A Survey on Neural Machine Translation

Xi Yang
Zhejiang University
xya81@sfu.ca

Abstract

Neural networks has been successfully applied to solve machine translation problem, enhanced state-of-the-art translation performance, and is showing great future potential. This paper gives a brief introduction to the current achievements of neural machine translation (NMT). We review the history of NMT by introducing classic models in details. Then, we discuss some recently published papers to analyze the future possibility of NMT research.

1 Introduction

Deep neural network is now popular in various areas of data science. It revolutionized many tasks in the field of natural language processing. A series of neural network models has been successfully applied to solve machine translation problem (Sutskever et al., 2014; Bahdanau et al., 2014; Vaswani et al., 2017). These methods are pushing the state-of-the-art performance of machine translation to a higher level compared to traditional statistical or rule based machine translation systems(Sutskever et al., 2014), and still have a large potential. They show that Neural Machine Translation(NMT) is a promising research area.

Neural machine translation is a type of statistical machine translation method. NMT problem can be described as solving a maximum likelihood problem. The source sentence, which is the input to the model, can be represented as a vector $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$, where $\boldsymbol{x}_i$ is a vector that represents a symbol in the source language. This symbol is a word in source language vocabulary. The output of the model is another vector $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_n)$ forming the output sentence, where $\boldsymbol{y_i}$ indicates a symbol in the target language. Similarly, it has a corresponding word in target language vocabulary. The goal of statistical machine translation model is to find the translated sequence $\boldsymbol{y}$ for a given input sequence $\boldsymbol{x}$ that maximizes the statistical likelihood, i.e. find

$$\arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}; \theta)$$

where θ is the set of parameters in the model. These parameters are to be learned by training with large parallel language pairs between the source and target languages.

Neural translation models are showing more advantages than translation accuracy. Most of these models do not require linguistic knowledge to design and implement, because they have nothing connected with syntactical or semantic features of the languages. This property enables those who know little about linguistics can also design well-performed NMT algorithms, freeing computer scientists from learning linguistic knowledge. Besides, NMT models are similar to some deep learning models for other tasks(Gehring et al., 2017), therefore they can be easily transferred from, or transferred to other deep learning based tasks. These nice properties, along with its impressive performance, imply that NMT is a promising method for machine translation. But also, NMT models have some disadvantages. They require large scale corpus to train, but is sometimes difficult to get such corpus. Furthermore, NMT models are almost uninterpretable, so the progress of NMT research are mostly based on experiments and empirical study.

In section 2 of this paper, we introduce some classical NMT models that are important in the short history of NMT. In section 3, we review some state-of-the-art NMT models in recent proceedings, and discuss the research potentials behind them. Finally, we draw the conclusion in section 4.

2 Neural Models for Machine Translation

This section reviews classical models for NMT. Models in this section are considered to be significant contributions in the history of NMT. Some of the models, for example, the LSTM, are not initially targeted for machine translation task, but they achieved success when applied at machine translation tasks.

2.1 Recurrent Neural Network as Language Model

The idea of Recurrent Neural Network(RNN) is first introduced in (Bengio et al., 2003) as a language model. A language model describes the probability distribution of a word $\boldsymbol{x}^{(t+1)}$ after a given word sequence. That is to find

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)})$$

where $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)}$ is the given word sequence in certain language.

The key idea in RNN language model is using a hidden state $\boldsymbol{h}^{(t)}$ for each vector in the input sequence. Suppose we are going to predict the next word $\boldsymbol{y}^{(T)}$ after a word sequence $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(T)}$. With this setup, the forward propagation process of RNN takes these steps. First, each input word, as an indicator vector in vocabulary, is mapped into a space of lower dimension by a word embedding matrix, i.e.

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

Then, with a initial hidden state $\boldsymbol{h}^{(0)}$, the hidden state of each word is computed from the input vector for current word and the previous hidden vector.

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1) \quad (1)$$

where $\sigma(\cdot)$ is an activation function, typically sigmoid function or $tanh(\cdot)$. Later we will discuss some awesome alternatives to the function. The final output of the model comes from the last hidden state

$$\hat{\boldsymbol{y}}^{(T)} = softmax(\boldsymbol{U}\boldsymbol{h}^{(T)} + \boldsymbol{b}_2)$$

where

$$softmax(\boldsymbol{x})_i = \frac{exp(\boldsymbol{x}_i)}{\sum_{j=1}^{k} exp(\boldsymbol{x}_j)} \; for \; i = 1, ..., k$$

is the softmax function to normalize the output. The softmax function gives the probability distribution of each symbol in vocabulary. This function so important that we can see it in almost every NMT model.

2.2 Training RNN

(Pascanu et al., 2013) described the approach to train RNN, as well as analyzed the gradient vanishing and explosion problem. The typical training approach used for RNN is Back Propagation Through Time (BPTT). In training RNN, the loss $J^{(i)}(\theta)$ is computed from the outputs $\hat{\boldsymbol{y}}_i$ in each recursive step, although we may not actually need these intermediate outputs. The final loss is the average of the overall loss. The loss function at each step usually uses cross-entropy. For each step i, the loss is

$$J^{(i)}(\theta) = -\sum_{j=1}^{k} y_j^{(i)} log \; \hat{y}_j^{(i)}$$

for $\boldsymbol{y}^{(i)} \in \mathbb{R}^k$. Then we find their average for the whole sequence

$$J(\theta) = \frac{1}{T}\sum_{i=1}^{T} J^{(i)}(\theta)$$

BPTT finds the gradient for each step by summing up all the products before this step.

$$\frac{\partial J}{\partial \theta} = \sum_{i=1}^{T} \frac{\partial J^{(i)}}{\partial \theta}$$

$$\frac{\partial J^{(i)}}{\partial \theta} = \sum_{k=1}^{i} \frac{\partial J^{(k)}}{\partial \boldsymbol{y}^{(i)}} \frac{\partial \boldsymbol{y}^{(i)}}{\partial \boldsymbol{h}^{(i)}} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{(k)}} \frac{\partial \boldsymbol{h}^{(k)}}{\partial \theta}$$

Here $\frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{(k)}}$ is a product of $i - k$ Jacobian Matrices(Bengio et al., 1994), and can become very small or large quickly, thus raises the vanishing or exploding gradient problem.

2.3 Improving RNN

As discussed in previous section, the gradients in RNN is not stable, can easily blow up or vanish. Several works aimed at solving the problem of the exploding or vanishing gradient in RNN. (Hochreiter and Schmidhuber, 1997) proposed Long Short-Term Memory (LSTM) unit for RNN. LSTM unit is an replacement for the activation function $\sigma(\cdot)$ in (1). A LSTM unit includes

several components with various proposes. For each unit, there is an input gate $\boldsymbol{i}_t$, an forget gate $\boldsymbol{f}_t$, an output gate $\boldsymbol{o}_t$, and a temporary memory cell $\tilde{\boldsymbol{c}}_t$. They are defined as

$$\begin{aligned}\boldsymbol{f}_t &= \sigma\left(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1}\right) \\ \boldsymbol{o}_t &= \sigma\left(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1}\right) \\ \boldsymbol{i}_t &= \sigma\left(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1}\right) \\ \tilde{\boldsymbol{c}}_t &= tanh\left(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1}\right)\end{aligned}$$

Then, the memory cell of LSTM $\boldsymbol{c}_t$ and final hidden state for RNN $\boldsymbol{h_t}$ of this unit is given by

$$\begin{aligned}\boldsymbol{c}_t &= \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \circ tanh(\boldsymbol{c}_t)\end{aligned}$$

where $\circ$ is the Hadamard product of vectors. The power of LSTM unit comes from its capability of selectively forgetting information from previous memory cell $\boldsymbol{c}_{t-1}$ and adopting the content of temporary memory cell $\tilde{\boldsymbol{c}}_t$. $\boldsymbol{f}_t$ and $\boldsymbol{i}_t$ indicate which information are useful in the future, and drop the irrelevant information. $\tilde{\boldsymbol{c}}_t$ provides a nonlinear combination of the input and previous hidden state, thus it can be more flexible and adaptive to learn.

Motivated by LSTM, (Cho et al., 2014) proposed Gated Recurrent Units (GRU), which provides another hidden unit alternative in RNN that selectively forgets and remembers like LSTM. However, GRU is less complicated to implement and compute, compared with LSTM. The architecture of GRU includes a reset gate $\boldsymbol{r}_t$, an update gate $\boldsymbol{z}_t$, and temporary hidden state $\tilde{\boldsymbol{h}}_t$. They are defined by

$$\begin{aligned}\boldsymbol{r}_t &= \sigma\left(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1}\right) \\ \boldsymbol{z}_t &= \sigma\left(\boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1}\right) \\ \tilde{\boldsymbol{h}}_t &= tanh(\boldsymbol{W} \boldsymbol{x}_t + \boldsymbol{r}_t \circ \boldsymbol{h}_{t-1})\end{aligned}$$

and the final hidden state $\boldsymbol{h}_t$ is computed by

$$\boldsymbol{h}_t = \boldsymbol{z}_t \circ \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \circ \tilde{\boldsymbol{h}}_t$$

The architecture of GRU resembles that of LSTM. But a main difference is that the leverage between using new information from $\tilde{\boldsymbol{h}}_t$ and using previous information from $\boldsymbol{h}_{t-1}$ is actually controlled by only one vector $\boldsymbol{z}_t$. This means each GRU unit selects information between current temporary memory cell and previous hidden state to use for future computation. In practice, both LSTM and GRU satisfactorily improved RNN performance.

2.4 Encoder-Decoder and Sequence to Sequence

Finally, after introducing all the preliminaries, we reach the topic of neural machine translation models. Besides GRU, (Cho et al., 2014) also proposed Encoder-Decoder model, which is an important milestone in NMT. The Encoder-Decoder model includes two RNNs. The first RNN encodes the source sentence into a fixed-length intermediate representation vector, and then the second RNN decodes the presentation vector into variable length sequence, which is the sentence in target language. More specifically, we use the final hidden state $\boldsymbol{h}^{(T)}$ of the encoder RNN as the intermediate representation, or here we way, the summary $\boldsymbol{c}$ of the input sequence. Then, we use the decoder RNN to generate the output sentence. However, different from the traditional RNN described above, each activation unit and output probability generate function of decoder RNN take not only the hidden state $\boldsymbol{h}^{(t-1)}$, but also the summary $\boldsymbol{c}$ as the input. Therefore, each hidden state $\boldsymbol{h}^{(t)}$ and word prediction $\hat{\boldsymbol{y}}^{(t)}$ is computed by

$$\begin{aligned}\boldsymbol{h}^{(t)} &= \sigma\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{y}^{(t-1)}, \boldsymbol{c}\right) \\ \boldsymbol{y}^{(t)} &= g\left(\boldsymbol{h}^{(t)}, \boldsymbol{y}^{(t-1)}, \boldsymbol{c}\right)\end{aligned}$$

where $\sigma(\cdot)$ is the activation function that can use LSTM or GRU, and $g(\cdot)$ is the function to generate final probability distribution of the symbol.

However, (Cho et al., 2014) did not use the Encoder-Decoder model to translate sentences. Instead, they used this model to capture linguistic regularities by scoring phrase pairs comes from another phrase-based system, thus enhanced translation quality. After Encoder-Decoder, (Sutskever et al., 2014) proposed sequence to sequence model to generate translated sequence fully based on RNN. This model is similar to Encoder-Decoder that contains two RNNs and a fixed-length middle variable. The difference is sequence to sequence model adopted LSTM for hidden unit in RNNs. They also introduced some tricks in training the model. As (Sutskever et al., 2014) claimed, reversing the input sequence of training and testing set can increase the performance of model output.

2.5 Attention Mechanism

(Bahdanau et al., 2014) introduced attention mechanism in Encoder-Decoder model to enhance its performance. They modified both the encoder

and decoder to add a context vector for indicating which part of the input sequence should current output symbol focus on.

For the encoder model, they used bidirectional RNN (BiRNN). The encoder of BiRNN consists of two RNNs that goes through the input sequence by different order. For each input symbol $\boldsymbol{x}_i$, there are two hidden states, $\overrightarrow{\boldsymbol{h}_i}$ from forward RNN and $\overleftarrow{\boldsymbol{h}_i}$ from backward RNN in the encoder of BiRNN. Then we get an annotation $\boldsymbol{h}_i = \left[\overrightarrow{\boldsymbol{h}_i}^\top, \overleftarrow{\boldsymbol{h}_i}^\top\right]$ for this word. This annotation contains information from both preceding and following words of current word, with its focus placed on current word $\boldsymbol{x}_i$.

For the decoder model, there is a lot difference in generating each symbol $\boldsymbol{y}_i$. First we use an alignment model to compute attention scores at each position for this symbol.

$$e_{ij} = a(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j)$$

where a is a alignment function to compute attention. Typically a can be dot attention $a(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v}$, while some other replacements are also available. In this case, $\boldsymbol{e}_i = \left[\boldsymbol{s}_{i-1}^\top \boldsymbol{h}_1, ..., \boldsymbol{s}_{i-1}^\top \boldsymbol{h}_N\right]$. Note that $\boldsymbol{h}_j$ is the hidden state that we derived above for input word $\boldsymbol{x}_j$ in encoder RNN. And here we use $\boldsymbol{s}_i$ to indicate the hidden state for current output word $\boldsymbol{y}_i$ in the decoder RNN. Then we compute the attention distribution of current target target word $\boldsymbol{y}_i$

$$\boldsymbol{\alpha}_i = softmax(\boldsymbol{e}_i)$$

here α_{ij} indicates the probability of a source word $\boldsymbol{x}_j$ aligns to target word $\boldsymbol{y}_i$. Next, we find the weighted sum of input hidden states $\boldsymbol{h}_i$ as context vector $\boldsymbol{c}_i$

$$\boldsymbol{c}_i = \sum_{j=1}^{N} \alpha_{ij} \boldsymbol{h}_j$$

And finally, in the decoder RNN we compute each hidden state $\boldsymbol{s}_i$ and output word $\boldsymbol{y}_i$ by using this context vector

$$\boldsymbol{s}_i = \sigma\left(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i\right)$$
$$\boldsymbol{y}_i = g\left(\boldsymbol{y}_{i-1}, \boldsymbol{s}_i, \boldsymbol{c}_i\right)$$

The successful application of attention mechanism in NMT improved its performance to a new level. The attention mechanism enables the decoder to focus on certain part of the input sequence, so that achieved better accuracy for generating target sequence. It also provides some interpretability to the NMT model by aligning words from source and target sequences. Visualizing the alignment can give us a sight on which part of the two languages are connected. This alignments also help us to debug the NMT model.

(Luong et al., 2015) further explored more probabilities of attention mechanism. They proposed global and local approaches for computing attention. The global approach always looks at the whole source sequence, while the local approach only concentrate on certain range of the source sequence at a time. They also compared various alignment functions a to find which functions fit which model better. Their experiments confirmed that NMT models with attention involved are superior to models without attention mechanism.

2.6 Transformer

Although RNN successfully pushed the boundaries of machine translation a lot, it has an obvious defect that each hidden state are dependent to the previous one. This step dependency between hidden units makes RNN model difficult to parallelize, which means it is unfriendly to large scale training and deployment. (Vaswani et al., 2017) pointed out this drawback, and proposed transformer model that abandoned RNN and thus fully based on attention mechanism. Similar to previous models, transformer also adopted encoder-decoder structure.

In the description of transformer model, the attention function takes a query Q and a set of key-value pairs K, V as input. All of Q, K, V and the output of the function are vectors. Queries in Q and keys in K are of d_k dimensions, and values in V are of d_v dimensions. We start from defining the scaled dot product as

$$Attention(Q, K, V) = softmax\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Note that if we remove the scale factor $\frac{1}{\sqrt{d_k}}$, it becomes the dot-product we defined at 2.5. By scaling, we can avoid the case that some large value in the softmax function makes its result peaked. Then, we define the multi-head attention by mapping Q, K, V into h different lower dimensional spaces with learned linear projection W_i^Q, W_i^K, W_i^V, for $i = 1, ..., h$. Next, we compute attention function in parallel for these

mapped queries, keys and values, concatenate them, and map them back to original space.

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

here $W_i^Q, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v} \times d_{model}$ are learned linear mappings. Usually, there is a useful feed-forward network after the output of the multi-head attention

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2$$

With all these setups, we can now describe the transformer model. Both encoder and decoder are composed from 6 identical layer. In each layer of the encoder, first we compute a multi-head attention that the queries, keys and values are all from the input of the layer, and we connect the feed-forward network to its output. In each layer of the decoder, first we compute the multi-head attention in the same way as encoder, taking the output from previous layer as input. Then, we add another multi-head attention, which takes the output of encoder as keys and values, and takes the output of previous attention function as queries. Finally there is also a feed-forward network connected after the last attention function, gives the output of this layer.

As the paper stated, transformer model reduced total computational complexity, but improved the amount of computation that can be parallelized. Transformer model is also better in learning long-range dependencies, since it provides shorter paths to forward and backward dependency signals.

3 Future Directions of NMT

Based on the significant achievements discussed in previous section, in this section we introduce some state-of-the-art results of NMT, and analyze the future research potentials behind them.

3.1 Fusion of Models

A recent work (Chen et al., 2018) compared these classic models, and discovered the design space of building hybrid models from them. They attempted to combine elements from different models at various levels. Consider the transformer model, note that both encoder and decoder contains 6 layer of the same structure. For each layer in encoder or decoder, we can assign different learning component, such as RNN or attention unit. At the encoder level, we can assemble different kinds of layers, in different number and different way to arrange (for example, sequential or parallel). At the model level, we can combine encoders and decoders from different models. Their experiments show that hybrid models can achieve better performance compared to baseline models.

(Domhan, 2018) also experimented various ways to combine existed models. They showed that self-attention mechanism in transformer model is more important in the encoder than in decoder, so replacing decoder with other model such as RNN decoder does not cause much performance loss. This work and previous work show that it is possible to search the space of combining existed model components to design better model architecture. In future, we can look for more ways to arrange these components, and provide connection mechanism to them.

3.2 Syntax-Awareness

Most of the previously introduced NMT models are completely based on statistical inference from a linear sequential representation, while ignoring linguistic features, such as syntax or semantics. Sequential RNN can frequently generate translation with syntax error(Chen et al., 2017a), especially can violate long-term syntactic information and generate incorrect translation(Wu et al., 2017). Thus, taking linguistic constraints into consideration is a reasonable way to improve NMT model.

(Tai et al., 2015) tried to modify the linear chain representation of LSTM units in RNN, proposed tree-LSTM networks that organizes LSTM units in a tree structure. They used tree-LSTM model constructed from parse tree of sentences to compare the similarity between sentence pairs. Although this work did not attempt machine translation task, it showed the possibility of adding syntactic information to RNN structures.

(Eriguchi et al., 2016) extended the tree-LSTM model to assist the encoder part of sequence to sequence model. Their model involves two intermediate variables. First they employs sequential LSTM units for a sentence to generate a medium vector, just like RNN in traditional sequence to sequence model. Then, the hidden states in the sequential LSTM are used as the leaf nodes in

the LSTM tree. The model constructs another binary tree-structured LSTM units in accordance to the bottom-up phrase structure grammar of the sentence, and this gives another intermediate vector. The decoder component derived its initial hidden state by combining the two medium vectors together. This model also adopted attention mechanism, in both sequential hidden units and tree-structured hidden units. Following this work, (Chen et al., 2017a) proposed another encoder-decoder model that employs tree-structure in encoder part. This model adopts bi-directional GRU unit to replace LSTM unit, and used phrase structure trees instead of head-drive phrase structure grammar (HPSG) trees. (Li et al., 2017) summarized three different ways to incorporate source syntax. They are parallel RNN encoder that uses one RNN for word and another for syntactic structure, hierarchical RNN encoder that contain two connected layers of RNNs with one for word and another for structural label, mixed RNN encoder that stitches words and structural label together. Their experiments showed the mixed RNN achieved best performance.

(Wu et al., 2017) proposed sequence to dependency model, of which the encoder is similar to RNN, but the decoder jointly generates both target translation and its dependency parse tree, with two differently trained RNNs, Words-RNN and Action-RNN. In each decoding step, the model first generates an action from the Action-RNN, and only generate a target word if the generated action is shift (i.e. the Action-RNN is traversing a node in the dependency tree, not an edge).

These works show that involving grammar in NMT can improve its accuracy and make the output more suitable to human language syntax. In future, we can expect more ways to involve linguistic knowledge in NMT.

3.3 Insufficient Corpus

Just like most other deep learning models, NMT models require large-scale parallel corpus to train. However, sometimes the existed corpus may not be sufficient for the model to converge. So here raises the problem of, how to train NMT model with less parallel sentence pairs.

An approachable way to tackle this problem is using a third language as a pivot to train translation model. The most naive idea is to first translate source language to pivot language, then translate pivot language to target language. However, this idea may not work satisfactorily since there are two stages of information loss. (Chen et al., 2017b) proposed a method to train a source-to-target translation model based on assumption that there are enough parallel corpus for source-to-pivot translation, and an existed model is available for pivot-to-target translation. Their method is using the pivot-to-target model as a "teacher" to teach the source-to-target model with source-to-pivot sentence pairs. (Ren et al., 2018) proposed a triangular architecture to solve the rare corpus problem. Their method enriches the target low-resource language pair with another rich-resource language pair using EM algorithm. All these two methods solved rare corpus problem to some extent.

3.4 Robustness

Another common issue of deep learning models are instability, and NMT suffers it as well. Usually the output of NMT is not stable enough, even small perturbation or grammar error can impact the translation result greatly. (Cheng et al., 2018) tried to use adversarial network to improve the robustness of NMT. Adversarial model has previously been successfully applied in computer vision to synthesize images. In this paper, they trained their model using original sentence and a small noisily perturbed sentence together. The intermediate representations of original sentence and the perturbed counterpart from encoder are then both sent into a discriminator, to ensure the difference between these two intermediate presentations are similar. Their model succeeded to improve both translation performance and stability. In future, we hope to see models that can both improve NMT performance and robustness, make it more reliable for industry level deployment.

4 Conclusion

Many deep neural network models in various types have successfully pushed the boundary of machine translation performance. We witnessed several neural machine translation models that refreshed the state-of-the-art performance of machine translation, including Recurrent Neural Network, attention mechanism and Transformer model. In the future, better performance can might be achieved by integrating existed models, or adding linguistic information to the model. In spite of translation

performance, we also need to concern about the stability of NMT models, and the case of corpus shortage.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017a. Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1936–1945. Association for Computational Linguistics.

Yun Chen, Yang Liu, Yong Cheng, and Victor O.K. Li. 2017b. A teacher-student framework for zero-resource neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1925–1935. Association for Computational Linguistics.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86. Association for Computational Linguistics.

Yong Cheng, Zhaopeng Tu, Fandong Meng, Junjie Zhai, and Yang Liu. 2018. Towards robust neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1756–1766. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics.

Tobias Domhan. 2018. How much attention do you need? a granular analysis of neural machine translation architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1799–1808. Association for Computational Linguistics.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 823–833. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. 2017. Modeling source syntax for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–697. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. 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.

Shuo Ren, Wenhu Chen, Shujie Liu, Mu Li, Ming Zhou, and Shuai Ma. 2018. Triangular architecture for rare language translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 56–65. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on*

Natural Language Processing (Volume 1: Long Papers), pages 1556–1566. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Shuangzhi Wu, Dongdong Zhang, Nan Yang, Mu Li, and Ming Zhou. 2017. Sequence-to-dependency neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 698–707. Association for Computational Linguistics.