PRELIMINARY VERSION: DO NOT CITE
The AAAI Digital Library will contain the published
version some time after the conference

Adaptive Beam Search Decoding for Discrete Keyphrase Generation

Xiaoli Huang,[1] Tongge Xu,[2*] Lvan Jiao,[1] Yueran Zu,[1] Youmin Zhang[3]
[1] School of Computer Science and Engineering, Beihang University
[2] School of Cyber Science and Technology, Beihang University
[3] Jiangxi Research Institute of Beihang University
{hxlist, xutg, jiaola, yueranzu, youmi}@buaa.edu.cn

Abstract

Keyphrase Generation compresses a document into some highly-summative phrases, which is an important task in natural language processing. Most state-of-the-art adopt greedy search or beam search decoding methods. These two decoding methods generate a large number of duplicated keyphrases and are time-consuming. Moreover, beam search only predicts a fixed number of keyphrases for different documents. In this paper, we propose an adaptive generation model-AdaGM, which is mainly inspired by the importance of the first words in keyphrases generation. In AdaGM, a novel reset state training mechanism is proposed to maximize the difference in the predicted first words. To ensure the discreteness and get an appropriate number of keyphrases according to the content of the document adaptively, we equip beam search with a highly effective filter mechanism. Experiments on five public datasets demonstrate the proposed model can generate marginally less duplicated and more accurate keyphrases. The codes of AdaGM are avaliable at: https://github.com/huangxiaolist/adaGM.

Introduction

Keyphrases generation is an important task that compresses a document into some salient and discrete phrases, which vary as widely as possible in terms of word choice, topic, and meaning. These phrases can help people understand the content quickly and further benefit downstream applications, such as summarization (Li et al. 2018), translation (Tang et al. 2016), and so forth. As depicted in Figure 1, the input is usually a document, and the output is a set of keyphrases. These keyphrases can be categorized into *present keyphrases* that appear in the document like “dynamic programming” and *absent keyphrases* that do not appear in the document like “craft woodworkers”.

To predict present and absent keyphrases, generative methods (Meng et al. 2017; Yuan et al. 2018; Chen et al. 2018; Chan et al. 2019; Chen et al. 2020) are proposed. Most of the above methods are implemented by the sequence-to-sequence (seq2seq) model (Sutskever, Vinyals, and Le 2014; Cho et al. 2014). Beam search (Meng et al. 2017; Chen et al. 2018, 2019) is adopted initially as the decoding algorithm. The standard beam search sets a large beam

[*] Corresponding author

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Document: building a better game through dynamic programming a flip analysis. and suggest modifications to the rules to make the game more marketable. in addition to being an interesting application of dynamic programming, this case shows the use of operations research in managerial decision making.
Keyphrases:
-Present: dynamic programming; flip analysis; operations research; managerial decision making;
-Absent: solitaire board game; craft woodworkers.

Figure 1: An example of an input document and keyphrase labels. We list the present and absent keyphrases, respectively, and present keyphrases are highlighted in red in the document. We can see that the first words (marked with underline) of these keyphrases are different, which highlight the importance of discreteness for keyphrases generation.

size (e.g., 200) to over-generate at every decoding step, and only top-K(e.g., K=10) phrases are kept, where K is consistent for each document. As a result, it suffers from huge computation cost and can only generate a fixed number of keyphrases for various documents. To solve these problems, (Yuan et al. 2018; Chan et al. 2019; Chen et al. 2020) utilize a greedy search decoding strategy with a new training setup. Concretely, they concatenate keyphrases using delimiters into a sequence to make the model determine the number of keyphrases, denoted as one-to-many (one2many) mode. However, these methods tend to generate a large number of repeated keyphrases, and the extra noise (i.e., delimiter) aggravates the duplication issue (Yuan et al. 2018; Chan et al. 2019).

In this work, we present an effective method to increase the discreteness of the keyphrases generation, intending to mitigate the issues mentioned above. Our method is motivated by the observation that the first word per keyphrase is usually different from each other. It indicates that the diversities of first words are the basis of keyphrases' discreteness. We verify our intuition by counting the percentage of keyphrases with diverse first words in five popular datasets, as shown in Table 1. The average proportion reaches to near 89% on all datasets and the percentage of three datasets even over 90%.

With regard to the crucial role of the first words to interpret the discreteness of keyphrases, we propose the **Ada**ptive

Dataset	total	diff	percentage
Inspec	4913	4422	90.0%
Krapivin	2339	2188	93.5%
NUS	2458	1961	79.8%
SemEval	1466	1256	85.7%
KP20k	2903008	2781524	95.8%

Table 1: Results of the number of keyphrases with different (**diff**) first words accounts for (in **percentage**) the number of **total** keyphrases for each dataset.

keyphrase **G**eneration **M**odel (AdaGM). To enhance the discreteness and accuracy of generated keyphrases, we introduce a reset state mechanism during training to maximize the difference between the first words. When inference, an adaptive beam search decoding strategy is proposed by equipping it with a novel filter mechanism. Note that, benefit from the effectiveness of the reset state mechanism during the training phase, our adaptive decoding method can set a small beam size and determine the number of keyphrases adaptively. Therefore, it possesses the following virtues: (a) ability to predict a reasonable number of keyphrases for each document, (b) high discreteness, and (c) fast inference.

We evaluate the **AdaGM** on five public science article datasets. Our results significantly outperform the start-of-the-art on most metrics: the improvement gain of $DupRaito$ is up to 42.9%. For present (absent) keyphrase predictions, the improvement gain of F1-measure at 5 is up to 18.3% (38.7%).

We summarize our contributions as follows:

- We propose a reset state mechanism, which can reduce the impact of delimiter noise in the one-to-many mode and generate the first words accurately.
- We propose an adaptive beam search decoding method, which can encourage the model to generate less duplicated keyphrases with an appropriate number.
- Our AdaGM achieves the SOTA on most datasets and has higher computational effectiveness.

Related Work

Keyphrase Extraction and Generation

Extractive methods select important information from a document as keyphrases. Most of them follow two steps: over-extracting present candidates firstly and then ranking them according to their different scoring mechanisms (Hulth 2003; Kim et al. 2010; Wang, Sheng, and Wu 2017; Florescu and Caragea 2017; Mahata et al. 2018; Prasad and Kan 2019). Recently, sequence tagging models (Gollapalli, Li, and Yang 2017; Luan, Ostendorf, and Hajishirzi 2017; Alzaidy, Caragea, and Giles 2019; Sahrawat et al. 2020) are proposed to recognize keyphrases. However, these methods cannot predict absent keyphrases, which are important to understand a document.

To produce extra absent keyphrases, generative models are proposed. CopyRNN (Meng et al. 2017) is the first attention-based encoder-decoder model (Bahdanau, Cho, and Bengio 2015) with a copy mechanism (Gu et al. 2016) to generate present and absent keyphrases, and multiple extensions are proposed based on it. CorrRNN (Chen et al. 2018) considers the correlation between phrases. TG-Net (Chen et al. 2019) incorporates title information to enhance the representation of the input document. All of the above generative models rely on beam search with beam size as K, beam depth as M, and select top-N as the final prediction. Each document, as a result, predicts consistent N keyphrases regardless of its content, which is unreasonable.

In order to choose various keyphrases with an appropriate number for each document, we propose a scheme that concatenates keyphrases with delimiters as a sequence. The proposed scheme considers the relationships between keyphrases and generates keyphrases sequentially. catSeq and catSeqD (Yuan et al. 2018) propose a target encoding module and orthogonal regularization. Kp-RL (Chan et al. 2019) utilizes a reinforcement learning approach with adaptive rewards. Concurrent to us, ExHiRD (Chen et al. 2020) also points out the importance of the first words in keyphrase generation, and they propose a complex exclusive hierarchical decoding framework that includes a hierarchical decoding process and two exclusion mechanisms. However, these models generally generate some duplicated keyphrases, which is shown in Figure 6. To avoid the duplicated keyphrases, some models have adopted the complex mechanisms, while the effect is little. Our model integrates an effective reset state mechanism and a simple-filter-based beam search method to resolve this problem, which can generate diverse keyphrases accurately.

Diversity Generation Method

These many one2many tasks adopt a beam search to generate a wide range of results, which also causes some redundant results. To solve this problem, (Cho 2016) adds noise to the hidden state of the decoder at each step. (Li and Jurafsky 2016) proposes group-mechanism to select the top g from each group as final predictions. (Vijayakumar et al. 2016) adopts an additional diversity-promoting term to the log-likelihood before re-ranking. (Tam et al. 2019) applies a clustering-based method to filter meaningless candidates. However, these methods are still limited by producing a fixed number of results. Different from existing works, we corporate beam search with a simple but effective filter mechanism to get the dynamic number of outputs at the first decoding step.

Methodology

Problem Formulation

Suppose that we have a dataset $D = \{x^i, y^i\}_{i=1}^{M}$, where M is the number of documents, $x^i = \left(w_1^i,\ w_2^i, \ldots, w_{T-1}^i, w_T^i\right)$, T is the number of words in one document. For $y^i = \{y^{i,j}\}_{j=1}^{N_i}$, N_i represents the number of keyphrases in the corresponding document and $y^{i,j} = \left(y_1^{i,j}, y_2^{i,j}, \ldots, y_{L_j}^{i,j}\right)$, where L_j represents the number of words in the keyphrase $y^{i,j}$. In an overview, by maximizing the probability of $\prod_{i=1}^{M}\prod_{j=1}^{N_i} P\left(y^{i,j}|x^i\right)$, we force our model to generate accurate keyphrases.

Keyphrase Generation Basic Model

Training setup. For each document keyphrases pair (x^i, y^i), we join all keyphrases in y^i into one sequence, written as $y^i = \langle bok\rangle y^{i,1}\langle eok\rangle y^{i,2}\langle eok\rangle ... \langle eok\rangle y^{i,N}\langle eok\rangle$, where $\langle bok\rangle$ and $\langle eok\rangle$ are special tokens that indicate the start of the first keyphrase and the end of each keyphrase respectively. Using such (x^i, y^i) sample as training data, the encoder-decoder model (Bahdanau, Cho, and Bengio 2015) can generate all keyphrases.

Encoder module. We use an embedding lookup table to map each word into a dense vector with size d^e. To merge words in one document, we use a bi-directional Gated-Recurrent Unit (Bi-GRU) (Cho et al. 2014) as the encoder. The encoder converts the input document $x = (w_1, w_2, \ldots, w_{T-1}, w_T)$ into a collection of hidden states $H = (h_1, h_2, \ldots, h_{T-1}, h_T)$. The i-th hidden state $h_i = [\overrightarrow{h}_i; \overleftarrow{h}_i]$, where [;] means connection.

Decoder Module. We use a single-layered GRU as the decoder. The t-th decoder hidden state s_t is represented as:

$$s_t = GRU(embed(y_{t-1}), s_{t-1}), \tag{1}$$

where $embed(y_t)$ is the embedding of t-th predicted word y_t. We set s_0 as the last hidden state h_T of the encoder and $embed(y_0)$ as $embed(\langle bos\rangle)$. We also apply attention mechanism from (Bahdanau, Cho, and Bengio 2015):

$$a_{ti} = softmax(W_d(tanh(W_h h_i + W_s s_t))), \tag{2}$$

$$C_t = \sum_{i=1}^{T} a_{ti} h_i, \tag{3}$$

where a_{ti} is an attention score for w_i in the document and C_t is the context vector at the t-th decoding step. Besides, all W terms are trainable parameters and we omit the bias units. To alleviate the out-of-vocabulary (OOV) issue, we use the copy mechanism in (See, Liu, and Manning 2017):

$$P(y_t) = p_{gen}^t P_{vocab}(y_t) + (1 - p_{gen}^t) \sum_{i:w_i=y_t} \alpha_{ti}, \tag{4}$$

$$p_{gen}^t = sigmoid(W_{gen}[C_t; s_t; embed(y_{t-1})]), \tag{5}$$

$$P_{vocab}(y_t) = softmax(W_{v'}(W_v[C_t; s_t])). \tag{6}$$

It indicates the word generation is dependent on the probability of vocabulary $P_{vocab}(y_t)$ and the sum of the word attention scores at the t-th decoding step in the document(i.e., $\sum_{i:w_i=y_t} \alpha_{ti}$). $P_{vocab}(y_t)$ is achieved by a 2-layer feed-forward neural net (FNN). We project the concatenation of context vector C_t, decoder hidden state s_t and the embedding of y_{t-1} into a scalar by a linear transformation, and further apply $sigmoid$ function to convert the scalar scale into $p_{gen}^t \in [0, 1]$. p_{gen}^t is a soft gate to choose between generating a word from the vocabulary and copying from the document at t time step. We minimize the negative log-likelihood loss to train our model:

$$loss = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N_i}\sum_{t=1}^{L_j} logP(y_t|y_{<t}; x; \theta), \tag{7}$$

where θ denotes all the learnable parameters.

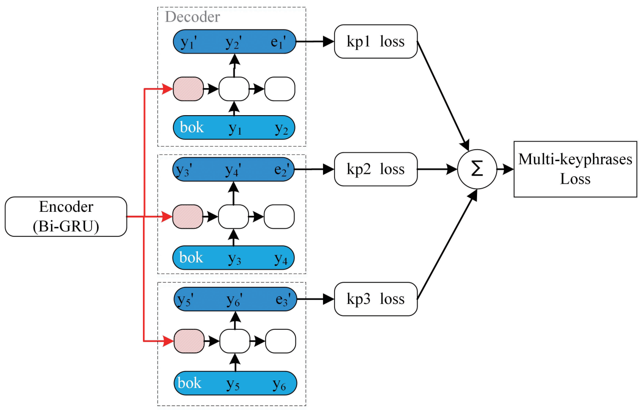


Figure 2: The model adopts Bi-GRU as encoder and GRU as decoder. We omit the encoder structure. The red areas denote the reset state mechanism. We reset decoder states including hidden states and input states when the last phrase is entered at the training stage. y'_n and e'_m represent decoder outputs.

Reset state mechanism

Keeping the difference of the first words of keyphrases and enhancing discreteness in mind, we implement the reset state mechanism by resetting the input and hidden states of the decoder in an one2many training mode, as shown in Figure 2. To be more specific, We reset the input state in the decoder with the character embedding $\langle bok\rangle$ and the hidden state with h_T from the encoder when the model input is a delimiter. The equation (1) is converted into:

$$s_t = GRU(embed(\langle bok\rangle), h_T), \tag{8}$$

unlike previous methods (Yuan et al. 2018; Chan et al. 2019) mark both the beginning and end of each keyphrase with the same token $\langle eok\rangle$, which tends to generate a large number of repeated predictions, the start and end of each keyphrase are distinguishable in our training process. As shown in Figure 2, just like a multi-classification task, the gradient of keyphrase can compete with each other and focus on the different parts of a document through the attention mechanism (Eq. 3), which helps to better model global semantics and maximize the discreteness of the first words at the first decoding step.

Adaptive beam search decoding method

The basis of discreteness of generated keyphrases is maximizing the difference between the first words of the keyphrases. A natural alternative is beam search during inference, which selects K different words at the first decoding step, as shown in Figure 3. However, huge computation cost

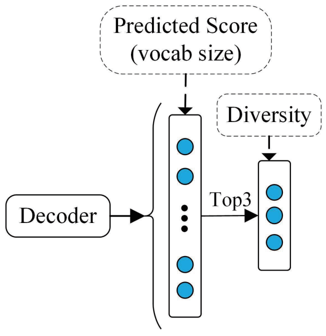


Figure 3: Standard beam search chooses different K words at the first decoding step. Here K is three.

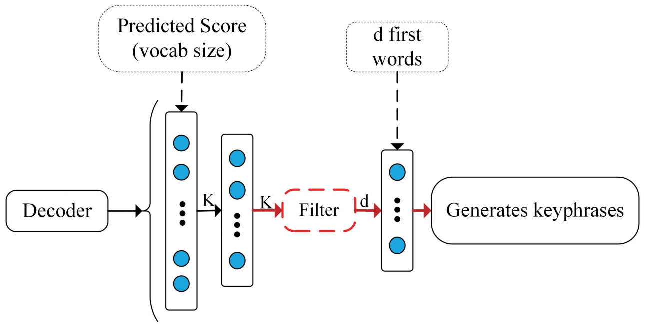
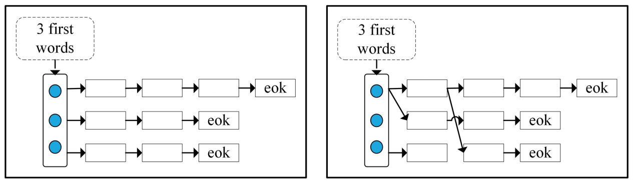


Figure 4: The process of adaptive beam search decoding method. At the first decoding step, we preserve Top-K first words. After applying the $Filter$ operation, we get $d\ first\ words$.

and number-fixed keyphrases generation make it hard for direct use. In order to satisfy the discreteness of the generated keyphrases but overcome the shortages of beam search, we propose an adaptive beam search decoding method.

Filter mechanism. To predict a reasonable number of keyphrases rather than fixed K for each document, we propose a simple but effective filter mechanism. Specifically, our model selects the top K different words at the first decoding time-step similar to standard beam search, and then filters out the words whose scores are lower than a threshold α. Note that since our reset state mechanism suppresses the score of wrong first words to a much lower degree (e.g., close to 0.0) than that of correct ones, this simple filter implemented with a threshold can preserve a reasonable number of the first words by filtering out the noise (i.e., the incorrect first words). Furthermore, our experiments show that the model can produce a high recall in the predicted first words, even set a very small K (e.g., 20) compared to a large number (e.g., 200) in previous methods. By avoiding the over-generation problem in standard beam search, a largely small number of K phrases helps to preserve memory-footprint. In Figure 4, we assume that d first words are kept after applying the filter mechanism, where d is a variable adaptive number for different documents.

Decoding method. Given the retained d first words, a direct way to generate keyphrases is in a fully parallelized



(a) fully parallelized generation method (b) semi-parallelized generation method

Figure 5: An example of generation processes in fully parallelized decoding and semi-parallelized decoding manner. Suppose that d =3 after applying the filter mechanism.

manner, as depicted in the left of Figure 5. The keyphrases generated in this manner have completely different first words. However, just as shown in Table 1, the **NUS** dataset still contains 18.7% repeated first words, which makes this method not optimal. A semi-parallelized manner based on beam search is adopted: the algorithm can share the same words in the previous decoding step in the current decoding step. It helps to preserve the opportunity to choose the identical first words for keyphrase generation adaptively.

Experiment Settings

Datasets

Experiments are carried out on five scientific publication datasets, including KP20k (Meng et al. 2017), Inspec (Hulth 2003), Krapivin (Krapivin and Marchese 2009), NUS (Nguyen and Kan 2007), and SemEval (Kim et al. 2010). We apply the same preprocess as (Chan et al. 2019): (a) removing all documents that are duplicated in itself, (b) filtering documents that have over 14 keyphrases or over 400 words in the training dataset. After the two operations, the training, validation, and testing samples of the KP20k dataset are 509,818, 20,000, 20,000, respectively. Following settings in (Meng et al. 2017; Yuan et al. 2018; Chen et al. 2018; Chan et al. 2019), we train models on KP20k and evaluate the models' performance on the testing part of all datasets.

Baseline Models and Evaluation Metric

Baseline models. We compare our model with seven generative models, including **CopyRNN** (Meng et al. 2017), **CorrRNN** (Chen et al. 2018), **TG-Net** (Chen et al. 2019), **catSeq**, **catSeqD** (Yuan et al. 2018), **Kp-RL** (Chan et al. 2019), and **ExHiRD** (Chen et al. 2020). We train the first five baselines with the setting mentioned in their published papers respectively. As for Kp-RL and ExHiRD, we use the best results in their papers.

Evaluation metrics. Following (Yuan et al. 2018; Chan et al. 2019; Chen et al. 2020), we adopt three important metrics, $F_1@5$ (F-measure@5), $F_1@M$, and $DupRatio$, for quantitative evaluation. $F_1@5$ uses the top 5 generated keyphrases and the ground-truth to compute F_1 score. If the model generates less than five predictions, we append some random answers to reach five. Thereby, similar $F_1@5$ and $F_1@M$ can be avoided. $F_1@M$ compares all predictions with the ground-truth without any modification, which is

Model	Inspec		Krapivin		NUS		SemEval		KP20k	
	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$
catSeq	0.235	0.273	0.272	0.357	0.309	0.376	0.247	0.292	0.288	0.365
catSeqD	0.223	0.264	0.256	0.340	0.318	0.393	0.230	0.279	0.280	0.359
Kp-RL	0.253	0.301	0.300	**0.369**	0.375	0.433	0.287	0.329	0.321	**0.386**
ExHiRD	0.235	0.291	0.286	0.347	—	—	0.284	0.335	0.311	0.374
AdaGM#	0.301	0.332	0.347	0.339	0.427	0.433	0.337	**0.346**	0.373	0.337
AdaGM	**0.305**	**0.348**	**0.363**	0.323	**0.442**	**0.438**	**0.343**	0.337	**0.388**	0.345

Table 2: Results of present keyphrases on five datasets. The best results are bold. The first four models adopt greedy search. AdaGM# and AdaGM denote that we adopt a fully- and semi- parallelized generation method, separately.

Model	Inspec		Krapivin		NUS		SemEval		KP20k	
	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$
catSeq	0.003	0.004	0.017	0.032	0.020	0.036	0.015	0.021	0.015	0.032
catSeqD	0.007	0.013	0.018	0.037	0.013	0.022	0.018	0.025	0.014	0.030
Kp-RL	0.012	0.021	0.030	0.053	0.022	0.037	0.021	0.031	0.027	0.050
ExHiRD	0.011	0.022	0.022	0.043	—	—	0.017	0.025	0.016	0.032
AdaGM#	0.010	0.018	0.026	0.049	0.021	0.033	0.026	0.036	0.022	0.042
AdaGM	**0.016**	**0.024**	**0.050**	**0.076**	**0.037**	**0.059**	**0.032**	**0.039**	**0.043**	**0.071**

Table 3: Results of absent keyphrases on five datasets. The best results are bold.

the real F_1 score. The $DupRatio$ evaluates the model's capability of avoiding generating repeated predictions, which equals the number of duplications divided by the number of predictions. For all of those, the macro average is reported. We also employ Porter Stemmer to match the two keyphrases on all the datasets except SemEval. The reason is that the keyphrases have been stemmed on the SemEval.

Implementation Details

In the preprocessing stage, following (Yuan et al. 2018; Chan et al. 2019), for each document, we lowercase all characters, replace digits with a special token <digit>, and further sort all the present keyphrase labels according to where they first appear in the document and append absent keyphrases. We set the vocabulary as the most frequent 50002 words and share it between the encoder and decoder. The word embedding is initialized by using a uniform distribution within $[-0.1, 0.1]$. We set the word embedding dimension as 100 and the hidden size of encoder and decoder both as 300. The initial state of the decoder is initialized as the encoder's last time-step's hidden state. Dropout with a rate of 0.1 is applied to both the encoder and decoder states.

During the training stage, we use the Adam optimization algorithm (Kingma and Ba 2014) with an initial learning rate of 0.001. The learning rate will be halved if the validation loss stops dropping. Early stopping is applied when validation loss stops decreasing for three contiguous checkpoints. We also set gradient clipping of 1.0, batch size of 32, and train our model for three epochs.

During the test stage, we apply the adaptive beam search method, in which beam-size=20 and threshold α=0.015, to generate the appropriate number of keyphrases. We calculate $F_1@5$ and $F_1@M$ after removing all the duplicated keyphrases.

Results and Analysis

Comparisons with the state-of-the-art methods

In this part of experiments, we compare AdaGM with state-of-the-art methods in terms of $F_1@5$ and $F_1@M$ on five datasets. According to whether the model can generate a dynamic number of keyphrases, we divide the seven baseline models into two categories: (a) catSeq, catSeqD, Kp-RL, and ExHiRD, which can generate a dynamic number of outputs. (b) CopyRNN, CorrRNN, and TG-Net, which preserve a fixed number of keyphrases. The comparison results among (a) are summarized in Table 2 and Table 3. In these tables, we also report the results of our model using the fully parallelized decoding method, denoted as AdaGM#. Since the models in (b) generate a fixed number of phrases, we only report $F_1@5$ results in Tables 4 and Table 5.

As shown in Table 2 and Table 3, our model achieves the best results on $F_1@5$ of all datasets and $F_1@M$ of most datasets. For instance, the present $F_1@5$ by AdaGM on NUS is 0.442, about 17.8% higher than that of Kp-RL at 0.375. However, AdaGM does not outperform the compared models on $F_1@M$ of Krapivin and KP20k in Table 2. This phenomenon is reasonable since our model tends to copy frequently occurring words in a document at the early stage, but these two datasets do not consider these words as the ground-truth. Note that AdaGM with fully parallelized decoding manner also gets great performance on most datasets, which indicates that even without considering the duplicated first words, our model still has better prediction results. From Tables 4 and 5, compared with the three baseline models using beam search, our model achieves the optimal or suboptimal $F_1@5$ on all datasets.

Model	**Inspec**	**Krapivin**	**NUS**	**SemEval**	**KP20k**
CopyRNN	0.258	0.342	0.432	0.336	0.371
CorrRNN	0.281	0.329	0.404	0.328	0.333
TG-Net	0.299	0.339	0.423	**0.349**	0.368
AdaGM	**0.305**	**0.363**	**0.442**	0.343	**0.388**

Table 4: Results of present keyphrases $F_1@5$ on five datasets. The best results are bold. All models employ beam search.

Model	**Inspec**	**Krapivin**	**NUS**	**SemEval**	**KP20k**
CopyRNN	0.015	**0.054**	**0.049**	0.027	0.041
CorrRNN	0.006	0.031	0.021	0.014	0.018
TG-Net	**0.016**	0.048	0.035	0.023	0.041
AdaGM	**0.016**	0.050	0.037	**0.032**	**0.043**

Table 5: Results of absent keyphrases $F_1@5$ on five datasets. The best results are bold. All models employ beam search.

Duplication Ratio of Predictions

We report the average DupRatio results in Table 6. It can be seen that our model reduces the duplication ratios on most of the datasets consistently and significantly. For instance, on the KP20k dataset, our model achieves 0.077, about 70.4% lower than catSeq, 71.9% lower than catSeqD, 77.4% lower than Kp-RL, and 23% lower than ExHiRD. Note that there are still some repeated results in our model, which is mainly caused by that we allow the model to select the same first words adaptively in a semi-parallelized manner.

Model	**Inspec**	**Krapivin**	**NUS**	**SemEval**	**KP20k**
catSeq	0.361	0.341	0.447	0.480	0.261
catseqD	0.371	0.327	0.374	0.333	0.274
Kp-RL	0.412	0.262	0.591	0.275	0.340
ExHiRD	**0.030**	0.140	—	0.091	0.100
AdaGM	0.067	**0.080**	**0.076**	**0.068**	**0.077**

Table 6: The results of average DupRatio on five datasets. The best results are bold.

Effect of two mechanisms

To analyze the impact of the reset state mechanism and the filter mechanism on the first words generation, we compare with three representative models: (a) CopyRNN, (b) Kp-RL, and (c) ExHiRD. In Table 7, we count the number of phrases (i.e., the number of first words) as $Oracle$ and the accurate first words generated by each model as $Num\#$ in each dataset. We also calculate a recall score, which equals $Num\#$ divided by $Oracle$. As shown in Table 7, our model achieves the best results on all datasets, which demonstrates

Dataset	**Oracle**	**Kp-RL**		**ExHiRD**		**CopyRNN**		**AdaGM**	
		Num#	recall	Num#	recall	Num#	recall	Num#	recall
Inspec	4913	1456	0.296	1613	0.328	2795	0.568	3323	**0.676**
Krapivin	2339	871	0.372	845	0.361	1354	0.579	1724	**0.737**
NUS	2458	21	0.008	307	0.125	1229	0.5	1249	**0.508**
SemEval	1466	299	0.204	—	—	575	0.392	699	**0.477**
KP20k	105560	43726	0.414	41823	0.396	73795	0.699	78824	**0.747**

Table 7: The first words recall scores. Oracle denotes the number of keyphrases in each test dataset, Num# means the number of the correct first words. The best results are bold.

Dataset	**catSeq**	**catSeqD**	**Kp-RL**	**AdaGM**	**Oracle**	
	MAE	**MAE**	**MAE**	**MAE**	**Avg#**	**Var#**
Inspec	6.82	7.32	5.43	**3.41**	9.57	22.42
Krapivin	4.88	4.15	**3.12**	6.37	5.24	6.64
NUS	8.71	8.37	6.86	**5.29**	11.54	64.57
SemEval	7.51	8.38	8.28	**3.63**	15.67	15.10

Table 8: Results of mean absolute error (MAE) on four datasets. The lower value is better. Oracle is the ground-truth keyphrase. Avg# and Var# denote the mean and variance of numbers of keyphrases per dataset, respectively.

these mechanisms can force our model to predict and preserve more accurate first words.

To further explain the effect of our filter mechanism, we adopt mean absolute error (MAE) to measure the differences between the number of predictions and the number of ground-truth keyphrases. The average and variance of the number of ground-truth are also reported in Table 8. We compare AdaGM with catSeq, catSeqD, and Kp-RL models, which all aim to generate a suitable number of keyphrases. From Table 8, our model can adaptively generate the appropriate number of keyphrases. For instance, on Inspec and NUS datasets, which have large differences in data distribution, our model still generates a smaller MAE than Kp-RL.

Inference time

To illustrate the advantage in the computation efficiency of our method, we compare the average inference time of our model with 6 baselines. For fair comparison, we choose the following settings: (a) use the same device, i.e., GTX-1080Ti, (b) set batch size=1, and (c) adopt the settings given in the corresponding papers. The results are summarized in Table 9. AdaGM achieves a considerably higher computational efficiency than other baselines, as shown in Table 9. For instance, AdaGM's time-consumption is only 89ms, which is about 3.6 times faster than that of CopyRNN.

Model	**CopyRNN**	**CorrRNN**	**catSeq**	**catSeqD**	**TG-Net**	**Kp-RL**	**AdaGM**
time(ms)	320	232	123	143	360	119	**89**

Table 9: The average time of predicting one document. The unit is milliseconds.

Threshold	present		absent		d
	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	
α=0.0 (no filter)	**0.392**	0.274	**0.054**	0.068	19.34
α=0.01	0.391	0.327	0.039	0.064	13.37
α=0.015	0.391	0.347	0.043	**0.075**	11.16
α=0.02	0.388	0.363	0.027	0.051	9.30
α=0.025	0.385	0.376	0.023	0.045	7.99
α=0.03	0.379	**0.386**	0.020	0.041	**7.01**

Table 10: Results of different thresholds on the KP20k validation set. α=n means the threshold is n. d refers to the average number of keyphrases predicted by the model under the current threshold.

Results on different threshold

In general, the selection of threshold α in the filter mechanism is related to the number of generated keyphrases. Hence, in this experiment, we aim to find the appropriate threshold of α. The following α are investigated: α= $\{0, 0.01, 0.015, 0.02, 0.025, 0.03\}$, and results are reported in Table 10. We also report the d value in Figure 4. As α increases, our model retains more reliable first words and generates fewer keyphrases. Therefore, the present $F_1@M$ is positively correlated with α. However, the present $F_1@5$ may lose some correct words, resulting in a decrease in its effectiveness. For the absent keyphrases, when α is greater than 0.015, $F_1@5$ and $F_1@M$ begin to decrease. This phenomenon is reasonable since the absent keyphrase is mainly dependent on the semantics of documents, the first word of absence may have little differences in scores. The performance of α=0.015 is competitive to no filter imposed (i.e., α=0) on $F_1@5$ and $F_1@M$ metrics. But the number of generated keywords is significantly reduced (nearly 50%), which fits the number of keyphrases in a document more closely. Hence, we set 0.015 as the final threshold.

Ablation Study

We perform an ablation study to further validate the reset state mechanism effect. Present keyphrases results are reported in Table 11. We set up two compared models: without any resetting operation and only without resetting the hidden state. From Table 11, the latter performs better than the former on Krapivin, $F_1@M$ on Inspec, and $F_1@5$ on SemEval datasets, which indicates that reset operation can eliminate some noise effect and generate more accurate keyphrases.

Case Study

We display an example of predictions of our model and four baselines in Figure 6. Compared to all baselines, our model can generate the appropriate number of keyphrases. In this case, the number of ground-truth keyphrases is 6, while only our model generates six keyphrases. Besides, our model can generate more discrete keyphrases. For instance, catSeqD and Kp-RL predict “game theory” keyphrase at least twice, while our model only generates it once. As for the present keyphrase, we note that all baseline models fail to predict the “operations research” keyphrase. But our AdaGM benefited

Model	Inspec		Krapivin		NUS		SemEval	
	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$	$F_1@5$	$F_1@M$
AdaGM	**0.305**	**0.348**	**0.363**	**0.323**	**0.442**	**0.438**	**0.343**	**0.337**
-reset hidden state	0.273	0.311	0.335	0.248	0.419	0.353	0.335	0.309
-reset all	0.287	0.309	0.315	0.247	0.423	0.354	0.328	0.310

Table 11: Results of present keyphrases on ablation experiment. “-reset all” represents we remove the holistic reset state mechanism and train the seq2seq model in a common manner. “-reset hidden state” means we do not reset the decoder hidden state, but only reset the input state. The best results are bold.

Document: building a better game through dynamic programming a flip analysis. and suggest modifications to the rules to make the game more marketable. in addition to being an interesting application of dynamic programming, this case shows the use of operations research in managerial decision making.
Keyphrases: dynamic programming; flip analysis; operations research; managerial decision making; solitaire board game; craft woodworkers.[6]

catSeq:	**dynamic programming**(2); **flip analysis**; flip flop; analysis(3); .; game; flip(2); programming(2).[13]
catSeqD:	**dynamic programming**; **flip analysis**; game theory(3).[5]
Kp-RL:	**dynamic programming**; **flip analysis**; game theory(3).[5]
ExHiRD:	**dynamic programming**(2); flip(2); game theory.[5]
Ours:	**dynamic programming**; game theory; **flip analysis**; **operations research**; decision making; managerial decision.[6]

Figure 6: Keyphrases generated by catSeq, catSeqD, ExHiRD, Kp-RL, and AdaGM. Bold keyphrases are correct and duplicated predictions are highlighted in blue. The digit in parentheses indicates the number of times that the repeated keyphrase has been generated. The digit in square brackets indicates how many phrases are generated.

from the reset state mechanism that can predict the accurate first word and then generates the corresponding keyphrase. In conclusion, our model has the advantage of generating more discrete keyphrases and achieving better results.

Conclusion and Future Work

In this paper, we emphasize the discreteness of the generated keyphrases. With the intuition that the first words of keyphrases are almost different from each other, we optimize the training stage with a reset state mechanism. As each document has a different number of keyphrases, the proposed AdaGM further embeds a filter-based adaptive beam search algorithm to enhance the discreteness and generate an appropriate number of keyphrases. Experiments demonstrate our model can yield better performance on most datasets. Our future work will focus on incorporating structure or syntax information.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2018YFC0830804) and the National Natural Science Foundation of China (61932014). We would like to thank Hou Pong Chan for their open-source code (Chan et al. 2019). Moreover, We sincerely thank Chongming Zhong and the five anonymous reviewers for their valuable comments.

References

Alzaidy, R.; Caragea, C.; and Giles, C. L. 2019. Bi-LSTM-CRF Sequence Labeling for Keyphrase Extraction from Scholarly Documents. In The World Wide Web Conference, WWW '19, 2551–2557.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In ICLR.

Chan, H. P.; Chen, W.; Wang, L.; and King, I. 2019. Neural Keyphrase Generation via Reinforcement Learning with Adaptive Rewards. In ACL, 2163–2174.

Chen, J.; Zhang, X.; Wu, Y.; Yan, Z.; and Li, Z. 2018. Keyphrase Generation with Correlation Constraints. In EMNLP, 4057–4066.

Chen, W.; Chan, H. P.; Li, P.; and King, I. 2020. Exclusive Hierarchical Decoding for Deep Keyphrase Generation. In ACL, 1095–1105.

Chen, W.; Gao, Y.; Zhang, J.; King, I.; and Lyu, M. R. 2019. Title-Guided Encoding for Keyphrase Generation. In AAAI.

Cho, K. 2016. Noisy Parallel Approximate Decoding for Conditional Recurrent Language Model. CoRR abs/1605.03835.

Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In EMNLP, 1724–1734.

Florescu, C.; and Caragea, C. 2017. A Position-Biased PageRank Algorithm for Keyphrase Extraction. In AAAI.

Gollapalli, S. D.; Li, X.-L.; and Yang, P. 2017. Incorporating Expert Knowledge into Keyphrase Extraction. In AAAI, AAAI'17, 3180–3187.

Gu, J.; Lu, Z.; Li, H.; and Li, V. O. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In ACL, 1631–1640.

Hulth, A. 2003. Improved Automatic Keyword Extraction Given More Linguistic Knowledge. In EMNLP, 216–223.

Kim, S. N.; Medelyan, O.; Kan, M.-Y.; and Baldwin, T. 2010. SemEval-2010 Task 5 : Automatic Keyphrase Extraction from Scientific Articles. In Proceedings of the 5th International Workshop on Semantic Evaluation, 21–26.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. ICLR abs/1412.6980.

Krapivin, M.; and Marchese, M. 2009. Large Dataset for Keyphrase Extraction. Technical Report DISI-09-055, DISI, Trento, Italy .

Li, C.; Xu, W.; Li, S.; and Gao, S. 2018. Guiding Generation for Abstractive Text Summarization Based on Key Information Guide Network. In ACL, 55–60.

Li, J.; and Jurafsky, D. 2016. Mutual information and diverse decoding improve neural machine translation. arXiv .

Luan, Y.; Ostendorf, M.; and Hajishirzi, H. 2017. Scientific Information Extraction with Semi-supervised Neural Tagging. In EMNLP, 2641–2651.

Mahata, D.; Kuriakose, J.; Shah, R. R.; and Zimmermann, R. 2018. Key2Vec: Automatic Ranked Keyphrase Extraction from Scientific Articles using Phrase Embeddings. In ACL, 634–639.

Meng, R.; Zhao, S.; Han, S.; He, D.; Brusilovsky, P.; and Chi, Y. 2017. Deep Keyphrase Generation. In ACL, 582–592.

Nguyen, T. D.; and Kan, M.-Y. 2007. Keyphrase Extraction in Scientific Publications. In ICADL.

Prasad, A.; and Kan, M.-Y. 2019. Glocal: Incorporating Global Information in Local Convolution for Keyphrase Extraction. In ACL, 1837–1846. Minneapolis, Minnesota.

Sahrawat, D.; Mahata, D.; Zhang, H.; Kulkarni, M.; Sharma, A.; Gosangi, R.; Stent, A.; Kumar, Y.; Shah, R. R.; and Zimmermann, R. 2020. Keyphrase Extraction as Sequence Labeling Using Contextualized Embeddings. In Jose, J. M.; Yilmaz, E.; Magalhães, J.; Castells, P.; Ferro, N.; Silva, M. J.; and Martins, F., eds., Advances in Information Retrieval, 328–335. Cham: Springer International Publishing.

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In ACL, 1073–1083.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In Advances in Neural Information Processing Systems 27, 3104–3112.

Tam, Y.-C.; Ding, J.; Niu, C.; and Zhou, J. 2019. Cluster-based beam search for pointer-generator chatbot grounded by knowledge. In AAAI.

Tang, Y.; Meng, F.; Lu, Z.; Li, H.; and Yu, P. L. H. 2016. Neural machine translation with external phrase memory. In arXiv.

Vijayakumar, A. K.; Cogswell, M.; Selvaraju, R. R.; Sun, Q. H.; Lee, S.; Crandall, D. J.; and Batra, D. 2016. Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models. arXiv abs/1610.02424.

Wang, Q.; Sheng, V. S.; and Wu, X. 2017. Keyphrase Extraction with Sequential Pattern Mining. In AAAI.

Yuan, X. E.; Wang, T.; Meng, R.; Thaker, K.; He, D.; and Trischler, A. 2018. Generating Diverse Numbers of Diverse Keyphrases. CoRR abs/1810.05241.