Depthwise Max Pooling in Asymmetric Encoder-Decoders with Attention and Copy for Keyphrase Generation

Kai Wong
UCLA
kaileymon@gmail.com

Derek Xu
UCLA
derekqxu@gmail.com

Anton Lykov
UCLA
anton5798@ucla.edu

Akikazu (Allen) Miyazawa
UCLA
allenmiyazawa@g.ucla.edu

ABSTRACT

Keyphrase generation on Community-based Question Answering (CQA) websites is the prediction of keywords associated with given text content. There are known supervised and unsupervised algorithms used for solving this problem, where the current state of the art technique utilizes deep learning algorithms in the form of Encoder-Decoder frameworks, in addition to techniques such as attention and copying mechanisms. However, Encoder-Decoder models were originally designed for language translation tasks, so despite strong empirical results, the canonical Encoder-Decoder architecture may not be as optimal for the specific problem domain of keyphrase generation. We propose several architectural changes to improve on existing Encoder-Decoder frameworks for keyphrase generation, including (1) an asymmetrically stacked architecture, as well as (2) a novel approach to aggregate cell states of a stacked LSTM via depthwise max pooling. Our current best model AsymLSTM+DepthwiseMaxPool also utilizes attention and copying mechanisms similar to previous studies. It is able to outperform these existing methods even without the use of bidirectional layers, which has been widely considered as the new baseline for various natural language processing tasks. Our experiments were conducted using the *"StackSample: 10% of StackOverflow Q&A"* dataset that is available on *Kaggle*.

KEYWORDS

keyphrase extraction, keyphrase generation, text summarization, sequence-to-sequence, encoder-decoder, asymmetrically stacked encoder-decoder, depthwise max pooling, LSTM cell state, attention mechanism, copying mechanism

1 INTRODUCTION

Keyphrase extraction is the task of automatically predicting a set of keywords associated with a given body of text. These keyphrases may be used by various downstream tasks such as ranking search results, producing recommendations, or providing statistics on a database. Some examples of keyphrases being used in industry include YouTube video tags, academic paper keywords, and StackOverflow tags. With the increasing supply of large corpora of text data, automatic keyword extraction is becoming increasingly important. Because keyphrases capture the semantic meanings within large bodies of text, they act as excellent features for information retrieval [11], text summarization [24], text clustering [9] and text mining [2]. Improvements in automatic keyphrase extraction would likely improve the performance of existing algorithms tackling other complex problems in natural language processing (NLP) and data mining.

Many prior methods exist for keyphrase extraction. These include both unsupervised and supervised algorithms. Statistical models, such as Tf-Idf [10], have been available since the 1970s for keyword extraction. More recent graph-based models, such as TextRank [17], SingleRank [22], and TopicRank [5] provide an unsupervised way of extracting keyphrases. Supervised models include feature-based machine learning algorithms, such as KEA [23] and WINGNUS [18]. The current state of the art utilizes deep learning algorithms, such as CopyRNN proposed in Deep Keyphrase Generation [16].

The major drawback of unsupervised models is their inability to tailor their predictions to be specific to the given application. Tags for the same body of text would be very different in a StackOverflow question versus a YouTube description. Unsupervised models perform better as text summarization tools than as keyphrase extraction tools, which limits their ability to capture context from text documents in specific domains (ex. multimedia and computer science). This hampers their overall usefulness as a keyword extraction algorithm.

The current state of the art method utilizes recurrent neural networks (RNNs) which are able to capture complex relationships between inputs and outputs. Specifically, a recent paper published in ACL in 2017 [16] utilizes a seq2seq model with various useful additions, including attention [1] and a copying mechanism [8]. Much progress has been made in both deep learning and NLP communities, and we wish to incorporate these ideas and introduce novel designs to improve on existing Encoder-Decoder models for the problem domain of keyphrase generation.

2 PROBLEM FORMALIZATION

Suppose we have N samples in our dataset, where the i^{th} sample is represented as the tuple $(x^{(i)}, y^{(i)})$ such that $x^{(i)}$ represents an input text sequence of variable length, and $y^{(i)}$ represents the target keyphrases. Letting T_i denote the number of target keyphrases for the i^{th} sample and L_i denote the length of the i^{th} input text sequence, we can write:

$$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_{L_i}^{(i)})$$
$$y^{(i)} = (y_1^{(i)}, y_2^{(i)}, ..., y_{T_i}^{(i)})$$

where $y_j^{(i)}$ is the j^{th} target keyphrase of data sample i such that $y_j^{(i)} = (y_{j,1}^{(i)}, y_{j,2}^{(i)}, ..., y_{j,P_{i,j}}^{(i)})$ is also a sequence of words of length $P_{i,j}$.

Our proposed method is based on sequence-to-sequence learning [6] [21], where the algorithm is tasked to learn the mapping between an input sequence and a target sequence. Thus, for the i^{th} data sample with T_i target keyphrases, we simply split $(x^{(i)}, y^{(i)})$ into T_i separate tuples:

$$(x^{(i)}, y^{(i)}) = (x^{(i)}, y_1^{(i)}), (x^{(i)}, y_2^{(i)}), ..., (x^{(i)}, y_{T_i}^{(i)})$$

3 DATA PREPROCESSING AND ANALYSIS

In this section, we will cover the steps of data preprocessing and analysis. For our dataset, we use 10% of StackOverflow's questions and answers, which StackOverflow released in 2016 and is still available on Kaggle. The dataset consists of questions, answers, and their associated tags (keyphrases) from their popular programming CQA website.

3.1 Data Preprocessing

The dataset is organized into three tables:

(1) `Questions`: Contains the title, body, creation date, closed date, score and owner ID.
(2) `Answers`: Contains the body, creation date, score, and owner ID for all the answers associated to each question.
(3) `Tags`: Contains the tags associated with each question.

Given the convenient structure of the dataset, we created a combined dataset with the three given tables. Because many attributes of the dataset were unimportant for our given problem, we created a joined dataset containing 'Id', 'Score', 'Title', 'Body', 'Answers', and 'Tags'. The first four attributes were given in the Questions dataset. For 'Answers', we found all answers directed to the same question according to the 'id' and concatenated them. For 'Tags', we found the tags directed to the same question in the Tags dataset.

We combined all data from the 'Body' and 'Answers' attributes into a single feature containing all data from a given StackOverflow post. Since the data was stored in raw html format, we extracted the text components from the data using the python module *bs4* and applied basic cleaning procedures to them (substituting abbreviations, removing some html components not extracted by *bs4*). We then applied the standard nltk tokenization scheme on this feature (removing special characters, stop words, splitting on spaces or punctuation). We utilized the python *nltk* package to obtain our stop words, taking care to not remove symbols associated with special tags such as "c#" or "asp.net". Using the tokens, we constructed a vocabulary for the entire data-set using the 50,000 most frequently occurring words. With the vocabulary and given feature tokens, we could then associate a word embedding vector for each unique token that was learned from an embedding layer preceding our main model.

3.2 Data Analysis

The dataset from StackOverflow has a total of 1102568 posts each associated to a unique 'id', with a total of 3257174 tags (a tag may occur multiple times) and 35487 unique tags. Each post has up to 5 tags, and the number of tags is approximately modeled by a normal distribution with a mean of 2.9542 and a standard deviation of 1.2044.

Moreover, the frequency of the tags abides by Zipf's law: The top 100 most frequent tags (0.28% of all unique tags) constitute 47.43% of all tags, and the top 1000 most frequent tags (2.82% of all unique tags) constitute 76.07% of all tags. In addition, 6778 unique tags (19.10% of all unique tags) appeared only once in the dataset.

Furthermore, the top 10 most frequent tags (0.028% of all unique tags) constitute 22.34% of all tags. The tags are (in order): javascript, java, c#, php, android, jquery, python, html, c++ and ios. By observation, many tags seem to fall in similar Computer Science domains.

Building on this observation, we analyzed the likelihoods of certain tags appearing together. In particular, we measured the frequency of the tag 'javascript' co-occurring with the tags 'php', 'jquery', 'css', and 'html', all of which are related to 'javascript'. Of the 108637 times that 'javascript' appeared, it co-appeared with 'jquery' 68% of the time, with 'html' 38% of the time, with 'css' 17% of the time, and with 'php' 14% of the time.

4 METHODOLOGY

In this section, we will introduce our proposed model in detail. First, we will provide an overview of our asymmetric Encoder-Decoder model and how it was applied to our problem. We will then discuss the details of the model as well as its attention and copying mechanisms.

4.1 Asymmetric Encoder-Decoder Model

The notion of asymmetrically stacked Encoder-Decoder models had been previously explored by (Majumdar et al., 2017) [15], and were shown to perform better on compression tasks. The core function of an asymmetric Encoder-Decoder model is essentially the same as its symmetric (conventional) counterpart; the encoder converts the input source text into hidden states, which is then used by the decoder to generate the corresponding keyphrases.

In our model, both encoder and decoder components are implemented using LSTM layers. For simplicity, in all of our experiments the decoder comprised a single unidirectional LSTM layer along with varying configurations for the encoder LSTM layers. The hidden dimensions of all LSTM layers were kept constant as well, which differs from the work of [15] where different hidden dimensions were used within the multi-layered encoder.

We represent our encoder with K layers as $f = (f_1, f_2, ..., f_K)$. Given a variable-length input sequence $x = (x_1, x_2, ..., x_T)$, the encoder layers will generate $h_K = (h_{K,1}, h_{K,2}, ..., h_{K,T})$, a vector containing the hidden states of the last encoder layer at each time step t. For each encoder layer k at time step t, we can write:

$$h_{k,t} = f_k(h_{k-1,t}, h_{k,t-1}) \tag{1}$$

where f_k's are non-linear functions represented by LSTM layer k. Only h_K is fed into the attention and copying mechanisms (sections 4.3, 4.4) to produce a context vector c, which we denote as:

$$c = q(h_{K,1}, h_{K,2}, ..., h_{K,T}) \tag{2}$$

for some non-linear function q. The decoder decompresses the context vector c to generate the predicted variable-length output sequence $y = (y_1, y_2, ..., y'_T)$. In order to do so, the decoder, denoted by g, generates hidden states given by:

$$s_t = g(y_{t-1}, s_{t-1}, c) \tag{3}$$

where s_t is the decoder hidden state at time step t. The predicted keyphrases are thus generated through a conditional language model:

$$p(y_t|y_{1,...,t-1}, x) = \sigma(y_{t-1}, s_t, c) \tag{4}$$

where σ denotes the softmax classifier which outputs the probabilities of all the words in the vocabulary. y_t is the predicted word at time step t, which is the word associated with the highest probability value after applying σ.

During training, both encoder and decoder are trained to minimize the negative conditional probability (negative log likelihood) of the target sequence given a source sequence. During validation and testing, beam search is used to further refine the predicted word sequences, similar to the method described in (Meng et al., 2017) [16].

4.2 Encoder Cell States

In our model, we propose a novel design choice involving the handling of the final cell states of the encoder layers. In conventional symmetrically stacked architectures with K encoder and decoder layers each, the final cell state of the k^{th} encoder layer e_k is typically used as the initial cell state for the corresponding k^{th} decoder layer, i.e., ${d_k}^{(0)} = e_k$ (see Luong et al., 2015) [14].

In our asymmetrically stacked architecture, we have the option of using only the final cell state from the last encoder layer e_K as the initial cell state for our single decoder layer, i.e., $d^{(0)} = e_K$, or we could compress the information from all e_k's into a single encoder state vector e_{cell} to be used as the initial decoder cell state:

$$d^{(0)} = e_{\text{cell}} = \gamma(e_1, e_2, ..., e_K) \tag{5}$$

where γ is some operator. In our experiments, we tested a simple depthwise sum operator and compared it with the case where $d^{(0)} = e_K$. The purpose of the test was to determine the degree to which the design choice affected the model's overall performance. The results suggest that using $d^{(0)} = e_{\text{cell}}$ significantly improves the proportion of relevant predictions made by our model; there is less variance between F_1@5 and F_1@10 scores, and we even see a significant proportion of test instances where the F_1@10 scores are higher than the F_1@5 scores. The inverse was true when we used $d^{(0)} = e_K$, despite similar F_1@5 scores in both cases.

Encouraged by these results, our current best model implements depthwise max pooling on the encoder cell states:

$$d^{(0)}(i) = \max_{1 \le k \le K}(e_k(i)) \tag{6}$$

where $d^{(0)}(i)$ denotes the i^{th} element in the decoder's initial cell state. With this approach, we significantly outperform our baseline comparison case where we use $d^{(0)} = e_K$, which already had very good results. More details about our experimental results will be covered in section 5.3.

An intuitive explanation for the design choice is that the LSTM cell state represents the memory state up to a particular time step t, where the memory state stores the most relevant features up until t. The hidden state output at t is based on the memory state, but a filtered version of it. In particular, contents of the memory state are regulated through LSTM gates across t; the final cell state of an LSTM layer could thus be interpreted as the most significant features over the entire input sequence.

Similar to why deeper convolutional architectures perform better in computer vision tasks, stacked LSTM layers facilitate the capturing of more non-linear associations in data, in large part due to the non-linear activation of the LSTM's output or sigmoid gates. As such, different LSTM layers in a stack can learn different hierarchical feature representations of the input text. Going back to the intuition of the LSTM cell state, the final cell state of each LSTM layer thus represents the most significant features of the entire input sequence *within the same hierarchical representation.*

Relating this intuition to $d^{(0)}$ (initial cell state of decoder), we see that the naive case of setting $d^{(0)}$ as the last encoder layer's final cell state discards hierarchically lower features learned by the upstream encoder layers. In the context of keyphrase generation, or text summarization in general, feature representations of the input text from lower hierarchies could capture important information that are left out by the last encoder layer, which could adversely affect the quality of the decoder outputs for summarization problems. By applying depthwise max pooling on the final cell states of the encoder layers, we aim to select the sharpest features across hierarchies (layers) at each time step t to feed into the decoder.

Note: Our current best model uses 4 unidirectional LSTM layers for the encoder.

4.3 Attention Mechanism

Our model adopts a simple attention mechanism (Bahdanau et al., 2015) [1] to allow the encoder to better preserve contextual information from the input text. The context vector c (2) is computed as the weighted sum of each hidden state in the last encoder layer $h_K = (h_{K,1}, h_{K,2}, ..., h_{K,T})$:

$$\begin{aligned} c_i &= \sum_{j=1}^{T} \alpha_{i,j} h_{K,j} \\ \alpha_{i,j} &= \frac{\exp(a(s_{i-1}, h_{K,j}))}{\sum_{j'=1}^{T} \exp(a(s_{i-1}, h_{K,j'}))} \end{aligned} \tag{7}$$

where $a(s_{i-1}, h_{K,j})$ is a soft-alignment scoring function that measures the similarity between the decoder hidden state at the previous time step (s_{i-1}) and the final encoder layer's hidden state at the current time step ($h_{K,j}$). Empirically, our model performed marginally better when we used the additive/concat scoring function as seen in [1] compared to other scoring functions like dot/'general' (Luong et al., 2015) [14].

In addition, our model also utilizes attentional input-feeding [14] in which context vectors generated during a preceding time step c_{i-1} are fed as inputs to the decoder during the current time step via concatenation with the current inputs (decoder hidden state from previous time step, s_{i-1}). The purpose of doing so is to inform the model about prior alignment choices.

4.4 Copying Mechanism

In a similar fashion to CopyRNN (Meng et al., 2017) [16], our model also adopts the copying mechanism proposed by (Gu et al., 2016) [8]. The purpose of the copying mechanism is to allow the model to generate keyphrases containing words not found in its vocabulary through the use of positional and syntactic information from the input text, even without knowledge of their semantic meanings.

By incorporating the copying mechanism, the conditional language model (4) defined earlier becomes a mixture of generative and copy components:

$$p(y_t|y_{1,\dots,t-1},x) = p_g(y_t|y_{1,\dots,t-1},x) + p_c(y_t|y_{1,\dots,t-1},x) \quad (8)$$

where $p_g(y_t|y_{1,\dots,t-1},x)$ denotes the probability of predicting the next word using the vocabulary, and $p_c(y_t|y_{1,\dots,t-1},x)$ denotes the probability of predicting the next word using only the input source text. The original authors [8] defined the copy component of (8) as follows:

$$\begin{aligned} p_c(y_t|y_{1,\dots,t-1},x) &= \frac{1}{Z}\sum_{j:x_j=y_t}\exp(\psi_c(x_j)), x_j \in \chi \\ \psi_c(x_j) &= \tanh({h_{K,j}}^T W_c)s_t \end{aligned} \quad (9)$$

where $\psi_c(x_j)$ is the score for 'copying' the word x_j, $W_c \in \mathbb{R}^{d_{h_K}\times d_s}$ is a learned weight matrix, Z is the normalization term shared by both generate and copy components, and χ is the set of unique words from the input text x. For more details, please refer to [8].

As with the case where only attention is used, we also perform input-feeding with the location-based context vector generated by the copying mechanism, in conjunction with input-feeding using the content-and-semantics-based context vector generated by the attention mechanism [8].

5 EXPERIMENTS

This section will first cover details about the evaluation metrics we used to evaluate our model, followed by our experimental results compared alongside some baselines for keyphrase extraction as well as CopyRNN [16], which we were aiming to improve on.

5.1 Evaluation Metrics

We use macro-averaged *precision*, *recall*, and *F_1-score* (*F-measure*) at top-5 and top-10 to evaluate the performance of the algorithms. As per standard definition, we have:

- `Precision`: The ratio of correctly-predicted keyphrases to all predicted keyphrases.
- `Recall`: The ratio of correctly-predicted keyphrases to all correct keyphrases.
- `F-measure`: The weighted average of *precision* and *recall*.

When comparing keyphrases, exact matching is used, i.e., a predicted keyphrase is correct if and only if it is exactly the same as the ground truth keyphrase after Porter stemming has been applied.

5.2 Experimental Settings

Our models were implemented using 4 unidirectional LSTM layers for the encoder and a single unidirectional LSTM for the decoder. Each LSTM used a constant hidden dimension size of 512, with a word embedding dimension of 150 that was initialized using a uniform distribution in $[-0.1, 0.1]$. The models were trained using the Adam optimizer [12] with an initial learning rate of 1.0×10^{-4}, as well as a dropout rate of 0.2.

Method	Top-5 F_1	Top-10 F_1
TextRank	0.0878	0.0794
SingleRank	0.0550	0.0536
TopicRank	0.0538	0.0516
MultipartiteRank	0.0874	0.0739
PositionRank	0.0755	0.0693
CopyRNN	0.4240	0.4157
AsymLSTM	0.4865	0.4759
AsymLSTM + DepthwiseSum	0.4920	0.4878
AsymLSTM + DepthwiseMaxPool	**0.5198**	**0.5136**

Table 1: F_1-scores on the StackOverflow CQA dataset

During evaluation, we use beam search to generate the predicted keyphrases with a max search depth of 8 and a beam size of 32. The models were trained and periodically (once every 1000 mini-batches) evaluated on a validation dataset, whereby training was stopped early when the model's F_1-score on the validation sets did not improve for over 10 validation steps.

In some experiments, we performed training using Professor Forcing [13] at varying ratios, but we did not observe any improvement in model performance or convergence rate.

5.3 Experimental Results and Analysis

As comparative baselines, we tested several unsupervised methods in addition to the CopyRNN model proposed by [16]. The baselines were implemented using the *pke* python module as proposed in [3], which included unsupervised methods such as TextRank [17], SingleRank [22], TopicRank [5], PositionRank [7] and MultipartiteRank [4], as well as supervised methods including KEA [23] and others. Results for several configurations (different depthwise operations) of our proposed model AsymLSTM are presented in Table 1.

Our results in Table 1 are generally consistent with what was reported in [16], that CopyRNN significantly outperforms existing unsupervised keyphrase extraction methods. Moreover, we show that our proposed architecture is able to achieve better F_1-scores at both Top-5 and Top-10 than CopyRNN, which was implemented using the hyperparameter settings reported by the authors [16]. AsymLSTM refers to our proposed architecture (section 5.2) where we initialize the decoder cell state using only the last encoder layer's final cell state. AsymLSTM + DepthwiseSum/DepthwiseMaxPool refers to the depthwise operation applied to the final encoder cell states as described in section 4.2.

An additional observation was that our proposed models were more resistant to overfitting compared to CopyRNN, which recorded its peak F_1-scores within slightly fewer training iterations, but thereafter exhibited decreasing performance on validation and test sets even while training loss continued to improve. While this observation could be a direct result of overfitting, it could also be a result of the inconsistency between evaluation metrics used during testing and the optimization objective used during training; maximizing the likelihood of correctly predicting keyphrases during training may not directly correlate to better F_1-scores during evaluation. We briefly discuss this challenge in section 6.2, but for the experiments conducted in this paper it is worth noting that we intentionally overlook this issue.

To the best of our knowledge, our model outperforms all known supervised methods on the same dataset.

6 RELATED WORK

6.1 Encoder-Decoder Models

The basis of our model builds off of the Encoder-Decoder architecture first proposed by Cho et al. (2014) [6] and Sutskever et al. (2014) [21] in the problem domain of machine translation. The architecture has been widely studied in the field of natural language processing and has proven to be a powerful tool in modelling variable length sequences in an end-to-end manner.

To improve on vanilla Encoder-Decoder models, the attention mechanism proposed by Bahdanau et al. (2014) [1] has set a performance benchmark. Attention works by learning a soft-alignment that allows the model to map final outputs to their respective relevant input components (context). In addition, some studies have proposed mechanisms that utilize syntactic and positional information within the input text to extract keyphrases, such as the copying mechanism proposed by Gu et al. (2016) [8].

An important study conducted by Meng et al. (2017) [16] combines Encoder-Decoder models with attention and copying mechanisms in the problem domain of keyphrase generation as well. Their proposed model CopyRNN uses a bidirectional gated recurrent unit (GRU) as the encoder and a forward GRU as the decoder (see Cho et al., 2014 [6]). One of the main motivations of the study involves predicting absent keyphrases (keyphrases containing words that are not present in the source text). While not the primary focus of our paper, the design of our model has been heavily influenced by their work.

6.2 Training Algorithms

While not explicitly explored in our paper, we also looked at training algorithms designed to address the incongruities between the training and evaluation of models used in problem domains similar to keyphrase generation. During training, the objective is to maximize the likelihood of predicting a keyphrase given the source text, but during evaluation non-differentiable, discrete metrics like BLEU or ROUGE (with precision, recall and F-measure) are used. As a result, training and evaluation performance may not exhibit strong correlation. Studies like Paulus et al. (2017) [19] and Rennie et al. (2016) [20] proposed the use of reinforcement learning to enable the inclusion of these evaluation metrics into the optimization objective as policies.

6.3 Asymmetric Stacked Autoencoder

We did not find many studies that explored asymmetrical designs for Encoder-Decoder models. However, one such study by Majumdar et al. (2017) [15] showed that asymmetrically stacked autoencoders produced better classification accuracies than their symmetrical counterparts, and also performed better on compression tasks.

7 CONCLUSION

In this work, we examine various design choices to improve the performance of Encoder-Decoder models in the task of keyphrase generation. We detail the results of our findings and the effectiveness of various deep learning techniques. We found that asymmetrically stacked Encoder-Decoder models perform better than their symmetrical counterparts (even without bidirectional layers), and are still able to benefit from existing techniques such as attention and copying mechanisms. Our main contribution is the introduction of depthwise max pooling on encoder cell states, and we showed empirically how this improved the performance of an asymmetric Encoder-Decoder model. We discussed the intuition behind this novel mechanism for constructing initial decoder cell state inputs, discussed its theoretical applications, and presented its effectiveness in the problem domain of keyphrase generation.

We hope that our project brings further interest to keyphrase extraction and our results help other NLP, machine learning, and data mining researchers in their downstream tasks. In the future, we hope to incorporate other techniques and methods such as transformer architectures and reinforcement learning, as well as learning for special domain-specific texts (e.g. programming language sections in StackOverflow CQA text). For those interested in keyphrase generation, we hope this work's contributions can serve as a good foundation.

ACKNOWLEDGMENTS

We would like to thank Professor Yizhou Sun and our TA Zeyu Li for their guidance and teaching. Special thanks to Rui Meng (co-author of [16]) for open-sourcing parts of CopyRNN's PyTorch base code, and for his helpful correspondence in sharing additional hyperparameter settings and training details not discussed in his paper.

Task	People
1. Data Collection and Preprocessing	Kai, Derek
2. Raw Data Analysis	Allen
3. Unsupervised Baselines Implementation	Anton, Derek
4. Embedding Mechanisms (for future)	Derek
5. Model Implementation	Kai
6. Algorithms Evaluation	Kai
7. Report	Anton, Allen, Derek, Kai

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2015).

[2] Gábor Berend. 2011. Opinion expression mining by exploiting keyphrase extraction. (2011).

[3] Florian Boudin. 2016. pke: an open source python-based keyphrase extraction toolkit. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations.* Osaka, Japan, 69–73. http://aclweb.org/anthology/C16-2015

[4] Florian Boudin. 2018. Unsupervised Keyphrase Extraction with Multipartite Graphs. *CoRR* abs/1803.08721 (2018). arXiv:1803.08721 http://arxiv.org/abs/1803.08721

[5] Adrien Bougouin, Florian Boudin, and BÃľatrice Daille. 2013. TopicRank: Graph-Based Topic Ranking for Keyphrase Extraction. 543–551.

[6] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR* abs/1406.1078 (2014). arXiv:1406.1078 http://arxiv.org/abs/1406.1078

[7] Corina Florescu and Cornelia Caragea. 2017. PositionRank: An Unsupervised Approach to Keyphrase Extraction from Scholarly Documents. 1105–1115. https://doi.org/10.18653/v1/P17-1102

[8] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. *CoRR* abs/1603.06393 (2016). arXiv:1603.06393 http://arxiv.org/abs/1603.06393

[9] Khaled M Hammouda, Diego N Matute, and Mohamed S Kamel. 2005. Corephrase: Keyphrase extraction for document clustering. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 265–274.

[10] Thorsten Joachims. 1996. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.* Technical Report. Carnegie-mellon univ pittsburgh pa dept of computer science.

[11] Steve Jones and Mark S Staveley. 1999. Phrasier: a system for interactive document retrieval using keyphrases. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 160–167.

[12] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

[13] Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. 2016. Professor Forcing: A New Algorithm for Training Recurrent Networks. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 4601–4609. http://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent-networks.pdf

[14] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *CoRR* abs/1508.04025 (2015).

[15] Angshul Majumdar and Aditay Tripathi. 2017. Asymmetric stacked autoencoder. *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), 911–918.

[16] Rui Meng, Sanqiang Zhao, Shuguang Han, Daqing He, Peter Brusilovsky, and Yu Chi. 2017. Deep Keyphrase Generation. *CoRR* abs/1704.06879 (2017). arXiv:1704.06879 http://arxiv.org/abs/1704.06879

[17] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order Into Texts. In *EMNLP 2004*.

[18] Thuy Dung Nguyen and Minh-Thang Luong. 2010. WINGNUS: Keyphrase extraction utilizing document logical structure. In *Proceedings of the 5th international workshop on semantic evaluation*. Association for Computational Linguistics, 166–169.

[19] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A Deep Reinforced Model for Abstractive Summarization. *CoRR* abs/1705.04304 (2017). arXiv:1705.04304 http://arxiv.org/abs/1705.04304

[20] Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2016. Self-critical Sequence Training for Image Captioning. *CoRR* abs/1612.00563 (2016). arXiv:1612.00563 http://arxiv.org/abs/1612.00563

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *CoRR* abs/1409.3215 (2014). arXiv:1409.3215 http://arxiv.org/abs/1409.3215

[22] Xiaojun Wan and Jianguo Xiao. 2008. Single Document Keyphrase Extraction Using Neighborhood Knowledge. In *AAAI*.

[23] Ian H Witten, Gordon W Paynter, Eibe Frank, Carl Gutwin, and Craig G Nevill-Manning. 2005. Kea: Practical automated keyphrase extraction. In *Design and Usability of Digital Libraries: Case Studies in the Asia Pacific*. IGI Global, 129–152.

[24] Yongzheng Zhang, Nur Zincir-Heywood, and Evangelos Milios. 2004. World wide web site summarization. *Web Intelligence and Agent Systems: An International Journal* 2, 1 (2004), 39–53.