Improving Neural Machine Translation with Neural Syntactic Distance

Chunpeng Ma[1,3], **Akihiro Tamura**[2], **Masao Utiyama**[3], **Tiejun Zhao**[1*], **Eiichiro Sumita**[3]
[1]Harbin Institute of Technology, Harbin, China
[2]Ehime University, Matsuyama, Japan
[3]National Institute of Information and Communications Technology, Kyoto, Japan
`{cpma, tjzhao}@hit.edu.cn` `tamura@cs.ehime-u.ac.jp`
`{mutiyama, eiichiro.sumita}@nict.go.jp`

Abstract

The explicit use of syntactic information has been proved useful for neural machine translation (NMT). However, previous methods resort to either tree-structured neural networks or long linearized sequences, both of which are inefficient. Neural syntactic distance (NSD) enables us to represent a constituent tree using a sequence whose length is identical to the number of words in the sentence. NSD has been used for constituent parsing, but not in machine translation. We propose five strategies to improve NMT with NSD. Experiments show that it is not trivial to improve NMT with NSD; however, the proposed strategies are shown to improve translation performance of the baseline model (+2.1 (En–Ja), +1.3 (Ja–En), +1.2 (En–Ch), and +1.0 (Ch–En) BLEU).

1 Introduction

In recent years, neural machine translation (NMT) has been developing rapidly and has become the de facto approach for machine translation. To improve the performance of the conventional NMT models (Sutskever et al., 2014; Bahdanau et al., 2014), one effective approach is to incorporate syntactic information into the encoder and/or decoder of the baseline model.

Based on how the syntactic information is represented, there are two categories of syntactic NMT methods: (1) those that use tree-structured neural networks (NNs) to represent syntax structures (Eriguchi et al., 2016; Hashimoto and Tsuruoka, 2017), and (2) those that use linear-structured NNs to represent linearized syntax structures (Li et al., 2017; Ma et al., 2017, 2018). For the first category, there is a direct corresponding relationship between the syntactic structure and the NN structure, but the complexity of NN structures usually makes training inefficient. In contrast, for the second category, syntactic structures are linearized and represented using linear-structured recurrent neural networks (RNNs), but the linearized sequence can generally be quite long and therefore training efficiency is still a problem. Although using a shorter sequence may improve the efficiency, some syntactic information is lost.

We propose a method of using syntactic information in NMT that overcomes the disadvantages of both methods. The basis of our method is the neural syntactic distance (NSD), a recently proposed concept used for constituent parsing (Shen et al., 2018; Gómez-Rodríguez and Vilares, 2018). NSD makes it possible to represent a constituent tree as a sequence whose length is identical to the number of words in the sentence (almost) without losing syntactic information. However, there are no previous studies that use NSD in NMT. Moreover, as demonstrated by our experiments, using NSD in NMT is far from straightforward, so we propose five strategies and verify the effects empirically. The strategies are summarized below.

- Extend NSD to dependency trees, which is inspired by the dependency language model (Shen et al., 2010).
- Use NSDs as input sequences[1], where an NSD is regarded as a linguistic input feature (Sennrich and Haddow, 2016).
- Use NSDs as output sequences, where the NMT and prediction of the NSD are simultaneously trained through multi-task learning (Firat et al., 2016).
- Use NSD as positional encoding (PE), which is a syntactic extension of the PE of the Transformer (Vaswani et al., 2017).

[*]Corresponding author

[1]Throughout this paper, ”input” means the input of an encoder or a decoder rather than the input of the NMT model (i.e., only source sentences), and ”output” is similar.

- Add a loss function for NSD to achieve distance-aware training (Shen et al., 2018).

2 Neural Syntactic Distance (NSD)

The NSD was firstly proposed by Shen et al. (2018). This is the first method of linearizing a constituent tree with a sequence of length n, without loss of information, where n is the number of words in the sentence.

Formally, given the sentence $\mathbf{w} = (w_1, \ldots, w_n)$, for any pairs of contiguous words (w_i, w_{i+1}), we can define an NSD $d(w_i)$,[2] where $i \in [1, n-1]$. In Shen et al. (2018), the NSD $d_S(w_i)$ is defined as the height of the lowest common ancestor (LCA) of the words.[3] Subsequently, in Gómez-Rodríguez and Vilares (2018), the NSD $d_G(w_i)$ was defined as the number of the common ancestors of the words. To make the definition complete, we define $d(w_n)$ as follows:[4]

$$d_S(w_n) = H, \quad d_G(w_n) = 0, \tag{1}$$

where H is the height of the constituent tree. It is easy to prove that

$$d_S(w_i) + d_G(w_i) = H, \quad i \in [1, n]. \tag{2}$$

We call d_S and d_G the *absolute* NSD.

Furthermore, Gómez-Rodríguez and Vilares (2018) define the *relative* NSD as follows:

$$d_R(w_i) = \begin{cases} d_G(w_1), & i = 1, \\ d_G(w_i) - d_G(w_{i-1}), & i \in [2, n]. \end{cases} \tag{3}$$

Figure 1 illustrates these NSDs. It is easy to see the one-to-one correspondence relationship between the constituent tree and the (absolute or relative) NSDs.

The effectiveness of all different NSDs has been proven on constituent parsing. However, there has been no attempt to use NSD in machine translation.

[2]Note that NSD is defined between two contiguous words. For convenience of notation, we use $d(w_i)$ rather than $d(w_i, w_{i+1})$ to denote an NSD.

[3]In Shen et al. (2018), NSD is defined as a real number that is a function of LCA. However, in practice, NSD is simply identical to the depth of the LCA.

[4]$d_S(w_n)$ and $d_G(w_n)$ are undefined in both of the original papers. We give the definitions here to enable the use of NSD in NMT later.

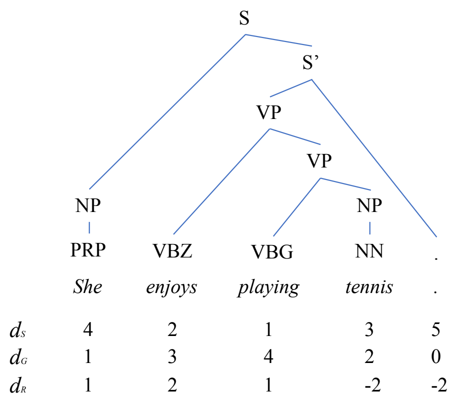

Figure 1: Example of different NSDs. This example is from Shen et al. (2018).

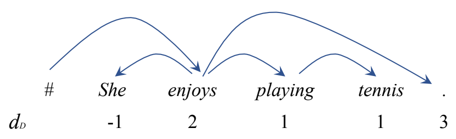

Figure 2: Example of dependency NSDs. "#" is the root. Dependency labels are omitted.

3 Strategies to improve NMT with NSD

3.1 Dependency NSD

There are many previous studies on using dependency trees to improve NMT (Nguyen Le et al., 2017; Wu et al., 2017). Therefore, we extend NSD to dependency trees. Formally, the dependency NSD between two nodes is defined as follows:

$$d_D(w_i) = i - h(i), \tag{4}$$

where $h(i)$ is the index of the head of w_i, and we let the index of root be 0. Note that $d_D(w_i)$ can be either positive or negative, representing the directional information. Figure 2 gives an example.

3.2 NSDs as Input Sequences

It is easy to see that for $\mathbf{w} = (w_1, \ldots, w_n)$, the lengths of d_S, d_G, d_R and d_D are all n. Denoting the NSD sequence as $\mathbf{d} = (d_1, \ldots, d_n)$, we can see that $d_i \in \mathbb{Z}, i \in [1, n]$, so we can obtain a sequence of embedding vectors $\mathbf{e}^d = (e_1^d, \ldots, e_n^d)$ as follows:

$$e_i^d = E_d[d_i^d + (\max(\mathbf{d}) - \min(\mathbf{d}) + 1)]. \tag{5}$$

We call E_d the distance embedding matrix and call $\mathbf{e}^d$ the syntactic embedding sequence. Note that $\mathbf{d}$ can be the NSD on either the source side or the target side, so there are two possible E_d, which are denoted as E_d^s and E_d^t, respectively. The embeddings are calculated as follows:

$$x_i^s = f_{emb}(E_w^s[w_i^s], e_i^{ds}), \tag{6}$$
$$x_i^t = f_{emb}(E_w^t[w_i^t], e_i^{dt}), \tag{7}$$

where e_i^{ds} and e_i^{dt} are defined in Eq. 5 on the source side and target side, respectively, and E_w^s and E_w^t are the word embedding matrices on both sides, respectively. Inspired by Sennrich and Haddow (2016), function f_{emb} is used to combine two vectors. This function has many different options, such as:

$$f_{emb}^{\|}(x, e) = x \| e, \tag{8}$$
$$f_{emb}^{+}(x, e) = x + e, \tag{9}$$
$$f_{emb}^{Wb}(x, e) = W_f \cdot (x \| e) + b_f, \tag{10}$$

where $x, e, b_f \in \mathbb{R}^d$ and $W_f \in \mathbb{R}^{d \times 2d}$. The operator "$\|$" is the concatenation of two vectors.

When NSD is used as the input sequence on the target side, there is one problem: $\mathbf{e}^{dt}$ is unknown during testing. For this case, we use NSDs for both the input and output sequences, let the decoder predict NSD on-the-fly using the strategy introduced in Section 3.3, and use the predicted NSD to calculate $\mathbf{e}^{dt}$.

3.3 NSDs as Output Sequences

An NSD can be used to form the output sequence to improve NMT using the idea of multi-task learning. Specifically, we train the model to predict the NSD sequence. When NSD is used as the output sequence of the *encoder*, we minimize the distance (e.g., cross entropy $\mathcal{L}_{dist}^{ent}$, see Section 3.5 for details) between the predicted and the golden NSD sequences. When NSD is used as the output sequence of the *decoder*, besides minimizing the distance, we use the predicted NSD as the input of the next time step.

Denote the hidden vector as $\mathbf{h} = (h_1, \ldots, h_n)$. For the encoder, $h_i = h_i^s$ and $n = n_s$, while for the decoder, $h_i = h_i^t$ and n is the current time step of decoding. Then, we can obtain a sequence of predicted syntactic distance $\hat{\mathbf{d}} = (\hat{d}_1, \ldots, \hat{d}_n)$, which is calculated as follows:

$$p(\hat{d}_i \mid h_i) = \mathrm{softmax}(W_d \cdot h_i + b_d), \tag{11}$$

where W_d and b_d are parameters to be learned. By minimizing the distance between $\hat{d}_i$ and d_i, NSD can be used to enhance NMT.

3.4 NSD as Positional Encoding (PE)

PE is used by the Transformer (Vaswani et al., 2017) to encode the positions of words. Formally, it is defined as follows:

$$x_i' = x_i + PE(i), \tag{12}$$
$$PE(i)_{2k} = \sin(i/10000^{2k/d}), \tag{13}$$
$$PE(i)_{2k+1} = \cos(i/10000^{2k/d}), \tag{14}$$

where x_i can be either x_i^s or x_i^t, and d is the dimension of the embedding vector. Similarly, we define syntactic PE as follows:

$$PE(i)_{2k} = \sin\left(\frac{i + \max(\mathbf{d}) - \min(\mathbf{d})}{{\lambda_{SPE}}^{2k/d}} \cdot 2\pi\right), \tag{15}$$

$$PE(i)_{2k+1} = \cos\left(\frac{i + \max(\mathbf{d}) - \min(\mathbf{d})}{{\lambda_{SPE}}^{2k/d}} \cdot 2\pi\right), \tag{16}$$

where λ_{SPE} is a hyperparameter to be tuned. In this way, the periods of these two functions vary from 1 to λ_{SPE}. We define syntactic PE in this way because (1) according to a quantitative analysis of the experimental datasets, we found that the ranges of possible values are quite different between NSD and word positions, so we tuned λ_{SPE} instead of fixed it to 10000 as in Eqs. 13 and 14, and (2) d_i may be negative, so we adjust it to be positive.

3.5 Distance-aware Training

Instead of using conventional cross-entropy loss function during training, we use the following loss function to make the NMT model learn NSD better:

$$\mathcal{L} = \mathcal{L}_{NMT} + \mathcal{L}_{dist} + \mathcal{L}_{dist}^{ent}. \tag{17}$$

The first item is the cross-entropy loss of the NMT model, which is

$$\mathcal{L}_{NMT} = - \sum_{\langle \mathbf{w}^s, \mathbf{w}^t \rangle \in \mathcal{D}} \log p(\mathbf{w}^t \mid \mathbf{w}^s), \tag{18}$$

where $\mathcal{D}$ is the training dataset. The second item is the distance-aware loss, which is inspired by Shen et al. (2018) and is as follows:

$$\mathcal{L}_{dist} = \sum_{\langle \mathbf{w}^s, \mathbf{w}^t \rangle \in \mathcal{D}} (\mathcal{L}_{dist}^s(\mathbf{w}^s) + \mathcal{L}_{dist}^t(\mathbf{w}^t)),$$

$$\mathcal{L}_{dist}^s(\mathbf{w}^s) = \sum_{i=1}^{n_s} (d_i - \hat{d}_i)^2 + \sum_{i, j > i} [1 - \mathrm{sign}(d_i - d_j)(\hat{d}_i - \hat{d}_j)]^+, \tag{19}$$

and $\mathcal{L}^{t}_{dist}$ can be defined similarly. The third item is the cross-entropy loss for NSD, which is as follows:

$$\mathcal{L}^{ent}_{dist} = \sum_{\langle \mathbf{w}^s, \mathbf{w}^t \rangle \in \mathcal{D}} (\mathcal{L}^{ent(s)}_{dist}(\mathbf{w}^s) + \mathcal{L}^{ent(t)}_{dist}(\mathbf{w}^t)),$$
$$\mathcal{L}^{ent(s)}_{dist}(\mathbf{w}^s) = - \sum_{d_i \in \mathbf{d}^s} p(d_i \mid h_i) \log p(\hat{d}_i \mid h_i), \tag{20}$$

and $\mathcal{L}^{ent(t)}_{dist}$ can be defined similarly.

4 Experiments

4.1 Configuration

We experimented on two corpora: (1) ASPEC (Nakazawa et al., 2016), using the top 100K sentence pairs for training En–Ja models and top 1M sentence pairs for training Ja–En models, and (2) LDC,[5] which contains about 1.2M sentence pairs, for training En–Ch and Ch–En models. To tackling the problem of memory consumption, sentences longer than 150 were filtered out, so that models can be trained successfully. Chinese sentences were segmented by the Stanford segmentation tool.[6] For Japanese sentences, we followed the preprocessing steps recommended in WAT 2017.[7]

The test set is a concatenation of NIST MT 2003, 2004, and 2005. Constituent trees are generated by the parser of Kitaev and Klein (2018)[8], and dependency trees are generated by the parser of Dyer et al. (2015)[9]. Note that although we only used syntactic information of English in our experiments, our method is also applicable to other languages.

We implemented our method on `OpenNMT`[10] (Klein et al., 2017), and used the Transformer as our baseline. As far as we know, there are no previous studies on using syntactic informations in the Transformer.

The vocabulary sizes for all languages are $50,000$. Both the encoder and decoder have 6 layers. The dimensions of hidden vectors and word embeddings are 512. The multi-head attention has 8 heads, and the dropout probability is 0.1 (Srivastava et al., 2014). The number of training epochs was fixed to 50, and we used the model which performs the best on the development set for testing.

As for optimization, we used the Adam optimizer (Kingma and Ba, 2014), with $\beta_1 = 0.9, \beta_2 = 0.998$, and $\epsilon = 10^{-9}$. Warmup and decay strategy for learning rate of Vaswani et al. (2017) are also used, with $8,000$ warmup steps. We also used the label smoothing strategy (Szegedy et al., 2016) with $\epsilon_{ls} = 0.1$.

4.2 Experimental Results

Table 1 compares the effects of the strategies. We evaluate the proposed strategies using character-level BLEU (Papineni et al., 2002) for Chinese and Japanese, and case-insensitive BLEU for English.

Comparison of different NSDs. The first five rows of Table 1 compare the results of using different NSDs. When NSD was used at the source side (En–Ja/En–Ch), all kinds of NSDs improved translation performance. This indicates that NSD can be regarded as a useful linguistic feature to improve NMT. In contrast, when NSD was used at the target side (Ja–En/Ch–En), d_S and d_G hurt the performance. This is because the values of d_S and d_G are volatile. A tiny change of syntactic structure often causes a big change of d_S and d_G. Since the model has to predict the NSD during decoding, once there is one error, the subsequent predictions will be heavily influenced. The use of d_R and d_D remedies this problem. Furthermore, the effects of d_S and d_G are similar, because they are equivalent in nature (refer to Eq. 2).

NSD as PE. Rows 5 to 8 of Table 1 evaluate the use of dependency NSD (d_D) as syntactic PE. Note that for all the experiments, we used not only the syntactic PE but the conventional PE. Experiment results show that this strategy is indeed useful. When the dominators of Eqs. 15 and 16, λ_{SPE}, were set to 10^4, there was no improvement. When they were set to 40, the improvement was remarkable. This indicates that our design of syntactic PE is reasonable.

NSD as input/output and source/target sequences. Rows 8 to 12 of Table 1 are the results of using dependency NSD (i.e., d_D) as the input and/or output sequences on both sides. First, for the choice of f_{emb}, we can see that $f^{||}_{emb}$ and f^{+}_{emb} are similar, while f^{Wb}_{emb} yields better performance. This is because the model has to learn W_f and

[5]LDC2002E18, LDC2003E07, LDC2003E14, Hansards portion of LDC2004T07, LDC2004T08, and LDC2005T06.

[6]`https://nlp.stanford.edu/software/stanford-segmenter-2017-06-09.zip`

[7]`http://lotus.kuee.kyoto-u.ac.jp/WAT/WAT2017/baseline/dataPreparationJE.html`

[8]`https://github.com/nikitakit/self-attentive-parser`

[9]`https://github.com/clab/lstm-parser`

[10]`http://opennmt.net`

	Type	I/O	SPE	Loss	En–Ja	Ja–En	En–Ch	Ch–En
1	N/A	N/A	No	Eq. 18	34.59	26.43	29.41	31.60
2	d_S	I(f_{emb}^{Wb})	No	Eq. 18,19,20	35.54	24.57	29.66	28.77
3	d_G	I(f_{emb}^{Wb})	No	Eq. 18,19,20	35.38	24.71	29.60	28.82
4	d_R	I(f_{emb}^{Wb})	No	Eq. 18,19,20	35.83	26.88	29.87	31.82
5	d_D	I(f_{emb}^{Wb})	No	Eq. 18,19,20	36.17	27.21	30.11	32.08
6	d_D	I(f_{emb}^{Wb})	10^4	Eq. 18,19,20	36.06	27.18	30.02	32.03
7	d_D	I(f_{emb}^{Wb})	10^2	Eq. 18,19,20	36.22	27.47	30.23	32.19
8	d_D	I(f_{emb}^{Wb})	40	Eq. 18,19,20	36.44	27.59	**30.59**	32.36
9	d_D	I($f_{emb}^{\|\|}$)	40	Eq. 18,19,20	36.17	27.21	30.15	32.11
10	d_D	I(f_{emb}^{+})	40	Eq. 18,19,20	36.08	27.32	30.21	32.29
11	d_D	O	40	Eq. 18,19,20	36.31	27.42	30.42	32.32
12	d_D	I(f_{emb}^{Wb})&O	40	Eq. 18,19,20	**36.69**	**27.71**	30.56	**32.55**
13	d_D	O	40	Eq. 18	21.08	10.22	18.63	15.61
14	d_D	O	40	Eq. 18,20	33.70	23.31	27.43	30.02
15	d_D	O	40	Eq. 18,19	34.18	25.19	29.14	31.74

Table 1: Comparison of strategies. I/O: use NSDs as the input or output sequences. Functions $f_{emb}^{||}$, f_{emb}^{+}, and f_{emb}^{Wb} are defined in Eqs. 8 to 10, respectively. SPE: use NSD as syntactic PE. Numbers are the values of λ_{SPE} in Eqs. 15 and 16. Loss: items used in the final loss function. Note that when NSD is used as the input sequence of the source language, $\mathcal{L}_{dist} + \mathcal{L}_{dist}^{ent} \equiv 0$, because the parsing tree is fixed.

b_f, which increases the model capacity. Second, performance improved for using NSDs both as input and output sequences, and combining both obtained further improvement. Third, NSDs improved the performance both on the source and the target sides. All these results indicate the robustness of NSDs.

Effects of distance-aware training. The last three rows compare the different effects of the items in the loss function. When only $\mathcal{L}_{NMT}$ are used, the performance is extremely poor. This is within expectations, because with only $\mathcal{L}_{NMT}$, weights related to NSDs are kept to the initial values and were not updated, and hence detrimental to learning. Adding $\mathcal{L}_{dist}^{ent}$ improves the results significantly, but the improvement is lower than that of $\mathcal{L}_{dist}$. This is because training with $\mathcal{L}_{dist}^{ent}$ treats different values of NSDs equally, while $\mathcal{L}_{dist}$ penalizes larger differences between the predicted NSD and the golden NSD more severely.

5 Conclusion

We proposed five strategies to improve NMT with NSD. We found relative NSDs and dependency NSDs are able to improve the performance consistently, while absolute NSDs hurt the performance for some cases. The improvement obtained by using NSDs is general in that NSDs can be used at both the source side and target side, both as input sequences and output sequences. Using NSDs as syntactic PE is also useful, and training with a distance-aware loss function is quite important.

Acknowledgements

We are grateful to the anonymous reviewers for their insightful comments and suggestions. Akihiro Tamura is supported by JSPS KAKENHI Grant Number JP18K18110. Tiejun Zhao is supported by the National Key Research and Development Program of China via grant 2017YFB1002102.

References

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

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. 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 334–343.

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.

Orhan Firat, Baskaran Sankaran, Yaser Al-Onaizan, Fatos T. Yarman Vural, and Kyunghyun Cho. 2016. Zero-resource translation with multi-lingual neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 268–277.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324.

Kazuma Hashimoto and Yoshimasa Tsuruoka. 2017. Neural machine translation with source-side latent graph parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 125–135.

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

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *arXiv preprint arXiv:1701.02810*.

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.

Chunpeng Ma, Lemao Liu, Akihiro Tamura, Tiejun Zhao, and Eiichiro Sumita. 2017. Deterministic attention for sequence-to-sequence constituent parsing. In *Proc. of AAAI-2017*, pages 3237–3243.

Chunpeng Ma, Akihiro Tamura, Masao Utiyama, Tiejun Zhao, and Eiichiro Sumita. 2018. Forest-based neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1253–1263.

Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara. 2016. ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 2204–2208.

An Nguyen Le, Ander Martinez, Akifumi Yoshimoto, and Yuji Matsumoto. 2017. Improving sequence to sequence neural machine translation by utilizing syntactic dependency information. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 21–29.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

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.

Libin Shen, Jinxi Xu, and Ralph Weischedel. 2010. String-to-dependency statistical machine translation. *Computational Linguistics*, 36(4).

Yikang Shen, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio. 2018. Straight to the tree: Constituency parsing with neural syntactic distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1180.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

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.