Pointer-Generator Network Summarization on TextRank-Preprocessed Documents

Long Tran
Department of Statistics
Stanford University
Stanford, CA 94305
longtran@stanford.edu

Abstract

Abstractive summarization on single documents remains a difficult task, despite recent improvements in neural network performance. We propose a framework that incorporates pointer-generator with coverage model, a state-of-the-art summarization framework, with TextRank preprocessing on input articles. Our model shows strong quantitative performance in terms of ROUGE scores, with potential qualitative advantages in factual error avoidance and novel n-gram generation.

1 Introduction

Document summarization is one of the most fundamental challenges in NLP. In its simplest form, the task involves *automatically* condensing a body of text into a shorter text while preserving the most critical information. If done correctly, summarization can have many important use cases in various domains, from generating preview snippets in web search to extracting knowledge from textual data [1]. The input of a summarization algorithm can be either a single text (single-document summarization) or several texts (multiple-document summarization). While multi-document summarization has seen significant progress recently, single-document summarization appears to have faced strong hurdles, largely due to the lack of redundancy over multiple input documents [2]. Nevertheless, the widespread deployment of deep learning in NLP, especially through the use of sequence-to-sequence models [3], has opened new possibilities in improving single-document summarization.

Generally speaking, there are two approaches to summarization: *extraction* and *abstraction*. Extractive summarization focuses on identifying important sections of the text input and organizing them verbatim, while abstractive summarization aims to generate out-of-text words and sequences. On one hand, abstractive summarization is closer to the human process of writing abstracts. On the other hand, extractive summarization has often performed better, because: a) copying words from the input usually guarantees a decent baseline, and b) abstractive summarization faces problems arising from semantic representation, inference, and natural language generation. Consequently, a majority of summarization research has focused on extractive summarization [1].

However, as mentioned above, recent advancements in sequence-to-sequence models and deep learning framework have made abstractive summarization methods more competitive [4, 5]. In this project, we seek to combine the advantage of extractive models in identifying key information with the capabilities of abstractive algorithms in representing semantics and generating language. More specifically, we apply the hybrid pointer-generator framework proposed by See, Liu, and Manning (2017) on input text that has been preprocessed by TextRank, an extractive sentence ranking algorithm that allows us to remove unimportant sentences [6]. The data used in this project come from *The New York Times Annotated Corpus* [7], consisting of more than 450,000 article-abstract pairs between 1987 and 2007. We show that this approach produces comparable performance compared to a pointer-generator network on full article input.

2 Background

Similar to neural machine translation, abstractive summarization has to overcome the challenge of aligning important words and phrases of the input (a full article) with words and phrases of the output (an abstract). To address this issue, Bahdanau, Cho, and Bengio (2014) proposed a sequence-to-sequence (or seq2seq) framework that allows for end-to-end neural network training and optimization on sequential input. This framework consists of two recurrent neural networks (RNN), an encoder RNN and a decoder RNN. The encoder reads the input sequence (i.e., a sequence of word vectors) and produces a sequence of encoder hidden state h_i as well as an encoding vector c. Each of the hidden states contains information on the whole input sequence, with additional emphasis on the information surrounding the i-th word of the sequence. The decoder, in contrast, predicts the next word $y_{t'}$, given the vector c and previously predicted word $\{y_1, \ldots, y_{t'-1}\}$.

While the basic conceptualization of sequence-to-sequence is useful, the model places a high burden on the last time step of the encoder to capture information from all previous timesteps. To mitigate this problem, Bahdanau, Cho, and Bengio (2014) also propose an attention mechanism in the decoder, which enables the decoder to decide which parts of the input sequence to pay attention to. Attention also alleviates the vanishing gradient issue often associated with RNN/LSTM, and potentially provides some forms of interpretability for the model. In the context of machine translation and document summarization, additional improvements include bidirectional RNN (which allows the encoder to take into account dependencies in both directions of the input sequence [8]), beam search decoding (a breadth-first tree search where the most promising sub-sequences are kept to maximize the conditional probability in the decoding phase), etc.

3 Approach

3.1 Baseline Model: Sequence-to-sequence with Attention

For the baseline, we implement a sequence-to-sequence attentional architecture with bidirectional encoding and beam search decoding. In this baseline framework, the encoder consists of a single-layer bidirectional LSTM, while the decoder uses a single-layer unidirectional LSTM. The attention distribution a^t is calculated as:

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + b_{\text{attn}}) \tag{1}$$

$$\alpha^t = \text{softmax}(e^t) \tag{2}$$

where v, W_h, W_s, and b_{attn} are trainable parameters, h_i is the encoder hidden state, and s_t is the decoder hidden state. The attention output a_t is calculated as a weighted sum of the encoder hidden states:

$$a_t = \sum_i \alpha_i^t h_i \tag{3}$$

The attention output a_t is then concatenated with the decoder hidden state s_t in order to produce the vocabulary distribution P_{vocab}, which is a probability distribution over the entire vocabulary:

$$P_{\text{vocab}} = \text{softmax}(V'(V[s_t, a_t] + b) + b') \tag{4}$$

where V', V, b, and b' are trainable parameters. The vocabulary distribution allows us to calculate the probability of a word w:

$$P(w) = P_{\text{vocab}}(w) \tag{5}$$

In addition, at decode-time, our baseline model uses beam search to keep track of the k most probable partial summaries. Finally, we use the negative log-likelihood to calculate the loss at each timestep t:

$$L_t = -\log P(w_t) \tag{6}$$

where w_t is the target word at that timestep. A visualization of the attention distribution is included in the supplementary materials.

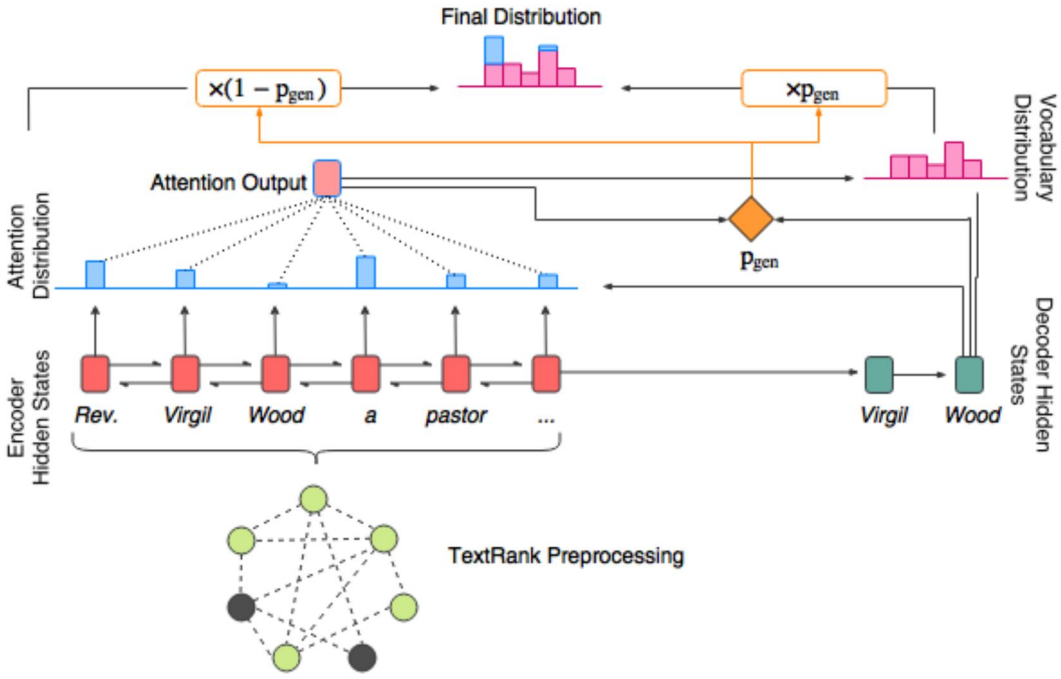

Figure 1: Pointer-generator model with TextRank preprocessing. Each vertex represents a sentence, while each edge implies that the two sentences share at least one token with each other. The TextRank score of each sentence is calculated, and only sentences with the highest scores (green vertices) are used as input. In the pointer-generator network, the generation probability p_{gen} is calculated. The final distribution is the weighted sum between the vocabulary distribution and the attention distribution, weighted by p_{gen}.

3.2 Extension: Extractive Preprocessing with TextRank

Despite improvements in performance of sequence-to-sequence variants, the overall framework is still relatively limited in its ability to effectively capture long input articles. As a result, many abstractive models use preprocessing to compress the input sequences, often through introducing a word cut-off. For example, See, Liu, and Manning find that using only the first 400 words in each article significantly increase ROUGE scores. Similarly, Paulus, Xiong, and Socher (2017) limit the input to the first 800 words. While it is true that English news articles often put the most important sentences in the beginning, this approach may be less effective depending on the author's style or the article's type (e.g., reviews, commentary, narrative, etc.).

Therefore, our preprocessing uses an extractive algorithm called TextRank [6], which allows us to rank the relative importance of sentences within an article. The algorithm works by first building a graph, in which each vertex represents a sentence in the article. The weight of the undirected edge between two vertices can then be defined using the Levenshtein distance between two sentences, which is the number of insertions, deletions, or substitutions of words to turn sentence S_i into sentence S_j, divided by the number of words in the longer sentence. The score of each vertex V_i can then be calculated as:

$$S(V_i) = (1-d) + d \sum_{V_j \in \text{In}(V_i)} \frac{\text{weight}_{ji}}{\sum_{V_k \in \text{Out}(V_j)} \text{weight}_{jk}} S(V_j) \tag{7}$$

where weight$_j i$ is the weight of the undirected edge between vertices V_i and V_j, In(V_i) and Out(V_i) are the set of vertices that share at least one word token with V_i (since the graph here is undirected, In(V_i) is the same as Out(V_i)). Factor d (the damping factor), which borrows from the conceptualization of PageRank [9], is the probability that a random surfer continue to "click" on vertex at random, and $1 - d$ is the probability that the surfer jumps to a new vertex. We use the default level of d at

0.85 in our experiments. Using the vertex scores, we can keep only the most important sentences as input to the sequence-to-sequence models.

3.3 Extension: Pointer-generator Network

Because the baseline model uses a fixed vocabulary, it tends to have difficulty learning the representations of out-of-vocabulary words. In many cases, these out-of-vocabulary words may contain important information that is specific to a particular article, but do not appear often enough across articles to be included into the vocabulary [4]. A common solution to this problem is to include some pointer in the decoder network that can point to specific phrases in the input sequence and copy them onto the output sequence [10]. Our implementation follows See, Liu, and Manning (2017)'s model, which calculates the generation probability $p_{\text{gen}} \in [0, 1]$ at each timestep:

$$p_{\text{gen}} = \sigma(w_a^T a_t + w_s^T s_t + w_x^T x_t + b_{\text{ptr}}) \tag{8}$$

where w_a, w_s, w_x, and $b_t extrmptr$ are trainable parameters, and x_t is the decoder input. For each document, considering the union set of the vocabulary and all words in that document as an extended vocabulary, we can derive the probability distribution over the extended vocabulary:

$$P_{\text{extended vocab}}(w) = p_{\text{gen}} P_{\text{vocab}}(w) + (1 - p_{\text{gen}}) \sum_{i:w_i=w} \alpha_i^t \tag{9}$$

See Figure 1 for an illustration of our TextRank preprocessing and pointer-generator network.

3.4 Extension: Coverage

The baseline sequence-to-sequence model also suffers from repetition (i.e., the output sequence contains the same phrase multiple times without much coherence). To address this issue, we use the approach proposed by See, Liu, and Manning (2017). In particular, the model keeps a coverage vector c^t, which simply sums the attention distribution over previous decoder timesteps:

$$c^t = \sum_{t'=0}^{t-1} \alpha^{t'} \tag{10}$$

This coverage vector is incorporated into the attention mechanism, so that:

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + w_c c_i^t + b_{\text{attn}}) \tag{11}$$

where w_c is a trainable parameter. We note that this is a modification from equation (1). Given that c^t contains information on which words in the input sequence have received more or less focus from the attention mechanism, the attention's focus at the current timestep is influenced by its previous choices. Intuitively, this enables the attention mechanism to avoid excessively attending to the same locations.

Additionally, See, Liu, and Manning also considers a coverage loss to further penalize repetition:

$$\text{covloss}_t = \sum_i \min(\alpha_i^t, c_i^t) \tag{12}$$

More specifically, this coverage loss disincentivizes the model from having an overlap between the attention distribution and the coverage up to time step t. This coverage loss is also added to the primary loss function previously specified in equation (6):

$$L_t = -\log P(w_t) + \lambda \sum_i \min(\alpha_i^t, c_i^t) \tag{13}$$

where λ is a hyperparameter representing the coverage weight.

4 Approach

4.1 Dataset

We use *The New York Times Annotated Corpus* [7], which contains articles published by The New York Times between January 1, 1987 and June 19, 2007 and article summaries written by library

scientists. We combine and modify the code of Li, Thadani, and Stent (2016) and See, Liu, and Manning (2017) to obtain 445,016 training article-summary pairs, 24,619 validation pairs, 24,789 test pairs. On average, the articles' full texts are 16-17 sentences (250-260 tokens) long, and the abstracts are approximately 3 sentences (45-50 tokens) long. We keep the original text largely non-anonymized, except for author's name that appears in the abstract (e.g., "John Doe writes a column on ..."). This is because it is often impossible to deduce author's name from the article's text.

4.2 Experiments

The scripts to run our experiments rely heavily on See, Liu, and Manning (2017)'s implementation, with modifications at the preprocessing step to incorporate the TextRank preprocessing. All our experiments use a vocabulary of 50k words for source and target. Word embeddings are not pre-trained and are learned during training. The Adaptive Gradient Algorithm (AdaGrad) is used for learning [12], together with a gradient clipping with a maximum norm of 2.

Unlike See, Liu, and Manning (2017), we do not truncate the article to 400 tokens, nor do we limit the length of the summary for training and testing. For our extractive preprocessing experiments, we use TextRank to assign an importance score between 0 and 1 (1 denoting the highest importance) for each sentence within an article. Our implementation of TextRank borrows from Adamor (2017). The default damping factor of 0.85 is used. For articles containing fewer than six sentences, all sentences are kept. For articles containing at least six sentences and at most ten sentences, we keep five sentences with the highest scores. For articles containing more then ten sentences, we keep the half the number of sentences. The positional order of sentences is preserved. A single GPU with a batch size of 16 runs our training. The decoding phase use a beam size of 4 for beam search.

All except two of our experiments have 128-dimensional hidden layers and 56-dimensional word embeddings. The other two experiments, which reflect our hyperparameter search on the pointer-generator model with TextRank preprocessing, have 256-dimensional hidden layers and 128-dimensional word embeddings. We train our models for approximately 180,000 iterations each (7 epochs, or approximately 2 days in real time), except for the baseline model, which we train for 5 epochs, or approximately 1.5 days in real time. To obtain our coverage models, we stop the pointer-generator network models after 6 epochs in training, add the coverage mechanism with a coverage weight of 1, and train for one more epoch. Finally, we implement early stopping by storing the loss on the validation set.

4.3 Evaluation Metrics

To evaluate the model performance for each experiment, we generate the summaries on the test set and calculated the standard ROUGE scores based on the reference summaries [14]. The F_1 scores for ROUGE-1, ROUGE-2, and ROUGE-l are reported. Respectively, these ROUGE scores measure word-overlap, bigram-overlap, and longest common sequence between the generated and reference summary.

5 Results

5.1 Quantitative Analysis

The results of our experiments are summarized in Table 1. The performance of our baseline is particularly poor, characterized by repetitions, incoherent phrasing, and unknown tokens. The weakness of the baseline relative to our other experiments may be explained by: a) the strength of pointer-generator in copying important out-of-vocabulary words, b) the coverage mechanism that mitigates repetition, and c) shorter training time for the baseline (5 epochs vs. 7 epochs for the other experiments). We note that the training time in all our experiments is shorter than that in See, Liu, and Manning (2017) (33 epochs).

The coverage mechanism appears to enhance the performance of the pointer-generator model. The improvement due to coverage is very noticeable when full input articles are used (i.e., no TextRank preprocessing): experiment 2 has a ROUGE-1 score of 38.84, compared to experiment 1's ROUGE-1 score of 36.99. Given that the ROUGE scores in these cases have a 95% confidence interval of

	ROUGE		
	1	**2**	**L**
seq2seq + attn, 128 hidden dim, 64 embed dim (baseline)	14.12	3.10	11.35
ptr-gen, 128 hidden dim, 64 embed dim (exp. 1)	36.99	23.49	31.53
ptr-gen + coverage, 128 hidden dim, 64 embed dim (exp. 2)	**38.84**	**24.63**	**32.67**
ptr-gen + TextRank, 128 hidden dim, 64 embed dim (exp. 3)	37.37	23.42	31.70
ptr-gen + coverage + TextRank, 128 hidden dim, 64 embed dim (exp. 4)	37.05	23.34	31.70
ptr-gen + TextRank, 256 hidden dim, 128 embed dim (exp. 5)	37.46	23.52	31.86
ptr-gen + coverage + TextRank, 256 hidden dim, 128 embed dim (exp. 6)	**37.79**	**23.84**	**32.20**
extractive model (Durett *et al.*)	**42.2**	**25.9**	

Table 1: Quantitative results on *The New York Times Annotated Corpus* test dataset.

at most $\pm$ 0.30, this difference is significant (see Figure 2). When using coverage on TextRank-preprocessed input, we observe a small, but insignificant improvement due to coverage between experiments 5 and 6, but interestingly, coverage lowers the ROUGE-1 and ROUGE-2 scores of experiment 4 (relative to experiment 3). A potential explanation is that summarization on full article input is particularly prone to repetition, in which case coverage can provide significant benefit. On the other hand, since the tokens in TextRank-preprocessed input are already important to begin with, coverage may excessively prevent the network from returning to these tokens even when it makes sense to do so, resulting in weaker summaries.

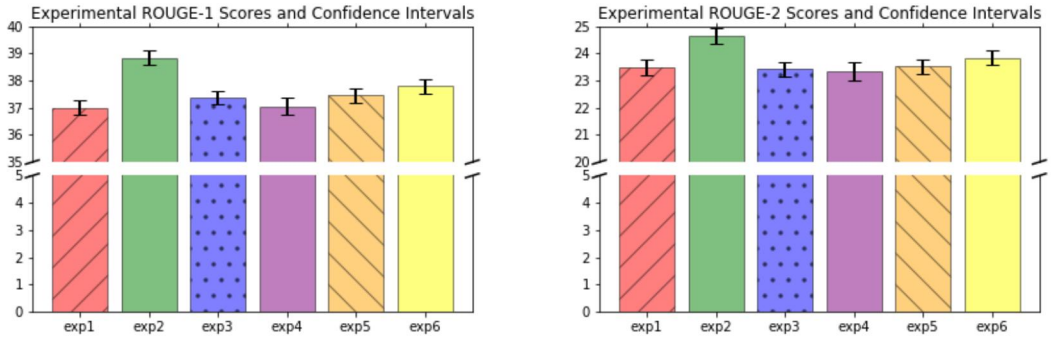

Figure 2: ROUGE-1 and ROUGE-2 F_1 scores with confidence intervals for our experiments.

The results from experiments 1, 3, and 5 suggest that when coverage is not used, using TextRank preprocessing leads to stronger performance than using the full article input. However, when the coverage mechanism is used, its benefit to the model trained on full article input is large enough to make experiment 2 the best model across our experiments. We also find that hyperparameter search through enlarging the hidden dimension and embedding dimension slightly increases the performance of the pointer-generator + coverage model with TextRank preprocessing (see experiments 5 and 6 vs. experiments 3 and 4). However, this improvement is small.

For additional context, we provide the results of the extractive model by Durrett, Berg-Kirkpatrick, and Klein (2016), with the caveat that Durrett, Berg-Kirkpatrick, and Klein uses a smaller subset of the New York Times data. The extractive model outperforms all of our experiments, illustrating the inherent advantage of extractive summarization.

5.2 Qualitative Analysis

To further examine the behaviors of our models, we also examine the individual output for several examples and compare them across our experiments. Table 2 considers one such example (additional examples are included in the supplementary materials). The baseline model produces incoherent sentences with a lot of repetition, resulting in a summary that bears little connection to the source

Source document: If the presidential candidates thought they were finished with bitter and parochial historical debates like whether the Confederate flag should fly over South Carolina's state capitol, they were mistaken. Tiny Rhode Island, which will hold its primary on March 7, could offer another uncomfortable controversy. At a breakfast last month in honor of the Rev. Dr. Martin Luther King Jr. , **the Rev. Virgil Wood**, a pastor in Providence and President of the Ministers' Alliance, a group of African-American clergy in the state, **announced a campaign to remove the word "plantations" from Rhode Island's official name. Most people probably don't realize that its full name is State of Rhode Island and Providence Plantations.** Resentment over the name has simmered in the state's small black population for years to little effect. But the conflict in South Carolina revived the issue, igniting a new debate over the state's slave-era history. Within a week of Mr. Wood's remarks, an editorial in the Providence Journal opposed changing the state's name, arguing that the word plantations had little to do with slavery at the time it was attached to the state name [...]
Ground truth: laurence zuckerman, in think tank column, notes that rev virgil wood has announced campaign to remove word 'plantation' from rhode island's official name; holds that most people probably do not realize that its full name is state of rhode island and providence plantations.
seq2seq + attn, 128 hidden dim, 64 embed dim (baseline): bad said new because **him's new** because **him bank** deeply confirm deputy, **which is first new because him's me bank, street that it more not be likely to make it suggest to make it suggest.**
ptr-gen + coverage, 128 hidden dim, 64 embed dim (exp. 2): tiny rhode island, which will hold its primary on march 7, **will remove word 'plantations'** from rhode island's official name; most people probably don't realize that its full name is state of rhode island and providence plantations.
ptr-gen + coverage + TextRank, 256 hidden dim, 128 embed dim (exp. 6): virgil wood, pastor in providence, **conn**, and president of ministers' alliance, group of african-american clergy in state, announce campaign to remove word 'plantations' from rhode island's official name.

Table 2: Example from *The New York Times Annotated Corpus* test dataset showing the output of our selected experiments. **Red** denotes factual errors, incoherent phrasing, and repetition. **Blue** denotes phrases containing key information in the source.

document. Interestingly, the pointer-generator with coverage model (experiment 2), while covering similar topics to the ground truth, makes a key factual error. On the other hand, the pointer-generator with coverage model on TextRank-preprocessed input (experiment 6) does not make this error, likely because it is easier to identify this key information with a smaller input sequence. However, it makes a smaller factual error and misses a detail in the ground truth (although it is arguable that this detail is not important).

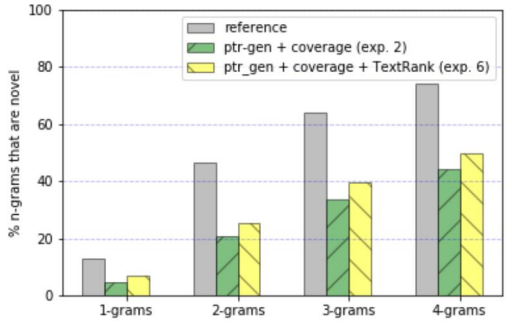

Figure 3: Our experiments appear to produce a substantial number of novel n-grams.

One important issue that See, Liu, and Manning (2017) face in their implementation of the pointer-generator model with coverage is its apparent "extractiveness". In other words, the pointer-generator mechanism encourages the model to copy a substantial number of words directly from the source input onto the generated summary, which makes the model appear less abstractive in nature. In Fig-

ure 3, we show the percentages of novel n-grams in the summaries (i.e., n-grams that appear in the summary but not in the input article). Surprisingly, both the pointer-generator model with coverage on complete articles and on the TextRank-preprocessed articles successfully produce a substantial number of novel n-grams. Furthermore, using TextRank-preprocessed appears to increase the number of novel n-grams. While it is unclear why our experiments result in more novel phrases than See, Liu, and Manning's results, we make two conjectures: a) their preprocessing step of keeping only the first 400 words might have forced the model to copy more than it should, and b) the nature of our dataset might be different from that of their *CNN/Daily Mail* dataset.

6 Conclusion

We have proposed a neural framework that incorporates the pointer-generator with coverage model and TextRank preprocessing. Our framework produces comparable results to a pointer-generator with coverage model on full article input, with potential advantages in avoiding factual errors and generating novel n-grams. This is encouraging, given that the training time in our experiments is limited. Finally, we believe that further investigation into our model's n-gram generating behavior may yield useful insights. If our findings are replicable, they may suggest that a combination of extractive preprocessing and abstractive summarization can produce summaries that are more similar to human writing than either extractive or abstractive method alone.

References

[1] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. A brief survey of text mining: classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*, 2017.

[2] Greg Durrett, Taylor Berg-Kirkpatrick, and Dan Klein. Learning-based single-document summarization with compression and anaphoricity constraints. *arXiv:1603.08887 [cs]*, March 29, 2016. arXiv: 1603.08887. URL: http://arxiv.org/abs/1603.08887 (visited on 02/08/2018).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[5] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[6] Rada Mihalcea and Paul Tarau. Textrank: bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[7] Evan Sandhaus. New york times corpus: corpus overview. *LDC catalogue entry LDC2008T19*, 2008.

[8] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[9] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[10] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[11] Junyi Jessy Li, Kapil Thadani, and Amanda Stent. The role of discourse units in near-extractive summarization. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, 2016.

[12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[13] David Adamor. Textrank. https://github.com/davidadamojr/TextRank, 2017.

[14] Chin-Yew Lin. Rouge: a package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.