Abstractive Text Summarization in News Headline Generation

Zicong Fan
11205168
zfan@alumni.ubc.ca

Si Yi (Cathy) Meng
32939118
mengxixi@cs.ubc.ca

Zixuan Yin
11687143
krystal_yzx@naver.com

Abstract

As the amount of textual data grows exponentially in the last few decades, effective information retrieval becomes central in high-level decision making. In this work, we provide an abstractive text summarization model under the attentional encoder-decoder framework. We employ one of the most cutting-edge language model – ELMo, along with the adaptive softmax technique for GPU memory efficiency. We evaluate our model performance both quantitatively and qualitatively on the Annotated English Gigaword dataset. The qualitative results demonstrate our model's adequacy in the headline generation task. A quantitative comparison of the effectiveness using different word embeddings under this framework is also provided.

1 Introduction

As the amount of textual data in the world grows in an ever-increasing rate, the ability to quickly index and search from documents becomes crucial. To provide a readable and comprehensive summary capturing all the relevant pieces often requires the knowledge of human experts, which is prohibitively costly as the amount of work scales. Furthermore, given a relatively neutral corpus to learn from, machine-generated summaries should be less prone to human-bias (either intentional or not). These issues motivate the use of modern technology to train computers to accomplish this task.

In this work, we explore a closely related task — news headline generation — as a reduced form of abstractive text summarization. We use deep learning architectures such as sequence-to-sequence recurrent neural network models with attention as the basic framework. The ELMo contextual word representation (Peters et al., 2018) was used as the source of our input word embeddings. To reduce GPU memory usage with a large vocabulary, we adopt the adaptive softmax technique (Grave et al., 2016) instead of the regular softmax.

This report is organized as follows. Section 2 provides a discussion of the related research to our task. Section 3 describes the Annotated English Gigaword dataset (Graff et al., 2003) that we use to train and evaluate our model. Section 4 provides a detailed description of our ELMo embedded global attentional model. Section 5 and 6 provide the evaluation setup and various aspects of the performance analysis. Lastly, section 7 concludes our work and contributions.

2 Related Work

2.1 Automatic Text Summarization

The task of automatic text summarization has been extensively studied (Gambhir and Gupta, 2017). The two general approaches are the extractive method and the abstractive method (Gupta and Lehal, 2010). Extractive methods explicitly select parts of the original document (verbatim words, phrases or even complete sentences) and combine them into a summary. On the other hand, abstractive methods focus on comprehending the original document and rephrasing the main ideas in a shorter form, often with words and phrases that do not occur in the original text. Early attempts in text summarization focused on extractive methods utilizing the lexical occurrence probabilities to extract parts of the input document (Mathis et al., 1973). Extractive methods are traditionally more straightforward to interpret and implement; however, as deep neural networks shed light on various natural language processing (NLP) problems,

abstractive methods become progressively feasible (Nallapati et al., 2016a,b; Rush et al., 2015). For instance, (Rush et al., 2015) proposed a local attention-based model by incorporating a neural language model, a contextual input encoder and a beam-search decoder to explore such tasks. Similarly, (Nallapati et al., 2016b) used an attentional encoder-decoder framework with additional rich linguistic features such as part-of-speech tags and named-entity recognition tags. Both works were evaluated quantitatively in a news headline generation setting on the Annotated English Gigaword dataset (Graff et al., 2003) with the ROUGE metrics (Lin, 2004), which will be discussed in more detail in section 6.

2.2 Recurrent Neural Networks

Neural network models have been widely adopted to solve various NLP problems. In particular, recurrent neural networks (RNNs) has gained popularity for its ability to handle arbitrary-length sequences using a hidden state vector that autoregressively encodes sequential information (Goodfellow et al., 2016). Since the vanilla RNN performs poorly with long sequences due to vanishing gradient and exploding gradient problems, the Long Short-Term Memory (LSTM) units (Hochreiter and Schmidhuber, 1997) and the Gated Recurrent Units (GRU) (Cho et al., 2014a) have been proposed. Both architectures have shown their successes in various tasks including supervised sequence labelling (Graves, 2012), image captioning (Vinyals et al., 2015) and speech recognition (Graves et al., 2013) with competitive performance (Yin et al., 2017).

2.3 Word Embedding

Text summarization systems (and other NLP systems) often rely on a competent underlying language model to accurately represent meanings in a vector form. Generally, there are two kinds of word representations: context-independent and context-dependent. The context-independent representations learn a *single* meaning for each word by either predicting a word based on its surrounding or by performing a matrix decomposition on the co-occurrence matrix. Canonical examples of context-independent word representations are `word2vec` (Mikolov et al., 2013), `FastText` (Bojanowski et al., 2017), and `GloVe` (Pennington et al., 2014). However, due to the polysemic nature of most natural languages, these representations are not flexible enough to capture word meanings in different contexts. To address this problem, novel insights in context-dependent representations start to emerge. Particularly, Embeddings from Language Models (`ELMo`) was introduced in (Peters et al., 2018) by combining all layers of a pre-trained deep bi-directional language model. The language model `ELMo` generates unique word representation when the word appears in different sentences instead of finalizing word embeddings into a fixed dictionary. Bidirectional Encoder Representations from Transformers (`BERT`) is another recent work introduced by (Devlin et al., 2018) where a deep bidirectional Transformer architecture was proposed, which has shown compelling results on many NLP benchmark tasks.

2.4 Machine Translation

Abstractive text summarization is closely related to machine translation, where variations of the encoder-decoder architecture (also known as the sequence-to-sequence model) are commonly adopted. The encoder — often an RNN such as LSTM — iterates forward in time over the input sequences while updating its hidden states to encode sequential information into a fixed-length vector. The decoder — another RNN — generates the output sequence by maximizing the probability of the next word found in the vocabulary space given the encoded vector (Sutskever et al., 2014; Cho et al., 2014b). This approach often works well for short sequences, but the performance deteriorates for longer input sequences, which is often the case in text summarization. To alleviate the problem, (Bahdanau et al., 2014) introduced the attention mechanism, which assigns weights over the entire input sequence to guide the decoder's focus on parts of the input sequence at a particular time step. However, computational bottleneck arises in the attention weight calculation for long sequences. A local attention approach was proposed by (Luong et al., 2015) to speed up the calculation by only computing attention over a window on the input sequence chosen by the current decoder hidden state. Finally, (Vaswani et al., 2017) proposed a highly parallelizable and efficient Transformer model to replace convolutional and recurrent components with only attention mechanism, and it showed significant improvement over the computational requirement while maintaining the

state-of-the-art performance. Many of these works were originally introduced in developing methods for machine translation tasks; however, one can think of text summarization being analogous to machine translation in a sense that we are trying to convert a signal to its condensed form. Thus, it is reasonable to apply similar techniques under the text summarization setting, which we will explore in this work.

3 Dataset

For training and evaluation, we use the Annotated English Gigaword dataset (Graff et al., 2003), which contains nearly 10 million documents from 7 news sources. Due to computational constraints, instead of using the full document as the input source, we extract only the first paragraph of each document as the input sequence, and the corresponding news headline is treated as our target summary. This formulation of the text summarization task is commonly adopted in the literature (Rush et al., 2015). We refer the input sequence and its target summary the *input-output pair*. For comparison purposes, we use the same data split [1] as in (Rush et al., 2015) to form the training, validation and test set (roughly 90-5-5). We tokenize the sequences, remove most punctuations, and convert all tokens to lowercase in the input-output pairs. Numeric tokens were left unchanged since they often contain important information for the model to generalize from. Due to limited computational resources, the training set is downsampled by randomly dropping 40% of the input-output pairs to reduce the vocabulary size as well as the number of embeddings that we need to store. During data cleaning, we remove pairs where the input text has less than 5 tokens, otherwise there are barely anything to summarize from. We also remove pairs with headline length greater than 30 to enforce summary conciseness. Furthermore, we replace low-frequency tokens (less than 4 occurences) with the "UNK" token. To prevent learning from uninformative examples, we remove pairs whose headlines have less than 3 known tokens.

Finally, we arrive at about 3.8 million training pairs with a vocabulary size of about $214,000$. The average sentence length is 9 tokens for the headlines and 30 tokens for the body text. The development set contains about $346,000$ pairs which we use for hyperparameter tuning. Following the same strategy as (Rush et al., 2015), the test set was generated by randomly sampling $2,000$ examples from the test split. The sentence length distribution and the word frequency distribution of the test set are similar to that of the training set.

[1] https://github.com/facebookarchive/NAMAS

4 Methods

4.1 Global Attention Framework

In an RNN encoder-decoder framework without attention mechanism, the RNN encoder reads the input sentence word by word and recursively updates its hidden state according to the previously hidden state and the current token's embedding. As a result, the final hidden state vector is used as a summarized representation of the input sentence. The decoder then takes the final hidden state vector as input and recursively chooses the token with highest probability from the vocabulary distribution until an end-of-sentence (EOS) token is obtained. This approach starts to deteriorate for longer input sentences because the summarization capacity of the encoder's final hidden vector is limited. Therefore, we adopted the global attentional encoder-decoder framework introduced by (Luong et al., 2015) because the news headline summarization task often has long input sentences.

$$score(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^T \bar{\boldsymbol{h}}_s & dot \\ \boldsymbol{h}_t^T \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & general \\ \boldsymbol{v}_a^T tanh(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]) & concat \end{cases} \quad (1)$$

In a nutshell, as shown in Fig. 1, (Luong et al., 2015) augmented the encoder RNN with a global attention mechanism to provide better summarization of the input sequence. Suppose that the decoder would like to sample a token at discrete time t. It first computes the current decoder hidden state $\boldsymbol{h}_t$ based on the previous decoder hidden state $\boldsymbol{h}_{t-1}$ and the previously chosen token. Then it scores each encoder hidden state $\bar{\boldsymbol{h}}_s$ for the input token s using a score function. Some score functions are provided in Eq. 1. In particular, we adopt the dot product score function for its computational efficiency to compute attention weights. To decode each token from its decoder hidden state, this scoring procedure allows us to decide how much attention each input token needs. Using the encoder input token hidden states $\bar{\boldsymbol{h}}_s$ and the attention weights $\boldsymbol{a}_t$, the context vector $\boldsymbol{c}_t$ is obtained by a sum of $\bar{\boldsymbol{h}}_s$ weighted by $\boldsymbol{a}_t$. As

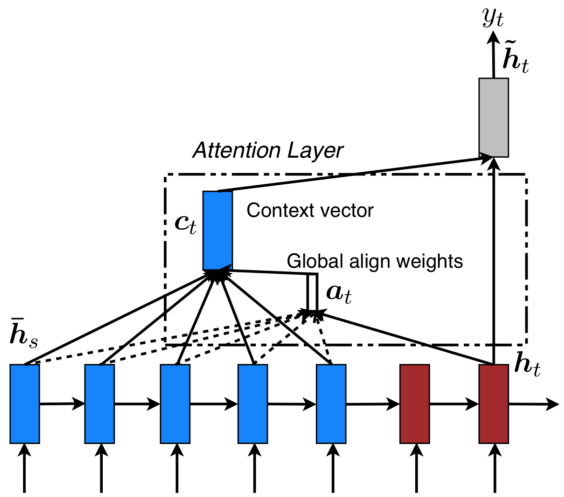

Figure 1: **Global attentional model**. The encoder computes an attention augmented hidden vector $\tilde{h}_t$ by the context vector c_t and the current decoder vector h_t. The context vector c_t is computed by scoring each encoder hidden state vector $\bar{h}_s$ by the current decoder hidden state h_t and taking a weighted average of the input hidden states using the attention weights a_t from the normalized scores. The figure is from the original paper (Luong et al., 2015).

a result, the context vector c_t has encoded attentional information of the input sentence. Finally, we concatenate the context vector c_t and the current decoder hidden state vector h_t and map to a vector $\tilde{h}_t$ such that $\tilde{h}_t, h_t, c_t$ share the same dimension.

4.2 ELMo Embedding

For model input, we use a pre-trained ELMo language model (Peters et al., 2018) to generate 256-dimension embeddings for our vocabulary. As discussed earlier, ELMo's main advantage is its ability to generate contextualized word embeddings. In other words, it is able to generate a different embedding for the same word that appears in different contexts. This is crucial as it allows our text summarization model to disambiguate between multiple meanings of a given word, but the implication is that we can no longer use a fixed dictionary as a pre-trained embeddings input. To address this issue, we run every sentence in our training set through the pre-trained ELMo model, and save the average of all the embeddings we receive for a given word. One may argue that by taking the average – thereby having the same embedding for a given word regardless of its context – defeats the purpose of employing the contextualized representation. However, we believe that the average essentially incorporates all the meanings of a given word into the same embedding, and therefore should do no harm in the downstream task. In Table 1, we compare the results of the same architecture using GloVe embeddings, FastText embeddings and averaged ELMo embeddings.

4.3 Adaptive Softmax

One of the main challenges in this work is handling the size of the dataset. Even after downsampling to retain only 60% of the training data and replacing low frequency words, our vocabulary is still too large to fit into the GPU memory. The bottleneck is that when the decoder predicts the next word, it relies on the last fully connected layer to map from hidden space to the output space in order to perform softmax, which is the same size as our vocabulary. The number of model parameters then scales linearly with the size of the vocabulary due to the fully connected layer. To circumvent this issue, we use adaptive softmax which is an efficient approximation to the regular softmax function (Grave et al., 2016). This strategy is the most effective when the output space is large and highly unbalanced, which is consistent with our vocabulary. Adaptive softmax works by partitioning the output space into clusters depending on the label frequencies, and thus approximating the regular softmax in a hierarchical fashion. Since the word frequencies roughly follows the Zipf's law (Wilson, 1949) where a large proportion of the articles is covered by only a small number of words in our vocabulary, it would make sense to prioritize the prediction of higher frequency words than the less frequent ones.

5 Experiments

The process of gathering ELMo embeddings is very time-consuming (takes about 5 hours) because it involves running each sentence in our corpus through ELMo. Therefore, we cache the gathered embeddings and reload them from disk for every subsequent experiment to amortize the cost.

There are some challenges in integrating ELMo into our encoder-decoder framework. The natural way to use ELMo for embeddings is to generate them on the fly as the training subroutine queries for the input embeddings. However, the results are unsatisfactory as the decoder also relies on the pre-

viously generated word to generate the next word. One option is to have ELMo return an embedding for every individual word, but there will be no context and therefore the embedding would not be very meaningful. Alternatively, if we run the already generated sequence through ELMo, at the beginning of training this sequence would likely be non-sensical (i.e. a sequence like "the the the"), rendering ELMo ineffective and therefore prohibits learning of the decoder as the error accumulates. Therefore, we take the averaging approach prior to training as discussed earlier.

The model is trained with a 2-layer Gated Recurrent Unit (GRU) on both the encoder and decoder with 200 hidden units for every hidden layer. The adaptive softmax is used, and it requires a mapping from the decoder's output to a 1024-dimensional hidden layer. This dimensionality is manually tuned using our development set. To avoid the issue of exploding gradient, we perform gradient clipping by monitoring the gradient's L2-norm during training. The gradient is capped to have a maximum L2-norm of 50.0.

Since our model has a large number of weights, regularization techniques are applied. Specifically, we set a weight decay of 0.0001 and used dropout with probability 0.25 to avoid overfitting. We use an Adam optimizer (Kingma and Ba, 2014) for each of the encoder and decoder. Since the decoder has a greater impact on the summary generated, we set the decoder's learning rate to be 5 times greater than that of the encoder throughout training. The initial learning rate of the encoder is set to 0.001, which is lowered by a quarter for a total of 4 times during the course of training. The learning rate decay schedule is determined by examining the convergence plot in Figure 2. For the output space partitions, we use cutoffs at $1,000$ and $20,000$ based on the frequency distribution of our vocabulary. This implies that using a vocabulary sorted by word frequencies, the first cluster contains the most frequent 1000 words, and similarly for the rest of the clusters.

Our model is trained on an NVIDIA GTX 1060 GPU using shuffled minibatches of size 32. Since we have a large training set, we only ran our model for 2 epochs and the loss converges in the end (see Fig. 2). Excluding any preprocessing time, training one model takes about 10 hours, which is the main bottleneck of our study.

We also compare the use of two other word embeddings (300-dimension GloVe embeddings, and 300-dimension FastText embeddings) against ELMo using the same architecture and training procedure. The results are shown in Table 1.

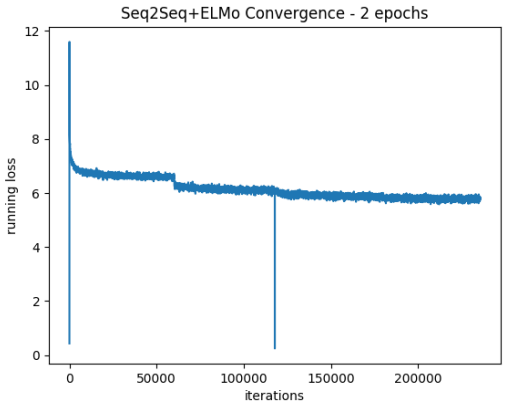

Figure 2: **Learning curve**. The first dip is around $60,000$ iterations, which is precisely the first time we reduce our learning rate. The two vertical lines are due to numerical instabilities at the beginning of an epoch while calculating the running loss. As one can see, the objective function plateaus after 2 epochs which suggests the end of training.

6 Results

Models	RG-1	RG-2	RG-L
Previous Work			
Re^3 Sum	37.04	19.03	34.46
words-lvt5k-1sent	36.40	17.70	33.71
ABS+	31.00	12.56	28.34
Our Models			
Seq2Seq + ELMo	24.54	8.44	22.91
Seq2Seq + Glove	22.86	6.80	20.41
Seq2Seq + fastText	19.67	5.38	18.41

Table 1: Gigaword test set summarization results with full-length F1 ROUGE scores (as previous work). The **Re^3 Sum**, **words-lvt5k-1sent**, and **ABS+** models are from (Cao et al., 2018; Nallapati et al., 2016b; Rush et al., 2015), respectively.

We use the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metrics (Lin, 2004) to evaluate our text summarization system. To be consistent with the literature, our models are evaluated with ROUGE-1 (overlap of 1-grams), ROUGE-2 (overlap of bigrams) and ROUGE-L (longest common subsequence) using full-length

F1-scores. Although ROUGE is the most widely adopted metric in text summarization, it is not always effective when measuring the success of such systems. As noted by (Cohan and Goharian, 2016), ROUGE calculates the scores purely based on the lexical overlap between the documents. Therefore, it can be misleading when the system outputs a paraphrase of the ground-truth summary but using different words, or even just a different ordering of the words. As an extreme example, the system and gold pair "fatal accident on highway 1" v.s. "two dead in motor vehicle collision in b.c. " describe the same event but has a ROUGE score of 0 due to the lack of n-gram overlaps. In particular, ROUGE-2 is always the lowest of the three metrics due to the different ordering of the system generated sentences, and the imperfection in the fluency is usually the cause behind the unsatisfactory performance on this finer-grained metric.

Since the ABS+ model (Rush et al., 2015) is one of the earliest work in abstractive text summarization using a sequence-to-sequence model that provides an evaluation on the same dataset splits, we treat this as our baseline. A comparison between our models and models from previous work is shown in Table 1. As one can see, our best performing model (Seq2Seq+ELMo) still has much room for improvement comparing to the baseline. There are several reasons behind the results we obtain. First, due to computational constraints, we were only able to utilize 60% of the training set as mentioned earlier, which significantly reduced the amount of information our model could potentially generalize from. In addition, we allow for only 2 epochs of training for each model since every epoch takes about 5 hours, and we would prefer faster iterations of tuning and changes to better understand our model's behaviour. Moreover, we were ambitious in expecting the model to learn the mappings between numbers in numerals and numbers spelled out (i.e. 11 people v.s. eleven people), since news headlines often use numerals to keep the length within a certain limit, while spelling the numbers in words in the text body for professionality. In most literature numbers are replaced with the # sign; however, since we did not perform such replacement, our ROUGE scores will be impacted whenever the numbers fail to match up.

It is interesting to note that using 200-dimension `GloVe` embeddings yield results are quite close to that of using the averaged `ELMo` embeddings under the same architecture and training procedure. We believe that this is because the `Glove` embeddings were trained on the same Annotated English Gigaword dataset which coincides to what we use for this task. On the other hand, the 300-dimension `FastText` embeddings were trained on a different dataset. Therefore, despite having a larger embedding size, it is not able to perform as well as using the `ELMo` embeddings.

Figure 3: Positive examples from the test set. **I** is the input, **G** is the gold headline, and **S** is system generated by our Seq2Seq+ELMo model.

Example 1:
I: ten people were killed when an explosion ripped through a minibus in the southern russian city of vladikavkaz on thursday the local interior ministry said quoted by interfax amid conflicting reports on the cause
S: 10 killed in blast in southern russia <EOS>
G: urgent 10 dead in southern russia blast report

Example 2:
I: malaysia beat bangladesh 1-0 in a world cup asian zone group one qualifying match in jeddah saudi arabia on monday
S: malaysia beat bangladesh in world cup <EOS>
G: malaysia beats bangladesh 1-0 in world cup qualifier

Example 3:
I: british agriculture minister douglas hogg on wednesday announced proposals to slaughter up to 40,000 cows in an effort to speed the elimination of mad cow disease from british cattle herds
S: british minister to fight against mad cow disease <EOS>
G: london proposes to eu to slaughter up to 40000 cows

For qualitative assessment of our summarization model, we handpick a few example summary pairs (gold and system) along with the input text to analyze. Figure 3 shows examples where our model was able to generate a decent summary. As one can see, despite minor grammatical issues (i.e. beat v.s. beats), our system-generated summaries are able to capture all the relevant information from the input text correctly in a concise manner. It is worth noting that in Example 3, the system-

Figure 4: Negative examples from the test set. **I** is the input, **G** is the gold headline, and **S** is system generated by our Seq2Seq+ELMo model.

Example 1:
I: the 14th budapest international book festival opened in the hungarian capital on thursday with italian author umberto eco as the special guest of the four-day event
S: new international book festival opens in germany <EOS>
G: budapest international book festival opens

Example 2:
I: thai cabinet has authorized the ministry of public health moph to amend a ministerial regulation to effectively ban visibility of cigarettes for sale
S: thai cabinet to be on health <EOS>
G: thai cabinet gives green light to restrict cigarette sales

Example 3:
I: brandie burton and betsy king mastered the wind the rain and a typically unhelpful links course thursday to card the only sub-par rounds and share the lead in the women s british open
S: unk and the unk <EOS>
G: burton king lead as pak struggles in the wind eds ams recasts adds more quotes detail

generated summary has more of a neutral tone, whereas the gold headline written by a human expert can be interpreted as being more biased in this case. We believe that to some readers, this particular human-written headline depicts a slightly negative image for the subject of interest (British government) by presenting only the more graphical details of the input without stating the subject's ultimate goal. However, the definition of neutrality in text summarization is open to debate and is beyond the scope of this report. In Figure 4, we can see that the model's capabilities is still far from the performance of a human expert in terms of overall coherence and content accuracy. In the first example, the system-generated summary clearly predicted a location more frequently seen in the corpus (Germany) in place of the correct location (Budapest). In the second example, although the system seems to have captured the main theme of the input, it misses some of the details and does not read coherently. The last example shows a situation where the input contains too many low frequency words that the model fails to generate anything meaningful other than the "UNK" tokens.

7 Conclusion

In this work, we have demonstrated the ability of sequence-to-sequence model in the task of abstractive text summarization. We conduct extensive research in comparing using different language models in conjunction with techniques to reduce computational requirement. Although our evaluation scores are not competitive with the state-of-the-art, the exploration provides us with deep understanding of the complexity of the task, and inspires us for future work in related area. Improvements to our system can potentially involve employing the large vocabulary trick mentioned in (Jean et al., 2014)'s work, using summary templates from (Cao et al., 2018) and applying pointer-generator networks which have been shown useful in (See et al., 2017).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *ICLR*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *ACL*.

Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. 2018. Retrieve, rerank and rewrite: Soft template based neural summarization. *ACL*.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder-decoder approaches. *SSST*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*.

Arman Cohan and Nazli Goharian. 2016. Revisiting summarization evaluation for scientific articles. *LREC*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Mahak Gambhir and Vishal Gupta. 2017. Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2003. English gigaword. *Linguistic Data Consortium*.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2016. Efficient softmax approximation for gpus. *ICML*.

Alex Graves. 2012. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*. Springer.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. *ICASSP*.

Vishal Gupta and Gurpreet Singh Lehal. 2010. A survey of text summarization extractive techniques. *Journal of emerging technologies in web intelligence*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *ICLR*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. *ACL*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *EMNLP*.

Betty A Mathis, James E Rush, and Carol E Young. 1973. Improvement of automatic abstracts by the use of structural analysis. *Journal of the American Society for Information Science*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Ramesh Nallapati, Bing Xiang, and Bowen Zhou. 2016a. Sequence-to-sequence rnns for text summarization. *CoNLL*.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016b. Abstractive text summarization using sequence-to-sequence rnns and beyond. *CoNLL*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. *EMNLP*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *NAACL*.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *EMNLP*.

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

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *NIPS*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS*.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. *CVPR*.

Logan Wilson. 1949. Human behavior and the principle of least effort.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of CNN and RNN for natural language processing. *arXiv preprint arXiv:1702.01923*.

Appendices

A Yin, Zixuan

- Implement data pre-processing routines.
- Implement ROUGE evaluation routines.
- Acquire the Gigaword training set.
- Attempt different neural network structure for better performance.
- Tune neural network by experimenting different hyperparameters.
- Write up programming routines to produce experimental results.
- First draft on the introduction section.
- First draft on the abstract section.
- First draft on the dataset section.
- Proofread the entire document.

B Fan, Zicong

- Design and implement the initial neural network structure.
- Literature review on related work section.
- Examine the data and report summary statistics from it.
- Attempt different neural network structure for better performance.
- Tune neural network by experimenting different hyperparameters.
- First draft on the related work section.
- Organize citations.
- Proofread the entire document.

C Meng, Si Yi (Cathy)

- Integrate `ELMo`, `GloVe`, `fastText` word embedding into the network.
- Integrate Adaptive Softmax into model.
- Attempt different neural network structure for better performance.
- Tune neural network by experimenting different hyperparameters.
- Write up code documentation on Juypter notebook.
- First draft on the method section.
- First draft on the experiment section.
- First draft on the result section.
- Proofread the entire document.