Approaches in Neural Machine Translation: Recurrent Network, Attention and Memory Network

Weicheng Zhu wz727@nyu.edu
Yiyi Zhang yz2092@nyu.edu
Zhengyuan Ding zd415@nyu.edu
Zihao Zhao zz913@nyu.edu

Abstract

As the improvement of neural network makes remarkable differences in natural language understanding and processing, handling hard tasks like machine translation becomes more possible. In this project, we tried multiple machine translation models including neural network and attention mechanism. With model selection and hyper-parameter tuning, the best model for Vietnamese-English translation has a 29.1 BLEU score on test set, and the best one for Chinese-English has a 23.6 BLEU score. We show that LSTM with the global attention of Luong et al.'s model performs better than others.

1 Introduction

In this project, we implemented and experimented on 4 models of translation systems for two language pairs, Vietnamese to English and Chinese to English: RNN based encoder-decoder with and without attention, self-attention based encoder, and fully self-attention translation.

The structure of the paper is as follows: In section 2 we present a brief background overview on the development of machine translation. Then we discuss the detailed architecture of various models used in this project in section 3. Section 4 talks about the training process and the corresponding results are given in section 5 with some analysis on different model configurations. Finally, we do comparisons of different models discussed in section 3 and give final conclusions in section 6.

2 Background

Machine translation is computers automatic translation of text or speech in human languages. Statistical Machine Translation (SMT) has been a popular option for machine translation (Brown et al., 1988)(Brown et al., 1990)(Brown et al., 1993) for decades, generally using a phrase-based implementation (Koehn et al., 2003). When the progress of SMT slowed down, the advent of the Neural Machine Translation(NMT) brought new inspirations to this field with a promising performance. Comparing to STM, NMT system requires fewer memory and has a simpler model structure, where all parts of the model are trained jointly to maximize the translation performance (Kalchbrenner and Blunsom, 2013)(Sutskever et al., 2014)(Cho et al., 2014a). NMT is typically done using a recurrent neural network with the encoder-decoder structure.

However, this traditional encoder-decoder model faces a problem when the input is long or very informative and the whole source sentence is required to be compressed into a fixed-size vector. This may be problematic since for translation each token generated is highly-related to certain part of the input text to keep an alignment (Young et al., 2018). Therefore, the attention mechanism was introduced according to this intuition (Bahdanau et al., 2014).

Furthermore, to address the sequential processing at encoder step, Vaswani et al. proposed the memory network based Transformer that dispenses the recurrence and relies entirely on the attention mechanism. This approach is significantly more parallelizable with less training time and as a result also gains higher translation quality (Vaswani et al., 2017).

In the following section, we will discuss the architectures of all of these models mentioned from simple encoder-decoder to attention based methods

in detail and experiment them on our two language pairs.

3 Model Architecture

Recently, most powerful deep neural translation models are using the encoder-decoder structure to deal with the sequential linguistic data (Sutskever et al., 2014). The encoder reads an input with variable length $x = [x_1, \cdots, x_{T_x}]$ and outputs a latent vector z with fixed length d. The decoder, then generates a sequence of elements $[y_1, \cdots, y_{T_y}]$ one at a time with variable length given the encoder output z. Every y_t is generated based on z and all the previously generated elements $y_{<t}$.

3.1 RNN Encoder

Every sentence has been transformed into a vector $x = [x_1, \cdots, x_{T_x}]$, where every entry represents a word in our vocabulary. Now, we apply an Embedding matrix $(d \times |V|)$ to transfer original sentence vector x to a matrix with size $m \times d$, where m is the sentence length. Then we apply a bidirectional recurrent neural network to this matrix in order to obtain the sequential info. The RNN, in fact, is a parametric recursive function that is able to read a sequence and use a hidden memory vector in order to understand natural languages. This recursive function takes as input both one input symbol x_t and the memory vector h, and it returns the updated memory vector. We apply bidirectional layer here to contain both the preceding sequential info and the reversed order for better capturing the context of a certain word. We concatenate every memory vector and sum up the vectors we get from two directions to get our encoder output $\mathbf{z} = [h_1, ..., h_{T_x}]$. The last memory vector are used as the initial hidden of the decoder.

GRU & LSTM: Instead of using general RNN, we experimented with two kinds of recurrent neural layer: GRU and LSTM. General RNN will face the difficulties of carrying information from earlier time steps and vanishing gradient problems. By introducing the gates that can regulate the flow of information in GRU (Cho et al., 2014b) and LSTM (Hochreiter and Schmidhuber, 1997), useful info could be carried longer and the vanishing gradient problem is also alleviated to some extent because those two structures introduced the short cut.

3.2 RNN Decoder Without Attention

The decoder acts as a generator to generate tokens one by one based on its categorical distribution. The decoder also uses the recurrent neural net and additionally, dependent on the encoder output. Because the decoder generates the token one by one, the detailed training process adopts a for loop: for the first time, we directly assign the '<sos>' token as the decoder input and the last hidden from the encoder as the decoder hidden. After that, we will take the current token embedding from target sentence (teacher forcing = 1) or the last token embedding we predicted (teacher forcing = 0) as the decoder input. Teacher forcing here is a ratio to represent how we rely on the ground truth token from the target sentence (=1). The last hidden from the decoder becomes the decoder hidden at current step. By concatenating the encoder output with the token embedding we choose, we will feed it with the decoder hidden to a RNN to get the output. Then, by applying it to a linear layer and a log softmax function, we finally obtain the categorical distribution over vocabulary to predict next token.

3.3 RNN Decoder With Attention

Many researchers realized that the simple encoder-decoder forces to encoder to compress the source sentence into one single vector, it may bring some irrelevant info for the decoder to generate its current output. In order to enable the decoder to better focus on the corresponding words in the source sentence, attention mechanism is widely applied in neural translation nowadays. Generally speaking, for the decoder, besides taking in the last hidden state and current token from target sentence (teacher forcing ratio = 1), the decoder is also conditioned on a context vector calculated based on the weighted sum over the input hidden state sequence. The attention mechanism is a mapping from a query vector (the last hidden vector from the decoder) and a set of key-value pairs to an output (Young et al., 2018). Bahdanau et al (Bahdanau et al., 2014) first applied the attention mechanism to machine translation and improved the performance especially for long sequences. After that, other scholars use various ways to improve the initial attention structure. Our model experimented mainly on three types attention mechanisms: the initial one (Bahdanau et al., 2014), global attention mechanism

and local attention mechanism (Luong et al., 2015).

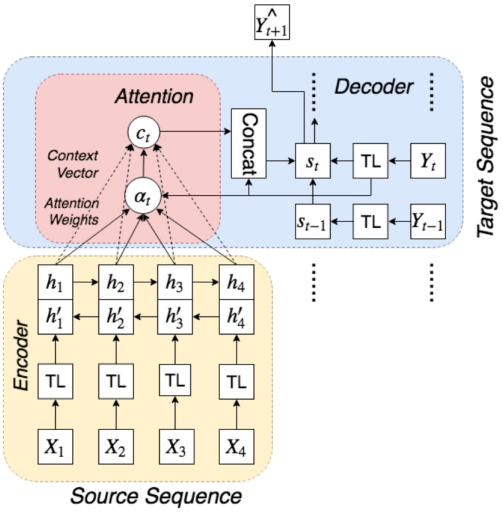

Figure 1: Bahdanau et al.'s model

Bahdanau et al.'s model: In this model, the attention weights were derived from the past hidden outputs of decoder (using unidirectional LSTM) and stacked encoder outputs of each word in source sentence. We compute the scores by using the query vector from the decoder and every memory vector from the encoder: $score(\mathbf{h_{t_{i-1}}}, \mathbf{s_j}) = \mathbf{v}^T \tanh(\mathbf{W s_j} + \mathbf{U h_{t_{i-1}}})$, where W and U are two fully connected layers. By applying the softmax function, we get the attention weights. The context vector is computed as the weighted sum of encoder outputs. Then the context vector was concatenated with the new feeding target word embedding vector as a new input into decoder LSTM. $p(y_i|y_{<i}, x) = g(y_{i-1}, \mathbf{h_{t_i}}, \mathbf{c_i})$, where $\mathbf{h_{t_i}} = f(\mathbf{h_{t_{i-1}}}, y_{i-1}, \mathbf{c_i})$

Luong et al.'s model: Global attention: Compared to Bahdanau et al.'s model, Luong et al.'s model improved it mainly in two ways: First, Luong et al. changed the initial structure of decoder: they first apply LSTM to the memory vectors, then compute the attention weights and the new context vector. After that, they concatenate c_t with the decoder outputs to produce $\tilde{h}_t$, and then make a prediction as detailed in the following equations: $\mathbf{\tilde{h_t}} = \tanh(\mathbf{W_c}[\mathbf{c_t}; \mathbf{h_t}])$, $p(y_t|y_{<t}, x) = softmax(\mathbf{W_s}\mathbf{\tilde{h_t}})$

On the other hand, at any time t, Bahdanau et al. build from the previous hidden state h_{t-1} and compute the attention weights at first. Then they

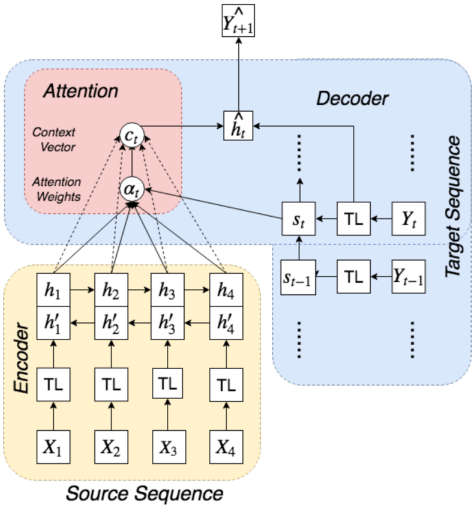

Figure 2: Luong et al.'s model

go through a LSTM layer as described above. The path Luong et al. has taken considered more past memory and included the current token info in the target sentence.

Secondly, Luong proposed several ways to compute the score between the query vector and the hidden vectors from the encoder other than simple concatenation in Bahdanau et al.'s model. We experimented both structures and methods of computing score and the results are shown in Figure 2.

Local attention: To save attention in a relatively smaller range, Luong et al. introduced the local attention. This mechanism selectively focuses on a small window of context and is differentiable. Based on our practice, we chose to use the predictive alignment P_t : $P_t = S \times sigmoid(\mathbf{v_p}^T \tanh(\mathbf{W_p h_t}))$, where W_p and v_p are the model parameters which will be learned to predict positions. S is the source sentence length. Then we place a Gaussian distribution centered around P_t to favor alignment point near P_t. We experimented with several window size and found out that Global attention still had better performance (See results in Table 1).

3.4 Self-Attention Encoder

Self-attention is an attention model that relates different positions of a single sequence to compute a representation of the sequence (Vaswani et al., 2017). A memory neural network here is used in self-attention encoder instead of RNN-based encoder. Memory neural network works like the memory of computers, storing keys and values and using

query to match with key in memory. With memory neural network, self-attention becomes a mapping from query and a set of key-value pairs in memory generated from original to attention weighted representation for the whole sequence.

Positional Encoding: As this encoder does not include RNN, we need to encode position of tokens in this sequence by adding positional information to embedding. We applied sin and cos with different periods to encode positional information.

$$PE_{pos,2i} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}})$$

Attention: The attention weight of the sentence is computed by the score evaluated by dot-product of queries and keys. To boost computation, we packed all queries, keys and values into matrix Q, K and V. We also normalize the score by dividing $\sqrt{d_k}$, $score(Q, K) = \frac{QK^T}{\sqrt{d_k}}$, and hence we can compute context vector as we did in previous attention model $c(Q, K, V) = softmax(score(Q, K))V$.

Multihead Attention: For the attention, we split each query, key and value in multihead h vectors and stack them together to compute in parallel. i.e. our attention function is transformed in the following way $MultiHead(Q, K, V) = [head_1, \cdots, head_h]W^O$ where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$. As d = 256 in the encoder, we use 4 heads.

Positionwise Feed-Forward Layer: To increase interaction between different layers of the attentions above, two linear fully connected layers are added after multihead attention, functioning similar as convolutions: $FF(x) = ReLU(x_{W_1} + b_1)W_2 + b_2$

3.5 Fully-Connected Self-Attention Network

With self-attention encoder, we can decode the sequence in two ways: RNN decoder and fully-connected self-attention decoder. To decode with RNN, we use the output of self-attention encoder in matrix of size sequence length $\times$ hidden size as the representation of source sentence, similar to RNN encoder. To construct a fully-connected self-attention, we build a multihead self-attention decoder which inherent the key and values in the memory of self-attention encoder.

In self-attention decoder, we use teaching force 1, meaning only use target sentence and encoded source sentence to train the decoded. For the target sentence, we process tokens with embedding and positional encoding as we did in self-attention encoder. Since we do not use RNN in fully connected network, we input the embedded target sentence as a whole rather than input a token at a time. To represent the positional information of target sentence, we need an extra masked self-attention layer: an upper-triangular square matrix with size of sentence lengths that masks one more token row-by-row. This stepwise mask together with multihead attention created a new query containing position information of target sentence, similar with the effect of token-wise RNN. In next attention layer, these query vectors then help compute the attention between source sentences and tokens in target sentence based on memory from encoder. Then we build connections between different heads with feed-forward layer and output the distribution of prediction with one more linear layer and softmax.

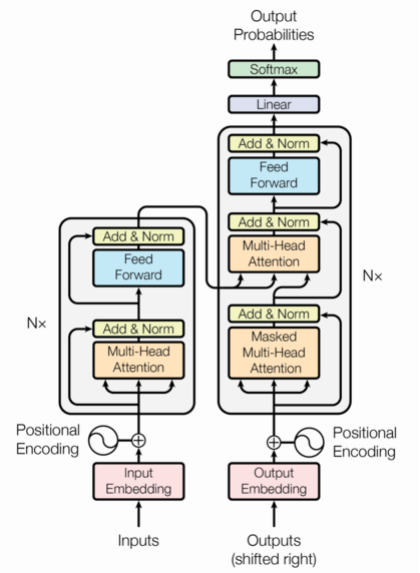

Figure 3: Fully-Connected Self-Attention network (Vaswani et al., 2017)

4 Training

4.1 Training Data and Batching

We have two language corpora including Vietnamese (VI) to English (EN) and Chinese(ZH) to English (EN). The VI-EN dataset consists of 133317 sentence pairs and we trimmed it to 133284 pairs by filtering out sentences with length larger than 200. For the ZH-EN dataset, there are 213376 pairs and is

trimmed to 213334 pairs by limiting sentence length to 200.

For string preprocessing, we normalize all the tokens: turn the unicode string to plain ASCII by removing accents, lowercase, trim, and remove non-letter characters. After string normalization, we then build two separated vocabularies for each language, adding tags like '<pad>', '<sos>', '<eos>', '<unk>' to store representations of padding, begin, end of a sentence and unknown tokens. As a result, Vietnamese has 42013 words and English has 41186 words. Chinese has 88784 words and corresponding English target words are 50875. After building a vocabulary for each of the language, we encode all the filtered sentences from strings of tokens to vectors by looking up the words in the corresponding vocabulary and replacing them by their indices. Also an '<eos>' token index is attached to the end of each sentence.

Sentence pairs are batched together to a batch size of 32 and are sorted within the batch. Then all the sentence vectors in a batch are padded by zeros to the same lengths as the maximum length of the batch sentences. By these procedures, sentences with close lengths are put together to minimize waste in parallel computing.

Masking: Tensors stored in batches has pad token with no meaning, which will affect the attention and loss computation. Hence, we masked pad positions in scoring of attention with 10^9 so that the attention weight from softmax will be close to zero. For loss computation, we also masked loss for padded positions since the loss between output distributions and pads are meaningless.

4.2 Optimizer

We use Adam with β_1=0.9, β_2=0.999 and $\epsilon = 10^{-8}$ (Contributors, 2018) as the optimizer. The method estimates the first and second moments of the gradients and computes an adaptive learning rate for each network weight (Kingma and Ba, 2014). This algorithm combines the advantages of both RMSprop and AdaGrad, which can deal with sparse gradients and non-stationary objectives.

4.3 Loss

We use negative log likelihood function to compute the loss based on log probability output and the target. Given the training set of sentence pairs $D = \{(X_1, Y_1), (X_2, Y_2), ..., (X_N, Y_N)\}$, we have: $loss = -\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_y} \log P(y_t^n | y_{<t}^n, x_n)$, where T_y is length of the target sentence; N is the number of data points; y_t is the t-th word of the target sentence; $y_{<t}$ is $[y_1, y_2, ..., y_{t-1}]$, i.e. all the preceding words before the t-th word, $\mathbf{x}$ is $[x_1, x_2, ..., x_{T_x}]$, i.e. all the words from the source sentence.

4.4 Regularization

We apply dropout 0.1 to the embedding and the positional encoding in both the encoder and the decoder. This dropout is also applied to each sub-layer output in the self-attention model.

5 Results

5.1 Text Generation

After we get the probability distribution $p(y_t|y_{<t}, x)$ over the whole vocabulary from the decoder, we need to form a sentence in target language as our translation. To get the exact best translation, we need to compute the probability over all possible translations, which is impossible due to the complexity. For example, suppose the vocabulary size is $|V|$, then we need to calculate $|V|^{|V|}$ probabilities at time 2, and $|V|^{|V|^{|V|}}$ probabilities at time 3, and so on and so forth. Therefore, we choose two more realistic algorithms.

5.1.1 Greedy Search

The simplest algorithm is to take only one best translation at each time step. For instance, we take $\hat{y_1} = argmax_y \log p(y_1|\mathbf{x})$ as the translation at the first position, and then based on this selection, we take $\hat{y_2} = argmax_y \log p(y_2|\hat{y_1}, \mathbf{x})$ as the translation at the second position. We repeat this procedure until the '<eos>' token has the highest probability over all other vocabulary.

5.1.2 Beam Search

In Greedy Search, we made decision of each token without thinking about the future. However, it is very likely that the translation which starts with a first token that has lower probability than other words turns out to have higher probability at time t. So beam search is used to optimize the greedy search. Instead of choosing only best token, beam search first choose k best tokens $w_1^1, \cdots, w_k^1$ at time $t = 1$. For each of the chosen token, the successor then will take into account all possible words, and compute the conditional probability $p(y_2 = w|y_1 = w_i^1, \mathbf{x})$ for all $w \in V$ and for all $i = 1, \cdots, k$. In the real implementation, we then choose k best tokens

which have the highest sum of log probabilities over all previous steps: $(w_1^1, w_1^2), \cdots, (w_k^1, w_k^2)$. It is easy to notice that beam search is close to the exact search as $k \to \infty$, and greedy search is the special case of beam search with $k = 1$. In this way, we need less computation than exact search, and we have a larger search space than beam search.

5.2 BLEU

To evaluate the model performance, we use corpus_bleu in sacrebleu API (Post, 2018) to compare the translation of our model to the target sentence in the dataset. Bleu (bilingual evaluation understudy) is an algorithm for evaluating the quality of machine translated text. It first calculate the precision of n-grams $p_n = \frac{m}{w_t}$ where m is number of n-grams from the translation $\hat{y}$ that are found in the reference y, and w_t is the total number of n-grams in the translation. Then it computes the brevity penalty $bp = \exp\{1 - \frac{|y|}{|\hat{y}|}\}$, if $|\hat{y}| < |y|$, and $bp = 1$ otherwise. At last, it returns a score between 0 and 1 by the formula: $bp \times \exp\{\frac{1}{4}\sum_{n=1}^{4} \log p_n\}$ (here sacrebleu use the precision up to 4-grams). A higher BLEU score indicates higher similarity between the translation and the target sentence. In corpus_bleu, the reported score is 100 $\times$ the original fraction.

5.3 Model Variation

Model	BLEU		PPL		# Para
	VI	ZH	VI	ZH	VI
LSTM w/o Attn	21.3	17.7	7.0	9.1	4×10^7
LSTM with Attn Bahdanau's model	27.8	20.0	4.0	6.6	4×10^7
LSTM with Attn Luong's global	30.0	23.1	2.9	4.1	4×10^7
LSTM with Attn Luong's local	27.1	21.4	3.8	4.8	4×10^7
GRU with Attn Luong's model	21.2	21.1	5.0	5.2	4×10^7
Self Attn Encoder	21.6	16.5	10.7	22.9	6×10^7
Fully-Connected	22.3	19.2	8.4	12.3	6×10^7

Table 1: Results

In Figure 4 and 5, we showed the best results for each model with greedy search after hyperparameter tuning. After fine tuning RNN models, we set hidden size=300, embedding size=300, dropout=0.1, learning rate=0.001, number of epochs=10, batch size=32. For the self-attention model and the fully-connected model, we change the learning rate to be 0.005 and batch size to be 16.

We also listed the highest BLEU score for each model under beam search in Table 1. From the results, we learned that attention can greatly improve the performance of machine translation; LSTM is generally better than GRU because of more gates control; self-attention models with memory network save resources of computation comparing to RNNs, but they occupy more memory permitting less batch size and thus, require more training time on P100 GPU. Besides, we noticed that VI-EN has better performance than ZH-EN over all models probably because of the difficult tokenization in Chinese.

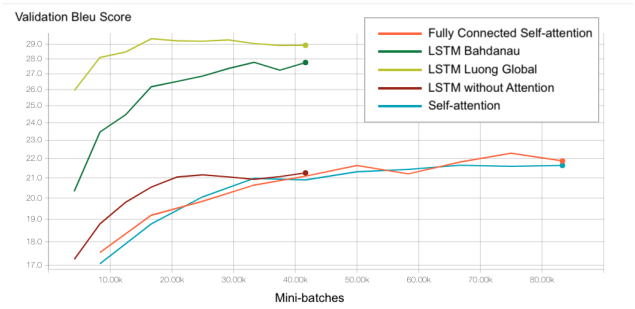

Figure 4: Model Selection for VI-EN

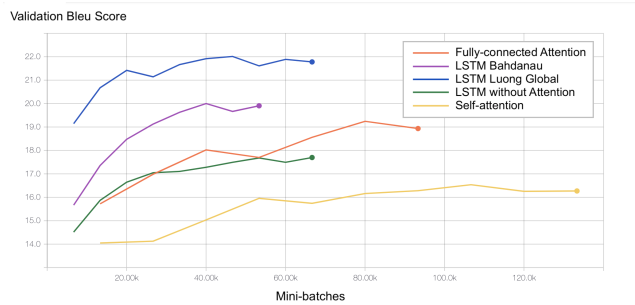

Figure 5: Model Selection for ZH-EN

6 Conclusion

For both language translations, we get the best results from LSTM with the global attention in Luong et al.s model. The final test result we got for VI-EN is BLEU 29.1 (with beam 3), and the final test result we got for ZH-EN is BLEU 23.6 (with beam 5).

Contribution Statements:

Weicheng Zhu: Coding, tuning, research

Yiyi Zhang: Coding, tuning, visualization

Zhengyuan Ding: Report, tuning, visualization

Zihao Zhao: Research, report composition, tuning

Code: https://github.com/jackzhu727/NLP-1011/tree/master/Final%20Project

References

[Bahdanau et al.2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Brown et al.1988] P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, R. Mercer, and P. Roossin. 1988. A statistical approach to language translation. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 1*, COLING '88, pages 71–76, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Brown et al.1990] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Comput. Linguist.*, 16(2):79–85, June.

[Brown et al.1993] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.*, 19(2):263–311, June.

[Cho et al.2014a] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.

[Cho et al.2014b] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[Contributors2018] Torch Contributors. 2018. Source code for torch.optim.adam. `https://pytorch.org/docs/stable/_modules/torch/optim/adam.html`.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jrgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Kalchbrenner and Blunsom2013] N. Kalchbrenner and P. Blunsom. 2013. Recurrent continuous translation models. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, (3):1700–1709.

[Kingma and Ba2014] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Koehn et al.2003] Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Luong et al.2015] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

[Post2018] Matt Post. 2018. A call for clarity in reporting BLEU scores. *CoRR*, abs/1804.08771.

[Sutskever et al.2014] I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, (3):3104–3112.

[Vaswani et al.2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

[Young et al.2018] T. Young, D. Hazarika, S. Poria, and E. Cambria. 2018. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13(3):55–75, Aug.