $\mathbf{r}_j$ and $\mathbf{z}_j$ being the reset and update gate activations with the trained model parameters $\mathbf{W}, \mathbf{U}, \mathbf{W}_r, \mathbf{U}_r, \mathbf{W}_z, \mathbf{U}_z$.

Note that the two GRU blocks are not individually recurrent, recurrence only occurs at the level

[3] All the biases are omitted for simplicity.

of the whole cGRU layer. This way of combining RNN blocks is similar to what is referred in the literature as *deep transition* RNNs (Pascanu et al., 2014; Zilly et al., 2016) as opposed to the more common *stacked* RNNs (Schmidhuber, 1992; El Hihi and Bengio, 1995; Graves, 2013).

deep output Given $\mathbf{s}_j$, y_{j-1}, and $\mathbf{c}_j$, the output probability $p(y_j|\mathbf{s}_j, y_{j-1}, \mathbf{c}_j)$ is computed by a softmax activation, using an intermediate representation $\mathbf{t}_j$.

$$p(y_j|\mathbf{s}_j,y_{j-1},\mathbf{c}_j) = \text{softmax}\left(\mathbf{t}_j\mathbf{W}_o\right)$$
$$\mathbf{t}_j = \tanh\left(\mathbf{s}_j\mathbf{W}_{t1} + \mathbf{E}[y_{j-1}]\mathbf{W}_{t2} + \mathbf{c}_j\mathbf{W}_{t3}\right)$$

$\mathbf{W}_{t1}, \mathbf{W}_{t2}, \mathbf{W}_{t3}, \mathbf{W}_o$ are the trained model parameters.

3 Training Algorithms

By default, the training objective in Nematus is cross-entropy minimization on a parallel training corpus. Training is performed via stochastic gradient descent, or one of its variants with adaptive learning rate (Adadelta (Zeiler, 2012), RmsProp (Tieleman and Hinton, 2012), Adam (Kingma and Ba, 2014)).

Additionally, Nematus supports minimum risk training (MRT) (Shen et al., 2016) to optimize towards an arbitrary, sentence-level loss function. Various MT metrics are supported as loss function, including smoothed sentence-level BLEU (Chen and Cherry, 2014), METEOR (Denkowski and Lavie, 2011), BEER (Stanojevic and Sima'an, 2014), and any interpolation of implemented metrics.

To stabilize training, Nematus supports early stopping based on cross entropy, or an arbitrary loss function defined by the user.

4 Usability Features

In addition to the main algorithms to train and decode with an NMT model, Nematus includes features aimed towards facilitating experimentation with the models, and their visualisation. Various model parameters are configurable via a command-line interface, and we provide extensive documentation of options, and sample set-ups for training systems.

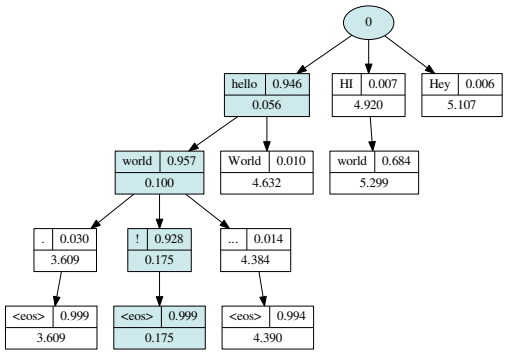

Figure 1: Search graph visualisation for DE→EN translation of "Hallo Welt!" with beam size 3.

Nematus provides support for applying single models, as well as using multiple models in an ensemble – the latter is possible even if the model architectures differ, as long as the output vocabulary is the same. At each time step, the probability distribution of the ensemble is the geometric average of the individual models' probability distributions. The toolkit includes scripts for beam search decoding, parallel corpus scoring and n-best-list rescoring.

Nematus includes utilities to visualise the attention weights for a given sentence pair, and to visualise the beam search graph. An example of the latter is shown in Figure 1. Our demonstration will cover how to train a model using the command-line interface, and showing various functionalities of Nematus, including decoding and visualisation, with pre-trained models.[4]

5 Conclusion

We have presented Nematus, a toolkit for Neural Machine Translation. We have described implementation differences to the architecture by Bahdanau et al. (2015); due to the empirically strong performance of Nematus, we consider these to be of wider interest.

We hope that researchers will find Nematus an accessible and well documented toolkit to support

[4]Pre-trained models for 8 translation directions are available at `http://statmt.org/rsennrich/wmt16_systems/`

their research. The toolkit is by no means limited to research, and has been used to train MT systems that are currently in production (WIPO, 2016).

Nematus is available under a permissive BSD license.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements 645452 (QT21), 644333 (TraMOOC), 644402 (HimL) and 688139 (SUMMA).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Boxing Chen and Colin Cherry. 2014. A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362–367, Baltimore, Maryland, USA.

Michael Denkowski and Alon Lavie. 2011. Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 85–91, Edinburgh, Scotland.

Salah El Hihi and Yoshua Bengio. 1995. Hierarchical Recurrent Neural Networks for Long-Term Dependencies. In *Nips*, volume 409.

Yarin Gal. 2015. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *ArXiv e-prints*.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

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

Marcin Junczys-Dowmunt and Alexandra Birch. 2016. The University of Edinburgh's systems submission to the MT task at IWSLT. In *The International Workshop on Spoken Language Translation (IWSLT)*, Seattle, USA.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Razvan Pascanu, Çağlar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2014. How to Construct Deep Recurrent Neural Networks. In *International Conference on Learning Representations 2014 (Conference Track)*.

Ofir Press and Lior Wolf. 2017. Using the Output Embedding to Improve Language Models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Valencia, Spain.

Jürgen Schmidhuber. 1992. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

Rico Sennrich and Barry Haddow. 2016. Linguistic Input Features Improve Neural Machine Translation. In *Proceedings of the First Conference on Machine Translation, Volume 1: Research Papers*, pages 83–91, Berlin, Germany.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Edinburgh Neural Machine Translation Systems for WMT 16. In *Proceedings of the First Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 368–373, Berlin, Germany.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Minimum Risk Training for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany.

Milos Stanojevic and Khalil Sima'an. 2014. BEER: BEtter Evaluation as Ranking. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 414–419, Baltimore, Maryland, USA.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, pages 3104–3112, Montreal, Quebec, Canada.

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

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5 - rmsprop.

WIPO. 2016. WIPO Develops Cutting-Edge Translation Tool For Patent Documents, Oct. `http://www.wipo.int/pressroom/en/articles/2016/article\_0014.html`.

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

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2016. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*.