Keyphrase Generation with GANs in Low-Resources Scenarios

Giuseppe Lancioni **Saida S.Mohamed** **Beatrice Portelli**
Giuseppe Serra **Carlo Tasso**

AILAB, UniUd - University of Udine, Italy
{lancioni.giuseppe,mahmoud.saidasaadmohamed,
portelli.beatrice}@spes.uniud.it
{giuseppe.serra, carlo.tasso}@uniud.it

Abstract

Keyphrase Generation is the task of predicting Keyphrases (KPs), short phrases that summarize the semantic meaning of a given document. Several past studies provided diverse approaches to generate Keyphrases for an input document. However, all of these approaches still need to be trained on very large datasets. In this paper, we introduce BeGan-KP, a new conditional GAN model to address the problem of Keyphrase Generation in a low-resource scenario. Our main contribution relies in the Discriminator's architecture: a new BERT-based module which is able to distinguish between the generated and human-curated KPs reliably. Its characteristics allow us to use it in a low-resource scenario, where only a small amount of training data are available, obtaining an efficient Generator. The resulting architecture achieves, on five public datasets, competitive results with respect to the state-of-the-art approaches, using less than 1% of the training data.

1 Introduction

A Keyphrase (KP) is a piece of text that conveys the main semantic meaning of a document. KPs can be either present (or extractive) or absent (or abstractive): present KPs are exact substrings of the document while absent KPs are not. Their automatic prediction is an important challenge for the community research as KPs are a key component for a wide range of applications such as text summarization (Zhang et al., 2004), opinion mining (Berend, 2011), document clustering (Hammouda et al., 2005), information retrieval (Jones and Staveley, 1999) and text categorization (Hulth and Megyesi, 2006).

Historically, the first approaches focused on simply extracting substrings of the text to be used as keyphrases candidates (Ye and Wang, 2018; Luan et al., 2017; Zhang et al., 2016).

Recently, the research community has focused on the broader task of Keyphrase Generation (Meng et al., 2017; Chen et al., 2018, 2019a). Keyphrase Generation aims to *produce* a set of phrases that summarize the essential information in a given text, as opposed to simply *look for them* in the text. This allows for greater flexibility.

Several approaches introduced generative models based on the Encoder-Decoder architecture (Meng et al., 2017; Chen et al., 2018). 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). The modules are trained jointly to learn the best intermediate representation to perform this mapping.

More recently, an approach based on GAN (Generative Adversarial Networks (Goodfellow et al., 2014)) architecture has been proposed to address the task (Swaminathan et al., 2019). Although all these solutions achieved interesting results, they require a very large amount of data in order to be trained.

Our aim is to improve training efficiency, so that a model can be trained using only small subsets of the data. We focus our research in the generation of present KPs and we propose a new conditional GAN architecture for Keyphrase Generation that can be trained with a relatively small set of samples. The key component of our solution is the Discriminator: a model based on BERT that is able to distinguish between human and machine-generated Keyphrases leveraging on the language modelling information obtained from finetuning in a low-resource scenario. A Reinforcement Learning (RL) strategy is then used to train the Generator, with rewards evaluated by the Discriminator. This encourages the model to generate more accurate

Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing, pages 89–96
Online, November 20, 2020.

and relevant KPs.

Thanks to the characteristics of our architecture, we are able to use only a small subset of the available data, using less than 1% of them to train our system. Compared to all the previous approaches that needed to be fully trained on large set of training samples, our architecture greatly reduces required resources, while still providing competitive results in the generation of present KPs.

2 Related Work

2.1 Keyphrase Extraction

Extractive methods aim at identifying Keyphrases in the span of the source text. Most of the algorithms in this field adopt a two steps pipeline to extract KPs. First, given a document, a list of candidates phrases is selected using heuristic methods (Wang et al., 2016; Le et al., 2016). Secondly, all candidates are scored against the document. The first step has a considerable impact on the ability of the whole model to correctly identify all KPs, so selecting a sufficiently high number of candidates is of utmost importance. The second step can be done either in a supervised or unsupervised manner (Mihalcea and Tarau, 2004; Witten et al., 1999; Nguyen and Kan, 2007). The top-scoring candidates are returned as KPs. Two interesting strategies that differ from the common pipeline approach have been proposed by Tomokiyo and Hurst (2003) and Zhang et al. (2016). The first method employs two statistical language-based models to extract Keyphrases. The latter introduces a model based on joint layer recurrent neural network to extract Keyphrases from tweets.

2.2 Keyphrase Generation

Recently, research has focused on the introduction of methods of text generation to predict Keyphrases. Most of these approaches rely on Encoder-Decoder framework in which the source text is first mapped to an encoded representation, and then decoded to the target text, that is the Keyphrases to predict.

Meng et al. (2017) proposed CopyRNN, a RNN-based generative model for KP Generation, which is an Encoder-Decoder model with copy mechanism. Chen et al. (2018) proposed CorrRNN model which is a sequence-to-sequence architecture for Keyphrase Generation that captures the correlations among Keyphrases. TG-Net model was introduced by Chen et al. (2019b) for improving automatic Keyphrase Generation using the information contained in the title of the document. Chen et al. (2019a) proposed an integrated approach for Keyphrase Generation which is a multitask learning framework that jointly learns an extractive model and a generative model.

Two recurrent generative based models, CatSeq and CatSeqD, were proposed by Yuan et al. (2018). One of their main characteristics is the ability to determine the appropriate number of Keyphrases for each input document. CatSeq is based on an Encoder-Decoder mechanism, which is used to identify relevant components of the source text (abstracts) and generate KPs (sequence-to-concatenated sequences) (Yuan et al., 2018; Chan et al., 2019). It employed the sequence-to-sequence framework combined with an attention mechanism and pointer softmax mechanisms in the Decoder. CatSeqD introduces the following techniques: orthogonal regularization, which prevents the model from predicting the same word after generating the constant KP separator; semantic coverage, which encodes again the decoded sequences and uses it as a representation of the target phrases. These representations are employed as further input during a self-supervised training phase with the aim of improve the semantic content of the predictions.

Chan et al. (2019) subsequently proposed a Reinforcement Learning approach with adaptive rewards to improve catSeq, CatSeqD, CorrRNN and TG-Net generative models, leading to a new version for each of them. These versions are called, respectively, catSeq-2RF1, catSeqD-2RF1, catSeqCorr-2RF1 and catSeqTG-2RF1.

Recently, (Swaminathan et al., 2019) proposed a GAN model conditioned on scientific articles for KP Generation. The author uses a catSeq model to implement the Generator, conditioning it on abstracts of scientific articles. The Discriminator is based on a hierarchical attention mechanism consisting of two GRU layers. The two layers model the relationship between the document and each generated KP to assess whether the KP is synthetic or human in origin.

To the best of our knowledge, no attempts have been made of either extracting or generating KPs in a low-resources scenario, in which only a small amount of the available data samples is used during training. Our proposed architecture, based on a Discriminator that relies on a language model, requires less than 1% of the available training data to achieve good results.

3 The proposed Approach

To generate present KPs in a low-resource scenario we propose an approach based on the GAN Framework that we call BeGan-KP. It mainly consists of three components: (1) a conditioned Generator model that produces a set of KPs, (2) a novel Bert Discriminator model that checks if the KPs are fake (generated) or real (human-curated), and (3) the Reinforcement Learning (RL) module that is involved in the training process of the system as a whole (see Figure 1).

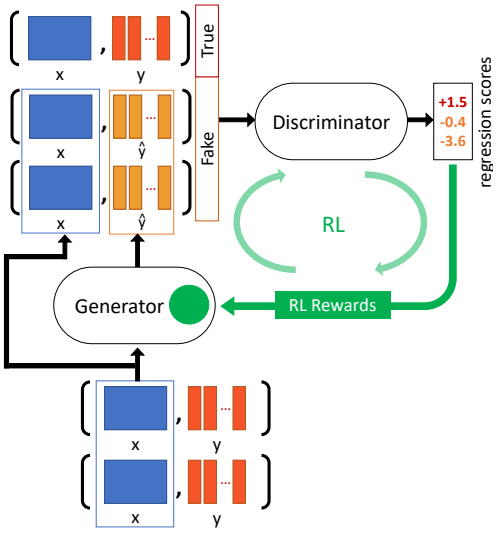

Figure 1: Schema of BeGan-KP.

3.1 Notations and Problem Definition

The samples available to train the system are pairs $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ is a document and $\mathbf{y} = (\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^M)$ is the set of M Keyphrases (True KPs) associated to $\mathbf{x}$. Note that 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.

The Generator takes as input $\mathbf{x}$ and outputs $\hat{\mathbf{y}} = (\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2, \ldots, \hat{\mathbf{y}}^J)$, that is the set of the J predicted KPs for $\mathbf{x}$ (Fake KPs).

The objective is to generate Fake KPs that match exactly the True KPs: $\hat{\mathbf{y}} \equiv \mathbf{y}$.

3.2 Generator

The Generator G takes as input the document $\mathbf{x}$ and generates as output the sequence of $\hat{\mathbf{y}}$ (Fake KPs).

Following the work of Swaminathan et al. (2019) we use the catSeq model as Generator. It consists in an Encoder-Decoder model in which the Encoder is a bidirectional Gated Recurrent Unit (GRU) and the Decoder is a forward GRU. It is based on CopyRNN by Meng et al. (2017).

We choose this component because it embeds some interesting features. It exploits the copying mechanism (Gu et al., 2016) to deal with long-tail words. These are words which are removed from the vocabulary due to their low frequency but are often topic-specific and therefore good candidates to be KPs. It also introduces the capability of predicting a variable number of Keyphrases for different documents. Furthermore it employs a beam-search strategy during the decoding step, meaning that at each time step the model decodes not just one word (greedy-search) but the top k most probable words. This allows generating more consistent sequences of words.

3.3 Discriminator

The Discriminator D receives as input the document $\mathbf{x}$ and a set of Keyphrases. These might be either the True KPs $\mathbf{y}$ or the Fake KPs $\hat{\mathbf{y}}$. Its task is to judge whether the KPs are True or Fake.

We introduce a novel Discriminator based on the language model BERT (Devlin et al., 2018). Differently from the previous literature, our idea is to exploit the strength of the language model characteristics to classify the quality of the input pair $(\mathbf{x}, \mathbf{y})$. This judgement is given as a *regression score*, which is lower for Fake KPs and higher for True KPs. In this way the regression score can be easily interpreted as the *reward* in the Reinforcement Learning module, giving to the system an inherent clarity. Moreover, different BERT-based models and reward configurations have been tested at an early stage, and the choice of a regression model provided the best results.

The language modelling component is able to achieve a better comprehension of the relationship of the two input sequences, while the robust pretraining allows us to use it efficiently even in a low-resource scenario.

In particular, the Discriminator model consists of four subcomponents (see Figure 2) :

- **Input preparation**. The input pairs $(\mathbf{x}, \mathbf{y})$ are tokenized and the tokens are concatenated to be compliant with the general pattern `[CLS]<x>[SEP]<y1><;>...<;><yn>[SEP]`.

`[CLS]` and `[SEP]` are special tokens which signal the start of the input and the end of text sequence respectively, `<x>` is the sequence of tokens for the document $\mathbf{x}$, `<yi>` is the sequence of tokens for the KP $\mathbf{y}^i$. Different KPs are separated by semicolon `<;>`. Note that the `[SEP]` token in the center is used to split the input sequence into document and KPs.

- **BERT modelling**. The input sequence is processed by a pretrained BERT model. It performs a word embedding of all the tokens and then passes them through 12 Encoder blocks. As it is basically a positional language model, it returns the last hidden states for each of the initial tokens.
- **Output aggregation**. Each of the outputs of the preceding step can be seen as an highly abstract embedding of the corresponding token. We aggregate the output of all the hidden states and evaluate their mean to obtain an embedding for the whole input sequence $E = E(\mathbf{x}, \mathbf{y})$. Note that in this way E is not generated using only the output obtained from the `[CLS]` token, but making use of the representations of all the tokens instead. Based on our preliminary experiments as well as literature references (Devlin et al., 2018), this value is considered to represent a better summary of the semantic content of the input.
- **Regression**. E is processed by the regression layer, a fully connected linear classifier, and a regression score is calculated. This is trained to be high for True KPs (human-curated) and low for Fake KPs (artificially generated), and is used as the reward in Reinforcement Learning.

The overall output of the Discriminator is therefore a regression score relative to the combination of input document and the related KPs.

3.4 Reinforcement Learning

To overcome the problem of non differentiability of the output layer of our architecture we extend the Reinforcement Learning strategy proposed by Yu et al. (2016) in the domain of KP Generation. In particular, we consider the Generator G as an agent whose action a at time step t is to generate a *word* y_t, which is part of the set of predicted KPs $\hat{\mathbf{y}}$ for the document $\mathbf{x}$. In this scenario the Discriminator

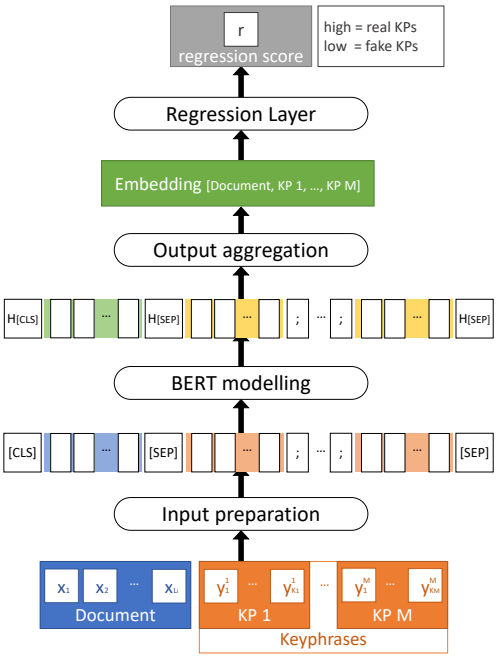

Figure 2: Schema of the Discriminator and its four processing phases.

D plays the role of the environment that evaluates the actions made by G and gives back a reward. Agent G acts following a policy

$$\pi = \pi(y_t | s_t, \mathbf{x}, \theta) \tag{1}$$

that is a function representing the probability distribution of y_t given the current state $s_t = (y_1, \ldots, y_{t-1})$, the sequence of words so far generated. The policy function is differentiable with respect to the set of parameters θ of G. Once the agent G generates the predictions, the environment D gives back a reward

$$r_t = f(y_1, \ldots, y_t | \mathbf{x}) \tag{2}$$

and moves to the state s_{t+1}. The reward is a quality measure of the action made by the agent G, and depends on the words generated up to the current time step (subset of $\hat{\mathbf{y}}$) given the input document $\mathbf{x}$. The agent G acts to maximize the reward, that is to maximize a differentiable optimization function $J(\theta)$ that gives a measure of the performance of G. According to the policy gradient theorem and the REINFORCE algorithm (Williams, 1992) the gradient of $J(\theta)$ can be expressed as:

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\sum_t r_t \nabla log(\pi(y_t | s_t, \mathbf{x}, \theta)) \right] \tag{3}$$

where the sum extends to all the time steps needed to generate the complete sequence $\mathbf{y}$.

The expectation $\mathbb{E}_\pi$ in Equation 3 can be approximated using a complete sequence $\hat{\mathbf{y}}$. In order to calculate the cumulative rewards of Equation 2 we use the regression score of a complete sequence of generated KPs: $r = D(\hat{\mathbf{y}})$.

Considering that maximizing the optimization function $J(\theta)$ is equivalent to minimizing its additive inverse, we can define the loss function of G as $L(\theta) = -J(\theta)$ and an estimator of its gradient as:

$$\nabla L(\theta) \approx -\sum_t (r - b) \nabla log(\pi(y_t|s_t, \mathbf{x}, \theta)) \quad (4)$$

where the regularization term b is introduced to reduce the variance of the above $\nabla L(\theta)$ estimator. It is essentially the cumulative reward $r = D(\bar{\mathbf{y}})$ where $\bar{\mathbf{y}}$ is a greedy decoded predicted sequence. The aim is to promote rewards that show effective improvements over greedy sequences (Rennie et al., 2017).

3.5 GAN Training

The first step is to train a first version G_0 of the Generator using the Maximum Likelihood Estimation (MLE). G_0 is then used to generate the Fake KPs $\hat{\mathbf{y}}$. $\hat{\mathbf{y}}$ and the ground truth $\mathbf{y}$ are used to train the first version of the Discriminator D_0 with Mean Squared Error (MSE) loss:

$$MSE(x, \hat{x}) = \frac{1}{N} \sum_{i=0}^{N} (x_i - \hat{x}_i)^2 \quad (5)$$

Starting from Generator G_1 training is performed using RL, so the loss is given by $L(\theta)$ as shown in Section 3.4. The training of the Discriminator remains the same. After each training iteration (G_j, D_j), predictions are tested to evaluate the scores $F1@M$ and $F1@5$.

4 Experiments and Evaluation

4.1 Datasets and Metrics

We compare our solution with state-of-the-art approaches on five datasets which are commonly used in literature:

KP20K (Meng et al., 2017) It consists of 567,830 titles and abstracts from computer science papers. The usual split is performed using 20,000 samples for testing, another 20,000 for validation, while the remaining 527,830 samples are used for training. In our low-resource scenario we only use 2,000 out of the >500,000 training samples.

INSPEC (Hulth, 2003) The complete dataset is composed of 2,000 abstracts from Computers and Control, and Information Technology disciplines. A subset of 500 samples is used for testing.

KRAPIVIN (Krapivin et al., 2009) The original released dataset is composed by 500 complete articles belonging to the domain of computer science. For KP Generation purposes only titles and abstract are used. The first 400 samples in alphabetical order are selected for testing.

NUS (Nguyen and Kan, 2007) A set of 211 scientific publications, all used for testing.

SEMEVAL2010 (Kim et al., 2010) 288 conference and workshop papers from the ACL Computer Library. 100 used for testing.

A brief report of main statistics of the test sets used is given in Table 1.

All datasets are preprocessed following Chan et al. (2019): duplicate papers are removed from KP20K, and for each document the list of KPs is sorted in order of appearance in the document. Digits in the input texts are replaced with the special token `<digit>`.

Results are evaluated using $F1$ score. In particular $F1@5$ and $F1@M$ are employed: the first is calculated considering only the top 5 high scoring KPs, the second is computed taking into account all the predictions.

All sample documents are annotated with human curated KPs. Of the above mentioned datasets, only KP20K is used for training; all the others are used only for testing and evaluation. Note that the strength of the language model of our Discriminator allows us to use only a small subset of the data samples during training: the whole architecture has been trained with a subset of 2,000 samples instead of the >500,000 used by the other state-of-the-art approaches.

4.2 Implementation Details

The initial MLE model G_0 is trained with a batch size of 12 and Adam optimizer (Kingma and Ba,

	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	

Table 1: Statistics on test samples for the 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 (Yuan et al., 2018)	-	**0.348**	-	0.276	-	**0.325**	-	0.374	-	**0.327**
catSeqCorr-2RF1 (Chan et al., 2019)	0.382	0.308	0.291	0.240	0.369	0.286	0.414	0.349	0.322	0.278
catSeqTG-2RF1 (Chan et al., 2019)	**0.386**	0.321	0.301	0.253	0.369	0.300	**0.433**	**0.375**	**0.329**	0.287
GAN (Swaminathan et al., 2019)	0.381	0.300	0.297	0.248	**0.370**	0.286	0.430	0.368	-	-
BeGan-KP (our approach)	0.318	0.309	**0.383**	**0.356**	0.332	0.317	0.388	0.366	**0.329**	0.319

Table 2: Results of present keyphrases for five datasets. Our approach is BeGan-KP.

2015); during RL training, batch size is 32. The Discriminator is trained with a batch size of 3 and AdamW optimizer (Loshchilov and Hutter, 2017). The pretrained BERT model is the base uncased version, with 12 layers, 12 attention heads, and hidden size of 768. The maximum input length after tokenization is fixed to 384 tokens. We use the implementation provided in the python library transformers by huggingface (Wolf et al., 2019)[1].

Training and experiments have been executed on a PC with a GeForce RTX 2080 GPU, 11GB.

4.3 Experimental Results

Our proposed solution BeGan-KP, trained on 2,000 samples, has then been compared with the following state-of-the-art approaches: catSeqD (Yuan et al., 2018); catSeqCorr-2RF1 and catSeqTG-2RF1 (Chan et al., 2019), and GAN (Swaminathan et al., 2019). The results of our tests are shown in Table 2.

First, we can note that BeGan-KP achieves results competitive with the best performing techniques, even using a limited set of samples (all the other approaches were trained on the whole KP20K).

Looking at the results in detail, we obtain by far the best performance for INSPEC both in $F1@5$ and $F1@M$.

Our approach has other good results in $F1@5$ metrics, specifically in KRAPIVIN and SEMEVAL2010 where our values are only slightly lower than the best. Since $F1@5$ is calculated considering the 5 predictions with the highest score, we can say that our model is capable of producing high quality Keyphrases reliably, and of outperforming or at least matching other best-performing models in this specific task. This confirms the strength and consistency of our architecture.

In addition, we obtain the best $F1@M$ score for SEMEVAL2010. Note that SEMEVAL2010 is a demanding test dataset as it is the smallest of the five, and the gross amount of KPs to predict is the lowest (612 present KPs out of a total of 1,443), leading to a great variance in the output.

Finally, consider that in Equation 3 the expectation of the policy function is evaluated using only one complete sequence $\hat{\mathbf{y}}$, inducing a high variance in the ∇J. This is a general issue of Reinforcement Learning applied to GANs for text generation and generally leads to unstable training process and slow convergence (Yu et al., 2016). Thanks to the capability of the language model embedded in our architecture, in our experiments the training process shows a quick convergence in terms of number of training iterations. In fact, the reported results have been achieved at the second iteration (G_2 generator).

5 Conclusion

In this paper we introduced an approach to the task of present Keyphrase Generation in a low-resources scenario, BeGan-KP. It is based on the GAN framework with a novel Bert based Discriminator model, trained by mean of the Reinforcement Learning paradigm. It has been tested on five public datasets showing performances competitive with state-of-the-art approaches while using less than 1% of the

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

available training data, achieving a great training efficiency.

References

Gábor Berend. 2011. Opinion Expression Mining by Exploiting Keyphrase Extraction. In *IJCNLP*.

Hou Pong Chan, Wang Chen, Lu Wang, and Irwin King. 2019. Neural Keyphrase Generation via Reinforcement Learning with Adaptive Rewards. In *ACL*.

Jun Chen, Xiaoming Zhang, Yu Wu, Zhao Yan, and Zhoujun Li. 2018. Keyphrase Generation with Correlation Constraints. In *EMNLP*.

Wang Chen, Hou Pong Chan, Piji Li, Lidong Bing, and Irwin King. 2019a. An Integrated Approach for Keyphrase Generation via Exploring the Power of Retrieval and Extraction. In *NAACL-HLT*.

Wang Chen, Yifan Gao, Jiani Zhang, Irwin King, and Michael R. Lyu. 2019b. Title-Guided Encoding for Keyphrase Generation. In *AAAI*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *ACL*.

Khaled M. Hammouda, Diego N. Matute, and Mohamed S. Kamel. 2005. CorePhrase: Keyphrase Extraction for Document Clustering. In *MLDM*.

Anette Hulth. 2003. Improved Automatic Keyword Extraction Given More Linguistic Knowledge. In *EMNLP*.

Anette Hulth and Beáta Megyesi. 2006. A Study on Automatically Extracted Keywords in Text Categorization. In *ACL*.

Steve Jones and Mark S. Staveley. 1999. Phrasier: A System for Interactive Document Retrieval Using Keyphrases. In *SIGIR*.

Su Nam Kim, Olena Medelyan, Min-Yen Kan, and Timothy Baldwin. 2010. SemEval-2010 Task 5 : Automatic Keyphrase Extraction from Scientific Articles. In *Workshop on Semantic Evaluation*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Mikalai Krapivin, Aliaksandr Autaeu, and Maurizio Marchese. 2009. Large Dataset for Keyphrases Extraction. Technical Report DISI-09-055, University of Trento.

Tho Thi Ngoc Le, Minh Le Nguyen, and Akira Shimazu. 2016. Unsupervised Keyphrase Extraction: Introducing New Kinds of Words to Keyphrases. In *Advances in Artificial Intelligence*.

Ilya Loshchilov and Frank Hutter. 2017. Fixing Weight Decay Regularization in Adam. *ICLR*.

Yi Luan, Mari Ostendorf, and Hannaneh Hajishirzi. 2017. Scientific Information Extraction with Semi-supervised Neural Tagging. In *EMNLP*.

Rui Meng, Sanqiang Zhao, Shuguang Han, Daqing He, Peter Brusilovsky, and Yu Chi. 2017. Deep Keyphrase Generation. In *ACL*.

Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order into Text. In *EMNLP*.

Thuy Dung Nguyen and Min-Yen Kan. 2007. Keyphrase Extraction in Scientific Publications. In *ICADL*.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-Critical Sequence Training for Image Captioning. In *CVPR*.

Avinash Swaminathan, Raj Kuwar Gupta, Haimin Zhang, Debanjan Mahata, Rakesh Gosangi, and Rajiv Ratn Shah. 2019. Keyphrase Generation for Scientific Articles using GANs. In *AAAI*.

Takashi Tomokiyo and Matthew Hurst. 2003. A language model approach to keyphrase extraction. In *ACL workshop on Multiword expressions*.

Minmei Wang, Bo Zhao, and Yihua Huang. 2016. PTR: Phrase-Based Topical Ranking for Automatic Keyphrase Extraction in Scientific Publications. In *ICONIP*.

Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*.

Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. 1999. KEA: Practical Automatic Keyphrase Extraction. In *ACM*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv:abs/1910.03771*.

Hai Ye and Lu Wang. 2018. Semi-Supervised Learning for Neural Keyphrase Generation. In *EMNLP*.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2016. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *AAAI*.

Xingdi Yuan, Tong Wang, Rui Meng, Khushboo Thaker, Daqing He, and Adam Trischler. 2018. Generating Diverse Numbers of Diverse Keyphrases. *ArXiv:abs/1810.05241*.

Qi Zhang, Yang Wang, Yeyun Gong, and Xuanjing Huang. 2016. Keyphrase Extraction Using Deep Recurrent Neural Networks on Twitter. In *EMNLP*.

Yongzheng Zhang, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2004. World Wide Web site summarization. *Web Intelligence and Agent Systems*.