Nematus: a Toolkit for Neural Machine Translation

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the EACL 2017 Software Demonstrations
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 Nadejde†
†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†, a new toolkit for Neural Machine Translation.

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 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 allow 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).

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 allow 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

<table>
<thead>
<tr>
<th>RNNSearch (Bahdanau et al., 2015)</th>
<th>Nematus (DL4MT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase Output - Input</td>
<td>Phase Output - Input</td>
</tr>
<tr>
<td>Look c_j ← s_{j-1}, C</td>
<td>Look c_j ← s_{j-1}, y_{j-1}, C</td>
</tr>
<tr>
<td>Generate y_j ← s_{j-1}, y_{j-1}, c_j</td>
<td>Update s_j ← s_{j-1}, y_{j-1}, c_j</td>
</tr>
</tbody>
</table>

We will here describe some details:

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 \(h_i\) be the annotation of the source symbol at position \(i\), obtained by concatenating the forward and backward encoder RNN hidden states, \(h_i = \left[ \overrightarrow{h}_i; \overleftarrow{h}_i \right]\), and \(s_j\) be the decoder hidden state at position \(j\).

**decoder initialization** Bahdanau et al. (2015) initialize the decoder hidden state \(s\) with the last backward encoder state.

\[
s_0 = \tanh \left( W_{init} \overleftarrow{h}_1 \right)
\]

with \(W_{init}\) as trained parameters. We use the average annotation instead:

\[
s_0 = \tanh \left( \frac{1}{T_x} \sum_{i=1}^{T_x} h_i \right)
\]

**conditional GRU with attention** Nematus implements a novel conditional GRU with attention, cGRU\(_{att}\). A cGRU\(_{att}\) uses its previous hidden state \(s_{j-1}\), the whole set of source annotations \(C = \{h_1, \ldots, h_{T_x}\}\) and the previously decoded symbol \(y_{j-1}\) in order to update its hidden state \(s_j\), which is further used to decode symbol \(y_j\) at position \(j\).

\[
s_j = \text{cGRU}_{att} (s_{j-1}, y_{j-1}, C)
\]

Our conditional GRU layer with attention mechanism, 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 \(s_{j-1}\) in order to generate an intermediate representation \(s'_j\) with the following formulations:

\[
s'_j = \text{GRU}_1 (s_{j-1}, s_{j-1}) = (1 - z'_j) \odot s'_j + z'_j \odot s_{j-1},
\]

\[
s''_j = \tanh (W'E[y_{j-1}] + r'_j \odot (U's_{j-1})),
\]

\[
r'_j = \sigma (W'E[y_{j-1}] + U's_{j-1}),
\]

\[
z'_j = \sigma (W'E[y_{j-1}] + U's_{j-1}),
\]

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

The attention mechanism, ATT, inputs the entire context set \(C\) along with intermediate hidden state \(s'_j\) in order to compute the context vector \(c_j\) as follows:

\[
c_j = \text{ATT} (C, s'_j) = \sum_i \alpha_{ij} h_i,
\]

\[
\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{kj})},
\]

\[
e_{ij} = \sum_{a=0} v_a^T \tanh (U_a s'_j + W_a h_i),
\]

where \(\alpha_{ij}\) is the normalized alignment weight between source symbol at position \(i\) and target symbol at position \(j\) and \(v_a, U_a, W_a\) are the trained model parameters.

Finally, the second transition block, GRU\(_2\), generates \(s_j\), the hidden state of the cGRU\(_{att}\), by looking at intermediate representation \(s'_j\) and context vector \(c_j\) with the following formulations:

\[
s_j = \text{GRU}_2 (s'_j, c_j) = (1 - z_j) \odot s'_j + z_j \odot s_j,
\]

\[
s_j = \tanh (Wc_j + r_j \odot (Us'_j)),
\]

\[
r_j = \sigma (Wc_j + U's'_j),
\]

\[
z_j = \sigma (Wc_j + U's'_j),
\]

similarly, \(s_j\) being the proposal hidden state, \(r_j\) and \(z_j\) being the reset and update gate activations with the trained model parameters \(W, U, W_r, U_r, W_z, U_z\).

Note that the two GRU blocks are not individually recurrent, recurrence only occurs at the level.
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 $s_j$, $y_j-1$, and $c_j$, the output probability $p(y_j|s_j, y_j-1, c_j)$ is computed by a softmax activation, using an intermediate representation $t_j$.

$$p(y_j|s_j, y_j-1, c_j) = \text{softmax}(t_jW_o)$$

$$t_j = \tanh(s_jW_{t1} + E[y_j-1]W_{t2} + c_jW_{t3})$$

$W_{t1}, W_{t2}, W_{t3}, W_o$ 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.

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.\footnote{Pre-trained models for 8 translation directions are available at http://statmt.org/rsennrich/wmt16_systems/}

5 **Conclusion**

(Sennrich et al., 2017)

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
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


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.


