Computers, Materials & Continua CMC, vol.61, no.2, pp.569-581, 2019

Keyphrase Generation Based on Self-Attention Mechanism

Kehua Yang[1, *], Yaodong Wang[1], Wei Zhang[1], Jiqing Yao[2] and Yuquan Le[1]

Abstract: Keyphrase greatly provides summarized and valuable information. This information can help us not only understand text semantics, but also organize and retrieve text content effectively. The task of automatically generating it has received considerable attention in recent decades. From the previous studies, we can see many workable solutions for obtaining keyphrases. One method is to divide the content to be summarized into multiple blocks of text, then we rank and select the most important content. The disadvantage of this method is that it cannot identify keyphrase that does not include in the text, let alone get the real semantic meaning hidden in the text. Another approach uses recurrent neural networks to generate keyphrases from the semantic aspects of the text, but the inherently sequential nature precludes parallelization within training examples, and distances have limitations on context dependencies. Previous works have demonstrated the benefits of the self-attention mechanism, which can learn global text dependency features and can be parallelized. Inspired by the above observation, we propose a keyphrase generation model, which is based entirely on the self-attention mechanism. It is an encoder-decoder model that can make up the above disadvantage effectively. In addition, we also consider the semantic similarity between keyphrases, and add semantic similarity processing module into the model. This proposed model, which is demonstrated by empirical analysis on five datasets, can achieve competitive performance compared to baseline methods.

Keywords: Keyphrase generation, self-attention mechanism, encoder-decoder framework.

1 Introduction

With the explosive growth of text information in recent years, people can access a large amount of text information every day, such as news, web pages, chat records, papers and so on. Extracting important content from a large amount of text has become an urgent need. Automatically extracting keyphrase provides an efficient solution. Keyphrase can provide users with highly condensed and valuable information that summarizes the major semantic meaning of longer texts. According to keyphrases, users can easily and accurately grasp the main content of the text, which not only saves time, but also

[1] College of Computer Science and Electronic Engineering and Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, Changsha, 410082, China.

[2] Oath Verizon Company, Manhattan, New York, 10007, USA.

[*] Corresponding Author: Kehua Yang. Email: khyang@hnu.edu.cn.

improves the efficiency of reading. Keyphrase extraction has been used in lots of fields, such as, text categorization [Hulth and Megyesi (2006)], text summarization [Zhang, Zincir-Heywood and Milios (2004)], information retrieval [Jones and Staveley (1999)], opinion mining [Berend (2011)], and retrieving encrypted data in cloud server [Tang, Lian, Zhao et al. (2018)]. Here, we think that both "keyword" and "keyphrase" mean the same thing, and there is no difference between them. In addition, we have to pay attention that "keyphrase" and "keyword" can be made of multiple words.

There are enormous usefulness of keyphrases, therefore, lots of studies have been done on the generation or automatic extraction of keyphrases, including supervised classification-based methods [Wang, Peng and Hu (2006)], ranking-based methods [Jiang, Hu and Li (2009)], clustering-based methods [Danilevsky, Wang, Desai et al. (2014)] and recurrent neural networks-based encoder-decoder methods [Meng, Zhao, Han et al. (2017); Zhang and Xiao (2018)]. The first three methods do not extract keywords from the semantic layer, which are extraction methods. They can only obtain keyphrases appearing in the source text, but they cannot obtain absent keyphrases [Meng, Zhao, Han et al. (2017)]. The last method uses recurrent neural networks as its encoder and decoder, which can generate present and absent keyphrases from text semantics and is a generation method. However, recurrent neural networks rely on the state of current and previous time steps. This essentially sequential nature excludes parallelism, and it takes a number of time steps to accumulate dependency features information between long-distance words, and the longer distance, the less possibility it is to capture dependency features information effectively. In order to overcome these shortcomings, we propose an encoder-decoder model that abandons the recurrent neural network and entirely relies on attention mechanism. The model creates global dependencies between input and output and can generate present and absent keyphrases. It solves the problem of weakening dependence caused by the long distance between words, has the property of parallelization, and considers the semantic correlation between keyphrases. To summarize, the main contributions of our research are as below:

1) A keyphrase generation model is proposed, which is entirely based on self-attention mechanism. As far as we know, this is the first time that self-attention mechanism applies to a keyphrase generation task.
2) We consider the semantic similarity between keyphrases and add semantic similarity processing module into the model.
3) We compare six important supervised and unsupervised models, and the results show that our model performs better.

In the rest part of the article, firstly, the related work will be shown in Section 2. Afterwards, we introduce our model in detail in Section 3. Then, we experimentally demonstrate the performance of our model in Section 4. At last, in Section 5, conclusions and future work will be given.

2 Related work

Due to the important role of keyphrases, there are many methods to extract keywords, and they usually have two steps for extracting keyphrases. The first step is to extract

keyword candidates from the source text and form a list. The longer length of the list, the greater the probability of getting the correct keyphrases. The methods for extracting candidates mainly include sequence matching [Le, Le Nguyen and Shimazu (2016)] and extract important n-gram phrases [Medelyan, Witten and Milne (2008)]. The second step is to sort keyword candidates according to the significance of the source text. The final result is the top keyphrases in the list of candidates. There are two main methods to rank the list of candidates: unsupervised and supervised. The former methods include finding the central nodes of text graph [Grineva, Grinev and Lizorkin (2009)], topical clusters [Liu, Huang, Zheng et al. (2010)] and so on. The latter takes it as a binary classification problem based on positive fractional ranking [Gollapalli and Caragea (2014)]. However, in scientific publications, instead of following the written content, authors usually assign keyphrases based on their semantics. Therefore, the above methods only judge the importance of the candidate keyphrases to the text according to the number of occurrences, and cannot reveal the complete semantics of the document content.

In addition to the above extraction methods, some scholars use a completely different way to generate keyphrases. Liu et al. [Liu, Chen, Zheng et al. (2011)] adopted a word alignment model, which translates the documents to the keyphrases. To a certain degree, this model reduces the vocabulary gaps between source and target. Zhang et al. [Zhang, Wang, Gong et al. (2016)] built a sequence labeling model, which extracted keywords by recurrent neural network from tweets. However, this model cannot fully understand the semantics behind the source text. Meng et al. [Meng, Zhao, Han et al. (2017)] proposed an encoder-decoder model, which combines attention mechanism and copy mechanism to generate keyphrases. However, this approach does not consider the semantic similarity between keyphrases, and when the sequence length is too long, it is difficult to capture the dependency features between words. The longer the distance, the less the possibility of effective capture. On the basis of Meng et al. [Meng, Zhao, Han et al. (2017)], Zhang et al. [Zhang and Xiao (2018)] used the coverage mechanism to solve the semantic similarity between keyphrases. However, the problem of weak dependence between long distance words and the inability to parallelize training still exist.

To overcome the shortcomings above, a keyphrase generation model is proposed, which is entirely based on self-attention mechanism. It abandons the recurrent neural network and takes into account the semantic similarity between keyphrases.

3 Methodology

Our keyword generation model includes an encoder, a decoder and a semantic similarity processing module. The encoder and decoder are shown in the upper and lower halves of Fig. 1, respectively. Given a source text, the encoder is used to learn the global dependency information between the words, the decoder combines the dependency information to generate the corresponding keyphrase, and the semantic similarity module handles the semantic similarity between the keyphrases.

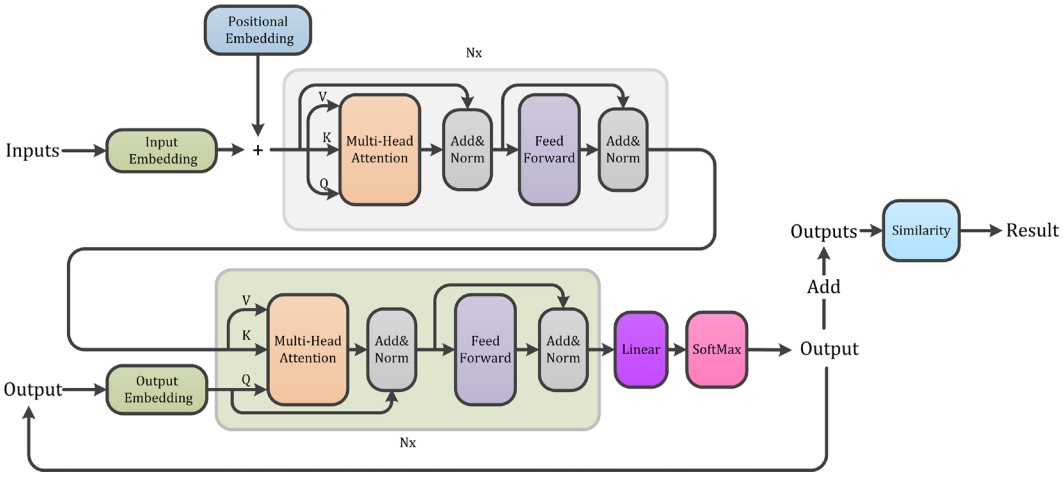

Figure 1: Our model architecture

3.1 Problem definition

For the keyphrase generation model, the task is to generate multiple keyphrases to summarize the source text by machine automatically. We suppose that dataset consists of N different samples, and the $\left(x^i, k^i\right)$ represents the *i*-th sample, which is composed of a source text x^i and n_i target keyphrases. We describe these target keyphrases as $k^i = \left(k^{(i,1)}, k^{(i,2)}, \ldots, k^{(i,n_i)}\right)$. Also both source text x^i and keyphrase $k^{(i,j)}$ are regarded as word sequences as below:

$$x^i = \left(x_1^i, x_2^i, \ldots, x_{l_{x^i}}^i\right) \tag{1}$$

$$k^{(i,j)} = \left(y_1^{(i,j)}, y_2^{(i,j)}, \ldots y_{l_{k^{(i,j)}}}^{(i,j)}\right) \tag{2}$$

l_{x^i} and $l_{k^{(i,j)}}$ respectively indicate the length of word sequence x^i and $k^{(i,j)}$ respectively. We should pay attention that the keyphrase sequences only contain a few words and are relatively short.

As mentioned above, each sample contains a source text x^i and multiple keyphrases k^i. We use an encoder and a decoder to generalize the hidden rules and learn the mapping relationship between the target keyphrases and source text. The source text sequence x^i is used as input, and the dependencies between each word are learned by the multi-attention mechanism in the encoder to obtain the global structure information of the source text. The previous one keyphrase $k^{(i,j)}$ that has been obtained is used as an additional input to the decoder, and then the decoder outputs the next keyphrase $k^{(i,j+1)}$ according to the global dependency information learnt by the encoder.

3.2 Encoder

Our encoder is basically the same as Vaswani et al. [Vaswani, Shazeer, Parmar et al. (2017)]. First, the input source text is transformed into vector matrix through input embedding. Because our keyphrase generation model consists of no convolution and no recurrence, in order to making use of the sequence order, we must inject some information about the absolute position or the relative of the words into the sequence. The same as Vaswani et al. [Vaswani, Shazeer, Parmar et al. (2017)], the "positional encoding" is added to the input embedding at the bottom of the encoder. Then, the results of their summation are input into the multi-layer multi-head attention mechanism and feed-forward network which is fully connected, and a sequence of consecutive representations is finally obtained. In order to preserve the original input information as much as possible, a residual connection [He, Zhang, Ren et al. (2016)] is adopted by every multi-head attention mechanism and feed-forward network, and then perform layer standardization [Ba, Kiros and Hinton (2016)].

3.2.1 Positional encoding

The self-attention mechanism is unable to distinguish different locations. To take advantage of the position information of the sequence, it is very significant to encode the position of every input word. There are many ways to encode the position. The simplest way is to use additional position embedding. In this work, we try the positional encoding approach proposed by Vaswani et al. [Vaswani, Shazeer, Parmar et al. (2017)], which is formulated as follows:

$$PE_{(pos,2i)} = \sin\left(pos / 10000^{2i/d_{model}} \right) \tag{3}$$

$$PE_{(pos,2i+1)} = \cos\left(pos / 10000^{2i/d_{model}} \right) \tag{4}$$

where pos is the position and i is the dimension. The positional encoding is simply added to the input embedding. Unlike the positional embedding approach, this approach does not introduce additional parameters.

After input embedding and positional encoding, our input source text is transformed into a matrix X, where $X \in R^{d_n \times d_{model}}$, d_n is the length of the input text, and d_{model} is the dimension of the embedding.

3.2.2 Multi-head attention

The attention mechanism has been widely applied to various tasks of natural language processing based on deep learning. We think a mapping of a query to a series of $\langle \text{key}, \text{value} \rangle$ data pairs can best describe the nature of the attention mechanism. Given a certain element $Query$ in the target, we calculate the similarity or correlation between it and the Key of each data pair in the $Source$ to obtain the weight coefficient of the corresponding $Value$, and then weight and sum the $Value$ according to the weight coefficient respectively, and the result of the summation is the attention value.

$$\text{Attention}\left(Query, Source\right) = \sum_{i=1}^{lx} Similarity\left(Query, Key_i\right) * Value_i \tag{5}$$

where lx represents the length of the $Source$, where the similarity or correlation of $Query$ and Key can be calculated by dot-product.

In this article, Q, K and V all represent the matrix of the $d_n \times d_{model}$ after input embedding and positional encoding. In order to better learn the global dependency information within the text, we use the multi-head attention proposed by Vaswani et al. [Vaswani, Shazeer, Parmar et al. (2017)]. It proved to be useful to project the Q, K and V matrices linearly h times with different, which respectively learned different linear projection to $d_n \times d_k$, $d_n \times d_k$ and $d_n \times d_v$ matrices. We perform the attention function parallel on each of these matrices of Q, K and V, resulting $d_n \times d_v$ output matrix. Then we connect the matrices and project them once again, yielding the final results, as presented in Fig. 2.

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{6}$$

$$MultiHead\left(Q, K, V\right) = Concat\left(head_1, \ldots, head_h\right)W^O \tag{7}$$

where the projections are parameter matrices $W_i^Q \in R^{d_{model} \times d_k}$, $W_i^K \in R^{d_{model} \times d_k}$, $W_i^V \in R^{d_{model} \times d_v}$ and $W^O \in R^{hd_v \times d_{model}}$.

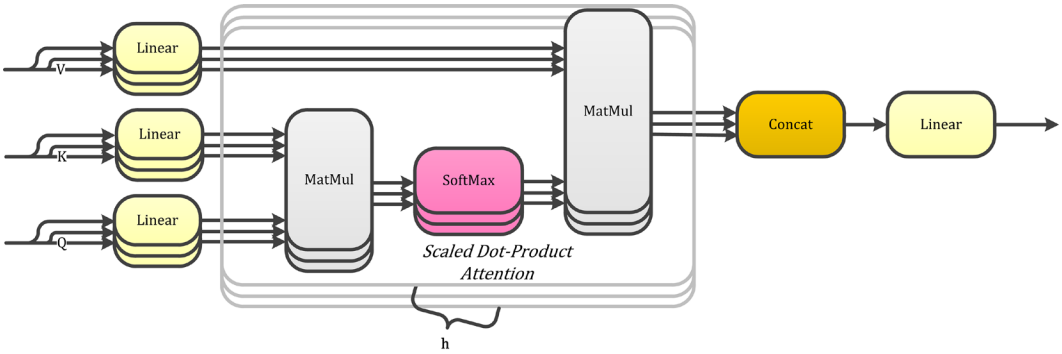

Figure 2: Multi-head attention

After projecting Q, K and V to subspaces of lower dimension, and then performing attention calculations in different subspaces, the subspace's dimension is lower without increasing the computational amount, which is conducive to parallelizing and capturing features of different subspaces.

If Q and K after linear projection are directly subjected to dot product operations, and the operation result is used as a weighting coefficient, there may be a problem that the dot product is too large and the gradient is too small. We try the method of Vaswani et al.

[Vaswani, Shazeer, Parmar et al. (2017)], scaling the dot product by $\frac{1}{\sqrt{d_k}}$, which is expressed as follows:

$$\text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) = softmax\left(\frac{QW_i^Q\left(KW_i^K\right)^T}{\sqrt{d_k}}\right)VW_i^V \tag{8}$$

where Q, K, $V \in R^{d_n \times d_{model}}$.

The self-attention mechanism can be expressed as $X = Q = K = V$, which means that attention mechanism works inside the sequence to find internal interdependencies. The self-attention mechanism solves the problem of weak dependence caused by the text being too long. So our multi-head attention can be simply described as:

$$Y = MultiHead\left(X, X, X\right) \tag{9}$$

where Y is the final value after connection and projection, $Y \in R^{d_n \times d_{model}}$.

3.2.3 Feed forward

In order to fit the training data better, we add a fully connected feed forward neural network layer to the encoder. It contains two linear layers and an intermediate nonlinear ReLU hidden layer. Formally, we have the following equation:

$$FFN\left(Y\right) = ReLU\left(YW_1 + b_1\right)W_2 + b_2 \tag{10}$$

where $W_1 \in R^{d_{model} \times d_f}$, $W_2 \in R^{d_f \times d_{model}}$, $Y \in R^{d_n \times d_{model}}$.

3.3 Decoder

Since the model of Vaswani et al. [Vaswani, Shazeer, Parmar et al. (2017)] is used to solve translation problem, they use the "Masked Multi-Head Attention" sublayer in the decoder to prevent future output words from being used during training, ensuring that the prediction of position *i* depends only on all known outputs less than *i*. However, our decoder part is different from them. Because the keyphrases we generated are short and independent, and there is no interdependence between them, our decoder only uses the keyphrase from the last time output, and then embedding it as the Q input of multi-head attention.

When we make predictions, we enter the source text x^i consisting of an abstract and a title at the encoder. After a series of processing by the encoder, we get a matrix Z, $Z \in R^{d_n \times d_{model}}$ which is used as the K and V input of the multi-head attention in the decoder, ie $Z = K = V$. In the decoder, the keyphrase $k^{(i,j)}$ outputted by the previous layer is processed by embedding to obtain a matrix which is used as another input Q of the multi-head attention layer in the decoder, $Q \in R^{d_m \times d_{model}}$, d_m is the length of the

keyphrase $k^{(i,j)}$. Q combines K, V to perform the multi-head attention and feed-forward neural network. Similar to the encoder, each sub-layer uses a residual connection [He, Zhang, Ren et al. (2016)], and then performs layer standardization [Ba, Kiros and Hinton (2016)]. Using the multi-head attention and feed-forward neural network stack Nx times to get the matrix S, $S \in R^{d_m \times d_{model}}$, and then through the linear layer and the softmax layer, predict the next most probable keyphrase $k^{(i,j+1)}$.

The linear layer is a neural network, which is simple and fully connected. It projects the matrix S, which is produced by the stack of decoder, into a very much larger vector. This vector is called logits vector. We suppose that our model could learn from its training dataset and know 10,000 unique keyphrases, which turns the logits vector into 10,000 cells wide. These cells are corresponding to the score of a keyphrase respectively. Then the softmax layer makes those scores probabilities, all of which are positive and have a sum of 1.0. The next keyphrase be produced, which is associated with the cell with the highest possibility.

Since the multi-head attention and the feed-forward neural network that used in our decoder have already been introduced in the encoder, we don't repeat them here.

3.4 Semantic similarity of keyphrases

In order to consider the semantic information between generated keyphrases, we add a semantic similarity module behind the encoder-decoder.

Firstly, we segment each keyphrase, then use Word2vec [Mikolov, Chen, Corrado et al. (2013)] to calculate the word vector of each word, and then add the word vectors in each keyphrase to get the average word vector as the current keyphrase. $w^i = \left(v_1^i, v_2^i, \ldots, v_n^i\right)$ indicates the word vector of the *i*-th keyphrase. We use the method of cosine similarity to calculate the similarity between keyphrases.

$$\mathrm{SIM}\left(w^o, w^p\right) = \frac{\sum_{i=1}^{n}\left(v_i^o \times v_i^p\right)}{\sqrt{\sum_{i=1}^{n}\left(v_i^o\right)^2} \times \sqrt{\sum_{i=1}^{n}\left(v_i^p\right)^2}} \tag{11}$$

where $\mathrm{SIM}\left(w^o, w^p\right)$ represents the cosine similarity of the *o*-th and *p*-th keyphrases.

4 Experiments

We introduce our experimental designs in this section. First, we describe the training dataset and the testing datasets. Then we introduce the baseline models and evaluation metrics. Finally, we give the experimental results and analysis.

4.1 Training dataset

As far, for evaluating keyphrase generation, we have a couple of publicly available datasets. The largest publicly available dataset is provided by Meng et al. [Meng, Zhao, Han et al. (2017)]. We use it to train and test our model. We mark it with "KP" in the later

description. The KP dataset contains plenty of high-quality computer science articles, including nearly 567,830 articles. We randomly select 40k articles to test and validate our model, and the rest of them were used as training dataset. About this 40k articles, 20k articles are used as a testing dataset, we mark it with KP20k, and the remaining 20k articles are regarded as validation dataset to judge the convergence of our model during the training process.

4.2 Testing dataset

In order to comprehensively evaluate the proposed keypharse generation model, we not only use the newly constructed test dataset KP20k to test our model, but also use other four scientific article datasets. The title and abstract of the article are used as source text. All the testing datasets and details are listed below:

1) **Inspec** [Hulth (2003)]: The abstracts of 2,000 articles are provided. We randomly select 500 abstracts as our testing dataset.
2) **Krapivin** [Krapivin, Autaeu and Marchese (2009)]: About 2,304 full-text papers and author-assigned keyphrases are provided. The abstracts of 400 articles among them are used as our testing dataset.
3) **NUS** [Nguyen and Kan (2007)]: The dataset provides 211 full-text papers and author-assigned keyphrases. We use all of them to test our model.
4) **SemEval-2010** [Kim, Medelyan, Kan et al. (2013)]: This dataset consists of 288 articles from the ACM Digital Library. We randomly select 100 papers to test our model.
5) **KP20k** [Meng, Zhao, Han et al. (2017)]: The dataset provides 20K scientific literatures about computer science with titles, abstracts and keyphrases.

Since the number of **Krapivin**, **Inspec**, **SemEval-2010** and **NUS** is too small to train our powerful model. Therefore, they only are used as testing dataset.

4.3 Baseline models

We use four supervised algorithms (Maui [Medelyan, Frank and Witten (2009)], RNN [Meng, Zhao, Han et al. (2017)], CopyRNN [Meng, Zhao, Han et al. (2017)], CovRNN [Zhang and Xiao (2018)]) and two unsupervised algorithms (TextRank [Mihalcea and Tarau (2004)], TF-IDF) as baselines. Zhang et al. [Zhang and Xiao (2018)] have experimentally verified these supervised and unsupervised methods, and we take the results of their experiments as the standard.

4.4 Evaluation metric

To measure the performance of our model, evaluation metrics consist of recall, macro-averaged precision and F-measure (F_1). The recall represents a comparison between the quantity of correctly-predicted keywords and the total quantity of sample keywords, while precision represents a comparison between the quantity of correctly-predicted keywords and the total quantity of predicted keywords. In fact, precision and recall are contradictory in some cases. In order to consider them comprehensively, we use the

comprehensive evaluation metric F_1 to measure the performance of the model when predicting present keyphrases.

$$F_1 = \frac{2P \times R}{P + R} \tag{12}$$

where R represents the recall and P represents the precision.

4.5 Results and analysis

To evaluate our model, a study on two different tasks is conducted. We classify keywords into two categories according to whether they appear in the source text: present keyphrase and absent keyphrase.

4.5.1 Predicting present keyphrases

To test the performance of our model on regular tasks, we compared it to six baseline models (TextRank, TF-IDF, RNN, Maui, CovRNN, CopyRNN). To be fair, we only consider models to predict the performance of present keyphrases. We use F_1 at top 4 and top 8 to measure the performance of each method. We highlight in bold for the best scores among the five benchmark datasets.

Table 1: Each model predicts the performance results of the present keyphrases

Method	Inspec		Krapivin		NUS		SemEval		KP20k	
	$F_1@4$	$F_1@8$	$F_1@4$	$F_1@8$	$F_1@4$	$F_1@8$	$F_1@4$	$F_1@8$	$F_1@4$	$F_1@8$
TF-IDF	0.223	0.317	0.130	0.167	0.141	0.183	0.126	0.190	0.111	0.131
TextRank	0.224	0.271	0.171	0.152	0.183	0.191	0.171	0.181	0.174	0.142
Maui	0.037	0.041	0.247	0.216	0.242	0.271	0.041	0.037	0.271	0.234
RNN	0.084	0.061	0.133	0.087	0.154	0.152	0.143	0.112	0.179	0.191
CopyRNN	0.271	0.340	0.310	0.256	0.320	0.316	0.292	0.294	0.321	0.260
CovRNN	0.280	0.350	0.312	0.257	0.321	0.340	0.301	0.295	0.323	0.270
Our model	**0.289**	**0.358**	**0.320**	**0.262**	**0.327**	**0.346**	**0.310**	**0.301**	**0.332**	**0.279**

The results of the experiment show that the four supervised models (Maui, RNN, CopyRNN, CovRNN) have a robust performance across different datasets. Overall, among all baseline models, CovRNN works extremely well. In KP20k dataset, its $F_1@4$ and $F_1@8$ values reach 0.323 and 0.270 respectively. These baseline models achieve good results from the study, but our proposed model based on self-attention mechanism performs the best.

In the predicting present keyphrases task, although our model with the self-attention mechanism performs best, it does not work well as we expected. For instance, the $F_1@8$ value of the CovRNN reach 0.270 on KP20k dataset, our model is 0.279. It probably

because that our model pays high attention to find the hidden semantic information behind the source text, which may result in generating general keyphrases that cannot be referenced from the source text.

4.5.2 Predicting absent keyphrases

It is an important concern to predict the absent keyphrases based on "comprehension" of content. It is worth mentioning that the prediction task is extreme challenging, for our knowledge, it can only be done by the seq2seq model. Therefore, only CopyRNN, CovRNN and our model experiments can be performed.

Here, in order to test the performance of each model to predict the absent keyphrases, we use the recall of top 10 and top 50 as evaluation metrics. The recall metric assists us analyze the predictive performance of each model. Tab. 2 shows the performance of CopyRNN, CovRNN and our model for predicting absent keyphrase. The best prediction results are shown in bold.

Table 2: Absent keyphrases prediction performance of CopyRNN, CovRNN and our model on five datasets

Dataset	CopyRNN		CovRNN		Our model	
	R@10	R@50	R@10	R@50	R@10	R@50
Inspec	0.045	0.102	0.048	0.113	**0.057**	**0.120**
Krapivin	0.115	0.191	0.131	0.202	**0.142**	**0.218**
NUS	0.059	0.118	0.064	0.121	**0.071**	**0.134**
SemEval	0.045	0.069	0.049	0.073	**0.051**	**0.082**
KP20k	0.125	0.191	0.129	0.213	**0.141**	**0.232**

In Tab. 2, the recall of CopyRNN, CovRNN and our model can reach about 7.8% (13.4%), 8.4% (14.4%) and 9.2% (15.7%) of accurate keyphrases at top 10 (50) predictions. In addition, our model shows better performance than CopyRNN and CovRNN. This performance shows that our model can better extract the semantic meaning hidden in the text.

Our proposed model completely relies on the self-attention mechanism to learn the global dependence information of the text, more comprehensively considers the dependencies between words, and solves the problem of weakening dependence caused by the long distance between words, has the property of parallelization. In addition, we consider the semantic relevance between keyphrases. The experimental results show the important role of the global dependence information of the text on keyphrase generation. This progress also shows that self-attention mechanism is extraordinary significant in the task of absent keyphrase prediction.

5 Conclusion and future work

In our work, we propose a keyphrase generation model based entirely on the self-attention mechanism, which is an encoder-decoder framework. In addition, we add the

semantic similarity module to our model. As far as we know, this is the first time that self-attention mechanism applies to a keyphrase generation task. Our model relies entirely on the self-attention mechanism to learn the global dependence information of text, without the limitation of the distance between words, has the property of parallelization, and can generate absent keyphrases based on text semantics. The effectiveness of our proposed model is demonstrated in the comprehensive empirical studies.

In the future, we consider applying our model to different types of testing datasets to test their performance. In addition, we also consider exploring a new way to deal with semantic correlation between keyphrases.

Acknowledgement: This work is supported by National Key R&D Program of China (No.2018YFC0831800) and Innovation Base Project for Graduates (Research of Security Embedded System).

References

Ba, J. L.; Kiros, J. R.; Hinton, G. E. (2016): Layer normalization. *arXiv:1607.06450*.

Berend, G. (2011): Opinion expression mining by exploiting keyphrase extraction. *5th International Joint Conference on Natural Language Processing*, pp. 1162-1170.

Danilevsky, M.; Wang, C.; Desai, N.; Ren, X.; Guo, J. et al. (2014): Automatic construction and ranking of topical keyphrases on collections of short documents. *SIAM International Conference on Data Mining*, pp. 398-406.

Gollapalli, S. D.; Caragea, C. (2014): Extracting keyphrases from research papers using citation networks. *Association for the Advance of Artificial Intelligence*, pp. 1629-1635.

Grineva, M.; Grinev, M.; Lizorkin, D. (2009): Extracting key terms from noisy and multitheme documents. *International Conference on World Wide Web*, pp. 661-670.

He, K.; Zhang, X.; Ren, S.; Sun, J. (2016): Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770-778.

Hulth, A. (2003): Improved automatic keyword extraction given more linguistic knowledge. *Conference on Empirical Methods in Natural Language Processing*, pp. 216-223.

Hulth, A.; Megyesi, B. B. (2006): A study on automatically extracted keywords in text categorization. *International Conference on Computational Linguistics and Meeting of the Association for Computational Linguistics*, pp. 537-544.

Jiang, X.; Hu, Y.; Li, H. (2009): A ranking approach to keyphrase extraction. *32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 756-757.

Jones, S.; Staveley, M. S. (1999): Phrasier: a system for interactive document retrieval using keyphrases. *22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 160-167.

Kim, S. N.; Medelyan, O.; Kan, M. Y.; Baldwin, T. (2013): Automatic keyphrase extraction from scientific articles. *Language Resources and Evaluation*, vol. 47, no. 3, pp.

723-742.

Krapivin, M.; Autaeu, A.; Marchese, M. (2009): *Large Dataset for Keyphrases Extraction*. University of Trento, Trento.

Le, T. T. N.; Le Nguyen, M.; Shimazu, A. (2016): Unsupervised keyphrase extraction: introducing new kinds of words to keyphrases. *Australasian Joint Conference on Artificial Intelligence*, pp. 665-671.

Liu, Z.; Chen, X.; Zheng, Y.; Sun, M. (2011): Automatic keyphrase extraction by bridging vocabulary gap. *Fifteenth Conference on Computational Natural Language Learning*, pp. 135-144.

Liu, Z.; Huang, W.; Zheng, Y.; Sun, M. (2010): Automatic keyphrase extraction via topic decomposition. *Conference on Empirical Methods in Natural Language Processing*, pp. 366-376.

Medelyan, O.; Frank, E.; Witten, I. H. (2009): Human-competitive tagging using automatic keyphrase extraction. *Conference on Empirical Methods in Natural Language Processing*, pp. 1318-1327.

Medelyan, O.; Witten, I. H.; Milne, D. (2008): Topic indexing with wikipedia. *AAAI WikiAI Workshop*, vol. 1, pp. 19-24.

Meng, R.; Zhao, S.; Han, S.; He, D.; Brusilovsky, P. et al. (2017): Deep keyphrase generation. *arXiv:1704.06879*.

Mihalcea, R.; Tarau, P. (2004): Textrank: bringing order into text. *Conference on Empirical Methods in Natural Language Processing*, pp. 404-411.

Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. (2013): Efficient estimation of word representations in vector space. *arXiv:1301.3781*.

Nguyen, T. D.; Kan, M. Y. (2007): Keyphrase extraction in scientific publications. *International Conference on Asian Digital Libraries*, pp. 317-326.

Tang, Y.; Lian, H.; Zhao, Z.; Yan, X. (2018): A proxy re-encryption with keyword search scheme in cloud computing. *Computers, Materials & Continua*, vol. 56, no. 2, pp. 339-352.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L. et al. (2017): Attention is all you need. *Advances in Neural Information Processing Systems*, pp. 5998-6008.

Wang, J.; Peng, H.; Hu, J. S. (2006): Automatic keyphrases extraction from document using neural network. *Advances in Machine Learning and Cybernetics*, pp. 633-641.

Zhang, Q.; Wang, Y.; Gong, Y.; Huang, X. (2016): Keyphrase extraction using deep recurrent neural networks on twitter. *Conference on Empirical Methods in Natural Language Processing*, pp. 836-845.

Zhang, Y.; Xiao, W. (2018): Keyphrase generation based on deep seq2seq model. *IEEE Access*, vol. 6, pp. 46047-46057.

Zhang, Y.; Zincir-Heywood, N.; Milios, E. (2004): World wide web site summarization. *Web Intelligence and Agent Systems: An International Journal*, vol. 2, no. 1, pp. 39-53.