Abstractive Text Summarisation Using Transformers and Bi-LSTMS with Attention

Aman Chopra*
Computer Science Department
New York University
New York, NY, USA
ac8511@nyu.edu

Niharika Sinha*
Computer Science Department
New York University
New York, NY, USA
ns4451@nyu.edu

Siddhartha Singh*
Computer Science Department
New York University
New York, NY, USA
ss13793@cs.nyu.edu

***Abstract*—Abstractive text summarization focuses on generating a meaningful summary of the given text as opposed to extractive text summarization which concatenates important sentences from the paragraph. In recent years most of the research being done in Abstractive text Summarization has been based on Recurrent Neural Networks, but RNN based approaches don't do well when long sequences are taken into consideration. In this paper, we have explored the state-of-the-art Transformer Neural Networks and have applied them to the problem of solving abstractive text summarization. We propose a Transformer based neural network and then compare the results obtained by Transformers with a Bi-Directional LSTM with Attention for the dataset we have used.**

Our experimental results show that Transformers perform as well as Bi-LSTMs on the dataset that we have chosen. Transformers however increase the training efficiency and decrease the training times per epoch by a whole lot.

The code of our project is present in [1].

***Index Terms*—Bi-LSTMS; Transformers; Attention; ROUGE-1; ROUGE-2; ROUGE-L; Self Attention; GLove; Inshorts News Dataset**

I. Introduction

Text summarization is a process wherein we extract meaningful context from a text which helps to understand the text in a concise manner. There are two strategies using which we can perform text summarization. First is Extractive Text Summarization. In Extractive Text Summarization, we pick up important parts of the text and concatenate them in the summary. Abstractive Text Summarizers on the other hand reconstruct parts of the text and may also introduce new words to explain the context of parts of the text in a concise manner. Since Abstractive Summarizers are more flexible hence they generate summaries similar to what a human may write after reading the text. In the recent past, a lot of models have been introduced with the aim of mapping input sequences to output sequences called sequence-to-sequence models. These models helps in speech recognition, machine translation, and video captioning. Despite having a lot of similarities with the above tasks, summarization is a very different task. The challenging part of summarization is to compress the original document without incurring loss such that the main context of the paragraph is preserved.

Recently a lot of researches have focused on using Recurrent Neural Networks for the purpose of text summarization. But, RNNs are not able to capture long-range context in a sentence. This restricts them from forming meaningful summaries. To help with long term contexts, researchers started using LSTM RNNs for this task. LSTM stands for Long Short Term Neural Networks. In this type of RNN, the neuron consists of gated structures that help the network to pass only relevant information and forget unnecessary information. This selective filtering mechanism assists the model to remember long-range dependencies. It would make much more sense to run the LSTMs network from both the beginning and end of the text. There are usually dependencies that a network can afford to learn only if it can peek into the upcoming portion of the text. Hence, we run LSTM RNNs from both the starting as well as from the end of the text in the opposite direction. In this paper, we make use of a Bi-directional LSTM with Attention. Even as humans, we don't read the whole text when we have to summarize a sentence. We look at the neighboring sentences to summarize the portion of the current sentence. Similarly, using the Attention mechanism, we try to force the model to pay attention to what's important.

Even though Bi-directional LSTMs do a decent task of summarization; they also have some shortcomings that restrict them from being very useful. They are very

*All the authors contributed equally to this work.

complex models which are slow to train and when the size of text increases beyond a certain range, they too loose contextual information. In the second section of the paper, we talk about Transformers and perform similar experiments as we did for the Bi-directional LSTM network. Transformers are the state-of-the-art solutions for most Natural Language problems. The paper has been laid out in the following manner. Section II, the objective section deals with the primary objectives and the problem description of this paper. Section III of this paper talks about the key studies being already done in this field. Section IV lays down the proposed idea and methodology. Section V discusses the dataset we have used along with the pre-processing. Section VI details both the models we have used along with their architecture. Section VII talks about the Results, and Sections VIII concludes this paper.

II. Objective

In this paper, the authors have taken the task to train a Neural Network model to learn the problem of abstractive text summarization. The aim is to compare the performance and the efficiency of Bi-LSTMs with Attention and Transfomer-based Neural Networks for this problem. This will help in having a deeper understanding of how both the models perform for this problem statement and the dataset.

III. Literature Survey

In this paper [2], the authors have modeled abstractive text summarization using Attentional Encoder-Decoder Recurrent Neural Networks. They have also proposed a new dataset that consists of multi-sentence summaries. They have also brought upon new benchmarks. They applied encoder-decoder RNN was developed for machine translation for the task of abstractive summarization. The model proposed by them outperformed the existing state-of-the-art models. In paper [3], the authors proposed an LSTM-CNN-based ATS framework. The framework can construct new sentences exploring more fine-grained fragments of the sentences. It is unlike abstractive text summarization and works in two stages. It first extracts phrases from source sentences and then generates text summaries using Deep Learning. In paper [4], the authors propose an adversarial process for abstractive text summarization. They used generative and discriminative models. The authors were able to achieve competitive results; i.e ROUGE scores for the present state-of-the-art methods on CNN/Daily mail dataset.

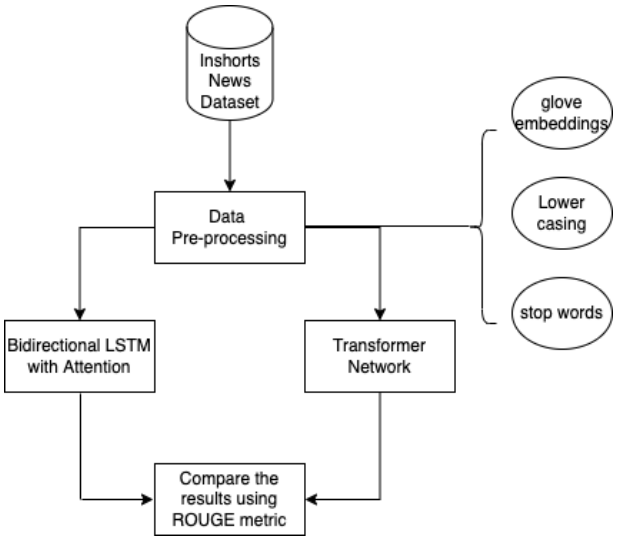

Figure 1. Methodology

In paper [5], the authors used a Convolutional sequence-to-sequence model. They added topic information into the model and used self-critical sequence training for optimization. They carried out experimental evaluation with state-of-the-art methods over the Gigaword dataset and demonstrated novel results. In paper [6], the authors introduced an abstractive summarization model that they named RC-Transformer. They claimed that the model was able to learn long-term dependencies compared to RNN bases encoder-decoder networks. The paper also claimed to have addressed the shortcoming of Transformer networks on insensitivity towards word order. They were able to achieve state-of-the-art results on the Gigaword dataset. The model was also able to outperform the RNN-based models in terms of speed.

IV. Methodology

Figure 1 shows the methodology we have followed doing this project. We have used the Inshorts news dataset. We have done some pre-processing to the dataset like changing the case, removing stop words, using glove embeddings, etc, before feeding the data to our model. Post that, we have tried to generate the headlines using news articles first using the Bidirectional LSTM Model with Attention layer and then using the Transformer Network by experimenting with the architecture and hyperparameters. We have then analyzed the performance of both models using the ROUGE metric. We have calculated the ROUGE-1, ROUGE-2, and the ROUGE-L scores in order to properly compare both the models.

V. Data

A. *Dataset*

The dataset used for this project was the Inshorts news dataset containing 55,104 news-headline pairs with over 100,000 unique words. The dataset was taken from [7]. The dataset was split into train and validation sets in a ratio of 9:1. This resulted in 49,594 training samples and 5,510 validation samples.

B. *Data pre-processing*

The steps involved in data pre-processing were similar for both the models, LSTM and Transformers, to allow a fair comparison of the model results. First, the dataset was shuffled using the sklearn library. The headline and the news article columns were extracted into 2 data frames.

Start and end tags were added to all the news summary data items. Every news summary was now beginning with $< startseq >$ and ending with $< endseq >$. Few sentences were very long, so the maximum length such that 90% of the sentences were shorter than was calculated. A vocabulary was created for all the words appearing a minimum of 3 times. The vocabulary used words both from the headline column and the news summary column.

To tokenize the words, they were converted to lowercase. Several abbreviations commonly found in every day news articles were replaced. Special characters like punctuation marks, numbers, etc. were replaced with an empty string. Spaces were removed from the beginning and the end of all headlines and news articles. The tokenization was done using krs.preprocessing.text.Tokenizer. <UNK> tag was used to replace out of vocabulary words. The vocabulary size was calculated for the encoder and the decoder. To convert words having similar meanings to similar representations, word embeddings were used. Finally, texts_to_seq was used to convert words to integer sequences. Pad sequence was used to pad these integer sequences to the same length.

VI. Models

In this section, we will talk in detail about both the models we have used for Abstractive Text Summarization.

A. *LSTMs*

Figure 2 shows the high-level architecture of Bidirectional LSTM with Attention Layer. The need for a complex architecture for the task of text summarization arose from the requirement of understanding the word

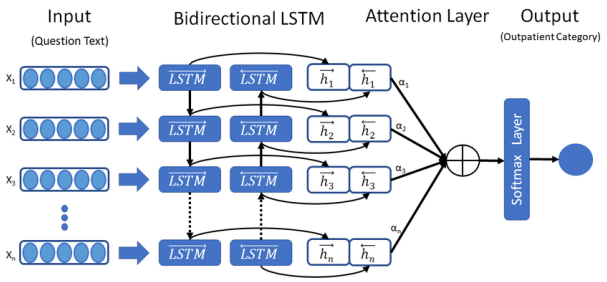

Figure 2. Bi-LSTM with Attention Layer

itself, and not just linking the word to a specific location. Since a normal neural network would fail to identify a named entity in different parts of the text, a shift to the RNN model for this task was seen.

The RNN network was successfully able to identify a named identity in different parts of the text, and it formed the base of seq2seq models. Apart from fulfilling the needed behavior of understanding named entity recognition irrespective of the location, the main advantages of RNN are that it takes time into consideration and takes the output from the previous step. Once the network is trained and executed, the intermediate input from the one-time step is passed on to the subsequent time step till the final output is reached. The end of the text is tagged with a token to indicate when the network should stop running. This type of RNN network is a many-to-many architecture requiring the same lengths for both the input and the output. To overcome this restriction, seq2seq comes in handy. Seq2seq is a special network that can take input of a particular length T_x and generates an output of a different length T_y, implementing an architecture called the encoder-decoder architecture.

There are 2 problems with the RNN i.e., exploding gradient and vanishing gradient. Exploding gradient issue arises in deep networks when during back-propagation, the gradients become too large. This could be solved by setting a gradient threshold, such that when the gradient exceeds that threshold, it is reset. The vanishing gradient issue also occurs when there is a large number of layers, wherein the gradient keeps getting smaller and smaller. As this happens, it becomes more difficult for the network to learn. Using LSTM helps solve these 2 issues.

LSTM, or Long Short Term Memory, is a modification to RNN that is built to remember long sequences of data. It deploys a memory cell within the network that helps store data at an early stage within the sequence so that it can be used at a later stage. This network helps retain

both long-term and short-term memories. LSTM uses the concept of gates in order to make calculations easy.

- **Learn Gate** – Takes an event E_t and the previous Short Term Memory STM_{t-1} as input, keeping only relevant information for prediction. This allows the important information learned from the STM to be applied to the current input.
- **Forget gate** – Takes the previous Long Term Memory LTM_{t-1} as input and decides what information should be retained and what should be discarded. Information that is not useful in the LTM is forgotten.
- **Remember gate** – The previous Short Term Memory STM_{t-1} and the current event E_t are combined to produce an output. This gate works as an updated LTM.
- **Use gate** – Combines the important information from the previous Long Term Memory and the Short Term Memory and creates STM for the next cell. The LTM, STM, and the Event together produce an output for the current event.

Sometimes, it becomes imperative to understand the context of a particular word in a sentence from both ends. This is a very important need in common NLP problems. To understand the significance of a word, sometimes we not only need to see the previous words, but also the words following it. For this reason, the authors implemented a bi-directional version of the LSTM architecture. In such a network, forward propagation is applied twice, once for forwarding cells and once for backward cells. Both forward and backward activations are considered to calculate the output y at a time t. To achieve even higher accuracy, the authors stacked multiple LSTM networks on top of each other. This helps improve accuracy but at the cost of time.

Adding an Attention network to the LSTM model helps the architecture to pay more attention to specific words which in turn helps generate better and more efficient summaries. Intuitively, the model is trying to mimic human behavior of paying attention to the neighboring words at a given instant, and not looking at the entire text to summarize together. This is achieved by constructing a simple neural network. An important structure in the Attention model is the context vector, which goes in between the encoder and the decoder. The context vector represents the amount of attention given to words.

*1) **Architecture**:* The architecture adopted by the authors for this project consists of 3 encoder and 1 decoder LSTM layers. The architecture begins with an encoder embedding layer. The embedding layer turns positive integers into dense vectors of fixed sizes. The output of this is passed onto the first encoded LSTM layer. The return sequences and the return state parameters are kept as True, and the dropout and recurrent dropout both are maintained at 0.4. The return sequence denotes whether to return the last output in the output sequence or the full sequence. The return state denotes whether to return the last state in addition to the output or not. The dropout signifies the fraction of units to be dropped for linear transformation of the inputs. The recurrent dropout signifies the fraction of units to be dropped for the linear transformation of the recurrent state. This layer is followed by 2 more LSTM encoder layers, maintaining the same hyper-parameters.

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 55)	0	
embedding_2 (Embedding)	(None, 55, 110)	2919070	input_3[0][0]
lstm_4 (LSTM)	[(None, 55, 200), (N	248800	embedding_2[0][0]
input_4 (InputLayer)	(None, None)	0	
lstm_5 (LSTM)	[(None, 55, 200), (N	320800	lstm_4[0][0]
embedding_3 (Embedding)	(None, None, 110)	885170	input_4[0][0]
lstm_6 (LSTM)	[(None, 55, 200), (N	320800	lstm_5[0][0]
lstm_7 (LSTM)	[(None, None, 200),	248800	embedding_3[0][0] lstm_6[0][1] lstm_6[0][2]
attention_layer (AttentionLayer	[(None, None, 200),	80200	lstm_6[0][0] lstm_7[0][0]
concat_layer (Concatenate)	(None, None, 400)	0	lstm_7[0][0] attention_layer[0][0]
time_distributed_1 (TimeDistrib	(None, None, 8047)	3226847	concat_layer[0][0]

Total params: 8,250,487
Trainable params: 8,250,487
Non-trainable params: 0

Figure 3. Bi-LSTM Architecture

Next is the decoder embedding layer, which is passed on to the decoder LSTM as input. The decoder LSTM keeps the recurrent dropout rate at 0.2, while the other hyper-parameters have the same values as the encoder layers. Finally, the encoder and the decoder outputs are passed to the Attention layer. The attention input and the decoder LSTM output are combined and passed to a densely connected neural network layer, using softmax as an activation function. Figure 3 shows the architecture of LSTM as adopted by the authors for the purpose of this project.

The LSTM architecture thus created is trained over the training set of the Inshorts news dataset for 100 epochs. However, convergence was observed after about 50-55 epochs, with the validation and training losses hitting 2.73 and 1.83 respectively. Since the LSTM network processes a sentence word by word, one epoch takes relatively longer to complete when compared to the Transformer model. Each epoch is completed in around

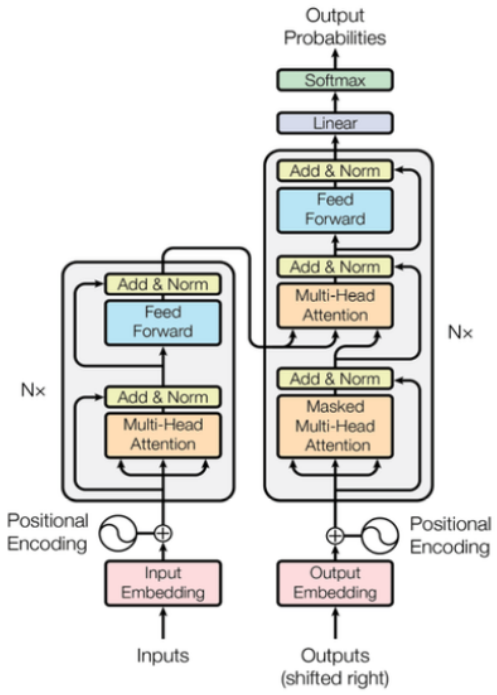

Figure 4. Transformer Neural Network

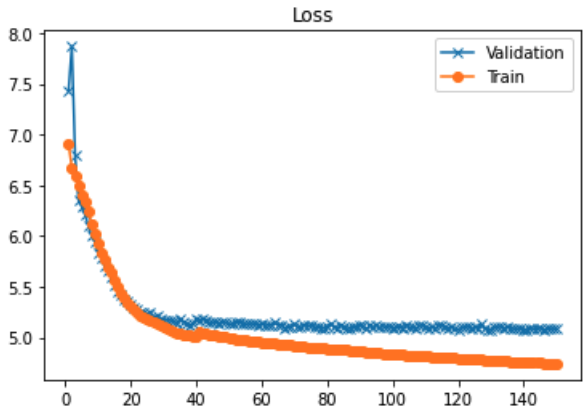

Figure 5. Loss vs Epochs

30-35 minutes using the NVIDIA Tesla P100 GPU. ROUGE scores were calculated once the model was trained and run on the validation set.

B. *Transformers*

RNNs are slow to train. They are so slow that we usually need to use a truncated version of back-propagation to train them. LSTMs can deal with longer sequences but if the sequence length is too long which is usually the case with the Text for Text Summary problems, LSTMs lose contextual information too. They are more complex and hence are slower to train. For LSTM, the input data needs to be passed serially as we need the inputs of the previous state to do any operation on the current state. This limits the usage of GPU as we cannot perform the tasks parallelly. In Transformers, the input sequence can be passed in parallel. The word embeddings of all the words of a sentence are generated simultaneously as there is no concept of time-step. Let's go into the different components of the Transformer Neural Network to better understand how Transformer achieves this.

1) ***Important components:*** **Positional Encoding:** Positional Encoder is a vector that gives context based on the position of a word in a sentence. The input embedding along with the positional Encoder helps in getting the context.

Multi-Head Attention: This is a part of both the Encoder and the Decoder block. Attention tells the network what part of input it should focus on. The Transformers use something called as Self-Attention which is slightly different than the normal Attention layer we used in the Bi-LSTM Networks. [8] details the differences between the two. Self-Attention is attention with respect to oneself. It tells the importance of a word in a sentence. The question we want to answer is, how relevant the i^{th} word is compared to other words of the same sentence. The attention vector may not be too strong. Hence, we determine multiple attention vectors per word and take a weighted average to calculate the final attention. This is why it is called as Multi-Head Attention. Each word has three vectors namely V, K, and Q and the attention Z is calculated as:

$$Z = softmax\left(\frac{Q.K^T}{\sqrt{Dimensionofvector Q, K or V}}\right).V$$

The decoder makes use of the similar components.

2) ***Architecture:*** The maximum length of sentences chosen for training was 60 and the maximum length of the summaries was taken as 12. The drop-out rate was kept at 0.1. The authors experimented with different layers. The best results were achieved with 12 layers. The number of neurons in the feed-forward neural network was kept at 512. The number of heads in the Multi-Head attention was 5. The feed-forward neural network used the ReLu Activation function and made use of *Lambda_L2* as the kernel regularizer. The point-wise feed-forward neural network block is basically a two-layer linear transformation that is used identically

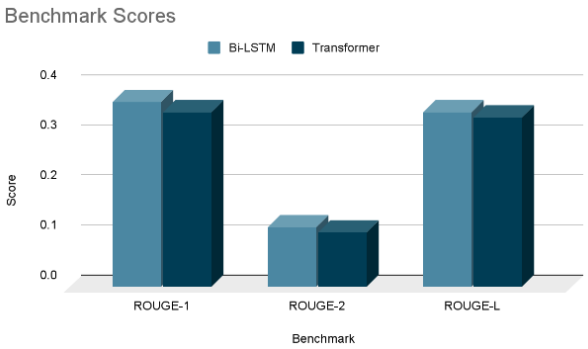

Figure 6. Results

throughout the model. Exponential Decay was used as the optimizer with a learning rate of $1e-3$ and the decay rate was kept at 0.95. The model was trained for 150 epochs. Each epoch is completed in around 3-4 minutes using the NVIDIA Tesla P100 GPU. The decay in Train and Validation loss with the number of epochs is shown in Figure 5.

VII. Results

Recall-Oriented Understudy for Gisting Evaluation is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in Natural Language Processing. ROUGE calculates precision and recall to find how much similar the machine-generated summary is to the actual summary. ROUGE-N measures the number of matching 'n-grams' between our model-generated text and a 'reference'. An n-gram is simply a grouping of tokens/words. A unigram (1-gram) would consist of a single word. A bigram (2-gram) consists of two consecutive words. ROUGE-L calculates the score for the Least Common Subsequences of the generated and reference summary. Figure 6 shows the comparison of ROUGE-1, ROUGE-2, and ROUGE-L for both the models. We can see that all the scores of both the models are similar but the efficiency of Transformers is very high. The time taken per epoch for Transformers was significantly low. We conclude that Transformers are better to use for this problem statement because of the following reasons:

1) At each step we have direct access to all the other steps (self-attention).
2) Sentences are processed as a whole rather than word by word.
3) Transformers are faster than RNN- based models as all the words of a sentence are ingested at once.

VIII. Conclusion and Future Work

In this paper, we have compared the two widely used models for the Abstractive Text Summarisation Problem. First, we experimented with a bidirectional LSTM model with Attention Layer to generate the text summaries. We experimented with different layers and hyperparameters to get a low loss value. On seeing that the time taken per epoch was significantly high, we realized the shortcomings of the LSTMs. We then shifted to the newer Transformer Networks. We understood the architecture of Transformers and applied this network to our dataset and experimented with the layers and hyperparameters. The time taken per epoch was significantly reduced using transformers and the accuracy was at par with the Bi-LSTMs with Attention.

To the best of our knowledge, this is the first time someone is using Transformers on this dataset. In the future, we want to use this model on different datasets as we believe that even the accuracy of Transformers will be more than that of Bi-LSTM Networks. The code of our project is present in [1].

References

[1] Abstractive text summarization. [Online]. Available: https://github.com/Black-Caesaris/Abstractive-Text-Summarization

[2] R. Nallapati, B. Xiang, and B. Zhou, "Sequence-to-sequence rnns for text summarization," *CoRR*, vol. abs/1602.06023, 2016. [Online]. Available: http://arxiv.org/abs/1602.06023

[3] S. Song, H. Huang, and T. Ruan, "Abstractive text summarization using lstm-cnn based deep learning," *Multimedia Tools and Applications*, vol. 78, no. 1, pp. 857–875, Jan 2019. [Online]. Available: https://doi.org/10.1007/s11042-018-5749-3

[4] L. Liu, Y. Lu, M. Yang, Q. Qu, J. Zhu, and H. Li, "Generative adversarial network for abstractive text summarization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12141

[5] L. Wang, J. Yao, Y. Tao, L. Zhong, W. Liu, and Q. Du, "A reinforced topic-aware convolutional sequence-to-sequence model for abstractive text summarization," 2018. [Online]. Available: https://arxiv.org/abs/1805.03616

[6] T. Cai, M. Shen, H. Peng, L. Jiang, and Q. Dai, "Improving transformer with sequential context representations for abstractive text summarization," in *Natural Language Processing and Chinese Computing*, J. Tang, M.-Y. Kan, D. Zhao, S. Li, and H. Zan, Eds. Cham: Springer International Publishing, 2019, pp. 512–524.

[7] Inshorts news data. [Online]. Available: https://www.kaggle.com/datasets/shashichander009/inshorts-news-data

[8] Attention vs self-attention. [Online]. Available: https://datascience.stackexchange.com/questions/49468/whats-the-difference-between-attention-vs-self-attention-what-problems-does-ea