Higher-order Syntactic Attention Network for Long Sentence Compression

Hidetaka Kamigaito[♡], **Katsuhiko Hayashi**[♢], **Tsutomu Hirao**[♠] **and Masaaki Nagata**[♠]

[♡] Institute of Innovative Research, Tokyo Institute of Technology
[♢] Institute of Scientific and Industrial Research, Osaka University
[♠] NTT Communication Science Laboratories, NTT Corporation

kamigaito@lr.pi.titech.ac.jp, katsuhiko-h@sanken.osaka-u.ac.jp,
{hirao.tsutomu,nagata.masaaki}@lab.ntt.co.jp

Abstract

Sentence compression methods based on LSTM can generate fluent compressed sentences. However, the performance of these methods is significantly degraded when compressing long sentences since it does not explicitly handle syntactic features. To solve this problem, we propose a higher-order syntactic attention network (HiSAN) that can handle higher-order dependency features as an attention distribution on LSTM hidden states. Furthermore, to avoid the influence of incorrect parse results, we train HiSAN by maximizing the probability of a correct output together with the attention distribution. Experiments on the Google sentence compression dataset show that our method achieved the best performance in terms of F_1 as well as ROUGE-1,2 and L scores, 83.2, 82.9, 75.8 and 82.7, respectively. In subjective evaluations, HiSAN outperformed baseline methods in both readability and informativeness.

1 Introduction

Sentence compression is the task of compressing long sentences into short and concise ones by deleting words. To generate compressed sentences that are grammatical, many researchers (Jing, 2000; Knight and Marcu, 2000; Berg-Kirkpatrick et al., 2011; Filippova and Altun, 2013) have adopted tree trimming methods. Even though Filippova and Altun (2013) reported the best results on this task, automatic parse errors greatly degrade the performances of these tree trimming methods.

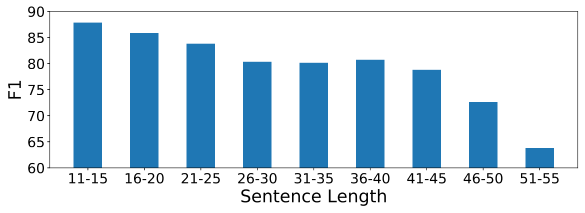


Figure 1: F_1 scores of LSTM-based sentence compression method for each sentence length.[1]

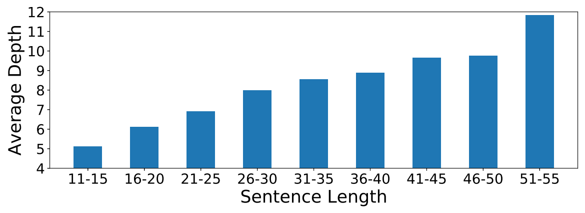


Figure 2: Average tree depths for each sentence length.[2]

Recently, Filippova et al. (2015) proposed an LSTM sequence-to-sequence (Seq2Seq) based sentence compression method that can generate fluent sentences without utilizing any syntactic features. Therefore, Seq2Seq based sentence compression is a promising alternative to tree trimming.

However, as reported for a machine translation task (Cho et al., 2014; Pouget-Abadie et al., 2014; Koehn and Knowles, 2017), the longer the input sentences are, the worse the Seq2Seq performances become. We also observed this problem in the sentence compression task. As shown in Figure 1, the performance of Seq2Seq is degraded when compressing long sentences. In particular, the performance significantly falls if sentence length exceeds 26 words. This is an important problem, because sentences longer than the average sentence length (=28 words) accounts for 42% of the Google sentence compression dataset.

As shown in Figure 2, long sentences have deep

[1]We used an LSTM-based sentence compression method (Filippova et al., 2015) in the evaluation setting as described in Section 4.1.

[2]We treat the maximum distance from root node to the leaf node as dependency tree depth.

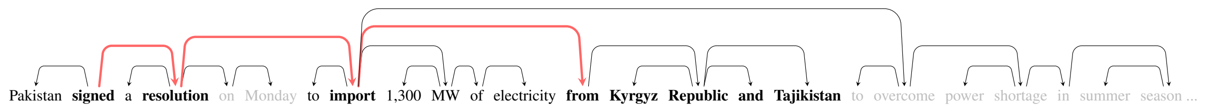

Pakistan signed a resolution on Monday to import 1,300 MW of electricity from Kyrgyz Republic and Tajikistan to overcome power shortage in summer season ...

Figure 3: An example compressed sentence and its dependency tree. The words colored by gray represent deleted words.

dependency trees, which have long distances from root node to words at leaf nodes. Therefore, improving compression performance for sentences with such deep dependency trees can help to compress longer sentences.

To deal with sentences that have deep dependency trees, we focus on the chains of dependency relationships. Figure 3 shows an example of a compressed sentence with its dependency tree. The topic of this sentence is *import agreement related to electricity*. Thus, to generate informative compression, the compressed sentence must retain the *country name*. In this example, the compressed sentence should keep the phrase "from Kyrgyz Republic and Tajikistan". Thus, the compressed sentence must also keep the dependency chain "import", "resolution" and "signed" because the phrase is a child of this chain. By considering such higher-order dependency chains, the system can implement informative compression. As can be seen from the example in Figure 3, tracking a higher-order dependency chain for each word would help to compress long sentences. This paper refers to such dependency relationships by the expression "d-length dependency chains".

To handle a d-length dependency chain for sentence compression with LSTM, we propose the higher-order syntactic attention network (HiSAN). HiSAN computes the deletion probability for a given word based on the d-length dependency chain starting from the word. The d-length dependency chain is represented as an attention distribution, learned using automatic parse trees. To alleviate the influence of parse errors in automatic parse trees, we learn the attention distribution together with deletion probability.

Evaluation results on the Google sentence compression dataset (Filippova and Altun, 2013) show that HiSAN achieved the best F_1, ROUGE-1,2 and L scores 83.2, 82.9, 75.8 and 82.7, respectively. In particular, HiSAN attained remarkable compression performance with long sentences. In human evaluations, HiSAN also outperformed the baseline methods.

2 Baseline Sequence-to-Sequence Method

Sentence compression can be regarded as a tagging task, where given a sequence of input tokens $\mathbf{x} = (x_0, ..., x_n)$, a system assigns output label y_t, which is one of three types of specific labels ("keep,""delete," or"end of sentence") to each input token x_t $(1 \leq t \leq n)$.

The LSTM-based approaches for sentence compression are mostly based on the bi-LSTM based tagging method (Tagger) (Klerke et al., 2016; Wang et al., 2017; Chen and Pan, 2017) or Seq2Seq (Filippova et al., 2015; Tran et al., 2016). Tagger independently predicts labels in a point estimation manner, whereas Seq2Seq predicts labels by considering previously predicted labels. Since Seq2Seq is more expressive than Tagger, we built HiSAN on the baseline Seq2Seq model.

Our baseline Seq2Seq is a version of Filippova et al. (2015) extended through the addition of bi-LSTM, an input feeding approach (Vinyals et al., 2015; Luong et al., 2015), and a monotonic hard attention method (Yao and Zweig, 2015; Tran et al., 2016). As described in the evaluations section, this baseline achieved comparable or even better scores than the state-of-the-art scores reported in Filippova et al. (2015). The baseline Seq2Seq model consists of embedding, encoder, decoder, and output layers.

In the embedding layer, the input tokens $\mathbf{x}$ are converted to the embeddings $\mathbf{e}$. As reported in Wang et al. (2017), syntactic features are important for learning a generalizable embedding for sentence compression. Following their results, we also introduce syntactic features into the embedding layer. Specifically, we combine the surface token embedding w_i, POS embedding p_i, and dependency relation label embedding r_i into a single vector as follows:

$$e_i = [w_i, p_i, r_i], \quad (1)$$

where $[]$ represents vector concatenation, and e_i is an embedding of token x_i.

The encoder layer converts the embedding $\mathbf{e}$ into a sequence of hidden states $\mathbf{h} = (h_0, ..., h_n)$

using a stacked bidirectional-LSTM (bi-LSTM) as follows:

$$h_i = \left[\overrightarrow{h_i}, \overleftarrow{h_i}\right] \quad (2)$$

$$\overrightarrow{h}_i = LSTM_{\overrightarrow{\theta}}(\overrightarrow{h}_{i-1}, e_i) \quad (3)$$

$$\overleftarrow{h}_i = LSTM_{\overleftarrow{\theta}}(\overleftarrow{h}_{i-1}, e_i), \quad (4)$$

where $LSTM_{\overrightarrow{\theta}}$ and $LSTM_{\overleftarrow{\theta}}$ represent forward and backward LSTM, respectively. The final state of the backward LSTM $\overleftarrow{h}_0$ is inherited by the decoder as its initial state.

In the decoder layer, the concatenation of a 3-bit one-hot vector which is determined by previously predicted label y_{t-1}, previous final hidden state d_{t-1} (explained later), and the input embedding of x_t, is encoded into the decoder hidden state $\overrightarrow{s}_t$ using stacked forward LSTMs.

Contrary to the original softmax attention method, we can deterministically focus on one encoder hidden state h_t (Yao and Zweig, 2015) to predict y_t in the sentence compression task (Tran et al., 2016).[3]

In the output layer, label probability is calculated as follows:

$$P(y_t \mid y_{<t}, \mathbf{x}) = softmax(W_o \cdot d_t) \cdot \delta_{y_t}, \quad (5)$$

$$d_t = [h_t, \overrightarrow{s}_t] \quad (6)$$

where W_o is a weight matrix of the softmax layer and δ_{y_t} is a binary vector where the y_t-th element is set to 1 and the other elements to 0.

3 Higher-order Syntactic Attention Network

The key component of HiSAN is its attention module. Unlike the baseline Seq2Seq, HiSAN employs a packed d-length dependency chain as distributions in the attention module. Section 3.1 explains the packed d-length dependency chain. Section 3.2 describes the network structure of our attention module, and Section 3.3 explains the learning method of HiSAN.

3.1 Packed d-length Dependency Chain

The probability for a packed d-length dependency chain is obtained from a dependency graph, which is an edge-factored dependency score matrix (Hashimoto and Tsuruoka, 2017; Zhang et al., 2017). First, we explain the dependency graph. Figure 4 (a) shows an example of the dependency graph. HiSAN represents a dependency graph as an attention distribution generated by the attention module. A probability for each dependency edge is obtained from the attention distribution.

[3]This is because the output length is the same as the input length, and each x_t can be assigned to each y_t in a one-to-one correspondence.

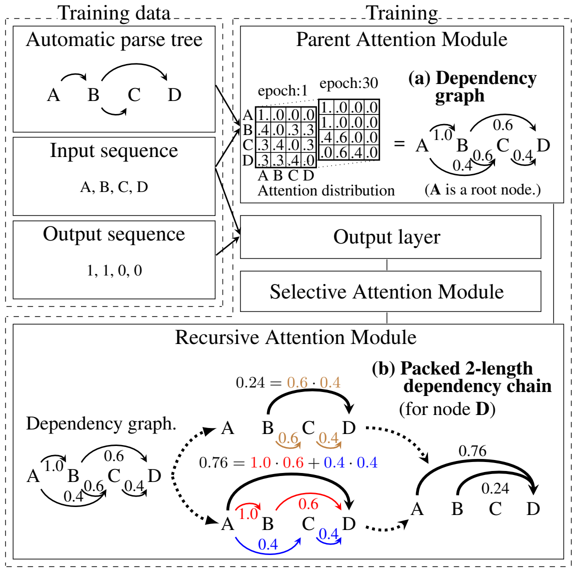


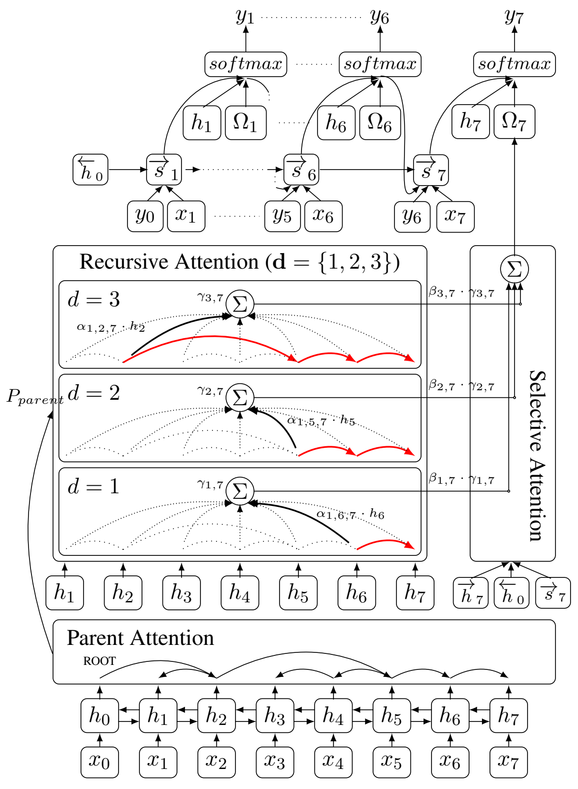
Figure 4: Dependency structures used in our higher-order syntactic attention network.

Figure 4 (b) shows an example of the packed d-length dependency chain. With our recursive attention module, the probability for a packed d-length dependency chain is computed as the sum of probabilities for each path yielded by recursively tracking from a word to its d-th ancestor. The probability for each path is calculated as the product of the probabilities of tracked edges. The probability for the chain can represent several d-length dependency chains compactly, and so alleviates the influence of incorrect parse results. This is the advantage of using dependency graphs.

3.2 Network Architecture

Figure 5 shows the prediction process of HiSAN. In this figure, HiSAN predicts output label y_7 from the input sentence. The prediction process of HiSAN is as follows:

1. **Parent Attention** module calculates $P_{parent}(x_j|x_t, \mathbf{x})$, the probability of x_j being the parent of x_t, by using h_j and h_t. This probability is calculated for all pairs of x_j, x_t. The arc in Figure 5 shows the most probable dependency parent for each child token.



y_1
y_6
y_7
$softmax$
h_1
Ω_1
h_6
Ω_6
h_7
Ω_7
$\overleftarrow{h}_0$
$\overrightarrow{s}_1$
$\overrightarrow{s}_6$
$\overrightarrow{s}_7$
y_0 x_1 y_5 x_6 y_6 x_7
Recursive Attention ($\mathbf{d} = \{1, 2, 3\}$)
$d = 3$
$d = 2$
$d = 1$
$\gamma_{3,7}$
$\gamma_{2,7}$
$\gamma_{1,7}$
$\alpha_{1,2,7} \cdot h_2$
$\alpha_{1,5,7} \cdot h_5$
$\alpha_{1,6,7} \cdot h_6$
$\beta_{3,7} \cdot \gamma_{3,7}$
$\beta_{2,7} \cdot \gamma_{2,7}$
$\beta_{1,7} \cdot \gamma_{1,7}$
Selective Attention
P_{parent}
h_1 h_2 h_3 h_4 h_5 h_6 h_7
$\overrightarrow{h}_7$ $\overleftarrow{h}_0$ $\overrightarrow{s}_7$
Parent Attention
ROOT
h_0 h_1 h_2 h_3 h_4 h_5 h_6 h_7
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7

Figure 5: Prediction process of our higher-order syntactic attention network.

2. **Recursive Attention** module calculates $\alpha_{d,t,j}$, the probability of x_j being the d-th order parent (d denotes the chain length) of x_t, by recursively using $P_{parent}(x_j|x_t, \mathbf{x})$. $\alpha_{d,t,j}$ is also treated as an attention distribution, and used to calculate $\gamma_{d,t}$, the weighted sum of $\mathbf{h}$ for each length d. For example, a 3-length dependency chain of word x_7 with highest probability is x_6-x_5-x_2. The encoder hidden states h_6, h_5 and h_2, which correspond to the dependency chain, are weighted by calculated parent probabilities $\alpha_{1,7,6}$, $\alpha_{2,7,5}$ and $\alpha_{3,7,2}$, respectively, and then fed to the selective attention module.

3. **Selective Attention** module calculates weight $\beta_{d,t}$ from its length, $d \in \mathbf{d}$, for each $\gamma_{d,t}$. $\mathbf{d}$ represents a group of chain lengths. $\beta_{d,t}$ is calculated by encoder and decoder hidden states. Each $\beta_{d,t} \cdot \gamma_{d,t}$ is summed to Ω_t, the output of the selective attention module.

4. Finally, the calculated Ω_t is concatenated and input to the output layer.

Details of each module are explained in the following subsection.

3.2.1 Parent Attention Module

Zhang et al. (2017) formalized dependency parsing as the problem of independently selecting the parent of each word in a sentence. They produced a distribution over possible parents for each child word by using the attention layer on bi-LSTM hidden layers.

In a dependency tree, a parent has more than one child. Under this constraint, dependency parsing is represented as follows. Given sentence $S = (x_0, x_1, ..., x_n)$, the parent of x_j is selected from $S \setminus x_i$ for each token $S \setminus x_0$. Note that x_0 denotes the root node. The probability of token x_j being the parent of token x_t in sentence $\mathbf{x}$ is calculated as follows:

$$P_{parent}(x_j|x_t, \mathbf{x}) = softmax(g(h_{j'}, h_t)) \cdot \delta_{x_j}, \quad (7)$$
$$g(h_{j'}, h_t) = v_a^T \cdot tanh(U_a \cdot h_{j'} + W_a \cdot h_t), \quad (8)$$

where v_a, U_a and W_a are weight matrices of g.

Different from the attention based dependency parser, $P_{parent}(x_j|x_t, \mathbf{x})$ is jointly learned with output label probability $P(\mathbf{y} \mid \mathbf{x})$ in the training phase. Training details are given in Section 3.3.

3.2.2 Recursive Attention Module

The recursive attention module recursively calculates $\alpha_{d,t,j}$, the probability of x_j being the d-th order parent of x_t, as follows:

$$\alpha_{d,t,j} = \begin{cases} \sum_{k=1}^{n} \alpha_{d-1,t,k} \cdot \alpha_{1,k,j} & (d>1) \\ P_{parent}(x_j|x_t, \mathbf{x}) & (d=1) \end{cases}. \quad (9)$$

Furthermore, in a dependency parse tree, *root* should not have any parent, and a token should not depend on itself. In order to satisfy these rules, we impose the following constraints on $\alpha_{1,t,j}$:

$$\alpha_{1,t,j} = \begin{cases} 1 & (t = 0 \wedge j = 0) \\ 0 & (t = 0 \wedge j > 0) \\ 0 & (t \neq 0 \wedge t = j) \end{cases} \quad (10)$$

The 1st and 2nd lines of Eq. (10) represent the case that the parent of *root* is also *root*. These constraints imply that *root* does not have any parent. The 3rd line of Eq. (10) prevents a token from depending on itself. Because the 1st line of Eq. (9) is similar to the definition of matrix multiplication, Eq. (9) can be efficiently computed on a CPU and GPU[4].

[4]In training, HiSAN with 1- and 3-length dependency chains took 25 and 26 minutes, respectively, per epoch on an Intel Xeon E5-2697 v3 2.60 GHz.

By recursively using the single attention distribution, it is no longer necessary to prepare additional attention distributions for each order when computing the probability of higher order parents. Furthermore, since it is not necessary to learn multiple attention distributions, it becomes unnecessary to use hyper parameters for adjusting the weight of each distribution in training. Finally, this method can avoid the problem of sparse higher-order dependency relations in the training dataset.

The above calculated $\alpha_{d,t,j}$ is used to weight the bi-LSTM hidden layer $\mathbf{h}$ as follows:

$$\gamma_{d,t} = \sum_{k=j}^{n} \alpha_{d,t,j} \cdot h_j. \tag{11}$$

Note that $\gamma_{d,t}$ is inherited by the selective attention module, as explained in the next section.

3.2.3 Selective Attention Module

To select suitable dependency orders of the input sentence, the selective attention module weights and sums the hidden states $\gamma_{d,t}$ to Ω_t by using weighting parameter $\beta_{d,t}$, according to the current context as follows:

$$\beta_{d,t} = softmax(W_c \cdot c_t) \cdot \delta_d, \tag{12}$$

$$\Omega_t = \sum_{d \in \{0,\mathbf{d}\}} \beta_{d,t} \cdot \gamma_{d,t}, \tag{13}$$

where W_c is the weight matrix of the softmax layer, $\mathbf{d}$ is a group of chain lengths, c_t is a vector representing the current context, $\gamma_{0,t}$ is a zero-vector, and $\beta_{0,t}$ indicates the weight when the method does not use the dependency features. Context vector c_t is calculated as $c_t = [\overleftarrow{h}_0, \overrightarrow{h}_n, \overrightarrow{s}_t]$ using the current decoder hidden state $\overrightarrow{s}_t$.

The calculated Ω_t is concatenated and input to the output layer. In detail, d_t in Eq. (5) is replaced by concatenated vector $d'_t = [h_t, \Omega_t, \overrightarrow{s}_t]$; furthermore, instead of d_t, d'_t is also fed to the input of the decoder LSTM at $t+1$.

3.3 Objective Function

To alleviate the influence of parse errors, we jointly update the 1st-order attention distribution $\alpha_{1,t,k}$ and label probability $P(\mathbf{y}|\mathbf{x})$ (Kamigaito et al., 2017). The 1st-order attention distribution is learned by dependency parse trees. If $a_{t,j} = 1$ is an edge between parent word w_j and child w_t on a dependency tree ($a_{t,j} = 0$ denotes that w_j is not a parent of w_t.), the objective function of our method can be defined as:

$$-logP(\mathbf{y}|\mathbf{x}) - \lambda \cdot \sum_{j=1}^{n}\sum_{t=1}^{n} a_{t,j} \cdot log\alpha_{1,t,j}, \tag{14}$$

where λ is a hyper-parameter that controls the importance of the output labels and parse trees in the training dataset.

4 Evaluations

4.1 Evaluation Settings

4.1.1 Dataset

This evaluation used the Google sentence compression dataset (Filippova and Altun, 2013)[5]. This dataset contains information of compression labels, part-of-speech (POS) tags, dependency parents and dependency relation labels for each sentence.

We used the first and last 1,000 sentences of `comp-data.eval.json` as our test and development datasets, respectively. Note that our test dataset is compatible wth that used in previous studies (Filippova et al., 2015; Tran et al., 2016; Klerke et al., 2016; Wang et al., 2017).

In this paper, we trained the following baselines and HiSAN on all sentences of `sent-comp.train*.json` (total 200,000 sentences)[6,7,8].

In our experiments, we replaced rare words that appear fewer than 10 times in our training dataset with a special token $\langle$UNK$\rangle$. After this filtering, the input vocabulary size was $23,168$.

4.1.2 Baseline Methods

For a fair comparison of HiSAN, we used the input features described in Eq. (1) for the following baseline methods:

[5]https://github.com/google-research-datasets/sentence-compression

[6]Note that Filippova et al. (2015) used 2,000,000 sentences for training their method, but these datasets are not publicly available.

[7]We also demonstrate an experimental evaluation on a small training set (total 8,000 sentences), that was used in previous research. The results of this setting are listed in our supplemental material.

[8]Note that the large training dataset lacks periods at the end of compressed sentences. To unify the form of compressed sentences in small and large settings, we added periods to the end of compressed sentences in the large training dataset.

Tagger: A method that regards sentence compression as a tagging task based on bi-LSTM (Klerke et al., 2016; Wang et al., 2017).

Tagger+ILP: An extension of **Tagger** that integrates ILP (Integer Linear Programming)-based dependency tree trimming (Wang et al., 2017). We set their positive parameter λ to 0.2.

Bi-LSTM: A method that regards sentence compression as a sequence-to-sequence translation task proposed by (Filippova et al., 2015). For a fair comparison, we replaced their one-directional LSTM with the more expressive bi-LSTM in the encoder part. The initial state of the decoder is set to the sum of the final states of the forward and backward LSTMs.

Bi-LSTM-Dep: An extension of Bi-LSTM that exploits features obtained from a dependency tree (named LSTM-PAR-PRES in Filippova et al. (2015)). Following their work, we fed the word embedding and the predicted label of a dependency parent word to the current decoder input of **Bi-LSTM**.

Base: Our baseline Seq2Seq method described in Section 2.

Attn: An extension of the softmax based attention method (Luong et al., 2015). We replaced h_t in Eq. (6) with the weighted sum calculated by the commonly used concat attention (Luong et al., 2015).

HiSAN-Dep: A variant of HiSAN that utilizes the pipeline approach. We fix $\alpha^{1,j,t}$ to 1.0 if x_j is a parent of x_t in the input dependency parse tree, 0.0 otherwise. In this baseline, $\mathbf{d} = \{1\}$ was used.

4.1.3 Training Details

Following the previous work (Wang et al., 2017), the dimensions of the word embeddings, LSTM layers, and attention layer were set to 100. For the Tagger-style methods, the depth of the LSTM layer was set to 3, and for the Seq2Seq-style methods, the depth of the LSTM layer was set to 2. In this setting, all methods have a total of six LSTM-layers. The dimensions of POS and the dependency-relation label embeddings were set to 40. All parameters were initialized by Glorot and Bengio (2010)'s method. For all methods, we applied Dropout (Srivastava et al., 2014) to the input of the LSTM layers. All dropout rates were set to 0.3.

During training, the learning rate was tuned with Adam (Kingma and Ba, 2014). The initial learning rate was set to 0.001. The maximum number of training epochs was set to 30. The hyper-parameter λ was set to 1.0 in the supervised attention setting. All gradients were averaged in each mini-batch. The maximum mini-batch size was set to 16. The order of mini-batches was shuffled at the end of each training epoch. The clipping threshold of the gradient was set to 5.0. We selected trained models with early stopping based on maximizing per-sentence accuracy (i.e., how many compressions could be fully reproduced) of the development data set.

To obtain a compressed sentence, we used greedy decoding, rather than beam decoding, as the latter attained no gain in the development dataset. All methods were written in C++ on Dynet (Neubig et al., 2017).

4.2 Automatic Evaluation

In the automatic evaluation, we used token level F_1-measure ($\mathbf{F_1}$) as well as recall of ROUGE-1, ROUGE-2 and ROUGE-L (Lin and Och, 2004)[9] as evaluation measures. We used $\Delta C = \mathit{system\ compression\ ratio} - \mathit{gold\ compression\ ratio}$ to evaluate how close the compression ratio of system outputs was to that of gold compressed sentences. The average compression ratio of the gold compression for input sentence was 39.8. We used micro-average for F_1-measure and compression ratio[10], and macro-average for ROUGE scores, respectively.

To verify the benefits of our methods on long sentences, we additionally report scores on sentences longer than the average sentence length ($= 28$) in the test set. The average compression ratio of the gold compression for longer input sentences was 31.4.

All results are reported as the average scores of five trials. In each trial, different random choices were used to generate the initial values of the embeddings and the order of mini-batch processing.

Table 1 shows the results. HiSANs outperformed the other methods. In particular, HiSAN ($d = \{1, 2, 4\}$) achieved the best score on F_1,

[9]We used the ROUGE-1.5.5 script with option "-n 2 -m -d -a".

[10]We also report the macro-average of F_1-measure and compression ratio in our supplemental material.

[11]Note that we used average of all metrics to decide the best score of the development dataset. The results are listed in our supplemental material.

		ALL					LONG				
		F_1	ROUGE			ΔC	F_1	ROUGE			ΔC
			1	*2*	*L*			*1*	*2*	*L*	
Tagger		82.8	81.1	72.4	80.9	-3.0	80.4	78.7	69.7	78.4	-2.8
Tagger+ILP		79.0	76.1	64.6	75.8	-4.1	74.3	73.7	62.1	73.2	-3.6
Bi-LSTM		81.9	81.1	73.7	80.9	-2.2	78.9	78.4	70.4	78.0	-2.1
Bi-LSTM-Dep		82.3	81.5	74.1	81.3	-2.1	79.6	78.9	71.0	78.5	-1.9
Attn		82.4	81.6	74.3	81.4	-2.3	79.8	79.4	71.4	78.7	-2.2
Base		82.7	81.9	74.7	81.7	-2.4	80.1	79.1	71.7	79.0	-2.3
HiSAN-Dep	$(\mathbf{d} = \{1\})$	82.7	82.1	74.9	81.9	-2.2	80.1	79.7	72.0	79.3	-1.9
HiSAN-Dep	$(\mathbf{d} = \{1,2\})$	82.7	81.9	74.6	81.7	-2.4	80.5	79.9	72.3	79.5	-2.2
	$(\mathbf{d} = \{1,2,3\})$	83.1	82.0	74.9	81.8	-2.5	80.5	79.5	71.9	79.2	-2.3
	$(\mathbf{d} = \{1,2,4\})*$	82.9	82.1	74.9	81.9	-2.4	80.4	79.7	72.1	79.4	-2.1
	$(\mathbf{d} = \{1,2,3,4\})$	82.7	82.1	74.8	81.9	-2.2	80.1	79.6	71.9	79.3	-2.0
HiSAN	$(\mathbf{d} = \{1\})$	83.0	81.7	74.5	81.5	-2.9	80.6	79.6	72.1	79.3	-2.5
	$(\mathbf{d} = \{1,2\})$	83.1	82.3	75.1	82.2	-2.1	**80.9**	80.5	72.8	80.2	-1.9
	$(\mathbf{d} = \{1,2,3\})$	82.9	82.5	75.2	82.1	-2.1	80.7	80.2	72.6	79.9	-2.1
	$(\mathbf{d} = \{1,2,4\})*$	**83.2**	**82.9**	**75.8**	**82.7**	**−1.7**	**80.9**	**80.6**	**73.2**	**80.3**	**−1.8**
	$(\mathbf{d} = \{1,2,3,4\})$	82.7	82.0	74.7	81.8	-2.3	80.6	79.8	72.6	79.5	-2.3

Table 1: Results of automatic evaluation. **ALL** and **LONG** represent, respectively, the results in all sentences and long sentences (longer than average length 28) in the test dataset. $\mathbf{d}$ represents the groups of d-length dependency chains. $*$ indicates the model that achieved the best score among the same methods with different $\mathbf{d}$ in the development dataset[11]. Bold values indicate the best scores.

ROUGE, and ΔC in all settings. The F_1 scores of HiSAN (**ALL**) were higher than the current state-of-the-art score of .82, reported by Filippova et al. (2015). The improvements in F_1 and ROUGE scores from the baselines methods in the **LONG** setting are larger than those in the **ALL** setting. ¿From these results, we can conclude that d-length dependency chains are effective for sentence compression, especially in the case of longer than average sentences. HiSAN ($d = \{1\}$) outperformed **HiSAN-Dep** in F_1 scores in **ALL** and **LONG** settings. This result shows the effectiveness of joint learning the dependency parse tree and the output labels.

4.3 Human Evaluation

In the human evaluation, we compared the baselines with our method, which achieved the highest F_1 score in the automatic evaluations. We used the first 100 sentences that were longer than the average sentence length (= 28) in the test set for human evaluation. Similar to Filippova et al. (2015), the compressed sentence was rated by five raters who were asked to select a rating on a five-point Likert scale, ranging from one to five for readability (**Read**) and for informativeness (**Info**). We report the average of these scores from the five raters. To investigate the differences between the methods, we also compared the baseline methods and HiSAN using the sentences for which the methods yielded different compressed sentences.

		Read	Info	CR
All *(100)*	Tagger	4.54	3.41	30.9
	Base	4.64	3.45	31.1
	HiSAN ($d=\{1,2,4\}$)	**4.68**	**3.52**	31.6
Diff *(41)*	Base	4.79	3.46	29.4
	HiSAN ($d=\{1,2,4\}$)	**4.89**	**3.64**	30.6

Table 2: Results of human evaluations. *All* denotes results for all sentences in the test set, and *Diff* denotes results for the sentences for which the methods yielded different compressed sentences. Parentheses ( ) denote sentence size. **CR** denotes the compression ratio. The average gold compression ratio for input sentence in *All* and *Diff* were 32.1 and 31.5, respectively. Other notations are similar to those in Table 1.

Table 2 shows the results. HiSAN ($d = \{1, 2, 4\}$) achieved better results than the baselines in terms of both readability and informativeness. The results agree with those obtained from the automatic evaluations. ¿From the results on the sentences whose compressed sentences were different between Base and HiSAN ($d = \{1, 2, 4\}$), we can clearly observe the improvement attained by HiSAN ($d=\{1, 2, 4\}$) in informativeness.

Input	Pakistan signed a resolution on Monday to import 1,300 MW of electricity from Kyrgyz Republic and Tajikistan to overcome power shortage in summer season, said an official press release .
Gold	Pakistan signed a resolution to import 1,300 MW of electricity from Kyrgyz Republic and Tajikistan .
Tagger	Pakistan signed a resolution to import 1,300 MW of electricity Tajikistan to overcome shortage .
Tagger-ILP	Pakistan signed resolution to import MW said .
Base	Pakistan signed a resolution to import 1,300 MW of electricity .
HiSAN-Dep ($d = \{1\}$)	Pakistan signed a resolution to import 1,300 MW of electricity .
HiSAN ($d = \{1, 2, 4\}$)	Pakistan signed a resolution to import 1,300 MW of electricity from Kyrgyz Republic and Tajikistan .
Input	US whistleblower Bradley Manning , charged with releasing over 700,000 battlefield reports from Iraq and Afghanistan to Wikileaks , received a sentence of 35 years in prison from a military court Wednesday .
Gold	Bradley Manning received a sentence of 35 years in prison .
Tagger	Bradley Manning received a sentence of 35 years .
Tagger-ILP	Bradley Manning received a sentence of years .
Base	Bradley Manning received a sentence of 35 years .
HiSAN-Dep ($d = \{1\}$)	Bradley Manning charged with releasing over 700,000 battlefield reports to Wikileaks received .
HiSAN ($d = \{1, 2, 4\}$)	Bradley Manning received a sentence of 35 years in prison .

Table 3: Example sentences and compressions.

5 Analysis

Table 3 shows examples of source sentences and their compressed variants output by baseline and HiSAN ($d = \{1, 2, 4\}$).

For both examples, the compressed sentence output by **Base** is grammatically correct. However, the informativeness is inferior to that attained by HiSAN ($d = \{1, 2, 4\}$). The compressed sentence output by **HiSAN-Dep** in the second example lacks both readability and informativeness. We believe that this compression failure is caused by incorrect parse results, because **HiSAN-Dep** employs the features obtained from the dependency tree in the pipeline procedure.

As reported in recent papers (Klerke et al., 2016; Wang et al., 2017), the F_1 scores of **Tagger** match or exceed those of the Seq2Seq-based methods. The compressed sentence of the first example in Table 3 output by **Tagger** is ungrammatical. We believe that this is mainly because **Tagger** cannot consider the predicted labels of the previous words. **Tagger-ILP** outputs grammatically incorrect compressed sentences in both examples. This result indicates that THE ILP constraint based on the parent-child relationships between words is insufficient to generate fluent sentences.

Compared with these baselines, HiSAN ($d = \{1, 2, 4\}$) output compressed sentences that were fluent and had higher informativeness. This observation, which confirmed our expectations, is supported by the automatic and human evaluation results.

	F_1	**ROUGE**			**ΔC**
		1	*2*	*L*	
Tagger	**82.8**	80.6	72.2	80.3	-3.2
Tagger+ILP	77.5	74.7	64.1	74.3	-4.6
Bi-LSTM	81.3	80.4	73.3	80.1	-2.2
Bi-LSTM-Dep	81.5	80.7	73.5	80.3	-2.1
Attn	81.9	81.0	73.9	80.6	-2.3
Base	82.1	81.0	73.9	80.7	-2.5
HiSAN-Dep					
$\mathbf{d} = \{1\}$	82.3	81.2	74.3	80.9	-2.4
$\mathbf{d} = \{1, 2\}$	81.9	80.8	73.8	80.4	-2.6
$\mathbf{d} = \{1, 2, 3\}$	82.6	81.2	74.3	80.9	-2.6
$\mathbf{d} = \{1, 2, 4\}$	82.0	80.7	73.7	80.4	-2.7
$\mathbf{d} = \{1, 2, 3, 4\}$	82.1	81.0	74.1	80.7	-2.5
HiSAN					
$\mathbf{d} = \{1\}$	82.7	81.4	74.5	81.1	-2.8
$\mathbf{d} = \{1, 2\}$	82.6	81.8	74.9	81.5	-2.1
$\mathbf{d} = \{1, 2, 3\}$	82.6	81.8	74.9	81.5	-2.3
$\mathbf{d} = \{1, 2, 4\}$	**82.8**	**82.2**	**75.5**	**82.0**	**−2.0**
$\mathbf{d} = \{1, 2, 3, 4\}$	82.4	81.3	74.4	81.0	-2.4

Table 4: Results of automatic evaluation using sentences with deep dependency trees (deeper than average depth 8). Bold results indicate the best scores.

We confirm that the compression performance of HiSAN actually improves if the sentences have deep dependency trees. Table 4 shows the automatic evaluation results for sentences with deep dependency trees. We can observe that HiSAN

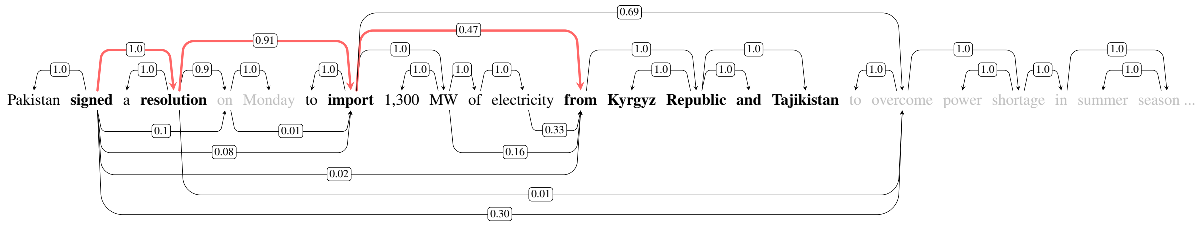

Pakistan signed a resolution on Monday to import 1,300 MW of electricity from Kyrgyz Republic and Tajikistan to overcome power shortage in summer season ...
1.0
1.0
1.0
0.9
1.0
0.91
1.0
0.47
1.0
1.0
1.0
1.0
0.69
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
0.1
0.01
0.33
0.08
0.16
0.02
0.01
0.30

Figure 6: An example compressed sentence and its dependency graph of HiSAN $\mathbf{d} = \{1, 2, 4\}$. The words colored by gray represent deleted words. The numbers for each arc represent a probabilistic weight of a relationship between a parent and child words. The arcs contained in the parsed dependency tree are located on the top side. The arcs not contained in the parsed dependency tree are located on the bottom side.

with higher-order dependency chains has better compression performance if the sentences have deep dependency trees.

Figure 6 shows a compressed sentence and its dependency graph as determined by HiSAN $\mathbf{d} = \{1, 2, 4\}$. Almost all arcs with large probabilistic weights are contained in the parsed dependency trees. Interestingly, some arcs not contained in the parsed dependency trees connecting words which are connected by the dependency chains in the parsed dependency tree (colored by red). Considering the training dataset does not contain such dependency relationships, we can estimate that these arcs are learned in support of compressing sentences. This result meets our expectation that the dependency chain information is necessary for compressing sentences accurately.

6 Related Work

Several neural network based methods for sentence compression use syntactic features. Filippova et al. (2015) employs the features obtained from automatic parse trees in the LSTM-based encoder-decoder in a pipeline manner. Wang et al. (2017) trims dependency trees based on the scores predicted by an LSTM-based tagger. Although these methods can consider dependency relationships between words, the pipeline approach and the 1st-order dependency relationship fail to compress longer than average sentences.

Several recent machine translation studies also utilize syntactic features in Seq2Seq models. Eriguchi et al. (2017); Aharoni and Goldberg (2017) incorporate syntactic features of the target language in the decoder part of Seq2Seq. Both methods outperformed Seq2Seq without syntactic features in terms of translation quality. However, both methods fail to provide an entire parse tree until the decoding phase is finished. Thus, these methods cannot track all possible parents for each word within the decoding process. Similar to HiSAN, Hashimoto and Tsuruoka (2017) use dependency features as attention distributions, but different from HiSAN, they use pre-trained dependency relations, and do not take into account the chains of dependencies. Marcheggiani and Titov (2017); Bastings et al. (2017) consider higher-order dependency relationships in Seq2Seq by incorporating a graph convolution technique (Kipf and Welling, 2016) into the encoder. However, the dependency information of the graph convolution technique is still given in pipeline manner.

Unlike the above methods, HiSAN can capture higher-order dependency features using d-length dependency chains without relying on pipeline processing.

7 Conclusion

In this paper, we incorporated higher-order dependency features into Seq2Seq to compress sentences of all lengths.

Experiments on the Google sentence compression test data showed that our higher-order syntactic attention network (HiSAN) achieved the better performance than baseline methods on F_1 as well as ROUGE-1,2 and L scores 83.2, 82.9, 75.8 and 82.7, respectively. Of particular importance, challenged with longer than average sentences, HiSAN outperformed the baseline methods in terms of F_1, ROUGE-1,2 and L scores. Furthermore, HiSAN also outperformed the previous methods for both readability and informativeness in human evaluations.

From the evaluation results, we conclude that HiSAN is an effective tool for the sentence compression task.

References

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 132–140. http://aclweb.org/anthology/P17-2021.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Simaan. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 1957–1967. https://www.aclweb.org/anthology/D17-1209.

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, pages 481–490. http://www.aclweb.org/anthology/P11-1049.

Yanju Chen and Rong Pan. 2017. Automatic emphatic information extraction from aligned acoustic data and its application on sentence compression. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*. pages 3422–3428.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics, Doha, Qatar, pages 103–111. http://www.aclweb.org/anthology/W14-4012.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 72–78. http://aclweb.org/anthology/P17-2012.

Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 360–368. http://aclweb.org/anthology/D15-1042.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, pages 1481–1491. http://www.aclweb.org/anthology/D13-1155.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. pages 249–256.

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*. Association for Computational Linguistics, Copenhagen, Denmark, pages 125–135. https://www.aclweb.org/anthology/D17-1012.

Hongyan Jing. 2000. Sentence reduction for automatic text summarization. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, pages 310–315. https://doi.org/10.3115/974147.974190.

Hidetaka Kamigaito, Katsuhiko Hayashi, Tsutomu Hirao, Hiroya Takamura, Manabu Okumura, and Masaaki Nagata. 2017. Supervised attention for sequence-to-sequence constituency parsing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Asian Federation of Natural Language Processing, Taipei, Taiwan, pages 7–12. http://www.aclweb.org/anthology/I17-2002.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *CoRR* abs/1609.02907. http://arxiv.org/abs/1609.02907.

Sigrid Klerke, Yoav Goldberg, and Anders Søgaard. 2016. Improving sentence compression by learning to predict gaze. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 1528–1533. http://www.aclweb.org/anthology/N16-1179.

Kevin Knight and Daniel Marcu. 2000. Statistics-based summarization-step one: Sentence compression. *AAAI/IAAI* 2000:703–710.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics, Vancouver, pages 28–39. http://www.aclweb.org/anthology/W17-3204.

Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*. Barcelona, Spain, pages 605–612. https://doi.org/10.3115/1218955.1219032.

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*. Association for Computational Linguistics, Lisbon, Portugal, pages 1412–1421. http://aclweb.org/anthology/D15-1166.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 1506–1515. https://www.aclweb.org/anthology/D17-1159.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980* .

J. Pouget-Abadie, D. Bahdanau, B. van Merrienboer, K. Cho, and Y. Bengio. 2014. Overcoming the Curse of Sentence Length for Neural Machine Translation using Automatic Segmentation. *ArXiv e-prints* .

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Nhi-Thao Tran, Viet-Thang Luong, Ngan Luu-Thuy Nguyen, and Minh-Quoc Nghiem. 2016. Effective attention-based neural architectures for sentence compression with bidirectional long short-term memory. In *Proceedings of the Seventh Symposium on Information and Communication Technology*. ACM, New York, NY, USA, SoICT '16, pages 123–130. https://doi.org/10.1145/3011077.3011111.

Oriol Vinyals, Ł ukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pages 2773–2781. http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language.pdf.

Liangguo Wang, Jing Jiang, Hai Leong Chieu, Chen Hui Ong, Dandan Song, and Lejian Liao. 2017. Can syntax help? improving an lstm-based sentence compression model for new domains. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 1385–1393. http://aclweb.org/anthology/P17-1127.

Kaisheng Yao and Geoffrey Zweig. 2015. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*. pages 3330–3334. http://www.isca-speech.org/archive/interspeech_2015/i15_3330.html.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Association for Computational Linguistics, Valencia, Spain, pages 665–676. http://www.aclweb.org/anthology/E17-1063.