

# DivGraphPointer: A Graph Pointer Network for Extracting Diverse Keyphrases

Zhiqing Sun  
Peking University  
Beijing, China  
1500012783@pku.edu.cn

Jian Tang  
Mila-Quebec Institute for Learning  
Algorithms  
HEC Montréal  
CIFAR AI Research Chair  
Montréal, Canada  
jian.tang@hec.ca

Pan Du  
Université de Montréal  
Montréal, Canada  
pandu@iro.umontreal.ca

Zhi-Hong Deng  
Peking University  
Beijing, China  
zhdeng@pku.edu.cn

Jian-Yun Nie  
Université de Montréal  
Montréal, Canada  
nie@iro.umontreal.ca

## ABSTRACT

Keyphrase extraction from documents is useful to a variety of applications such as information retrieval and document summarization. This paper presents an end-to-end method called DivGraphPointer for extracting a set of diversified keyphrases from a document. DivGraphPointer combines the advantages of traditional graph-based ranking methods and recent neural network-based approaches. Specifically, given a document, a word graph is constructed from the document based on word proximity and is encoded with graph convolutional networks, which effectively capture document-level word salience by modeling long-range dependency between words in the document and aggregating multiple appearances of identical words into one node. Furthermore, we propose a diversified point network to generate a set of diverse keyphrases out of the word graph in the decoding process. Experimental results on five benchmark data sets show that our proposed method significantly outperforms the existing state-of-the-art approaches.

## CCS CONCEPTS

• **Information systems** → **Information retrieval**; *Information extraction*; Information retrieval diversity; Summarization.

## KEYWORDS

graph neural networks, keyphrase extraction, diversified pointer network, document-level word salience

## ACM Reference Format:

Zhiqing Sun, Jian Tang, Pan Du, Zhi-Hong Deng, and Jian-Yun Nie. 2019. DivGraphPointer: A Graph Pointer Network for Extracting Diverse Keyphrases. In *Proceedings of the 42nd International ACM SIGIR Conference on Research*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

SIGIR '19, July 21–25, 2019, Paris, France

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-6172-9/19/07...\$15.00

<https://doi.org/10.1145/3331184.3331219>

and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3331184.3331219>

## 1 INTRODUCTION

Keyphrase extraction from documents is useful in a variety of tasks such as information retrieval [20], text summarization [34], and question answering [24]. It allows to identify the salient contents from a document. The topic has attracted a large amount of work in the literature.

Most traditional approaches to keyphrase extraction are unsupervised approaches. They usually first identify the candidate keyphrases with some heuristics (e.g., regular expressions), and then rank the candidate keyphrases according to their importance in the documents [14]. Along this direction, the state-of-the-art algorithms are graph-based ranking methods [25, 30, 43], which first construct a word graph from a document and then determine the importance of the keyphrases with random walk based approaches such as PageRank [5]. By constructing the word graph, these methods can effectively identify the most salient keyphrases. Some diversification mechanisms have also been investigated in some early work [4, 27] to address the problem of over-generation of the same concepts in keyphrase extraction. However, these methods are fully unsupervised. They rely heavily on manually designed heuristics, which may not work well when applied to a different type of document. In experiments, we also observe that the performance of these methods is usually limited and inferior to the supervised ones.

Recently, end-to-end neural approaches for keyphrase extraction have been attracting growing interests [29, 46, 47]. The neural approaches usually studied keyphrase extraction in the encoder-decoder framework [39], which first encodes the input documents into vector representations and then generates the keyphrases with Recurrent Neural Networks (RNN) [31] or CopyRNN [13] decoders conditioned on the document representations. These neural methods have achieved state-of-the-art performance on multiple benchmark data sets with end-to-end supervised training. The end-to-end training offers a great advantage that the extraction process can

adapt to the type of documents. However, compared to the unsupervised graph-based ranking approaches, existing end-to-end approaches only treat documents as sequences of words. They do not benefit from the a more global graph structure that provides useful document-level word salience information such as long-range dependencies between words, as well as a synthetic view on the multiple appearances of identical words in the document. Another problem of these end-to-end methods is that they cannot guarantee the diversity of the extracted key phrases: it is often the case that several similar keyphrases are extracted. Therefore, we are seeking an approach that can have the advantage of modeling document-level word salience, generating diverse keyphrases, and meanwhile be efficiently trained in an end-to-end fashion.

In this paper, we propose an end-to-end approach called DivGraphPointer for extracting diversified keyphrases from documents. Specifically, given an input document, we first construct a word graph from it, which aggregates identical words into one node and captures both the short- and long-range dependency between the words in the document. Afterwards, the graph convolutional neural network [22] is applied to the word graph to learn the representations of each node, which effectively models the word salience. To extract diverse keyphrases from documents, we propose a diversified pointer network model [42] over the word graph, which dynamically picks nodes from the word graph to construct the keyphrases. Two diversity mechanisms are proposed to increase the diversity among the generated keyphrases. Specifically, we employ a coverage attention mechanism [40] to address the over-generation problem in keyphrase extraction at lexical level and a semantic modification mechanism to dynamically modify the encoded document representation at semantic level. Figure 1 illustrates our approach schematically. The whole framework can be effectively and efficiently trained with back-propagation in an end-to-end fashion. Experimental results show that our proposed DivGraphPointer achieves state-of-the-art performance for keyphrase extraction on five benchmarks and significantly outperforms the existing supervised and unsupervised keyphrase extraction methods.

The contribution of this paper is twofold:

- We propose a graph convolutional network encoder for keyphrase extraction that can effectively capture document-level word salience.
- We propose two complementary diversification mechanisms that help the pointer network decoder to extract diverse keyphrases.

## 2 RELATED WORK

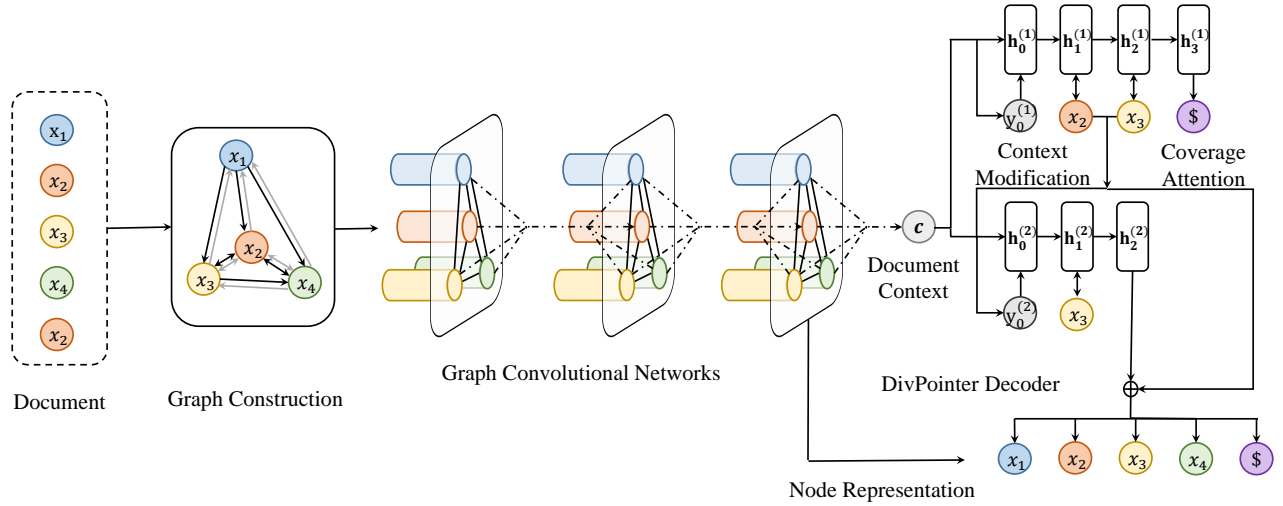
The most traditional keyphrase extraction method is based on Tf-Idf [37]: It identifies words that appear frequently in a document, but do not occur frequently in the entire document collection. These words are expected to be salient words for the document. The same idea can be applied on a set of keyphrase candidates identified by some syntactic patterns [9, 17]. However, the drawback of this family of approaches is that each word (or phrase) is considered in isolation. The inherent relations between words and between phrases are not taken into account. Such relations are important in keyphrase extraction.

To solve this problem, approaches based on word graphs have been proposed. In a word graph, words are connected according to some estimated relations such as co-occurrences. A graph-based extraction algorithm can then take into account the connections between words. The TextRank algorithm [30] was the first graph-based approach for keyphrase extraction. Given a word graph built on co-occurrences, it calculates the importance of candidate words with PageRank. The importance of a candidate keyphrase is then estimated as the sum of the scores of the constituent words. Following this work, the DivRank algorithm [27] was proposed to balance the importance and diversity of the extracted keyphrases. The TopicRank algorithm [4] was further proposed for topic-based keyphrase extraction. This algorithm first clusters the candidate phrases by topic and then chooses one phrase from each topic, which is able to generate a diversified set of keyphrases. In TopicRank, a complete topic graph is constructed to better capture the semantic relations between topics. The graph-based document representation can effectively model document-level word salience. However, these methods are fully unsupervised: the way that keyphrases are identified from a word graph is designed manually. Such a method lacks the flexibility to cope with different types of documents. In our proposed methods, we will use an end-to-end supervised training in order to adapt the extraction process to documents. In experiments, this also yields better performance.

Indeed, end-to-end neural approaches to keyphrase extraction have attracted a growing attention in recent studies. Zhang et al. [46] treated keyphrase extraction as a sequence labeling task and proposed a model called joint-layer RNN for extracting keyphrases from Twitter data. Meng et al. [29] first proposed an encoder-decoder framework for keyphrase extraction. However, their RNN-based encoder and decoder treat a document as a sequence and ignore the correlation between keyphrases. Afterwards, Zhang et al. [47] further proposed a CNN-based model for this task. The copy mechanism [13] is employed to handle the rare word problem in these encoder-decoder approaches, which allows to copy some words from the input documents. All the above approaches are based on word sequences, which inherit the well known problem that only local relations between words can be coped with. Compared to them, our model encodes documents with graphs, which are able to model more global document-level word salience.

Deep graph-based methods have been used for other tasks. They are also relevant to our work. For example, Yasunaga et al. [45] proposed to use graph convolutional networks to rank the salience of candidate sentences for document summarization. Marcheggiani and Titov [26] studied encoding sentences with graph convolutional neural networks for semantic role labeling. Bastings et al. [2] studied graph convolutional encoders for machine translation. In this paper, we target a different task. This is the first time that graph convolutional neural networks are used for keyphrase extraction.

Diversity is an important criterion in keyphrase extraction: it is useless to extract a set of similar keyphrases. Diversity has been studied in IR for search result diversification [6, 35]. Two main ideas have been used: selecting results that are different from those already selected; or selecting a set of results that ensure a good coverage of topics. In keyphrase extraction, there are also a few attempts to deal with diversity. Similar to our work, Zhang and Xiao [48] and Chen et al. [7] also employed a coverage mechanism to



**Figure 1: Illustration of our encoder-decoder architecture for keyphrase extraction. In this example, the document is a sequence of words, namely,  $d = \langle x_1, x_2, x_3, x_4, x_2 \rangle$ , and we have generated the first keyphrase  $y^1 = \langle x_2, x_3 \rangle$ . We are predicting  $y^2$ , the second word for keyphrase  $y^2$ , which will be selected from the nodes within the graph and the ending token  $\$$  for a keyphrase. Note that multiple appearances of  $x_2$  in the document is aggregated into only one node in the constructed word graph. (Better viewed in color)**

address the diversification problem. Chen et al. [7] further proposed a review mechanism to explicitly model the correlation between the keyphrases. However, their approaches are different from ours in multiple ways. First, this paper focuses on keyphrase extraction, while their approaches focus on keyphrase generation, which also requires generating keyphrases that cannot be found in the document. Furthermore, we use a hard form of coverage attention mechanism that penalizes the attention weight by one-hot vectors, while they used the same form in machine translation as the original paper [40]. We believe that in the keyphrase extraction task where source and target share the same vocabulary, a hard coverage attention could avoid the error propagation of attentions. Moreover, by directly applying the coverage attention on the word graph, we efficiently penalize all appearances of identical words. Finally, the review mechanism was employed in the decoding phase of RNN in Chen et al. [7], while our context modification mechanism modifies the initial semantic state of RNN. We expect that our context modification and coverage attention mechanisms can explicitly address the over-generation problem in keyphrase extraction at both semantic level and lexical level, and thus are complementary to each other. In contrast, the review mechanism and the coverage mechanism used in Chen et al. [7] provide both an attentive context, which tend to play a similar role and are somehow duplicated.

### 3 DEEP GRAPH-BASED DIVERSIFIED KEYPHRASE EXTRACTION

Our Diversified Graph Pointer (DivGraphPointer) is based on the encoder-decoder framework. We first construct a graph from a document from word adjacency matrices and encode the graph with the Graph Convolutional Networks (GCN) [22]. Afterwards, based on the representation of the graph, we generate the keyphrases one

by one with a decoder. Note that during the generation process, we restrict the output to the nodes of the graph. In other words, we only extract words appearing in the documents. Figure 1 presents an illustration of the DivGraphPointer framework.

#### 3.1 Graph Convolutional Network Encoder

In this part, we present our graph-based encoders. Traditional unsupervised graph-based ranking approaches for keyphrase extraction have shown promising performance at estimating the salience of words, which motivated us to develop deep graph-based encoders. Compared to sequence-based encoders such as RNN and CNN, graph-based encoders have several advantages. For example, graph-based encoders can explicitly leverage the short- and long-term dependency between words. Moreover, while the sequence-based encoders treat different appearances of an identical word independently, the graph representation of document naturally represents the identical word at different positions as a single node in the word graph. In this way, the graph-based encoding approaches can aggregate the information of all appearances of an identical word when estimating its salience, and thus employ a similar idea as in Tf-Idf [37] and PageRank [5] methods: important words tend to be more frequent in the document and widely linked with other important words.

**3.1.1 Graph Construction.** Our model represents a document by a complete graph in which identical words are nodes and edges are weighted according to the strength of the structural relations between nodes. Instead of manually designing and extracting connections between words (e.g. based on co-occurrence weights), we rely on the basic proximity information between words in the sentences, and let training process learn to use it. The basic assumption is that the closer two words in a sentence, the stronger their relation.

Specifically, we construct a directed word graph from the word sequences of a document in both directions. The adjacency matrices are denoted as:  $\overleftarrow{A}$  and  $\overrightarrow{A}$ . We assume that two words are related if they appear close to each other in a document. This extends the traditional adjacency relation to a more flexible proximity relation. The strength of the relation between a pair of words depends on their distance. Specifically, similar to [4], we define the weight from word  $w_i$  to word  $w_j$  in the two graphs as follows:

$$\overleftarrow{A}_{ij} = \sum_{p_i \in \mathcal{P}(w_i)} \sum_{p_j \in \mathcal{P}(w_j)} \text{relu}\left(\frac{1}{p_i - p_j}\right) \quad (1)$$

$$\overrightarrow{A}_{ij} = \sum_{p_i \in \mathcal{P}(w_i)} \sum_{p_j \in \mathcal{P}(w_j)} \text{relu}\left(\frac{1}{p_j - p_i}\right), \quad (2)$$

where  $\mathcal{P}(w_i)$  is the set of the position offset  $p_i$  of word  $w_i$  in the document. The function  $\text{relu}(\cdot) = \max(\cdot, 0)$  aims to filter the uni-directional information and help the graph-based encoder focusing on the order of the sequence.

In order to stabilize the iterative message propagation process in graph convolutional network encoder, we normalize each adjacency matrix  $A \in \{\overleftarrow{A}, \overrightarrow{A}\}$  by  $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ , where  $\tilde{A} = A + I_N$  is the adjacency matrix  $A$  with self-connections, and  $\tilde{D} = \sum_j \tilde{A}_{ij}$  is the degree matrix. The purpose of this re-normalization trick [22] is to constrain the eigenvalues of the normalized adjacency matrices  $\overleftarrow{\hat{A}}, \overrightarrow{\hat{A}}$  close to 1.

Such a graph construction approach differs from the one used in TextRank [30] that captures the co-occurrence in a limited window of words.  $\overleftarrow{\hat{A}} + \overrightarrow{\hat{A}}$  forms a complete graph where all nodes are interconnected. As stated in [4], the completeness of the graph has the benefit of providing a more exhaustive view of the relations between words. Also, computing weights based on the distances between offset positions bypasses the need for a manually defined parameter such as window size.

**3.1.2 Graph Convolutional Networks.** Next, we encode the nodes with multi-layer Graph Convolutional Networks (GCNs) [22]. Each graph convolutional layer generally consists of two stages. In the first stage, each node aggregates the information from its neighbors; in the second stage, the representation of each node is updated according to its current representation and the information aggregated from its neighbors. Given the node representation matrix  $H_l$  in the  $l$ -th layer, the information aggregated from the neighbors, denoted as  $f_l(H_l)$ , can be calculated as follows:

$$f_l(H_l) = \overleftarrow{\hat{A}} H_l \overleftarrow{W}_l + \overrightarrow{\hat{A}} H_l \overrightarrow{W}_l + H_l W_l \quad (3)$$

which aggregates the information from both the neighbors defined in the two matrices and the node itself. Here,  $\overleftarrow{W}_l, \overrightarrow{W}_l, W_l$  are layer-specific trainable weight matrices.

Once the information from the neighbors are aggregated, inspired by He et al. [15] and Gehring et al. [11], we updated the node representation with a residual Gated Linear Unit (GLU) activation:

$$H_{l+1} = H_l + f_l(H_l) \otimes \sigma(g_l(H_l)) \quad (4)$$

where  $\sigma$  is the sigmoid function and  $\otimes$  is the point-wise multiplication.  $g_l$  is another function defined in a similar way as  $f_l$ , which is used as the gating function of the information collected from the

neighbors.  $H_0$  is initialized with the pretrained word embedding matrix, and the residual connection is omitted in its activation.

The representation of the entire graph (or the document representation)  $c$  is then obtained by averaging the aggregation of the last layer's node representations  $f_L(H_L)$ , where  $L$  denotes the total number of GCN layers.

Based on the encoded document representation  $c$ , we propose a decoder, named DivPointer, to generate summative and diverse keyphrases in the next section.

## 3.2 Diversified Pointer Network Decoder

In this part, we introduce our approach of keyphrase extraction based on the graph representation. Most previous end-to-end neural approaches select keyphrases independently during the decoding process. However, ignoring the diversity among phrases may lead to multiple similar keyphrases, undermining the representativeness of the keyphrase set. Therefore, we propose a DivPointer Network with two mechanisms on semantic level and lexical level respectively to improve the diversity among keyphrases during the decoding process.

**3.2.1 Pointer Network.** The decoder is used to generate output keyphrases according to the representation of the input document. We adopt a pointer network [42] with diversity enabled attentions to generate keyphrases. A pointer network is a neural architecture to learn the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in the original data space. The graph nodes corresponding to words in a document are regarded as the original data space of the pointer network in our case.

Specifically, the pointer decoder receives the document representation  $c$  as the initial state  $\mathbf{h}_0^{(i)}$ , and predicts each word  $y_t^{(i)}$  of a keyphrase  $y^{(i)}$  sequentially based on  $\mathbf{h}_t^{(i)}$ :

$$\mathbf{h}_t^{(i)} = \text{RNN}(\mathbf{y}_{t-1}^{(i)}, \mathbf{h}_{t-1}^{(i)}) \quad (5)$$

where  $\mathbf{y}_{t-1}^{(i)}$  denotes the node representation of the word  $y_{t-1}^{(i)}$  that keyphrase  $y^{(i)}$  generated at the previous step.  $\mathbf{h}_t$  is the hidden state of an RNN. The word  $y_t^{(i)}$  is then selected with a pointer network according to certain attention mechanism based on  $\mathbf{h}_t^{(i)}$ .

A general attention [1] score  $e_{t,j}$  on each graph node  $x_j \leq N$  with respect to the hidden state  $\mathbf{h}_t^{(i)}$  can be computed by:

$$e_{t,j}^{(i)} = \mathbf{v}^T \tanh(W_h \mathbf{h}_t^{(i)} + W_x \mathbf{x}_j + \mathbf{b}) \quad (6)$$

where  $\mathbf{x}_j$  is the node representation of  $x_j$  taken from  $H_L$  and  $\mathbf{v}^T, W_h, W_x$ , and  $\mathbf{b}$  are parameters to be learned. We can then obtain the pointer distribution on the nodes by normalizing  $\{e_{t,j}^{(i)}\}$ :

$$p(y_t^{(i)} = x_j) = \frac{\exp(e_{t,j}^{(i)})}{\sum_{k=1}^N \exp(e_{t,k}^{(i)})} \quad (7)$$

With this distribution, we can select a word with the maximum pointer probability as  $y_t^{(i)}$  from the graph nodes.

However, the attention mechanism above is merely built on the global hidden state, namely the document representation  $c$ . Diversity among the generated keyphrases can hardly be addressed

without taking the previously generated keyphrases into consideration in the decoding process. Two mechanisms aiming at achieving diversity among keyphrases generated are introduced in the following sections.

**3.2.2 Context Modification.** During the decoding process, the document representation  $\mathbf{c}$  is the key to maintain a global semantic context for each word to be generated. However, the constant context may lead to generating similar keyphrases repeatedly, which hurts the representativeness of the generated keyphrase set in reality.

Therefore, we propose to update the context  $\mathbf{h}_0^{(i)}$  dynamically based on the previously generated keyphrases  $\mathbf{y}^{(1:i-1)}$  while decoding the words for the  $i^{th}$  phrase  $\mathbf{y}^{(i)}$ , as follows:

$$\mathbf{y}_0^{(i)} = W_y \left[ \mathbf{c}, \overline{\mathbf{y}^{(1:i-1)}} \right] + \mathbf{b}_y \quad (8)$$

$$\mathbf{h}_0^{(i)} = \tanh \left( W_h \left[ \mathbf{c}, \overline{\mathbf{y}^{(1:i-1)}} \right] + \mathbf{b}_s \right) \quad (9)$$

where we introduce the average representation of the previous generated keyphrases  $\overline{\mathbf{y}^{(1:i-1)}}$  into the model to learn a updated context.  $\overline{\mathbf{y}^{(i)}}$  is initialized as  $\mathbf{0}$ .  $\mathbf{y}_0^{(0)}$  and  $\mathbf{h}_0^{(0)}$  are initialized accordingly.

We find the modified context helps the pointer network to focus on the keyphrases with different meanings yet to come. Therefore, it can solve the over-generation problem in the previous deep generation models.

**3.2.3 Coverage Attention.** In addition to the context modification mechanism, we also adopt the coverage mechanism to enhance diversity among keyphrases on lexical level. Coverage mechanism has been well investigated in machine translation [40], search result diversification [35] and document summarization [36]. When the criterion is used in a neural network, it generally maintains a coverage vector to keep track of the attention history. More specifically, they directly use the sum of previous alignment probabilities as the coverage for each word in the input.

Since keyphrases are usually short and summative, their meanings are more sensitive to the term replacement than those of sentences or documents. Based on this observation, we propose a coverage attention on lexical level with the help of one-hot representations of the previously generated keyphrases. Specifically, we use the sum of the one-hot vectors of the previously generated keyphrases as a coverage representation.

$$\mathbf{c}_j^{(i)} = \mathbf{c}_j^{(i-1)} + \sum_t \mathbf{o}_{t,j}^{(i-1)} \quad (10)$$

where  $\mathbf{c}_j^{(i)}$  is the coverage value of the  $j^{th}$  node for the  $i^{th}$  keyphrase and  $\mathbf{o}_{t,j}^{(i)}$  is the  $j^{th}$  element of the  $t^{th}$  one-hot vector in the  $i^{th}$  keyphrase. It is 1 if the  $t^{th}$  generated word is the  $j^{th}$  node in the graph, 0 otherwise.

The coverage representation is then incorporated into the attention mechanism of our DivPointer network as follows:

$$\mathbf{e}_{t,j}^{(i)} = \mathbf{v}^T \tanh(W_h \mathbf{h}_t^{(i)} + W_x \mathbf{x}_j + \mathbf{w}_c \mathbf{c}_j^{(i)} + \mathbf{b})$$

Note that our coverage attention mechanism differs from the original form [40] that was designed for machine translation. The reason for the change is that in the keyphrase extraction task,

the source and the target share the same vocabulary. So a hard coverage attention could avoid the error propagation of attentions to similar words. Instead, the soft attention distribution cannot precisely represent the previous generated keyphrases.

## 4 TRAINING AND DECODING

In both training and decoding, the \$ symbol is added to the end of the keyphrases to help learn to stop adding more words. The whole model is trained to maximize the log-likelihood of the words in the keyphrases according to the given input document. Specifically, the training objective is defined as below:

$$p(\mathbf{y}^{(i)}) = \prod p(y_j^{(i)} | \mathbf{y}_{1:j-1}^{(i)}, \mathbf{y}^{(1:i-1)}, \mathbf{c})$$

$$L = -\frac{1}{\sum K_d} \sum_{d=1}^D \sum_{k=1}^{K_d} \log p(\mathbf{y}^{(d,k)}) \quad (11)$$

where  $D$  is the number of document and  $K_d$  is the number of keyphrases in each document.  $p(\mathbf{y}^{(i)})$  is the generative probability of the  $i^{th}$  keyphrase given the previous generated keyphrases and the context  $\mathbf{c}$ . For simplicity, we omit the conditional notation here.  $\mathbf{y}^{(d,k)}$  is the  $k^{th}$  keyphrase in the  $d^{th}$  document, which is also conditioned on  $\mathbf{y}^{(d,1:k-1)}$  and the context of the  $d^{th}$  document. The order of the keyphrases for the same document are randomly shuffled in each epoch of training.

We use Adam [21] with a mini-batch size of  $n = 256$  to optimize model parameters, with an initial learning rate  $\alpha_1 = 0.002$  in the first 6,000 steps and  $\alpha_2 = 0.0002$  in the rest steps. A gradient clipping  $clip = 0.1$  is applied in each step. We also use early-stop in the training with a validation dataset. Model parameters are initialized by normal distributions [12]. Dropouts [38] are applied on the word embedding with dropout rate  $p = 0.1$  and on the GCN output with dropout rate  $p = 0.5$  to reduce over-fitting. We also apply batch normalization [18] after the last graph convolutional layers to accelerate the training process.

In the decoding, we generate keyphrases based on the negative log generative probability of candidate keyphrases, with a beam-search in window size = 100 and search depth = 5. In practice, we find that the model tends to generate short keyphrases when using the negative log-likelihood as the keyphrase scores. Therefore, we propose a simple keyphrase length penalization, where we normalize the score of candidate keyphrase  $\mathbf{y}$  by its length  $|\mathbf{y}|$ :

$$\bar{s}(\mathbf{y}) = \frac{s(\mathbf{y})}{\alpha + |\mathbf{y}|} \quad (12)$$

where  $s(\mathbf{y}) = -\log p(\mathbf{y})$  is the original score (negative log probability) and  $\bar{s}$  is the normalized score.  $\alpha$  is the length penalty factor, where larger  $\alpha$  tends to generate shorter keyphrases, and smaller  $\alpha$  generates longer keyphrases.

## 5 EXPERIMENTS

### 5.1 Experimental Setting

**Data.** We use the data set Kp20k [29] for training. Kp20k contains a large amount of high-quality scientific metadata in the computer science domain from various online digital libraries [28]. We follow the official setting of this dataset and split the dataset into training

Dataset	#paper	#keyphrase	length
<b>training data</b>			
Kp20k	527830	2.94	1.80
<b>validation data</b>			
Inspec	1500	7.39	2.28
NUS	five-fold cross-validation		
SemEval	188	3.84	1.97
Krapivin	1904	2.52	1.94
Kp20k	20,000	2.94	1.80
<b>test data</b>			
Inspec	500	7.70	2.28
NUS	211	5.37	1.84
SemEval	100	5.73	1.93
Krapivin	400	3.24	2.01
Kp20k	20,000	2.94	1.80

**Table 1: Statistics of five datasets. We use the original training data from the Inspec, SemEval, and Krapivin data sets as validation data to select the best length penalty factor  $\alpha$ .**

(527,830 articles), validation (20,000 articles) and test (20,000 articles) data. We further test the model trained with KP20k on four widely-adopted keyphrase extraction data sets including Inspec [17], NUS [32], SemEval-2010 [19] and Krapivin [23]. Following [29], we take the concatenation of the title and the abstract as the content of a paper for all these datasets. No text pre-processing steps are conducted. In this paper, we focus on keyphrase extraction. Therefore, only the keyphrases that appear in the documents are used for training and evaluation. Table 1 provides the statistics on the number of papers, the average number of keyphrases and the corresponding average keyphrase length for each benchmark datasets.

*Model Setting.* A 3-layer Gated Recurrent Unit (GRU) [8] is used as the RNN recurrent function. The dimension of both the input and the output of GRU is set to 400, while the word embedding dimension is set to 300. The number of GCN layers is empirically set to 6. The length penalty factor  $\alpha$  is selected according to the validations on different data sets. All the other hyper-parameters are the same when we evaluate our different models on different datasets. The word embeddings (or the node embeddings) are fixed to the pre-trained fastText model [3], which is trained on Wikipedia 2017, UMBC web-base corpus and statmt.org news dataset (16B tokens). It breaks words into sub-words, which can help handle the problem of out-of-vocabulary (OOV).

*Baseline.* We compare our models with four supervised algorithms: RNN [29], CopyRNN [29], CNN[47], and CopyCNN[47]. Considering that the unsupervised ranking-based methods motivate our proposed graph-based encoder solution, we also compare our models with four well-known unsupervised algorithms for keyphrase extraction including Tf-Idf [37], TextRank [30], SingleRank [43], ExpandRank [43].

We also compare several different variants of our algorithms. We compare with the method of encoding documents with 2-layer bi-directional LSTM [16] and decoding the keyphrases with pointer

networks, marked as SeqPointer. We also compare with the method with graph-based encoder and vanilla pointer decoder, marked as GraphPointer. No diversity mechanisms are used during decoding for both SeqPointer and GraphPointer. The hyper-parameters of these two models are also selected according to the validation data.

*Evaluation.* Following the literature, the macro-averaged precision, recall and F1 measures are used to measure the overall performance. Here, precision is defined as the number of correctly-predicted keyphrases over the number of all predicted keyphrases; recall is computed as the number of correctly predicted keyphrases over the total number of data records, and F1 is the harmonic average of precision and recall.

Besides the set-based metrics, we also evaluate our models with a rank-based metric, Normalized Discounted Cumulative Gain (NDCG) [44]. Since most keyphrase extraction models' output is sequential, we believe that the rank-based metric could better measure the performance of those models.

In the evaluation, we apply Porter Stemmer [33] to both target keyphrases and predicted keyphrases when determining the match of keyphrases and match of the identical word. The embedding of different variants of an identical word are averaged when fed as the input of graph-based encoder.

## 5.2 Results

The performance of different algorithms on five benchmarks are summarized in Table 2. For each method, the table presents the performance of generating 5 and 10 keyphrases with respect to F1 measure. We also include the truncated NDCG measure of generating 10 keyphrases in the table. The best results are highlighted in bold. We can see that for most cases (except RNN and CNN), the supervised models outperform all the unsupervised algorithms. This is not surprising since the supervised models are trained end-to-end with supervised data. The RNN model and CNN do not perform well on nearly all datasets (e.g., Inspec, NUS, SemEval, and Krapivin). The reason is that they cannot generate words that are not in the vocabulary (OOV words) during the decoding process, while in this paper we only allow to generate words appeared in the given documents. This problem is addressed by the SeqPointer model, which utilizes the pointer network to copy words from the source text, and hence the performance is significantly improved. We can also see such phenomenon on CopyCNN.

Comparing SeqPointer and CopyRNN, we can observe that SeqPointer outperforms CopyRNN on all data sets. This could be due to the fact that generation mechanism interferes with the copy mechanism in CopyRNN. Implementation details could also contribute to the performance difference. For example, we utilize fastText as word embedding to handle the OOV problem, while Meng et al. [29] randomly initialize their word embeddings. Moreover, We use a length penalty mechanism to solve the problem of short phrase generation, while Meng et al. [29] applied a simple heuristic by preserving only the first single-word phrase and removing the rest. We also use different hidden dimension, learning rate, dropout rate, beam depth, and beam size settings.

By replacing the RNN encoder with the graph convolutional network encoder, the performance of GraphPointer is further improved. This shows that although sequence-based encoders are effective

	Inspec		NUS		SemEval		Krapivin		Kp20k	
	F <sub>1</sub> @5	F <sub>1</sub> @10	F <sub>1</sub> @5	F <sub>1</sub> @10	F <sub>1</sub> @5	F <sub>1</sub> @10	F <sub>1</sub> @5	F <sub>1</sub> @10	F <sub>1</sub> @5	F <sub>1</sub> @10
unsupervised methods										
Tf-Idf	0.223	0.304	0.139	0.181	0.120	0.184	0.113	0.143	0.105	0.130
TextRank	0.229	0.275	0.195	0.190	0.172	0.181	0.172	0.147	0.180	0.150
SingleRank	0.214	0.297	0.145	0.169	0.132	0.169	0.096	0.137	0.099	0.124
ExpandRank	0.211	0.295	0.137	0.162	0.135	0.163	0.096	0.136	N/A	N/A
supervised methods										
RNN	0.000	0.000	0.005	0.004	0.004	0.003	0.002	0.001	0.138	0.009
CopyRNN	0.292	0.336	0.342	0.317	0.291	0.296	0.302	0.252	0.328	0.255
CNN	0.088	0.069	0.176	0.133	0.162	0.127	0.141	0.098	0.188	0.203
CopyCNN	0.285	0.346	0.342	0.330	0.295	0.308	0.314	0.272	0.351	0.288
SeqPointer	0.347	0.386	0.383	0.345	0.328	0.324	0.318	0.274	0.333	0.281
GraphPointer	0.375*	0.387	0.421*	0.375**	0.377*	0.350**	0.340**	0.280**	0.341**	0.282*
DivGraphPointer	<b>0.386**</b>	<b>0.417**</b>	<b>0.460**</b>	<b>0.402**</b>	<b>0.401**</b>	<b>0.389**</b>	<b>0.363**</b>	<b>0.297**</b>	<b>0.368**</b>	<b>0.292**</b>
Normalized Discounted Cumulative Gain @10 (NDCG@10)										
SeqPointer	0.448		0.501		0.440		0.476		0.463	
GraphPointer	0.479		0.536		0.482		0.499		0.498	
DivGraphPointer	<b>0.503</b>		<b>0.591</b>		<b>0.518</b>		<b>0.534</b>		<b>0.532</b>	

Table 2: The performance of keyphrase extraction on five benchmarks. The results of the former six methods are taken from [28] and the results of CNN and CopyCNN are taken from [47]. The significance levels (\*\* 0.01, \* 0.1) between different methods (GraphPointer v.s SeqPointer, DivGraphPointer v.s GraphPointer) are also provided.

	Inspec		Krapivin	
	F <sub>1</sub> @5	F <sub>1</sub> @10	F <sub>1</sub> @5	F <sub>1</sub> @10
SeqPointer	0.347	0.386	0.318	0.274
1-layer GCN	0.351	0.365	0.320	0.261
3-layer GCN	0.373	0.382	0.334	0.268
6-layer GCN	<b>0.375</b>	0.387	0.340	0.280
9-layer GCN	0.374	<b>0.397</b>	<b>0.362</b>	<b>0.284</b>
12-layer GCN	0.373	0.394	0.344	0.283

Table 3: Effectiveness of encoding documents as graphs with graph convolutional neural networks.

at capturing sequential information such as word order and adjacency, such sequential information is not sufficient in the keyphrase extraction task. It turns out that the long-term dependency and information aggregation are more important for precisely extracting keyphrases.

Our proposed model DivGraphPointer achieves the best performance by modeling document-level word salience during the encoding process (the graph encoder), and increasing the diversity of keyphrases during decoding with diversified pointer networks.

### 5.3 Model Analysis

We further conducted a detailed analysis of the proposed DivGraphPointer model. We take two data sets Inspec and Krapivin as examples.

**5.3.1 Effectiveness of Encoding Documents as Graphs.** In this part, we evaluate the effectiveness of encoding documents with graphs. We compare SeqPointer and GraphPointer with different numbers

	Inspec		Krapivin	
	F <sub>1</sub> @5	F <sub>1</sub> @10	F <sub>1</sub> @5	F <sub>1</sub> @10
+neither	0.375	0.387	0.340	0.280
+coverage	0.363	0.381	0.356	0.283
+context	0.379	0.400	0.360	0.290
+both	<b>0.386</b>	<b>0.417</b>	<b>0.363</b>	<b>0.297</b>

Table 4: Investigating the performance of DivGraphPointer with different diversity mechanisms. Here "coverage" denotes coverage attention, and "context" denotes context modification.

	AIC@5		AIC@10	
Inspec	0.057	<b>0.034</b>	0.062	<b>0.048</b>
NUS	0.056	<b>0.041</b>	0.063	<b>0.054</b>
SemEval	0.048	<b>0.031</b>	0.051	<b>0.044</b>
Krapivin	0.047	<b>0.033</b>	0.050	<b>0.044</b>
Kp20k	0.049	<b>0.030</b>	0.053	<b>0.040</b>

Table 5: The Average Index of Coincidence (AIC@N) of the extracted keyphrases from GraphPointer (left) and DivGraphPointer (right) on five benchmarks

of graph convolutional layers. To focus on comparing the effectiveness of different encoders, we do not use the diversity mechanism and set the length penalty factor  $\alpha = 1$  in all the compared algorithms.

Title: theoretical and experimental investigations on coherence of **traffic noise transmission** through an **open window** into a **rectangular room** in **high-rise buildings**

Abstract: a method for theoretically calculating the coherence between **sound pressure** inside a **rectangular room** in a **high-rise building** and that outside the **open window** of the room is proposed. the traffic noise transmitted into a room is generally dominated by **low-frequency components**, to which active noise control (anc) technology may find an application. however, good coherence between reference and error signals is essential for an effective noise reduction and should be checked first. based on **traffic noise prediction methods**, **wave theory**, and **mode coupling theory**, the results of this paper enabled one to determine the potentials and limitations of anc used to reduce such a transmission. experimental coherence results are shown for two similar, empty **rectangular rooms** located on the 17th and 30th floors of a 34 floor **high-rise building**. the calculated results with the proposed method are generally in good agreement with the experimental results and demonstrate the usefulness of the method for predicting the coherence

**CopyRNN :**

active noise control, traffic noise, coherence, mode coupling, **sound pressure**, active noise, noise reduction, **wave theory**, **mode coupling theory**, noise control

**GraphPointer-100 :**

coherence, active noise control, **high-rise buildings**, traffic noise, traffic noise control, **wave theory**, **rectangular room**, **mode coupling theory**, traffic noise predict, mode coupling

**DivGraphPointer-100 :**

coherence, traffic noise, active noise control, **high-rise buildings**, **mode coupling theory**, **wave theory**, **rectangular room**, **open window**, anc, **sound pressure**

**DivGraphPointer-1 :**

active noise control, **mode coupling theory**, traffic noise, **high-rising build**, **wave theory**, coherence, **rectangular room**, **open window**, traffic noise predict, **traffic noise transmission**

Figure 2: An example of keyphrase extraction results with CopyRNN and our models. Phrases in bold are true keyphrases that predicted by the algorithms.

	Inspec		Krapivin	
	F <sub>1</sub> @5	F <sub>1</sub> @10	F <sub>1</sub> @5	F <sub>1</sub> @10
$\alpha = 0$	<b>0.394</b>	0.411	0.348	0.282
$\alpha = 1$	0.386	<b>0.417</b>	0.363	<b>0.297</b>
$\alpha = 2$	0.379	0.415	0.365	<b>0.297</b>
$\alpha = 5$	0.364	0.400	<b>0.370</b>	0.296
$\alpha = 20$	0.349	0.389	0.363	0.295
$\alpha = 100$	0.341	0.382	0.362	0.296

Table 6: The performance of DivGraphPointer w.r.t. the length penalty factor  $\alpha$ .

The results on Inspec and Krapivin are summarized in Table 3. First, we can see that encoding documents as graphs significantly outperforms the SeqPointer, which encodes documents with bi-directional RNN, especially when more GCN layers are used. Especially, we notice that even 1-layer GCN can outperform RNN encoder on some metrics. This shows that the graph-based representation are very suitable for the keyphrase extraction task and demonstrates the effectiveness of graph-based encoder in aggregating information of multiple appearances of identical words and modeling long-term dependency.

Increasing the number of layers will increase the performance in the beginning. If too many layers are used, the performance will decrease due to over-fitting. In all our other experiments, we choose

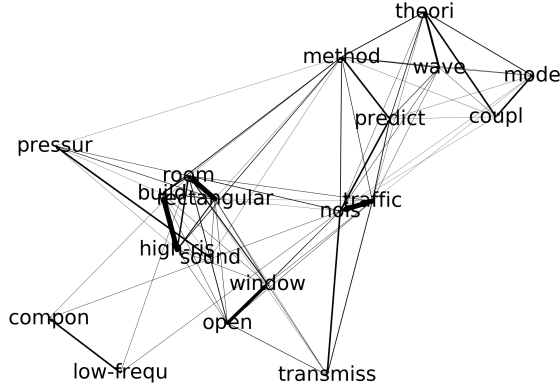
the number of graph convolutional layers as six by balancing the effectiveness and efficiency of the graph convolutional networks.

**5.3.2 Diversity Mechanism.** Next, we investigate the effectiveness of the proposed diversity mechanisms: context modification and coverage attention. We compare the following DivGraphPointer variants: (1) with neither of them, (2) with only context modification, (3) with only coverage attention and (4) with both of them. Here we also set the length penalty factor  $\alpha = 1$ . The results are presented in Table 4.

We can see that the performance of adding context modification significantly improves comparing to the vanilla pointer networks. The results of adding coverage attention are mixed. On the Krapivin data set, the performance improves while on the Inspec data set, the performance decreases. The performance of adding both mechanism are very robust, significantly better than vanilla pointer networks. This shows that the coverage attention and context modification focus on lexical-level and semantic-level duplication problem, respectively, thus are complementary to each other.

We also provide the Average Index of Coincidence (AIC) [10] of the extracted keyphrases from GraphPointer and DivGraphPointer in Table 5. AIC represents the probability of the identical words appearing in the extracted keyphrases. A smaller AIC indicates smaller redundancy. We can find that our proposed mechanism is very effective in improving keyphrase diversity. We also observe that this effect is more significant for top 5 keyphrases than top 10





**Figure 3: A sub-graph on keyphrase words of the full word graph for the document in Figure 2. The edge width is proportional to the edge weight.**

keyphrases. It shows that the diversification mechanism tends to take effect at the begin of the keyphrase extraction process.

**5.3.3 Length Penalty Factor.** Finally, we investigate how the length penalty factor  $\alpha$  affects the performance of the DivGraphPointer. Results with different values of length penalty factor  $\alpha$  are presented in Table 6. Results show that the length penalty factors affect the model performance significantly. Either a small value of  $\alpha$  (e.g.,  $\alpha=0$ ), which tends to generate long phrase, or a big value of  $\alpha$  (e.g.,  $\alpha=100$ ), which tends to generate short phrases, yields inferior results. The best choice of  $\alpha$  is between 0 and 1 for Inspec and between 1 and 5 for Krapivin, which indicates that both extreme cases - totally normalization  $\alpha=0$  or no normalization  $\alpha=100$  - are not good choices.

## 5.4 Case Analysis

Finally, we present a case study on the results of the keyphrases extracted by different algorithms. Figure 2 presents the results of CopyRNN [29] and different variants of our algorithms. "Model- $\alpha$ " means that the length penalty factor  $\alpha$  is used in the evaluated model. We also visualize a sub-graph on the words that appear in the keyphrases for clear illustration of encoding phase of graph-based methods. From the figures, we have the following observations:

- (1) The diversity mechanism is quite effective to increase the diversity of the generated keyphrases. For example, "CopyRNN" generates five similar keyphrases, which all contain the word "noise", "GraphPointer-100" generates four such keyphrases, while "DivGraphPointer-100" and "DivGraphPointer-1" only generates two.
- (2) The length penalty factor  $\alpha$  can efficiently control the granularity of generated keyphrases. For example, the keyphrase with only a single word "coherence" ranks the first in "DivGraphPointer-100", but only ranks the sixth in "DivGraphPointer-1", when the same trained model is used.

- (3) The graph-based encoders can better estimate the salience of words by their relations. For example, "high-rise building" and "rectangular room" are highly relevant in the word graph, and thus are selected by the "GraphPointer" model, while "CopyRNN" finds "sound pressure", which appears only once and only weakly connected to other keyphrases. The strong connection between "traffic" and "noise" also explains why the graph-based methods rank "traffic noise" higher than true keyphrase "traffic noise transmission" or "traffic noise prediction methods".

## 6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose an end-to-end method called DivGraphPointer for extracting diverse keyphrases. It formulates documents as graphs and applies graph convolutional networks for encoding the graphs, which efficiently capture the document-level word salience by modeling both the short- and long-range dependency between words in documents and aggregate the information of multiple appearances of identical words. To avoid extracting similar keyphrases, a diversified pointer network is proposed to generate diverse keyphrases from the nodes of the graphs. Experiments on five benchmark data sets show that our proposed DivGraphPointer model achieves state-of-the-art performance, significantly outperforming existing state-of-the-art supervised and unsupervised methods.

Our research can be extended in many directions. To begin with, currently our diversified pointer network decoders extract keyphrase in an auto-regressive fashion. We could further leverage reinforcement learning to address the exposure bias as well as consequent error propagation problem in the sequential generation process. Moreover, our graph convolutional network encoder aggregates the word relation information through manually designed edge weights based on proximity at present. We would like to further explore utilizing graph attention networks (GATs) [41] to dynamically capture correlation between words in word graph. Finally, utilizing linguistic information when constructing edges in word graphs to ease keyphrase extraction is also an interesting future direction.

## ACKNOWLEDGEMENTS

We would like to thank Rui Meng for sharing the source code and giving helpful advice.

## REFERENCES

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
- [2] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675* (2017).
- [3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* (2016).
- [4] Adrien Bouguin, Florian Boudin, and Béatrice Daille. 2013. Topicrank: Graph-based topic ranking for keyphrase extraction. In *International Joint Conference on Natural Language Processing (IJCNLP)*. 543–551.
- [5] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* 30, 1-7 (1998), 107–117.
- [6] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of*

- the 21st annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 335–336.
- [7] Jun Chen, Xiaoming Zhang, Yu Wu, Zhao Yan, and Zhoujun Li. 2018. Keyphrase generation with correlation constraints. *arXiv preprint arXiv:1808.07185* (2018).
  - [8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
  - [9] Eibe Frank, Gordon W Paynter, Ian H Witten, Carl Gutwin, and Craig G Nevill-Manning. 1999. Domain-specific keyphrase extraction. In *16th International joint conference on artificial intelligence (IJCAI 99)*, Vol. 2. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 668–673.
  - [10] William Frederick Friedman. 1922. *The index of coincidence and its applications in cryptography*. Aegean Park Press.
  - [11] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional Sequence to Sequence Learning. *arXiv preprint arXiv:1705.03122* (2017).
  - [12] 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*. 249–256.
  - [13] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393* (2016).
  - [14] Kazi Saidul Hasan and Vincent Ng. 2014. Automatic keyphrase extraction: A survey of the state of the art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 1262–1273.
  - [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
  - [16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
  - [17] Anette Hulth. 2003. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*. Association for Computational Linguistics, 216–223.
  - [18] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. 448–456.
  - [19] Su Nam Kim, Olena Medelyan, Min-Yen Kan, and Timothy Baldwin. 2010. Semeval-2010 task 5: Automatic keyphrase extraction from scientific articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation*. Association for Computational Linguistics, 21–26.
  - [20] Youngsam Kim, Munhyong Kim, Andrew Cattle, Julia Otmakhova, Suzi Park, and Hyopil Shin. 2013. Applying graph-based keyword extraction to document retrieval. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*. 864–868.
  - [21] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
  - [22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
  - [23] Mikalai Krapivin, Aliaksandr Autaeu, and Maurizio Marchese. 2009. *Large dataset for keyphrases extraction*. Technical Report. University of Trento.
  - [24] Guangda Li, Haojie Li, Zhaoan Ming, Richang Hong, Sheng Tang, and Tat-Seng Chua. 2010. Question answering over community-contributed web videos. *IEEE MultiMedia* 17, 4 (2010), 46–57.
  - [25] Zhiyuan Liu, Wenyi Huang, Yabin Zheng, and Maosong Sun. 2010. Automatic keyphrase extraction via topic decomposition. In *Proceedings of the 2010 conference on empirical methods in natural language processing*. Association for Computational Linguistics, 366–376.
  - [26] Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826* (2017).
  - [27] Qiaozhu Mei, Jian Guo, and Dragomir Radev. 2010. Divrank: the interplay of prestige and diversity in information networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1009–1018.
  - [28] Rui Meng, Shuguang Han, Yun Huang, Daqing He, and Peter Brusilovsky. 2016. Knowledge-based content linking for online textbooks. In *Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on*. IEEE, 18–25.
  - [29] Rui Meng, Sanqiang Zhao, Shuguang Han, Daqing He, Peter Brusilovsky, and Yu Chi. 2017. Deep keyphrase generation. *arXiv preprint arXiv:1704.06879* (2017).
  - [30] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*.
  - [31] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.
  - [32] Thuy Dung Nguyen and Min-Yen Kan. 2007. Keyphrase extraction in scientific publications. In *International Conference on Asian Digital Libraries*. Springer, 317–326.
  - [33] Martin F Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.
  - [34] Vahed Qazvinian, Dragomir R Radev, and Arzucan Özgür. 2010. Citation summarization through keyphrase extraction. In *Proceedings of the 23rd international conference on computational linguistics*. Association for Computational Linguistics, 895–903.
  - [35] Rodrygo LT Santos, Craig Macdonald, and Iadh Ounis. 2010. Exploiting query reformulations for web search result diversification. In *Proceedings of the 19th international conference on World wide web*. ACM, 881–890.
  - [36] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368* (2017).
  - [37] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.
  - [38] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research* 15, 1 (2014), 1929–1958.
  - [39] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.
  - [40] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811* (2016).
  - [41] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* 1, 2 (2017).
  - [42] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. 2692–2700.
  - [43] Xiaojun Wan and Jianguo Xiao. 2008. Single Document Keyphrase Extraction Using Neighborhood Knowledge. In *AAAI*, Vol. 8. 855–860.
  - [44] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A theoretical analysis of NDCG type ranking measures. In *Conference on Learning Theory*. 25–54.
  - [45] Michihiro Yasunaga, Rui Zhang, Kshitij Meelu, Ayush Pareek, Krishnan Srinivasan, and Dragomir Radev. 2017. Graph-based Neural Multi-Document Summarization. *arXiv preprint arXiv:1706.06681* (2017).
  - [46] Qi Zhang, Yang Wang, Yeyun Gong, and Xuanjing Huang. 2016. Keyphrase extraction using deep recurrent neural networks on Twitter. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 836–845.
  - [47] Yong Zhang, Yang Fang, and Xiao Weidong. 2017. Deep keyphrase generation with a convolutional sequence to sequence model. In *Systems and Informatics (ICSAI), 2017 4th International Conference on*. IEEE, 1477–1485.
  - [48] Yong Zhang and Weidong Xiao. 2018. Keyphrase Generation Based on Deep Seq2seq Model. *IEEE Access* 6 (2018), 46047–46057.