Headline Writer: Better Abstractive Text Summarization with Attention and Pointer Generator Network

Henglin Wu henglin@seas
Ruilin Zhao rzhao15@seas
Roy Wu wuroy@seas
Chenyuan Li li15@seas

1 Introduction

Sequence-to-sequence (Seq2Seq) neural networks have been proved to be effective in tasks that involve transforming text from one form to another (e.g. machine translation and speech recognition). They can also be used to generate summarization from text input. In this project, we compare summarization results of a non-deep learning method and results of different implementations of the seq2seq network. The deep learning implementations include using Bi-directional long short-term memory (BiLSTM), additive attention, and teacher forcing in the seq2seq network. More advanced approaches using pointer-generator and coverage are also applied to improve the summarization results.

2 Related Work

Our project is motivated by and built upon previous research in machine translation. Sutskever et al. (2014) et al proposed the Seq2Seq structure, in which a LSTM layer is used to encode the input text to a vector of fixed dimensionality and another LSTM layer is used to decode the target sequence from the vector. Expanding on the structure of Seq2Seq, Bahdanau et al. (2014) showed that adding an additive attention extension may help the network search a set of positions where the most relevant information is concentrated and predicts the target word based on the context vectors associated with these source positions. Vaswani et al. (2017) built on the self-attention model and created multi-attention, which allows the model to jointly attend to information from different representation subspaces at different positions. Going back to the early days of recurrent neural networks (RNNs), a method called teacher forcing was used to help RNNs converge faster. When the predictions are unsatisfactory in the beginning and the hidden states would be updated with a sequence of wrong predictions, the errors would accumulate. Teacher forcing was shown to mitigate this problem (Williams and Zipser, 1989). In 2017, See et al. (2017) applied a new structure using a pointer-generator network that can copy words from the source text via pointing and coverage to keep track of the content that has been summarized to avoid repetition. Our project adopts these breakthrough networks and features step by step and shows the improvements on our specific abstractive summarization task.

3 Methods

3.1 Data

We use "All the news" dataset published by Thompson (2019). This dataset contains 204,135 news articles with headlines from 18 different American publications. It is an updated version of tbe dataset posted on Kaggle, containing over 50,000 more articles from a great number of publications.

3.2 Preprocessing

We use TorchText to preprocess our data. We define a field called TEXT for both the news articles and headlines. First, we define the TEXT field to be sequential and padded to a fixed length of 50. The articles and headlines are tokenized using spaCy and padded with a start token "⟨ SOS ⟩" and end token "⟨EOS⟩." The final training dataset contains 170,000 pairs of articles and headlines, while the validation dataset contains 20,000. We then build the vocabulary and create the word embeddings using GloVE with dimension size of 100. Finally, the training and validation dataset are passed into BucketIterator to generate data loaders for PyTorch in batch size of 32 and sorted by the length of articles.

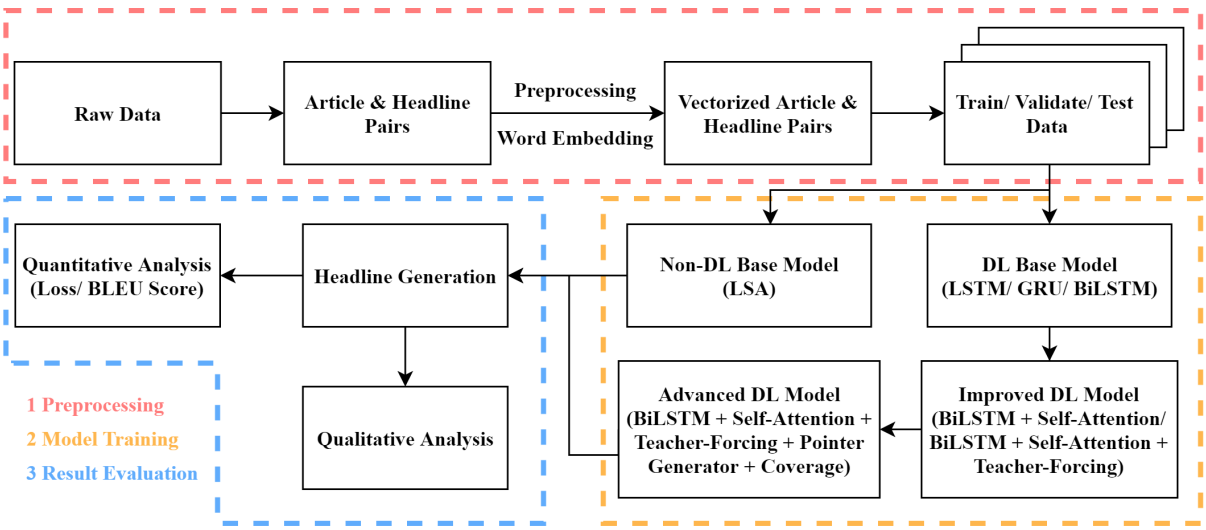

Figure 1: Project Pipeline

Method	Model	Highlights	Architecture	Hyperparameters
Non-DL Baseline	LSA Summarizer	Linear, rule-based	TF-IDF matrix singular value decomposition	None
DL Baseline	Seq2seq	Baseline - LSTM - BiLSTM - GRU	RNN Encoder + RNN Decoder	Default*
Improved	Additive Attention + Teacher Forcing	Attention: creates context vectors that are weighted sums of the encoder hidden states TF: uses ground truth from previous time step as decoder input	RNN Encoder + Attention + RNN Decoder, TF algorithm	Default*, Teacher Forcing Ratio = 0.3
Advanced	Pointer-Generator + Coverage	Pointer Generator: copy words from the source text Coverage: penalize repeating words	Pointer-Generator: adds generation probability Coverage: adds vector to attention, additional loss term	Default*

Table 1: Default*: All neural network models, if not otherwise specified, have learning rate = 0.001, number of neurons in the encoder output hidden layer = 256, word embedding dimension = 100, input text length = 50, output text length = 50, single BiLSTM layer in encoder, single BiLSTM in decoder, loss function = Cross Entropy, optimizer = Adam.

3.3 Model

In this section we describe the methods we tried over the course of this project. First, we use (1) LSA as a non-deep learning baseline. For our deep learning baseline model, we start by implementing (2) sequence-to-sequence (Seq2Seq) model, a basic encoder-decoder model with RNNs, then improve our results by (3) adding attention mechanism and teacher forcing to the Seq2Seq model, and finally introduce (4) pointer-generator model with a coverage mechanism as an advanced deep learning approach.

3.3.1 LSA

Latent semantic analysis (LSA) (Steinberger, 2014) uses singular value decomposition (SVD) on the term frequency-inverse document frequency (TF-IDF) matrix of a text input to extract the highest weighted sentence from the right singular matrix to be the output summary. This extractive summarization method is suitable for a non-deep learning baseline because it is efficient and requires very little computing resources.

3.3.2 Baseline Seq2seq

Our basic Seq2Seq model is based on the Sequence to Sequence Learning with Neural Networks paper (Sutskever et al., 2014). This encoder-decoder model uses Recurrent Neural Network (RNNs) to encode the input text into a single vector, then decodes this vector by a second RNN, which learns to output the summary by generating one token at a time. At each time-step, the encoder RNN takes in the embedding of the current word $e(x_t)$, and the hidden state from the previous time-step h_{t-1}, and then outputs a new hidden state h_t. Once the final word is passed through the encoder, the decoder takes as input the hidden layers created by the encoder. The decoder has a similar structure as the encoder with an additional fully-connected layer after the output from each token to form each word of the generated headline. Each generated word is subsequently fed into the decoder as an input for generating the next word of the headline.

In our experiment, we tested various RNN architectures; these include Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bi-directional LSTM (BiLSTM) units, all which were chosen for their ability to handle sequential data. For our hyperparameters, we use only a single layer that contains 256 hidden units,

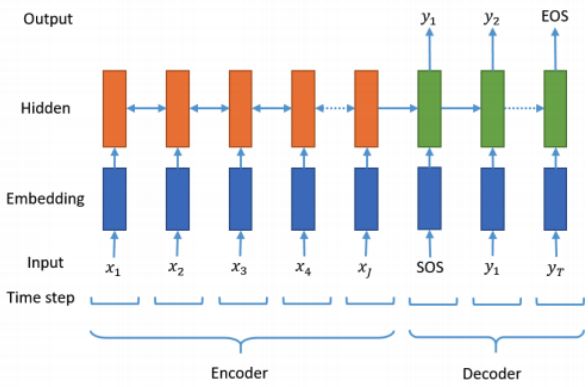

Figure 2: Baseline Seq2seq (Shi et al., 2018)

with no dropouts (Lopyrev (2015)'s observations showed that dropout does not improve the model's performance). We also use Cross Entropy as our loss function and Adam as our optimizer with a learning rate set to 0.001. These will be the default hyperparameters as we go forward.

3.3.3 Attention

In our baseline Seq2Seq model, we pass the context vector to the decoder at every time-step and pass the context vector, embedded input word and hidden state to the linear layer to make a prediction. Although this reduces some information compression, the context vector still needs to contain all of the information about the source sentence. To solve this, we improve our previous model by introducing the additive attention mechanism from the Bahdanau et al. (2014)'s paper, which computes attention weights that allow the network to remember certain aspects of the input better. The attention distribution a^t represents a probability distribution over the source words and is calculated as follows:

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

$$a^t = \text{softmax}(e^t) \quad (2)$$

In the equations above, v, W_h, W_s, $b_{\text{attention}}$ are learnable parameters. The attention distribution is then used to calculate a weighted source vector (context vector) h_t^* such that the weights sum to 1 and represent a weighted average over the last hidden layers after processing all of the input words:

$$h_t^* = \sum_{i=0} a_i^t h_i \quad (3)$$

The context vector can be viewed as a representation of what has been read from the source with a fixed size. This is computed at every

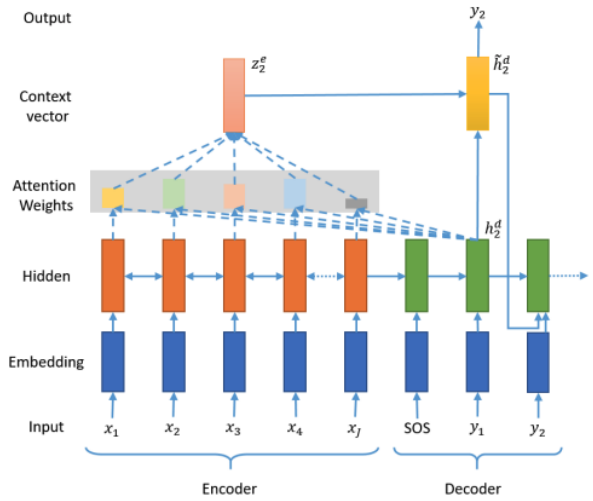

Figure 3: Seq2Seq with Attention (Shi et al., 2018)

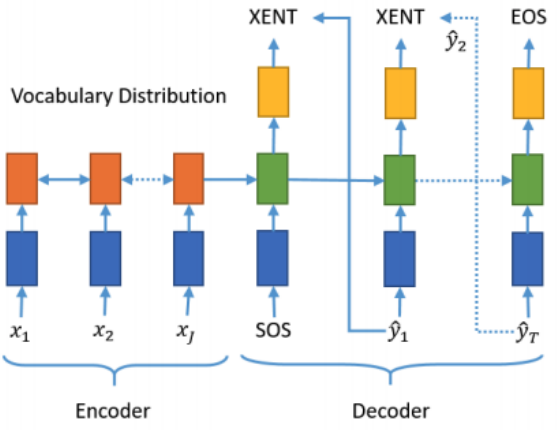

Figure 4: Teacher Forcing Algorithm (Shi et al., 2018)

time-step when decoding, which is concatenated with the decoder state s_t and fed as input into a fully connected layer to output the vocabulary distribution, providing a final distribution from which the decoder makes a prediction. This component allows the model to pay attention to which words from the source are most important when generating a summary.

We incorporate this attention model to our baseline Seq2Seq model with BiLSTM units, which closely follows Nallapati et al. (2016)'s work. As before, we use the same hyperparameters during our training.

3.3.4 Teacher Forcing

In our models described so far, the tokens in the decoder are generated one after another at each timestep, using the prediction of the previous decoder as the input of the subsequent decoder. The problem with this is that if the model predicts an incorrect word early on, subsequent predictions will continue to diverge from the target. This can cause issues such as slow convergence, model instability, and overall poor performance. To combat this, we improve our previous models by implementing Teacher Forcing algorithm (Williams and Zipser, 1989), a strategy that for a chosen probability, uses the ground truth of the next token in the sequence instead of the output of the previous decoder. In this experiment, we add this component to the training process of our previous models, keeping all the model architecture and hyperparameters fixed. We used a teacher forcing ratio = 0.3. During testing, previous ground truth tokens are unknown, so they are replaced with tokens generated by the model itself.

3.3.5 Pointer-Generator and Coverage

Pointer-Generator:
The pointer-generator network combines our baseline Seq2Seq with the attention model with a pointer network that allows it to copy words for the source text. In our implementation, the attention and vocabulary distribution is calculated as before. Additionally, we compute a generation probability $p_{\text{gen}} \in [0, 1]$, representing the probability of generating a word from the vocabulary rather than copying from the source. At timestep t, p_{gen} is calculated as follows:

$$p_{\text{gen}} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{\text{pointer}}) \quad (4)$$

Where h_t^* is the context vector, s_t is the decoder state, x_t is the decoder input, and weights w and scalar b_{pointer} are learnable parameters. p_{gen} is used to weight and combine the vocabulary distribution used for the generator and the attention distribution a^t into a final distribution:

$$P(w) = p_{\text{gen}} P_{\text{vocab}}(w) + (1 - p_{\text{gen}}) \sum_{i:w_i=w} a_i^t \quad (5)$$

The probability of word w being produced is equal to the sum of the probability of generating it from the vocabulary (with probability of p_{gen}) and the probability of pointing it some place in the source (with probability of $1 - p_{\text{gen}}$). When training this model, we use our default hyperparameters.

Coverage:
As explained in the paper, coverage is a technique that uses the attention distribution to keep track

of what's been covered so far and penalize the network for attending to the same words over and over again. We build this on top of our point-generator model with a coverage vector c^t, which is the sum of attention distributions over all previous decoders:

$$c^t = \sum_{t'=0}^{t-1} a^{t'} \tag{6}$$

This is added as an additional input to the attention mechanism described previously:

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + b_{\text{attention}} + w_c c_i^t) \tag{7}$$

We also need to introduce an extra loss term to penalize attending to the same locations

$$\text{covloss}_t = \sum_i \min(a_i^t, c_i^t) \tag{8}$$

In our experiment, we build this coverage technique on top of the pointer-generator network, maintaining the same hyperparameters as before.

3.4 Evaluation

In our analysis, we use the BLEU evaluation metric (Papineni et al., 2002), which calculates the fraction of the words (n-grams) in the machine generated summaries that appeared in the human reference summaries and also compares the length of the generated summary with the length of the true headline. This can be thought of as a measure of precision, and it is a popular metric used in NLP that is reported to have high correlation with human judgement, which is appropriate for our text summarization task.

4 Analysis

4.1 Quantitative Result Comparison

To reduce the computational burden, we first conducted all the experiments on a size of 10,000 article and headline pairs' subset data, and the result is as shown in Table 2. The approaches we have tried can be roughly divided into 4 categories, including Non Deep Learning Base model, Deep Learning Base model, Improved Deep Learning model, and Advanced Deep Learning model.

First and foremost, as for the base model, we implemented a linear LSA summarizer, and 3 base deep learning model using different types of RNN units, including LSTM, GRU, and BiLSTM. As for LSA summarizer, since it is a rule-based model, it does not have a loss. We can see that the BLEU score of LSA is merely 0.005, which means the model output doesn't quite make sense. However, this is consistent with our intuition, because summarization is too complex a task for a linear model to achieve. As for deep learning models, BiLSTM is showing the best results of 0.015 BLEU score compared to the other two. Possible reason might be that it utilized the information not only from the previous context, but also the posterior context. Therefore, based on this, we decided to use BiLSTM as the base RNN unit to carry on with further experiments.

Then, during the improvement part, we tried out 2 kinds of approaches, including teacher forcing and self-attention. It is clear to see that both teacher forcing and self-attention have managed to improve the BLEU score from 0.015 to 0.017, and 0.034 respectively, and together they can dramatically increase the BLEU to 0.045. This result can actually be credited to the advantage of self-attention and teacher forcing, because self-attention enables us to learn a probability distribution that tells us which word in the source text should be paid more attention to, which makes up for the flaw of BiLSTM that tends to forget the long-term dependencies. Besides, teacher forcing is also an effective training method to boost the performance by preventing the error from previous prediction being passed into the next state. Therefore, both of the methods are proven to be effective in our application scenario.

In order to further improve our model performance in certain scenarios, (for example, to avoid generating faulty information), we implemented the pointer generator and coverage structure on the basis of BiLSTM with self-attention as the advanced model (using teacher-forcing to train). Surprisingly, the loss and BLEU score after adding the pointer generator and coverage is not as good as before. We think possible reasons might be that, on the one hand, we changed the loss function by adding coverage loss into it, which just increased component of loss function; on the other hand, although the metrics we used do not suggest improving, after doing qualitative analysis, we find the headline generation is actually improved,

Subset data				
	Model	*Train Loss*	*Validation BLEU*	*Validation Loss*
Base Non-DL Model	LSA Summarizer	/	0.005	/
Base DL Model	LSTM	6.554	0.009	7.005
	GRU	6.474	0.013	6.827
	BiLSTM	6.055	0.015	6.879
Improved DL Model	BiLSTM + TF	5.917	0.017	6.617
	BiLSTM + Att	5.832	0.034	6.764
	BiLSTM + Att + TF	5.502	0.045	6.502
Advanced DL Model	Pointer Generator + Coverage	5.617	0.041	6.660

Table 2: Result comparison on subset data with 10,000 rows. DL for deep learning, Att for attention, TF for teacher forcing.

which will be further explained in the next part.

After the exploration phase, we retrained the models using the full dataset and the corresponding results and training loss curves are presented in Figure 5 and Table 3. From the loss curve we can see that all of them are demonstrating similar optimization velocity, and the trend is in consistency with our findings in the subset data that teacher forcing and self-attention helps improve the performance. As for the result, it is not hard to find out that although the metrics are improved by introducing more data, the relative trend keeps the same, and our best model's (BiLSTM with self-attention, pointer generator, and coverage structure) BLEU score 0.075 actually beats our main source paper with BLEU of 0.010 (Lopyrev, 2015).

4.2 Qualitative Result Comparison

The sample below, one piece of the news from the test dataset, demonstrates our implemented models' performances qualitatively.

Original Text: [ ] a confidential document from tech billionaire peter thiel 's venture capital firm raises doubts about the valuation of palantir technologies , the highly touted silicon valley data analysis company of which thiel is co - founder and chairman . the document also reveals that thiel 's firm 's own investment results are among the strongest in the industry , at least on paper .

Original summary: confidential document shows how peter thiel really feels about palantir
BiLSTM summary (base model): the 's the to the the the the
BLEU Score 0.0204689180374571
Attention + Teacher Forcing summary (improved model): confidential document 's twitter to to
BLEU Score 0.08263103182212594
Pointer Generator + Coverage summary (advanced model): confidential document $ peter thiel to thiel
BLEU Score 0.11255157994168306

Figure 6: Comparison of headline generation results

The original text shows about the first fifty words of an original piece of news, which is used as the model input. The model output summary shows that the base model BiLSTM fails to capture any meaningful information in the original news by outputting some random frequent words that don't make any sense. The improved model with attention mechanism and teacher forcing manages to capture the information of "confidential document". The advanced model does even a better job by successfully capturing another important information of "Peter Thiel".

The sample outputs validate that the attention mechanism improved the model performance by better locating where to look at in the original news when generating summarization. The capture of "Peter Thiel" shows how the pointer generator network takes the benefit of including the probability of outputting words directly from the original news. Words like "Peter Thiel", a person's name, could be very hard to capture by a model that makes summarization by merely generating similar words based on the word embeddings and probability distributions over the vocabulary in the entire dataset, like the one implemented as our base model or improved model. The improved model and the advanced model also achieve higher BLEU scores in this sample.

5 Discussion and Conclusion

According to our work, our findings are as follow:

1. BiLSTM is the base RNN unit that has the best performance in our application scenario,

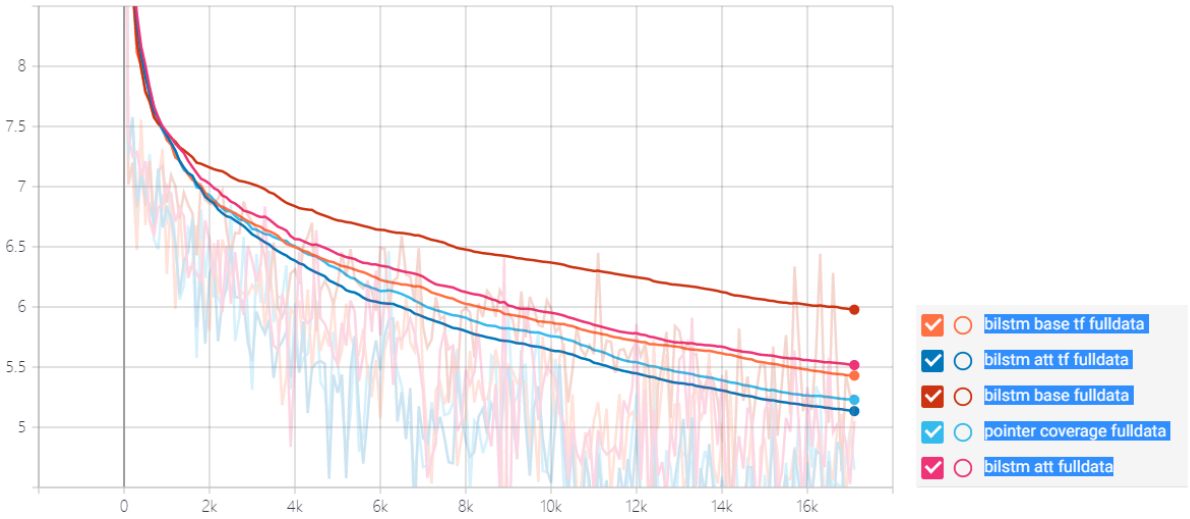

Figure 5: Training loss curve on full dataset

Full data

	Model	*Train Loss*	*Validation BLEU*	*Validation Loss*
Base Non-DL Model	LSA Summarizer	/	0.005	/
Base DL Model	BiLSTM	5.262	0.064	5.739
Improved DL Model	BiLSTM + TF	5.011	0.073	5.195
	BiLSTM + Att	4.778	0.070	5.801
	BiLSTM + Att + TF	4.363	0.079	5.143
Advanced DL Model	Pointer Generator + Coverage	4.449	0.075	5.355

Table 3: Result comparison on full dataset.

(possible reason may be that BiLSTM can utilize the information from both previous and posterior context);

2. Self-attention can help improve generator's performance when dealing with long source article (by learning a probability distribution that tells us which part of the source text should be paid more attention to, which makes up for the flaw of BiLSTM that tends to forget the long-term dependencies)

3. Teacher forcing is an effective training approach that can be used for Seq2Seq model scenario, (to prevent the error from previous prediction being passed into next state);

4. Pointer generator and coverage approach can significantly reduce the occurrence of the model generating faulty information by including the probability of directly outputting words from the source text;

5. Our best model (BiLSTM with self-attention, pointer generator, and coverage structure) beats our main source paper's BLEU score of 0.010 (Lopyrev, 2015) by 0.075.

If given more time, we would better clean the data, which is one of the main barriers that limits our model performance at this time. To further improve our results, we can:

1. Use dynamic word embedding like ELMO, instead of just static embedding like GloVe, because dynamic word embedding can disambiguate multiple sense of the input word, thus better represent the meaning of the word in the context of input, thus very likely to help the model learn the dependencies;

2. Try more state-of-the-art models. As an experiment, we attempted to implement the transformer model from Vaswani et al. (2017)'s paper. A common misconception in NLP is that-Seq2Seq models only use RNN units. Instead, we can use Transformers, which is solely based on attention mechanisms to achieve any task that can be realized by RNN. Though unfortunately we were unsuccessful to implement this on our own due to computational limits, if given more time, we would leverage the open-sourced pre-trained model such as BERT (Devlin et al., 2018) to compare state-of-the-art results and our own.

References

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

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*.

Konstantin Lopyrev. 2015. Generating news headlines with recurrent neural networks. *arXiv preprint arXiv:1512.01712*.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

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

Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K Reddy. 2018. Neural abstractive text summarization with sequence-to-sequence models. *arXiv preprint arXiv:1812.02303*.

Josef Steinberger. 2014. Using latent semantic analysis in text summarization and summary evaluation.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Andrew Thompson. 2019. All the news. Https://components.one/datasets/all-the-news-articles-dataset/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.