Improving a Syntactic Graph Convolution Network for Sentence Compression

Yifan Wang and Guang Chen

Beijing University of Posts and Telecommunications, Beijing, China

yifan_wang@bupt.edu.cn,chenguang@bupt.edu.cn

Abstract. Sentence compression is a task of compressing sentences containing redundant information into short semantic expressions, simplifying the text structure and retaining important meanings and information. Neural network-based models are limited by the size of the window and do not perform well when using long-distance dependent information. To solve this problem, we introduce a version of the graph convolutional network (GCNs) to utilize the syntactic dependency relations, and explore a new way to combine GCNs with the Sequence-to-Sequence model (Seq2Seq) to complete the task. The model combines the advantages of both and achieves complementary effects. In addition, in order to reduce the error propagation of the parse tree, we dynamically adjust the dependency arc to optimize the construction process of GCNs. Experiments show that the model combined with the graph convolution network is better than the original model, and the performance in the Google sentence compression dataset has been effectively improved.

Keywords: sentence compression; graph convolution network; sequence-to-sequence

1 Introduction

Sentence compression is a standard NLP task that simplifies sentences while preserving the most important content. Deletion-based sentence compression treats the task as a word deletion problem: given an input source sentence $\mathbf{x} = (x_0, x_1, \ldots, x_N)$ (each symbol represents a word in the sentence), and the goal is to generate a sentence summarization by deleting the words in the source sentence $\mathbf{x}$. There have been many studies related to syntactic-tree-based methods[1,2,3,4] or machine-learning-based methods[5, 6, 7] in the past. A common approach is to use the syntactic tree to produce the most readable and informative compression [1,3]. Meanwhile, Filippova et al. [5] treated the sentence compression task as a sequence labeling problem, using the recurrent neural network (RNN), which has become a main-stream solution. However, the two methods still have the following problems:

1. Automatic generation of compressions by simply pruning the syntactic tree is inevitably vulnerable to error propagation as there is no way to recover from an incorrect parse tree.
2. The sequence labeling-based recurrent neural network (RNN) approach focuses more on the word itself than on the overall sentence structure; due to the timing characteristics of the RNN, the jumping information with long distance is difficult to capture.

The purpose of our paper is to study how to combine the syntactic structure with the neural network to achieve complementary effects and reduce the error propagation of the parse tree.

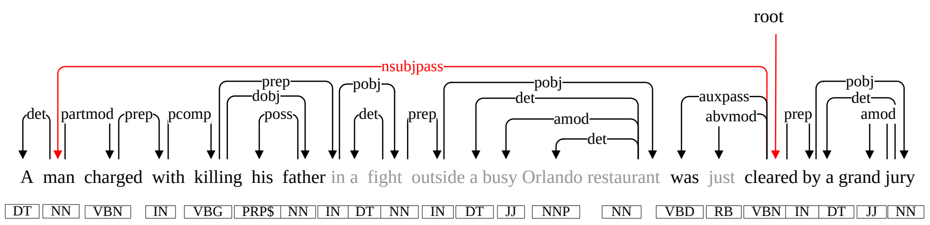
root
nsubjpass
prep
dobj
poss
pobj
det
prep
pobj
det
amod
det
auxpass
abvmod
prep
pobj
det
amod
det
partmod
prep
pcomp
A man charged with killing his father in a fight outside a busy Orlando restaurant was just cleared by a grand jury
DT NN VBN IN VBG PRP$ NN IN DT NN IN DT JJ NNP NN VBD RB VBN IN DT JJ NN

Fig. 1. An example of sentence compression and its syntactic parsing results in Google sentence compression dataset. Words in gray indicate are supposed to be deleted. The tags below the words indicate the corresponding part-of-speech tags in the tree.

Semantic representation is closely related to syntactic features. Intuitively, in a syntactic dependency tree, if the parent node has a high probability to be retained, the child node directly connected to it should also be retained with a certain probability.

For example, Figure 1 shows an example of compressed sentence and its syntactic dependency tree. The word "*cleared*" connected to the “*root*” is retained with a great probability. The word "*man*" which directly connected to “*cleared*” by dependent arc should also be given a high score, although the two words in the sentence are far apart (15 words apart). Furthermore, following the dependency chain, the phrases "*charged with*" and "*by a grand jury*" have a greater tendency to be retained too. As a result, given the similarities of dependency trees and compressed sentences, syntactic information can be used naturally when generating sentence summaries.

In particular, in this paper we introduce graph convolutional network (GCNs) [8,9] in the sentence compression task, combining it with the sequence-to-sequence model (Seq2Seq). GCNs is a multi-layer neural network acting on graphs and is widely used in irregular data tasks. By using dependency relations, syntactic context representation is generated for each node in the graph (each word in the text). Further, this representation is incorporated into the encoding structure of the Seq2Seq model in order to obtain more relevant information. In Section 3.4 we will discuss two combination methods of GCNs and Seq2Seq: stacked GCNs and parallel GCNs. As we use the parallel method, we achieve the state-of-the-art performance on the widely used dataset for sentence compression.

2 Related work

The general techniques of text summarization are mainly divided into two categories: extractive methods and abstractive methods. The method of extraction is designed to select valid fragments, sentences or paragraphs from the text, and generate a shorter summary. The abstract summary is closer to natual language expression and intended to be generated using natural language generation techniques. This paper focuses on extractive summaries based on deletions.

Previous work used syntactic-tree-based approach to produce readable and informative compression. Jing [1] used a variety of external resources to determine which part of the sentence can be removed, eventually the component with little relevance to the context was removed. Knight et al. [10] proposed applying the noise channel model to the sentence compression task. Clarke et al. [3] first used the ILP framework for sentence compression. Filippova et al. [4] operated directly on the syntactic parsing tree to get the compressed sentence and constructed a large-scale parallel corpus, which laid the foundation for the subsequent research.

In recent work, researchers have attempted to use neural networks in sentence compression tasks. Filippova et al. [5] treated the sentence compression task as a sequence labeling problem. The model based on the recurrent neural network (RNN) was trained on a large-scale parallel corpus. In terms of introducing the syntactic features, Wang et al. [7] used explicit syntactic features and introduced syntactic constraints through Integer Linear Programming (ILP). In order to solve the problem of long sentence compression, Kamigaito et al. [13] handled dependency features as an attention distribution on LSTM hidden states. Marcheggiani et al. [9] introduced GCNs into the semantic role labeling task and stacked it with LSTMs. On this basis, we explore more ways to combine GCNs and RNN, and the structure of GCNs is further optimized to mitigate the effects of error propagation of the parse tree.

3 Approach

3.1 Problem definition

Sentence compression based on deletion is regarded as a sequence labeling task. The original sentence is represented as $\mathbf{x} = (x_0, x_1, \ldots, x_N)$ (where N represents the length of the sentence). By retaining or deleting words, a sequence that still retains important information is obtained. The output is a binary decision sequence $\mathbf{y} = (y_0, y_1, \ldots, y_N)$, $y_i \in \{0,1\}$. Here, $y_i = 1$ represents RETAIN, and $y_i = 0$ represents REMOVE.

3.2 Base Seq2Seq model

First we introduce our baseline model, an extended model of Seq2Seq. In the deleted based task, LSTM-based tagging model [7][14] and Seq2Seq model [5][13][15] are widely used. Considering that Seq2Seq model can capture the overall sentence information, we improve Seq2Seq model [13] as the basis for experiments.

In the input layer, the input sentence is converted to a dense vector representation. According to the setting of Wang et al. [7], besides pre-training word vectors $\mathbf{w} = (w_0, w_1, \ldots, w_N)$, we also consider part-of-speech tags and dependency relations as additional feature inputs. The part-of-speech tag embedding is denoted as $\mathbf{t} = (t_0, t_1, \ldots, t_N)$ and the dependency relation embedding is denoted as $\mathbf{r} = (r_0, r_1, \ldots, r_N)$. The three are concatenated to form a new word representation as input:

$$\begin{aligned} \mathbf{e} &= (e_0, e_1, \ldots, e_N) \\ e_t &= [w_t, t_t, r_t] \end{aligned} \tag{1}$$

Here [] represents the concatenation of vectors, and e_t is the vector input when the time step is *t*.

In the encoder layer, we use the standard bidirectional LSTM model to encode the input embedding vector into a series of hidden layer vectors $\mathbf{h} = (h_0, h_1, \ldots, h_N)$. The representation of the encoder output h_t is:

$$h_t = \left[\overrightarrow{h_t}, \overleftarrow{h_t}\right] \tag{2}$$

$$\overrightarrow{h_t} = \overrightarrow{LSTM}\left(\overrightarrow{h_{t-1}}, e_t\right) \tag{3}$$

$$\overleftarrow{h_t} = \overleftarrow{LSTM}\left(\overleftarrow{h_{t-1}}, e_t\right) \tag{4}$$

In the decoder layer, LSTM unit is used as the basic unit of the RNN. Unlike Kamigaito et al. [13], in each step *i*, the decoder state s_i is determined by the previous decoder state s_{i-1}, the previously predicted label y_{i-1}, the aligned encoder hidden layer output h_i, and the context vector c_i:

$$s_i = f_{LSTM}\left(s_{i-1}, y_{i-1}, h_i, c_i\right) \tag{5}$$

Where y_i is the predicted label, f_{LSTM} is the state transition operation inside LSTM unit, and the context vector c_i is calculated as the weighted sum of the encoder states[16]:

$$c_i = \sum_{j=1}^{N} \alpha_{i,j} h_j \tag{6}$$

$$\begin{aligned} \alpha_{i,j} &= \frac{\exp(e_{i,j})}{\sum_{k=1}^{N} \exp(e_{i,k})} \\ e_{i,k} &= g(s_{i-1}, h_k) \end{aligned} \tag{7}$$

Where h_j is the output of the encoder at different time step *j*. e_{ij} represents the matching score of the input in the position *j* and the output in the position *i*, which is calculated by the hidden layer state s_{i-1} and h_j .

The final output layer calculates the probability of each label:

$$p\left(y_i \mid y_{i-1}, \mathbf{x}\right) = softmax(Ws_i + b) \tag{8}$$

Where *W* and *b* are the parameter matrices and bias to be learned.

3.3 Syntactic GCNs

GCNs aims to capture the dependency relations between nodes. Each node in the graph constantly changes its state due to the influence of neighbors and farther points. The closer the neighbor is, the greater its influence is. For a graph $G = (V, E)$, the input **x** is a *N*D* matrix. There is also an adjacency matrix **A** of the graph. It is desirable to obtain a *N*F* feature matrix **Z**, representing the feature of each node, where *F* is the dimension of the desired representation.

In Kipf et al. [8], GCNs is proved to be very effective for node classification tasks. In this paper, we introduce it into the sentence compression task. Here *G* is the parse tree for the input sentence **x**, $V = (v_1, v_2, ..., v_n)$ (|*V*|=n) is the word in **x**, and $\mathrm{K} = \left(v_i, v_j\right) \in E$ is the directed dependent arc between the words in **x**. In addition to the forward arc, we also consider the reverse flow of information: the reverse arc, and the self-circulating arc [9]. As shown in Figure 1, there is an arc *K("man", "charged") = partmod* (verb modification relationship) between *"man"* and *"charged"*. Naturally, there is a modified relationship $K("charged","man") = partmod'$ between the two nodes, with the apostrophe indicating that the information flows in the opposite direction of the arc.

For one layer of GCNs, the relationship of direct neighbor nodes can be captured; for a k-layers stacked GCNs, the relationship of up to k-order chains can be learned. In the k-th syntactic convolutional network layer, we calculate the graph convolution vector of the node $v \in V$ by the following method:

$$h_v^{(k+1)} = f\left(\sum_{u \in N(v)} \left(W_{K(u,v)}^{(k)} h_u^{(k)} + b_{L(u,v)}^{(k)}\right)\right) \tag{9}$$

Where $K(u,v)$ and $L(u,v)$ denotes the type of dependency relations; $W_{K(u,v)}^{(k)}$ and $b_{L(u,v)}^{(k)}$ are weight matrices and bias; $N(v)$ is the set of neighbors of *v*, including *v* (because it contains a self-loop); f is the activation function.

Since the parameters *W* are specific to the dependency tags, assuming that the number is N, then there will be (2N + 1) parameter matrices for one layer of GCNs to be trained. Specifically, for the Google sentence compression dataset used in this paper, there are 54 dependency relations, and 109 parameter matrices will be learned. The training costs make model fall into the predicament of over-parameterization. Following the settings of Marcheggiani et al. [9], we use different dependency relations on the bias terms, but simplify *W*, only to show three edges (1) forward; (2) reverse; (3) self-looping. The parameters *W* are shared in the same direction side of the different tags, and the dependency is expressed as:

$$K(w_i, w_j) = \begin{cases} along, & (w_i, w_j) \in E \\ rev, & i != j \,\&\, (w_j, w_i) \in E \\ loop, & i == j \end{cases} \tag{10}$$

However, considering the sentence summary task, on the one hand, the structure of GCNs is completely determined by the syntactic parsing result, which is unchangeable during the training process. In order to reduce the influence of the error propagation, we set a dynamic adjustment of the inputs' dependency arc. On the other hand, in the compression process, there is no need to rely equally on the edges contained in the graph. It is necessary to add different weights to the edges. In response to these problems, we make the following two changes:

1. According to equation (9), the input of node *v* is determined by its parent *u*. For a given relationship, the structure of GCNs is determined. In this paper, the dependent parent of the input parsing is dynamically adjusted to make it structurally variable during the training process. Define $p = (u_0, u_1, \ldots u_N)$ as parent set of sentence **x**, u_t is the parent node of v_t. The one-hot representation of *p* is denoted by p_v , *W* and *b* are trainable parameter matrices, and σ is the logistic sigmoid function. The adjusted dependent parent p' is represented as：

$$p' = argmax(\sigma(Wp_v + b)) \tag{11}$$

2. Inspired by Marcheggiani et al. [9], a gating unit is added for each node output, using weights to get different importance distributions:.

$$g_{u,v}^{(k)} = \sigma\left(h_u^{(k)} \cdot V_{K(u,v)}^{(k)} + b_{L(u,v)}^{(k)}\right) \tag{12}$$

$V_{K(u,v)}^{(k)}$ and $b_{L(u,v)}^{(k)}$ are the parameter matrix and bias of the gating unit. After adding the gate, the syntactic GCNs's final output is:

$$h_v^{(k+1)} = ReLU\left(\sum_{u \in N(v)} g_{u,v}^{(k)}(W_{K(u,v)}^{(k)} h_u^{(k)} + b_{L(u,v)}^{(k)})\right) \tag{13}$$

3.4 Combining GCNs and Seq2Seq

GCNs can capture the dependency relations between associated arcs, which indicates that GCNs has the ability to learn semantic information over long distance, but may not perform well on local sequential information mining. Studies have shown that, the LSTMs model has outstanding performance in processing context information in this task [6,7].

At this point, it can be considered that LSTMs and GCNs play complementary roles. LSTMs can handle timing-related context information. GCNs establishes the connection of semantically related words, although they may be far apart in the same sentence.

In this paper, GCNs structure is integrated into the encoder of Seq2Seq, and the different connection methods of GCNs and LSTMs are compared.

Stacked Model

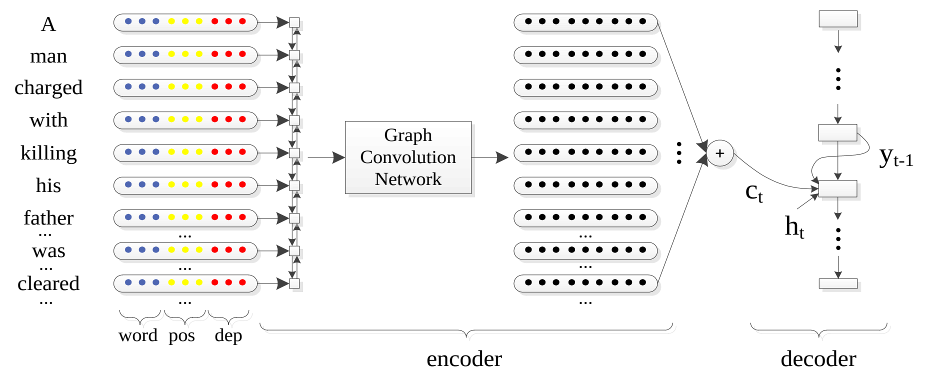

Fig. 2. Stacked model, including input layer, LSTMs layer, GCNs layer, and decoder portion.

As shown in Figure 2, similar to Marcheggiani et al. [9], LSTMs model and GCNs model are stacked. The output of bidirectional LSTMs layer is obtained from equation (2), as an input to GCNs. The local context information is adaptively acquired by LSTMs.

Parallel Model

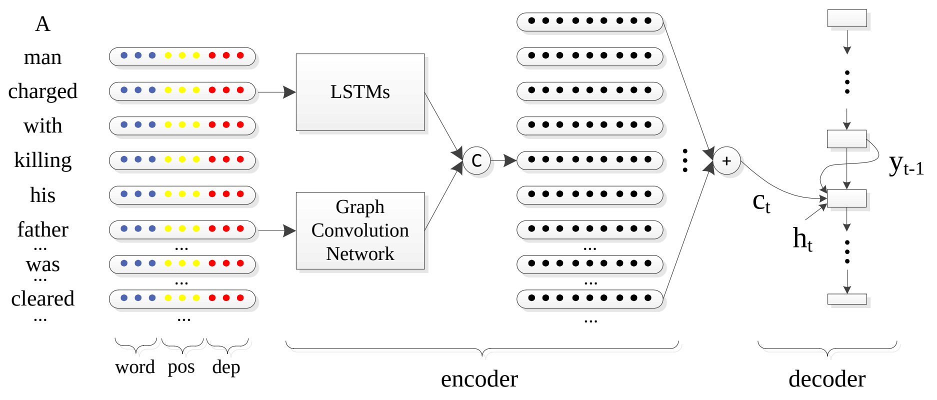

Fig. 3. Parallel model, parallelizing LSTMs layer and GCNs layer in the encoder section. The function C is introduced by formulars 15 and 16.

The above model represents the encoder as a sequential superposition of Bi-LSTMs layer and GCNs layer. However, previous work did not explicitly explore why LSTMs and GCNs should be stacked: this paper attempts to use a parallel form to dynamically control the output of the encoder by calculating the probability of using LSTMs or GCNs. As shown in Figure 3, the final output of the encoder is determined jointly by the output of LSTMs and GCNs. We demonstrate the validity of this idea in later experiments. The previous model can be considered as stacked, and the model proposed next can be considered as parallel.

After the input layer, the same feature vector is input to Bi-LSTMs layer and GCNs layer, which means using the original input sentence instead of the output of LSTMs as input to GCNs:

$$e = (e_0, e_1, \cdots, e_N) \tag{14}$$

Calculate the probability of using LSTMs or GCNs for each word:

$$g_t^{\alpha} = \sigma(h_t^{\alpha} w_{\alpha} + b_{\alpha}) \tag{15}$$

α represents LSTMs layer or GCNs layer, h_t^{α} is the hidden layer output, *W* and *b* are trainable parameter matrices and bias, and σ is the logistics activation function.

The output of the encoder contains the information of LSTMs hidden layer h^{lstm} and GCNs hidden layer h^{gcn}, calculated by the following formula:

$$h_t = [g_t^{lstm} \cdot h_t^{lstm}, g_t^{gcn} \cdot h_t^{gcn}] \tag{16}$$

4 Experiment

4.1 Dataset and Parameters

We use Google sentence compression dataset [5], as shown in Table 1, containing 200,000 sentence compression pairs[1]. The dataset contains each sentence's compressed tags, part-of-speech (POS) tags, dependency parents and dependency tags. For comparison with previous studies [5,7,13,15,18], the first 1000 and last 1000 sentences of comp-data.eval.json are used as the test dataset and validation dataset, and the rest is used as training dataset.

Table 1. Google sentence compression dataset

	Google dataset
Size	200,000
Compression Ratio	0.40
Vocab Size	42621

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

In the experiment, for the input layer, the embedding vector size of pre-training word, the part-of-speech tag and the dependency is 100. The word vector is initialized by GloVe-100 dimension pre-training embedding, and the part of speech and dependency embedding is randomly initialized. The bidirectional LSTMs in the Seq2Seq model encoder is set to two layers, and the depth of decoder is also set to two. As a result, the total number of layers is six, which is comparable to Wang et al. [7] and Kamigaito et al. [13]. Dropout [19] rates are 0.3. The batchsize is set to 16. All experiments use the Adam [20] optimization algorithm, with the initial learning rate being 0.0001.

4.2 Evaluation

Table 2. Results in Google dataset

Google dataset	F1	Accuracy	CR
&1 Traditional ILP (Clarke el al., 2008)[2]	54.0	56.0	62.0
&2 LSTMs (Filippova el al., 2015)	82.0	-	38.0
&3 Seq2Seq (Kamigaito el al., 2018)	82.7	-	37.4
&4 HiSAN (Kamigaito el al., 2018)	83.2	-	38.1
&5 Seq2Seq (Our implementation)	82.6	86.2	38.0
&6 Seq2Seq+stacked GCNs (k=1)	83.0	86.4	38.3
&7 Seq2Seq+stacked GCNs (k=1), without adjusted	82.9	86.2	38.0
&8 Seq2Seq+stacked GCNs (k=2)	82.6	86.2	38.1
&9 Seq2Seq+parallel GCNs (k=1)	**83.3**	**86.5**	**39.0**
&10 Seq2Seq+parallel GCNs (k=1), without adjusted	82.7	86.1	38.6
&11 Seq2Seq+parallel GCNs (k=2)	83.1	86.4	38.5
&12 Seq2Seq+GCNs(no LSTMs)(k=1)	82.0	85.4	38.0
&13 Seq2Seq+GCNs(no LSTMs)(k=2)	81.9	85.4	39.0

We use Google sentence compression dataset with true-value tags for automated evaluation. Word-level accuracy (Accuracy) and F1-score are used to measure the validity of the model. Accuracy is defined as the correct percentage of each predicted label yi in the sentence. The F1 score is calculated based on the precision (Precision) and recall rate (Recall) in terms of tokens kept in the golden and the generated compressions. In addition, we measure the difference between truth and prediction by calculating the Compression Ratio (CR). To compare with the previous work, Table 2 shows the performance of the experiments in our work.

1. First, we implement a powerful baseline model &5 that is comparable to other research results. This version of the Seq2Seq model, which does not add GCNs, is 0.6% better than LSTMs model &2 with larger datasets in F1 score.

[2] Note that we quote the results from Wang et al [7].

2. In order to verify the validity of GCNs layer, we compare our GCNs model with baseline under the same settings. It was observed that the stacked model &6 and the parallelized model &9 respectively increased the F1 score by 0.4% and 0.7%, compared to the baseline. The performance of parallel model is better than the best results [13] on the current dataset. The compression ratio of our model is also closer to the original dataset. From model &7 and model &10, which don't adjust the dependent parents, we observe the effectiveness of this improvement.
3. It can be seen from experiments &8 and &11 that increasing the number of layers of GCNs does not improve the experimental effect. After adding a layer of GCNs (k=2) to the stacked model and the parallel model, the F1 score decreased by 0.4% and 0.2%, respectively. This may be because the combination of LSTMs and one layer of GCNs has obtained most of the information. The multi GCNs layer adds redundant information, which causes interference to the prediction results.
4. Experiments &6 and &9 compare stacked GCNs and parallel GCNs models. The F1 score, accuracy, and compression ratio of the parallel model all performed better.
5. Finally, in experiments &12 and &13, we compare the versions that GCNs are employed alone in the encoder. The results are incomparable with the model combined with LSTMs, which prove the irreplaceability of LSTMs.

4.3 Results and discussion

Table 3. Examples of original sentence, compressed sentence, and its predicted compresseion

Source：	A man charged with killing his father in a fight outside a busy Orlando restaurant was just cleared by a grand jury .
Ground truth：	A man charged with killing his father was cleared by a grand jury.
Seq2Seq：	A man charged with killing his father in a fight was cleared .
Stacked model (k=1):	A man charged with killing his father was cleared by a grand jury.
Parallel model (k=1)：	A man charged with killing his father was cleared by a jury.

To further analyze the experimental results, Table 3 shows examples of the original sentence, the compressed sentence, and its predicted compression in this paper.

Seen from the example sentences, although the output of the base Seq2Seq model is syntactically coherent, it lacks key information such as "*by a grand jury*". In the model with GCNs, either the stacked model or the parallel model contains this information. As observed in Figure 1, the word "*cleared*" directly connected to the root is retained with high probability, and the phrase "*by a grand jury*" directly connected by the dependent arc also has a greater retention propensity, retaining in the compressed sentence. This is in line with the experimental expectations of this paper and is confirmed during the evaluation phase.

From the results of the overall evaluation, the performance of parallel model is better than stacked model. We think this may be because that the direct utilization of GCNs leads to an increase in compression ratio and fits the real data better.

5 Conclusion

In this paper, we introduce and improve the graph convolutional neural network in the sentence compression task, and explore a variety of its combinations with the sequence-to-sequence model. The experiment results show that the model based on the graph convolutional neural network can obtain better compression effect, have different degrees of improvement in F1 and accuracy score, and further improve the compression ratio.

Reference

1. Jing H. Sentence reduction for automatic text summarization[C]//Sixth Applied Natural Language Processing Conference. 2000.
2. McDonald R. Discriminative sentence compression with soft syntactic evidence[C]//11th Conference of the European Chapter of the Association for Computational Linguistics. 2006.
3. Clarke J, Lapata M. Global inference for sentence compression: An integer linear programming approach[J]. Journal of Artificial Intelligence Research, 2008, 31: 399-429.
4. Filippova K, Altun Y. Overcoming the lack of parallel data in sentence compression[J]. 2013.
5. Filippova K, Alfonseca E, Colmenares C A, et al. Sentence compression by deletion with lstms[C]//Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015: 360-368.
6. Zhao Y, Senuma H, Shen X, et al. Gated neural network for sentence compression using linguistic knowledge[C]//International Conference on Applications of Natural Language to Information Systems. Springer, Cham, 2017: 480-491.
7. Wang L, Jiang J, Chieu H L, et al. Can syntax help? Improving an LSTM-based sentence compression model for new domains[C]//Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2017: 1385-1393.
8. Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint arXiv:1609.02907, 2016.
9. Marcheggiani D, Titov I. Encoding sentences with graph convolutional networks for semantic role labeling[J]. arXiv preprint arXiv:1703.04826, 2017.
10. Knight K, Marcu D. Statistics-based summarization-step one: Sentence compression[J]. AAAI/IAAI, 2000, 2000: 703-710.
11. Zhao Y, Luo Z, Aizawa A. A Language Model based Evaluator for Sentence Compression[C]//Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). 2018: 170-175.
12. Fevry T, Phang J. Unsupervised Sentence Compression using Denoising Auto-Encoders[J]. arXiv preprint arXiv:1809.02669, 2018.
13. Kamigaito H, Hayashi K, Hirao T, et al. Higher-Order Syntactic Attention Network for Longer Sentence Compression[C]//Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). 2018: 1716-1726.
14. Chen Y, Pan R. Automatic emphatic information extraction from aligned acoustic data and its application on sentence compression[C]//Thirty-First AAAI Conference on Artificial Intelligence. 2017.

15. Tran N T, Luong V T, Nguyen N L T, et al. Effective attention-based neural architectures for sentence compression with bidirectional long short-term memory[C]//Proceedings of the Seventh Symposium on Information and Communication Technology. ACM, 2016: 123-130.
16. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[J]. arXiv preprint arXiv:1409.0473, 2014.
17. Dauphin Y N, Fan A, Auli M, et al. Language modeling with gated convolutional networks[C]//Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017: 933-941.
18. Klerke S, Goldberg Y, Søgaard A. Improving sentence compression by learning to predict gaze[J]. arXiv preprint arXiv:1604.03357, 2016.
19. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The Journal of Machine Learning Research, 2014, 15(1): 1929-1958.
20. Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.