Efficient Keyphrase Generation with GANs

Giuseppe Lancioni[0000−0001−6211−9195], Saida S. Mohamed[0000−0002−2552−3356], Beatrice Portelli[0000−0001−8887−616X], Giuseppe Serra[0000−0002−4269−4501], and Carlo Tasso[0000−0001−5162−185X]

Università degli Studi di Udine, Udine, Italy
http://ailab.uniud.it/
{lancioni.giuseppe,mahmoud.saidasaadmohamed,
portelli.beatrice}@spes.uniud.it
{giuseppe.serra,carlo.tasso}@uniud.it

Abstract. Keyphrase Generation is the task of predicting keyphrases: short text sequences that convey the main semantic meaning of a document. In this paper, we introduce a keyphrase generation approach that makes use of a Generative Adversarial Networks (GANs) architecture. In our system, the Generator produces a sequence of keyphrases for an input document. The Discriminator, in turn, tries to distinguish between machine generated and human curated keyphrases. We propose a novel Discriminator architecture based on a BERT pretrained model fine-tuned for Sequence Classification. We train our proposed architecture using only a small subset of the standard available training dataset, amounting to less than 1% of the total, achieving a great level of data efficiency. The resulting model is evaluated on five public datasets, obtaining competitive and promising results with respect to four state-of-the-art generative models.

Keywords: Keyphrase Generation · GAN · Reinforcement Learning.

1 Introduction

A keyphrase is a sequence of words that summarizes the content of a whole document and expresses its core concepts. High quality keyphrases (KPs) can facilitate the understanding of a document and they are used to provide and retrieve information regarding the whole document on a high level. The world wide growth of digital libraries has made the task of automatic KP prediction both useful and necessary. There are many topics in which such ability could be positively applied, such as text summarization [34], opinion mining [1], document clustering [9], information retrieval [12] and text categorization [11].

KPs can be either present or absent. Present KPs are exact substrings of the document and can be *extracted* from its text, while absent KPs are sequences of

words which do not exist in the text, but can be *abstracted* from its contents. They are also referred to as extractive and abstractive KPs, respectively. The research community has made great efforts in the task of predicting KPs so far, and the proposed solutions all rely on the following two main approaches: 1. extraction of sequences of words from the document *(automatic extractive methods)*; 2. generation of words and phrases related to the document *(automatic abstractive methods)*.

Extractive approaches are only able to deal with present KPs [29, 18, 33]: the greatest drawback in this case is that predicted KPs are related to the written content of the source document, and not to its semantic meaning.

The other main approach is the abstractive one, which has been introduced to address the limitations of the extractive approaches and to better mimic the way humans assign KPs to a given document. Despite it being a more recent line of research, there are a good number of studies tackling this problem [19, 3, 4]. Abstractive methods are designed to produce sets of KPs which are not strictly related to the words of source text. In principle this method could be used to predict both absent and present KPs from a given source text. Generative models are best suited for abstractive approaches. A lot of examples can be found in literature in which such kind of models are used, mainly leveraging the Encode-Decoder framework [19, 3]. This architecture works by compressing the contents of the input (e.g. the text document) into a hidden representation using an Encoder module. The same representation is then decompressed using the Decoder module, which returns the desired output (e.g. a sequence of KPs). Recently, Generative Adversarial Networks (GANs) have been introduced in text generation task [31], and in particular in keyphrase generation [24]. GANs are based on an architecture that simultaneously trains two models: a generative model that captures the data distribution, and a discriminative model that estimates the probability that a sample came from the (real) training data rather than from the generator. The aforementioned approaches rely on the use of large datasets to perform training, with a high consumption in terms of computational resources.

In this paper we introduce a new GAN architecture for keyphrase generation with a focus on data efficiency: the aim is to only use a small subset of the training data and still achieve reasonably good results. The main contribution of our approach is the introduction of a novel Discriminator model based on BERT that is able to distinguish between human and machine-generated keyphrases leveraging on the powerful language model of BERT. A Reinforcement Learning strategy has been used in our architecture to overcome the problems given by the direct application of GAN in text generation. Our architecture achieved competitive results using only 1% of the available training samples compared to previous approaches.

2 Related Work

2.1 Automatic Keyphrase Extraction

Many different extractive approaches have been proposed in the literature, but most of them consist of the following two steps. Firstly, a reasonable number of KP candidates are extracted. The number of candidates usually exceeds the number of correct candidates and it is selected using heuristic methods. Secondly, a ranking algorithm is used to give a score to each candidate based on the source text. This whole process can be performed either in a supervised or unsupervised fashion. For supervised methods, this task is treated as a binary classification task [26, 21], and gives positive scores to the correct candidates in the list. Unsupervised methods aim to find central nodes of the text graph [20], or detect phrases from topical clusters [16].

There are also other studies that differ from the previously described pipeline. For example the authors in [33] applied an alignment model to learn the conversion from the source text to target KPs. Also, recurrent neural network have been used to build sequence labeling models to extract KPs from tweets [33].

2.2 Automatic Keyphrase Generation

Abstractive methods represent an important approach which is gaining a growing attention, as they allow to generate results which are more in the line of human expectations. Sequence-to-sequence (Seq2seq) models showed great success in keyphrase generation and can generate human-like results. They are based on the Encoder-Decoder framework, where the Encoder generates a semantic representation of the source text and the Decoder is responsible for generating target texts based on such semantic representation. CopyRNN [19] was the first specific Encoder-Decoder model to be used in the topic of keyphrase generation; it incorporates an attention mechanism. CorrRNN [3] was introduced later and focused on capturing the correlation between KPs. TG-Net [5] exploits the information given by the title to learn a better representation for the input documents. Chen et al. [4] leveraged extractive models to improve the performance of the (abstractive) keyphrase generation one. Ye et al. [30] proposed a semi-supervised approach considering a limited training dataset to improve the performance. All the previous approaches used the beam search algorithm to generate large number of KPs from which to choose the k-best ones as final predictions. CatSeq and CatSeqD [32] were the first two recurrent generative models with the ability to predict the appropriate number of predicted KPs for each document (instead of predicting a fixed number of KPs for each sample). CatSeq proposed several novelties. Firstly, an orthogonal regularization module to prevent the model from predicting the same word after generating the KP separator token. Secondly, semantic coverage, a self-supervised technique with the aim of enhancing the semantic content of the predictions.

Reinforcement Learning has been used in a wide range of text generation tasks [28, 22]. The generative models CatSeq, CatSeqD, CorrRNN and TG-Net have been improved by applying a Reinforcement Learning (RL) approach

with adaptive reward to produce their improved versions catSeq-2RF1, catSeqD-2RF1, catSeqCorr-2RF1 and catSeqTG-2RF1 [2]. In [24], the authors propose a keyphrase generation approach using Generative Adversarial Networks (GAN) conditioned on scientific documents. The architecture is composed of a CatSeq model as Generator and a hierarchical attention-based model as Discriminator. This was the first attempt to apply GAN in the Keyphrase generation task. This approach was able to show improvements in the generation of abstractive KPs, but no significant improvements in extractive KPs.

3 The proposed approach

The novelty of the approach presented in this paper is two-fold: first, we introduce a BERT Discriminator as part of a Generative Adversarial Networks (GAN) architecture for the keyphrase generation task; and second, we train our system with only a small amount of the available data to pursue data efficiency.

A general overview of the implemented system is given in Figure 1. It is based on two main components: a state-of-the-art Generator that relies on the Encoder-Decoder model and is able to generate a list of KPs for a given input text, and the new BERT-based Discriminator that is trained to separate the true KPs from the fake ones by giving them a score: the higher the score, the more likely is the keyphrase list to be real.

To overcome the well known problems of differentiability that arise when employing GAN architectures for text generation, Reinforcement Learning (RL) paradigm is adopted for training the system [31].

3.1 Formal Problem Definition

A source document $\mathbf{x}$ and the related list of M ground-truth keyphrases $\mathbf{y} = (\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^M)$ (True KPs) are represented by the pair $(\mathbf{x}, \mathbf{y})$. Both $\mathbf{x}$ and $\mathbf{y}^i$ are sequences of words:

$$\mathbf{x} = x_1, x_2, \ldots, x_L$$
$$\mathbf{y}^i = y_1^i, y_2^i, \ldots, y_{K_i}^i$$

where L and K_i are the number of words of $\mathbf{x}$ and of its i-th KP respectively.

A keyphrase generation model will predict a set of keyphrases $\hat{\mathbf{y}} = (\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2, \ldots, \hat{\mathbf{y}}^N)$ (Fake KPs) with the aim to reproduce the true ones, so that $\hat{\mathbf{y}} \equiv \mathbf{y}$.

3.2 Details of the System

Generator The task of the Generator G is to take a source document $\mathbf{x}$ and generate a sequence of predicted KPs $\hat{\mathbf{y}}$. For our system we chose catSeq [32] that is based on the CopyRNN [19], a generative model optimized for KP generation. It introduces the ability to predict a sequence of KPs that is obtained by concatenating together target KPs separated by a special token. In this way the

training schema moves from one-to-many to one-to-seq, and the system can be trained to generate a variable number of KPs. It also employs the Copy Mechanism [8] to deal with long-tail words, which are the less frequent words in the vocabulary of the input samples. They are removed to gain efficiency during training, but being frequently very specific for the topic of the document, they could be part of KPs. The Copy Mechanism employs a positional attention to give a score to the words surroundings the ones which were removed, recovering the best scoring ones. Implementation relies on a bidirectional Gated Recurrent Unit (GRU) [6] for the encoder, and a forward GRU for the decoder.

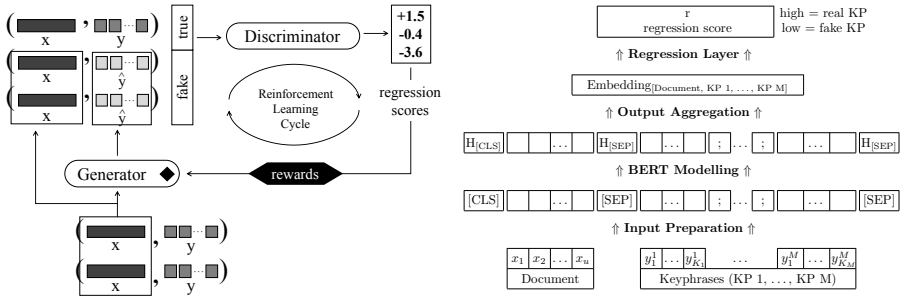

Fig. 1. *Left.* Schema of the proposed approach. *Right.* Detailed schema of the implemented BERT Discriminator.

Discriminator The Discriminator D is basically a binary classifier whose aim is to separate the true samples $(\mathbf{x}, \mathbf{y})$ from the fake ones $(\mathbf{x}, \hat{\mathbf{y}})$. It performs this task by computing a regression score for each sample, giving a high value to the reputedly real samples and a low value to the others.

We introduce a novel BERT-based model for our Discriminator. BERT [7] is part of the Transformer architecture, and since its introduction has achieved state-of-the-art results in many Natural Language Processing tasks. Our implementation is based on a BERT pretrained model, fine-tuned for Sequence Classification. The input samples are processed in four steps as (see Figure 1):

1. Input Preparation. Input pairs $(\mathbf{x}, \mathbf{y})$ are first lower-cased and tokenized, then the tokens are concatenated together in the form

`[CLS]<x>[SEP]<y1><;>...<;><yn>[SEP]`

where `<x>` and `<yi>` are the sequences of tokens of the input document and of the i-th KP; `[CLS]` is the BERT special token marking the start of the sequence; `[SEP]` is the BERT special token for marking the end of the sequence, also used to separate the input document from the list of related KPs; and the semicolon `<;>` is the KP separator.

2. BERT Modelling. The prepared input sequence is passed through the 12 consecutive Encoder blocks of the pretrained BERT model. Pretrained weights

act as initialization, and are optimized during training. Since BERT processing is positional, each input token is mapped to its corresponding output.

3. Output Aggregation. Output tokens are averaged together to obtain an Embedding of the whole input sequence. Note that generally when using a BERT-based model, the output of the `[CLS]` token is usually considered as a sentence embedding. Nevertheless, we averaged all the output tokens, as this aggregated value has proven to be a better estimate of the semantic content of the input (see also [7]).

4. Regression Layer. The sentence embedding is passed through a dense layer that evaluates a Regression score. This score is used to perform the classification of the input samples: the higher the score is, the more probable that the sample is a real one. The same score is also used as the Reward given by the Discriminator to the Generator in the Reinforcement Learning schema.

Reinforcement Learning with Policy Gradient We follow the Reinforcement Learning paradigm to train the system, as proposed in [31, 24].

In detail, we consider the Generator G as an agent whose action a at step t is to generate a *word* $\hat{y}_t$ which is part of the set $\hat{\mathbf{y}}$ of predicted KPs for the document $\mathbf{x}$. Action a is performed following the policy $\pi(\hat{y}_t|s_t, \mathbf{x}, \theta)$ that represents the probability of sampling $\hat{y}_t$ given the state $s_t = (\hat{y}_1, \ldots, \hat{y}_{t-1})$, the sequence of words generated up to step $t-1$. The policy is differentiable with respect to the parameters θ of G. As the agent G generates the predicted list of KPs, the Discriminator D, that plays the role of the environment, evaluates them and gives back a reward:

$$R(\hat{\mathbf{y}}) = r(\hat{y}_T|s_T) = D(\hat{\mathbf{y}}|\mathbf{x}) \tag{1}$$

where $r(\hat{y}_t|s_t)$ is the expected accumulative reward at step t and T denotes the steps needed to generate the whole prediction $\hat{\mathbf{y}}$. The aim of the agent G is to maximize the function $J(\theta)$ defined as the expected value of the final reward under the distribution of probability given by the policy π:

$$J(\theta) = \mathbb{E}_\pi \left[R(\hat{\mathbf{y}})\right] = \sum_{\hat{y}} r(\hat{y}|s) \cdot \pi(\hat{y}|s, \mathbf{x}, \theta) \tag{2}$$

The gradient of $J(\theta)$ is evaluated by means of the policy gradient theorem and the REINFORCE algorithm [25]:

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\sum_{t=1}^{T} r(\hat{y}_t|s_t) \cdot \nabla log\big(\pi(\hat{y}_t|s_t, \mathbf{x}, \theta)\big) \right] \tag{3}$$

Expectation $\mathbb{E}_\pi$ in Equation 3 can be approximated by sampling with $\hat{\mathbf{y}} \sim \pi(\cdot|\mathbf{x}, \theta)$. Then, defining the loss function of G as $L(\theta) = -J(\theta)$, an estimator of its gradient is:

$$\nabla L(\theta) \approx -\sum_{t} \big(r(\hat{y}_t|s_t) - b_t\big) \cdot \nabla log\big(\pi(\hat{y}_t|s_t, \mathbf{x}, \theta)\big) \tag{4}$$

A regularization term b_t has been introduced as an expected accumulative reward evaluated on a greedy decoded sequence of predictions, as suggested in [23]. Its aim is two-fold: to lower the variance of the process, and to support the predictions with a higher reward with respect to the greedy decoded sequence.

GAN Training Training is an iterative process in which G and D are trained separately, see Algorithm 1.

At the first step, an initial version G_0 of the Generator is trained using Maximum Likelihood Estimation (MLE) loss. Its predictions $(\mathbf{x}, \hat{\mathbf{y}})$ together with ground truth $(\mathbf{x}, \mathbf{y})$ are used to train the first version of the Discriminator D_0 with Mean Squared Error (MSE) loss. The regression scores evaluated by D_0 are then employed in the training of next Generator G_1, using the RL optimization as defined in Section 3.2. All the subsequent generators G_i are trained in the same way by means of the rewards given by D_{i-1}. Discriminators are always trained with MSE loss. *g-steps* and *d-steps* refer to the updating iterations during G and D training.

Algorithm 1: GAN training

Data: Samples $(\mathbf{x}, \mathbf{y})$
Pre-train G_0 with *MLE loss*; generate $\hat{\mathbf{y}}_0 = G_0(\mathbf{x})$;
Pre-train D_0 with *MSE loss*; evaluate $D_0(\mathbf{y})$ and $D_0(\hat{\mathbf{y}}_0)$;
while $D_i(\hat{\boldsymbol{y}}) \ll D_i(\boldsymbol{y})$ **do**
  i=i+1;
  for *g-steps* **do**
    Generate predictions: $\hat{\mathbf{y}} = G_i(\mathbf{x})$;
    Evaluate rewards: $R = D_{i-1}(\hat{\mathbf{y}})$;
    Update G_i with *Policy Gradient RL* maximizing R;
  end
  for *d-steps* **do**
    Generate predictions: $\hat{\mathbf{y}} = G_i(\mathbf{x})$;
    Evaluate $D_i(\mathbf{y})$ and $D_i(\hat{\mathbf{y}})$;
    Update D_i with MSE loss;
  end
end
G is evaluated on test datasets

4 Experiments and Results

4.1 Datasets

Five well known datasets largely used in literature have been considered in this work:

KP20k [19] 567,830 titles and abstracts from papers about computer science; of them, 20,000 samples are usually employed for testing, another 20,000 for validation, and the remaining 527,830 samples for training. This is the only

dataset used for training duties. In our data-efficient training approach we only use 2,000 out of the >500,000 training samples.

Inspec [10] 2,000 abstracts from disciplines: Computers and Control, and Information Technology. Only 500 samples are used for testing.

Krapivin [15] 500 articles about computer science. Since no hint is given by the authors on how to split testing data, the first 400 samples in alphabetical order are taken for testing.

NUS [21] 211 papers selected from scientific publications; used for testing.

Semeval2010 [13] 288 articles from the ACL Computer Library, of whose 100 are used for testing.

Some statistics about test samples are given in Table 1. Procedures that are standard protocol in KP generation are applied to data (see for example [2]): all duplicate documents are removed from training set; for each document the sequence of KPs is given in order of appearance; digits are replaced with the `<digit>` token; out of vocabulary words are replaced with the `<unk>` token.

The vocabulary of the generator V_G consists of the 50,000 most frequent words in the training dataset. The vocabulary of the discriminator V_D is the one of the pretrained BERT base uncased (english version): it contains 30,522 words and chunks of words (called wordpieces) eventually used to compose all the possible flections (e.g.: `'hexahedral'` is tokenized as `'he', '##xa', '##hedral'`).

Table 1. Statistics on test samples for the five datasets.

	KP20k		**Inspec**		**Krapivin**		**NUS**		**Semeval2010**	
	#	*%*	*#*	*%*	*#*	*%*	*#*	*%*	*#*	*%*
Present KPs	66,267	62.91	3,602	73.59	1,297	55.57	1,191	52.26	612	42.41
Absent KPs	39,076	37.09	1,293	26.41	1,037	44.43	1,088	47.74	831	57.59
Total KPs	105,343	100.00	4,895	100.00	2,334	100.00	2,279	100.00	1,443	100.00
Test samples	20,000		500		400		211		100	

4.2 Details of Implementation

Optimization of generator G is performed with Adam [14]. G_0 is trained with MLE loss and a batch size of 12; following G_i are trained with RL optimization and batch size of 32. Optimization of the discriminator D is performed with AdamW optimizer [17]. It is trained with MSE loss and a batch size of 3. For the Discriminator, we refer to the BERT implementation provided by huggingface [27][1]. The model is a bert-base-uncased with 12 layers, 12 attention heads, and hidden size of 768. Input sequences are trimmed to 384 tokens. The model is fine-tuned for Sequence Classification with one label (regression). Training and testing run on a PC with a Titan RTX GPU, 24GiB.

[1] https://github.com/huggingface/transformers

4.3 Comparative Results

The system has been trained with 2,000 samples randomly extracted from KP20K training dataset, and then evaluated using the five baseline datasets described in Section 4.1. A comparison has been carried out with four state-of-the-art approaches, namely catSeqD [32]; catSeqCorr-2RF1 and catSeqTG-2RF1 [2], and GAN [24]. Results are shown in terms of $F1$ score: $F1@5$ is evaluated over the top 5 high scoring KPs, while $F1@M$ takes into account all the predictions. Results are shown in Table 2.

Our approach achieves competitive results with respect of the above mentioned models. In depth, it is by far the best on INSPEC, both in $F1@M$ and $F1@5$ scores. Also it performs very well on SEMEVAL2010, where we match the best $F1@M$ score and are close to the best $F1@5$. Note that SEMEVAL2010 is the smallest of the five testing datasets and contains the smallest amount of target KPs. This makes it a very difficult test set to perform well on.

We point out that our approach shows good performance in the $F1@5$ score. Since the $F1@5$ score is evaluated taking the best 5 predicted KPs, we claim that our approach is able to generate good quality KPs.

A final consideration has to be made about Equation 3: the expectation of the policy function is evaluated using only complete sequences $\hat{\mathbf{y}}$, and this determines relatively large oscillations in the ∇J, inducing instability in the training process [31]. Our system has shown a great efficiency in dealing with this problem, even in a training scenario characterized by the scarce availability of resources in terms of data. In fact, we observed a trend to a quick convergence of the training, obtaining the best results at second iteration of the Generator (G_2). We consider that this quick convergence was achieved due to the strength of the language model embedded in the architecture.

Table 2. Results of present keyphrases for five datasets.

Model	**KP20k**		**Inspec**		**Krapivin**		**NUS**		**Semeval2010**	
	F1@M	*F1@5*	*F1@M*	*F1@5*	*F1@M*	*F1@5*	*F1@M*	*F1@5*	*F1@M*	*F1@5*
catSeqD [32]	-	**0.348**	-	0.276	-	**0.325**	-	0.374	-	**0.327**
catSeqCorr-2RF1 [2]	0.382	0.308	0.291	0.240	0.369	0.286	0.414	0.349	0.322	0.278
catSeqTG-2RF1 [2]	**0.386**	0.321	0.301	0.253	0.369	0.300	**0.433**	**0.375**	**0.329**	0.287
GAN [24]	0.381	0.300	0.297	0.248	**0.370**	0.286	0.430	0.368	-	-
Our approach	0.318	0.309	**0.383**	**0.356**	0.332	0.317	0.388	0.366	**0.329**	0.319

Ranking Analysis In order to better analyze our method of data-efficient training, we performed a comparison in terms of ranking metrics between our system (trained on 2,000 samples) and the original catSeq model (trained on the whole training set). Two evaluation measures have been used: the Mean Average Precision *MAP* and the normalized Discounted Cumulative Gain *nDCG*.

Table 3. Ranking measures for present KPs. Comparison of our approach (trained on 2,000 samples) and catSeq (trained on the whole dataset), on the five test datasets.

Model	KP20k		Inspec		Krapivin		NUS		Semeval2010	
	MAP	*nDCG*	*MAP*	*nDCG*	*MAP*	*nDCG*	*MAP*	*nDCG*	*MAP*	*nDCG*
catSeq	**0.305**	**0.585**	0.164	0.570	0.303	**0.576**	0.285	**0.740**	0.189	0.663
Our approach	0.300	0.560	**0.268**	**0.720**	**0.308**	**0.576**	**0.290**	0.736	**0.228**	**0.683**

MAP is defined as the mean of the average precision P scores evaluated for each set of predicted KPs. It is a measure of the proportion of relevant KPs among the predicted ones. *nDCG* is a measure of the usefulness, or gain, of a document based on its position, or rank, in the result list. It is widely used in information retrieval, specifically in web search and related tasks. For both the metrics the higher the scores, the better the accordance between the relevance of the predicted KPs with respect to the real ones.

Results are shown in Table 3. For four out of five datasets and for both measures, our approach achieves better or nearly the same results than the baseline catSeq, clearly showing the strength of our method.

5 Conclusion

In this paper we presented a system for Keyphrase Generation using a GAN architecture with Reinforcement Learning. Thanks to the characteristics of our approach, we have been able to train the system in a data-efficient way using only a small fraction of the available data. We tested it on five baseline datasets, achieving results that are competitive with some state-of-the-art generative models. To the best of our knowledge, this is the first attempt to train such a complex architecture for the demanding task of Keyphrase Generation in an scenario in which only a small amount of data is available.

References

1. Berend, G.: Opinion Expression Mining by Exploiting Keyphrase Extraction. In: IJCNLP (2011)
2. Chan, H.P., Chen, W., Wang, L., King, I.: Neural Keyphrase Generation via Reinforcement Learning with Adaptive Rewards. In: ACL (2019)
3. Chen, J., Zhang, X., Wu, Y., Yan, Z., Li, Z.: Keyphrase Generation with Correlation Constraints. In: EMNLP (2018)
4. Chen, W., Chan, H.P., Li, P., Bing, L., King, I.: An Integrated Approach for Keyphrase Generation via Exploring the Power of Retrieval and Extraction. In: NAACL-HLT (2019)
5. Chen, W., Gao, Y., Zhang, J., King, I., Lyu, M.R.: Title-Guided Encoding for Keyphrase Generation. In: AAAI (2019)
6. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: EMNLP (2014)

7. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: NAACL-HLT (2018)
8. Gu, J., Lu, Z., Li, H., Li, V.O.K.: Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In: ACL (2016)
9. Hammouda, K.M., Matute, D.N., Kamel, M.S.: CorePhrase: Keyphrase Extraction for Document Clustering. In: MLDM (2005)
10. Hulth, A.: Improved Automatic Keyword Extraction Given More Linguistic Knowledge. In: EMNLP (2003)
11. Hulth, A., Megyesi, B.: A Study on Automatically Extracted Keywords in Text Categorization. In: ACL (2006)
12. Jones, S., Staveley, M.S.: Phrasier: A System for Interactive Document Retrieval Using Keyphrases. In: SIGIR (1999)
13. Kim, S.N., Medelyan, O., Kan, M.Y., Baldwin, T.: SemEval-2010 Task 5 : Automatic Keyphrase Extraction from Scientific Articles. In: Workshop on Semantic Evaluation (2010)
14. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: ICLR (2015)
15. Krapivin, M., Autaeu, A., Marchese, M.: Large Dataset for Keyphrases Extraction. Technical Report DISI-09-055, University of Trento (2009)
16. Liu, Z., Li, P., Zheng, Y., Sun, M.: Clustering to Find Exemplar Terms for Keyphrase Extraction. In: EMNLP (2009)
17. Loshchilov, I., Hutter, F.: Fixing Weight Decay Regularization in Adam. ICLR (2017)
18. Luan, Y., Ostendorf, M., Hajishirzi, H.: Scientific Information Extraction with Semi-supervised Neural Tagging. In: EMNLP (2017)
19. Meng, R., Zhao, S., Han, S., He, D., Brusilovsky, P., Chi, Y.: Deep Keyphrase Generation. In: ACL (2017)
20. Mihalcea, R., Tarau, P.: TextRank: Bringing Order into Text. In: EMNLP (2004)
21. Nguyen, T.D., Kan, M.: Keyphrase Extraction in Scientific Publications. In: ICADL (2007)
22. Ranzato, M., Chopra, S., Auli, M., Zaremba, W.: Sequence Level Training with Recurrent Neural Networks. In: ICLR (2016)
23. Rennie, S.J., Marcheret, E., Mroueh, Y., Ross, J., Goel, V.: Self-Critical Sequence Training for Image Captioning. In: CVPR (2017)
24. Swaminathan, A., Gupta, R.K., Zhang, H., Mahata, D., Gosangi, R., Shah, R.R.: Keyphrase Generation for Scientific Articles using GANs. In: AAAI (2019)
25. Williams, R.J.: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning (1992)
26. Witten, I.H., Paynter, G.W., Frank, E., Gutwin, C., Nevill-Manning, C.G.: KEA: Practical Automatic Keyphrase Extraction. In: ACM (1999)
27. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Brew, J.: HuggingFace's Transformers: State-of-the-art Natural Language Processing. ArXiv:abs/1910.03771 (2019)
28. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., Dean, J.: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. CoRR (2016)
29. Ye, H., Wang, L.: Semi-Supervised Learning for Neural Keyphrase Generation. In: EMNLP (2018)

30. Ye, H., Wang, L.: Semi-Supervised Learning for Neural Keyphrase Generation. arXiv preprint arXiv:1808.06773 (2018)
31. Yu, L., Zhang, W., Wang, J., Yu, Y.: SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In: AAAI (2016)
32. Yuan, X., Wang, T., Meng, R., Thaker, K., He, D., Trischler, A.: Generating Diverse Numbers of Diverse Keyphrases. ArXiv:abs/1810.05241 (2018)
33. Zhang, Q., Wang, Y., Gong, Y., Huang, X.: Keyphrase Extraction Using Deep Recurrent Neural Networks on Twitter. In: EMNLP (2016)
34. Zhang, Y., Zincir-Heywood, A.N., Milios, E.E.: World Wide Web site summarization. Web Intelligence and Agent Systems (2004)