Abstractive Text Summarization using Transformers

Kalyan Kumar Paladugula
University of Illinois at Chicago
kpalad4@uic.edu

Yash Mahajan
University of Illinois at Chicago
ymahaj3@uic.edu

Harikrishna Nagarajan
University of Illinois at Chicago
hnagar6@uic.edu

Abstract

For many years, the field of text summarization has been dominated by attention-based Seq2Seq architectures. After being introduced in 2017, Transformers have become state-of-the-art for the sequence generation tasks, such as Machine Translation and Text Summarization. In this paper, we introduce a model for abstractive text summarization using BART (Bi-Directional Auto-Regressive Transformer). BART is a denoising autoencoder built using a sequence-to-sequence model that has a Bidirectional encoder (like BERT) and an Auto-Regressive decoder (like GPT). We applied this model on the selected CNN summarization dataset and obtained an increase of at least 6% in BERT Scores and 50% in ROUGE scores, suggesting a significant improvement compared to the baseline seq2seq LSTM model.

1 Introduction

Nowadays, a lot of textual data is being generated and it is only going to keep getting bigger. The internet, for example, is flooded with web pages that contain lots of different textual data in the form of news articles, blogs, forums and more. Most of the articles are quite massive and it is difficult to understand the gist of them. So, news organizations started summarizing them manually and provided the summary in the form of highlights above the articles. But over time it became difficult for humans to manually summarize the articles due to the colossal amount of information.

Automatic text summarization got a lot of attention because of this problem. By definition, it's goal is to develop techniques by which a machine can generate summaries that successfully imitate summaries generated by human beings. There are two ways of summarizing text: Extractive and Abstractive summarization. Extractive summarization involves selecting a subset of sentences and phrases from the source text. This is done by ranking the sentences and phrases and choosing the best of them. Abstractive summarization, on the other hand, generates summaries that contain new words and sentences generated based on the source text similar to human summaries. The former is easier as it just has to copy sentences and phrases from the original text and it also ensures some sense of grammar in them. But abstractive summarization has to understand the original text first and creates the words and phrases on its own. Hence, initial work and research were focused mostly on Extractive summarization. [12]

However, with the advent of RNNs, the Seq2Seq models became popular for abstractive summarization. Though these models achieved good ROUGE scores, they are sequential and work in isolation using pre-trained word embeddings and these are fixed and don't change with the context. These models treat the sentences as "bag of words" and don't consider the positions of words in the sentences. They generate a representation of the whole text in the form of a single vector which cannot capture all the aspects of the large sentences. Due to this, the summaries generated by these models don't encapsulate all the important information in the source text.

To resolve these issues, researchers proposed contextual word embeddings that are pre-trained using RNN language models and are fine tuned on the given text to impart context. In 2017, Vaswani [14] proposed a transformer model that relies entirely on the attention mechanism called self-attention to extract dependencies between the source and target text. Self-attention (intra-attention) is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. This attention mechanism has been successful in various tasks like reading comprehension, and abstractive summarization. Transformers quickly became the state-of-the-art in Machine Translation and achieved significantly better BLEU (bilingual evaluation understudy) scores than the Seq2Seq models.

All the techniques[8], that used Transformers for text summarization until now, evaluated the summaries based on the metrics like ROUGE and METEOR, etc., which calculate the score based on the number of n-gram matches. In this paper, we used a different and better evaluation method called BERTScore. It calculates the score based on the pairwise dot products of the contextual word embeddings of the tokens of human summaries and the predicted summaries. These embeddings are generated using the BERT transformer.

For our model, we explored a recently-proposed transformer model called BART which uses a seq2seq architecture with a bidirectional encoder (like BERT) and a left-to-right decoder (like GPT). The pre-training task involves randomly shuffling the order of the original sentences and a novel in-filling scheme, where spans of text are replaced with a single mask token. BART is particularly effective when fine tuned for sequence generation tasks. It matches the performance of RoBERTa with comparable training resources on GLUE and SQuAD, achieves new state-of-the-art results on a range of abstractive dialogue, question answering, and summarization tasks, with gains of up to 6 ROUGE [9]. We applied this model on two different variations of the CNN dataset and obtained an increase of at least 6% in BERTScore and 50% in ROUGE - L scores in comparison to those of the baseline LSTM model.

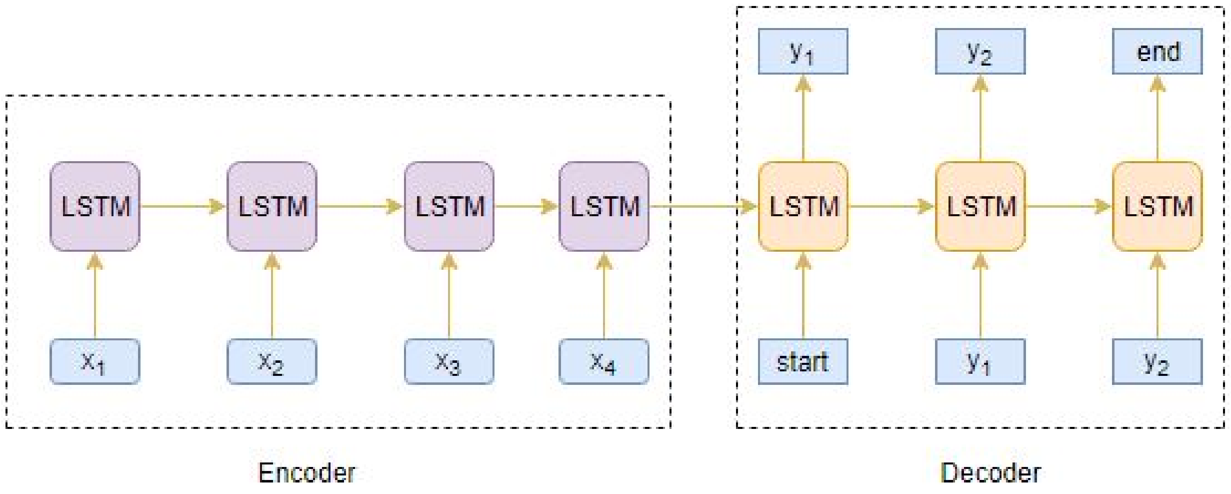


Figure 1: Baseline Model Architecture

2 Related work

A vast majority of past work in summarization has been extractive, which consists of identifying key sentences or passages in the source document and reproducing them as summary. The task of abstractive summarization has been standardized using the DUC2003 and DUC-2004 competitions. The data for these tasks consists of news stories from various topics with multiple reference summaries per story generated by humans. The best performing system on the DUC-2004 task, called TOPIARY [16], used a combination of linguistically motivated compression techniques, and an unsupervised topic detection algorithm that appends keywords extracted from the article onto the compressed output.

With the emergence of deep learning as a viable alternative for many NLP tasks [5], researchers have started considering this framework as an attractive, fully data-driven alternative to abstractive summarization. In [4], the authors use convolutional models to encode the source, and a context-sensitive attentional feed-forward neural network to generate the summary, producing state-of-the-art results on Gigaword and DUC datasets. In an extension to this work, [4] used a similar convolutional model for the encoder, but replaced the decoder with an RNN, producing further improvement in performance on both datasets.

One such approach is the [10] where they adapted the DeepMind question-answering dataset [6] for summarization, resulting in the CNN/Daily Mail dataset, and provided the first abstractive baselines. The same authors then published a neural extractive approach [11], which uses hierarchical RNNs to select sentences, and found that it significantly outperformed their abstractive result with respect to the ROUGE metric.

Their work starts with the same framework as [7], where they use RNNs for both source and target, but they go beyond the standard architecture and propose novel models that address critical problems in summarization. They also note that this work is an extended version of [10]. In addition to performing more extensive experiments compared to that work, they also propose a novel dataset for document summarization on which they establish benchmark numbers too.

Another significant contribution to this task was from [14] where they pre-trained a large Transformer language model and fine-tuned it for text summarization, demonstrating the model's sample efficiency. In order to use pre-trained weights more efficiently, they used a Transformer-based decoder only network during fine-tuning. This approach (1) avoids the redundancy of loading copies of the same pre-trained weights into the encoder and decoder, (2) uses fewer parameters compared to encoder-decoder networks, and most importantly (3) ensures all model weights, including those controlling attention over source states, are pre-trained.

3 Problem Statement

To build a transformer model that can summarize the CNN stories better than the baseline LSTM Seq2Seq encoder decoder model in terms of BERTScore and ROUGE scores.

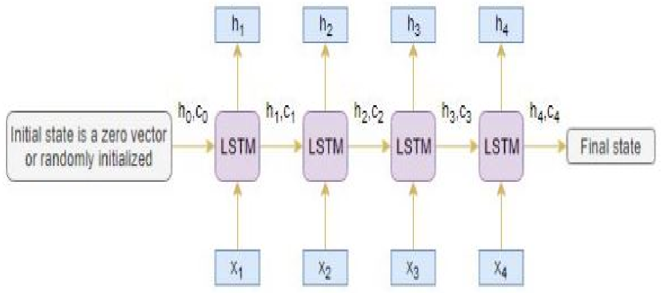


Figure 2: Encoder Module

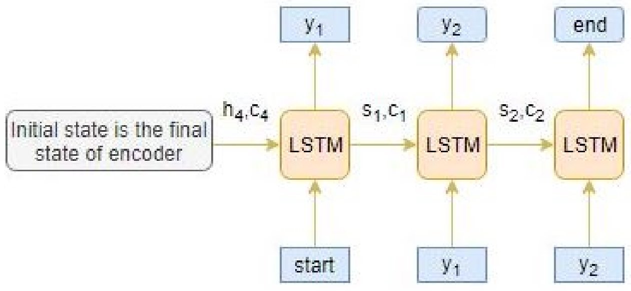


Figure 3: Decoder Module

4 Technical Approach

4.1 Baseline Model

Our baseline model (Figure 1) is a Sequence-2-Sequence model. This model consists of an encoder and decoder modules using Long Short Term Memory model (LSTM).

4.1.1 Encoder Module

An Encoder LSTM reads the entire input sequence and at each timestep, one word is fed into the encoder. It then processes the information at every timestep and captures the contextual information present in the input sequence (See Figure 2).The hidden state (hi) and cell state (ci) of the last time step are used to initialize the decoder, this is because the encoder and decoder are two different sets of the LSTM architecture.

4.1.2 Decoder Module

The decoder is also an LSTM network which reads the entire target sequence word-by-word and predicts the same sequence offset by one timestep. The decoder is trained to predict the next word in the sequence given the previous word (See Figure 3). <start> and <end> are the special tokens which are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting the target sequence by passing the first word into the decoder which would be always the <start> token. And the <end> token signals the end of the sentence.

4.1.3 Decoding the test sequence

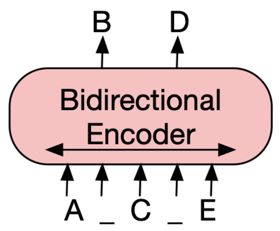
To decode a test sequence we first encode the entire input sequence and initialize the decoder with internal states of the encoder. It then passes the input tokens to the decoder. The output will be the probability for the next word. The word which has the maximum probability will be selected. It then passes the sampled word as input in the next time step and updates the internal states with the current time step. We repeat this process until we hit the end of the sequence or we reach the maximum length for the target sequence.

4.2 Our BART Model [9] [15]

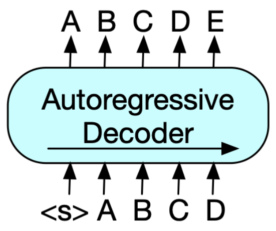
BART is a denoising autoencoder built using a sequence-to-sequence model that can be applied to a range of tasks. It pre-trains a model combining a Bidirectional encoder (like BERT) and an Auto-Regressive decoder (like GPT). Pre-training involves two steps, (1) text is corrupted using an arbitrary noise function and (2) a sequence-to-sequence model is learned to reconstruct the original text.

BART uses the standard sequence-to-sequence Transformer architecture from (Vaswani et al., 2017) [14], except, following GPT, that we modify ReLU activation functions to GeLUs and initialize parameters. For our “facebook/bart-large-cnn” model, we use 24 layers in the encoder and decoder. The architecture is closely related to that used in BERT, with the following differences: (1) each layer of the decoder additionally performs cross-attention over the final hidden layer of the encoder (as in the transformer sequence-to-sequence model); and (2) BERT uses an additional feed-forward network before word prediction, which BART does not. In total, BART contains roughly 10% more parameters than the equivalently sized BERT model.

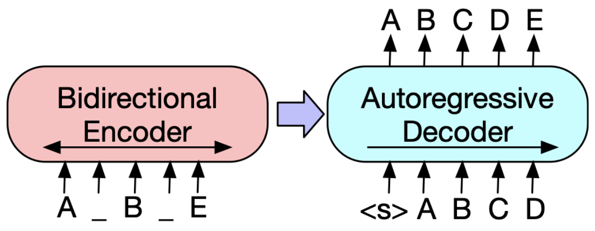
A key advantage of this setup is the noising flexibility; arbitrary transformations can be applied to the original text, including changing its length. There are lots of noise techniques, for example, randomly shuffling the order of the original sentences and using a novel in-filling scheme, where arbitrary length spans of text (including zero-length) are replaced with a single mask token. This approach generalizes the original word masking and next sentence prediction objectives in BERT by forcing the model to reason more about overall sentence length and make longer-range transformations to the input. BART is trained by corrupting documents and then optimizing a reconstruction loss—the cross-entropy between the decoder’s output and the original document. Unlike existing denoising autoencoders, which are tailored to specific noising schemes, BART allows us to apply any type of document corruption. In the extreme case, where all information about the source is lost, BART is equivalent to a language model.



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Figure 4: BART Architecture

4.3 Simple Transformers Library

We used the simple transformers library as one of the options for the project. Simple Transformers is designed around the way a person will typically use a Transformers model. At the highest level, Simple Transformers is branched into common NLP tasks such as text classification, question answering, and language modeling. Each of these tasks have their own task-specific Simple Transformers model. While all the task-specific models maintain a consistent usage pattern (initialize, train, evaluate, predict), this separation allows the freedom to adapt the models to their specific use case.

We import the BART [1] module from the library and the prepossessed data. For summarization tasks, encoder_decoder_type must be 'bart' and encoder_decoder_name can be chosen from any of the lists of standard pre-trained models. Also, the Seq2Seq function has a lot of parameters we can fine-tune. We only tuned repetition penalty, but we can always experiment with other parameters in the future. We also considered implementing Pointer-Generator networks proposed by Abigail See [13] with transformers as the generator network, but we could not do it due to its complexity and the computational constraints.

4.4 What is new and why our model works

The transformer networks became the state-of-the-art in NLP because of their parallel processing of words, the multi-head attention mechanisms, faster training and execution compared to RNNs. They also take advantage of GPUs because of the parallel processing. OpenAI have demonstrated that their transformer models GPT-2 and GPT-3 can generate extremely human-like texts [2].

The transformer architecture BART uses contextual word embeddings for tokens instead of traditional word embeddings. The contextual word embeddings, which are pre-trained using BERT and tuned with the training data, capture better semantic meanings of the words. They also use Positional word embeddings, that store the sequential information of the words, along with the contextual word embeddings. Positional word embeddings are an important part of transformers as the meaning of a word changes depending on its position in the sentence.

Also, the BART transformer, an improvement over the BERT, is an autoencoder model that can learn the representation of the input text better than the LSTM Encoder as it first corrupts the input text and uses Seq2Seq model to reconstruct the original input text. With a good representation of the input text, a decoder can predict good summaries.

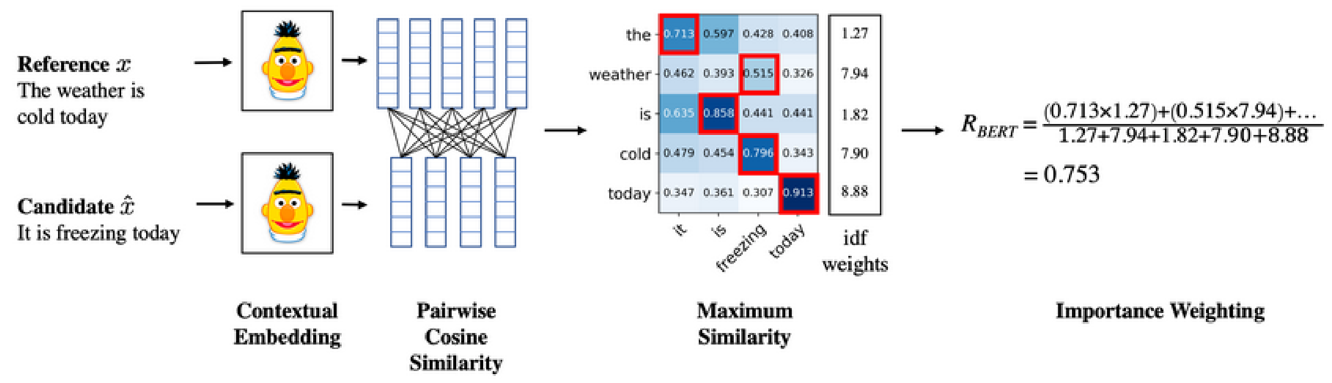


Figure 5: BERTScore Calculation

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x}_i^\top \hat{\mathbf{x}}_j \;, \quad P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \mathbf{x}_i^\top \hat{\mathbf{x}}_j \;, \quad F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}}$$

Figure 6: BERTScore Formulae: Recall, Precision, and F1 Score

5 Experimental Setup

5.1 Data

We used CNN dataset which contains 92,579 new articles paired with multi-sentence summaries. Due to computational constraints, we randomly selected 10000 for training, 1000 for validation, and 1000 for testing. We further divided each of these datasets into two datasets:

- Data 2_2: For the text, I selected the first 2 sentences. For the actual summary, I selected the first sentence.
- Data 3_2: For the text, I selected the first 2 sentences and the last sentence. For the actual summary, I selected the first 2 sentences.

5.2 Preprocessing

One of the most popular datasets that is freely available is the CNN News story dataset. The dataset was mainly developed for a Question Answering task but has been adapted successfully to be used for text summarization tasks by many previous works [3]. The dataset essentially consists of two parts: (1) the story and (2) multiple highlights for each story. We can use these highlights as multiple reference summaries for each news article.

Data cleaning is a challenging problem and must be tailored for the specific application of the system. In our case, we loaded all the stories and their respective highlights. We removed the source CNN office line if it exists. We normalize all the words to lowercase and remove all punctuations from the story and highlight as well. We also remove all the non-alphabetic characters from the dataset. We finally save the data as pickle files for future use.

There are many ways to improve on this aspect of the project but for now we are sticking to this level of data cleaning and we can improve on this in the future.

5.3 Hyper-Parameter tuned for the BART model

Repetition_penalty: Penalty for the repetition of tokens. 1.0 means no penalty. This resembles the coverage mechanism proposed by Abigail See [13].

We selected the values for the maximum input sequence length, and maximum target length through the data analysis of input text and the target text.

5.4 Evaluation Metrics Used

ROUGE (Recall-Oriented Understudy for Gisting Evaluation): This is like the BLEU metric but without penalty for shorter sentences. This measures the n-gram overlaps. In this paper they used three different ROUGE metrics:

(1) ROUGE-1: unigram overlap
(2) ROUGE-2: bigram overlap
(3) ROUGE-L: longest common subsequence overlap

Rouge is a popular evaluation metric for text summarization. But it gives a score based on the number of n-gram matches between the predicted summary and the human summary. Human summaries are subjective, and an article can have infinite summaries. Hence, Rouge, even though useful, is not a reliable evaluation metric. Sometimes, even low rouge scored summaries capture a lot of information from the article and high scored summaries could be bad.

Hence, we used a better evaluation metric called BERTScore1, which uses the contextual word embeddings from BERT and finds pairwise cosine similarity between the tokens of the predicted summary and the human summary, optionally weighted with their inverse document frequencies from the test dataset (see Figure 6).

We can also calculate precision, Recall, and F1 BERTScores, which are useful for the evaluation of text summarization predictions. Given a reference sentence x = (x1, x2, ... , xk) and a candidate sentence x̂ = (x̂1, x̂2, ... , x̂l) the recall, precision and F1 scores are given in Figure 5. x1, x2, x3 are contextual word embeddings of the tokens.

Rouge vs BERTScore example:

For the following example:

candidate = It is freezing today

reference = The weather is cold today.

The Rouge scores are Rouge-1: 0.45, Rouge-2: 0.0, Rouge-L: 0.51

The F1 BERTScores are 0.75 (with idf weights) and 0.94 (without idf weights) Note: We observed that BERTScore is immune to repetition of words. It gave lesser scores for summaries with repetition (check the BERTScore ipynb file).

6 Results

Note: all the reported scores are the average scores over the whole test dataset.

For Data 2_2, the best hyperparameters for the BART model are:

- maximum sequence length = 80
- maximum target length = 15
- repetition penalty = 1.112

The BERTScore and Rouge Scores on the test dataset are given in Table 1.

Model Name	F1 Bert Score	ROUGE Scores		
		1	2	L
LSTM Seq-2-Seq	79.92	10	1	11
BART	85.17	20	6	22

Table 1: F1 BERTScores and Rouge Scores on Data 2_2

For Data 3_2, the best hyperparameters for the BART model are:

- maximum sequence length = 90
- maximum target length = 25
- repetition penalty = 1.12

The BERTScore and Rouge Scores on the test dataset are given in Table 2.

Model Name	F1 Bert Score	ROUGE Scores		
		1	2	L
LSTM Seq-2-Seq	79.44	16	2	16
BART	84.72	24	6	24

Table 2: F1 BERTScores and Rouge Scores on Data 3_2

Our BART model gave better F1 BERTScores and Rouge Scores than the baseline LSTM Seq2Seq model. This is expected as the BART has 16 attention heads for encoder and decoder. Each of the attention heads focus on different positions in a sentence and thus capture different aspects of the sentence like semantics and syntactics etc. Hence, the decoder will get a representation that is an accumulation of representations of different aspects.

7 Conclusion and future work

Upon manual inspection of the predicted summaries of the both test datasets, we observed that some of the summaries didn't end properly due to the maximum length parameter. It would have been better if there is a parameter in the simpletransformers BART model to specify how many number of sentences we want, it just has the maximum number of output tokens parameter. And our model cannot generate the summaries in form of sentences, not just tokens. These are some interesting things to work on.

In some cases, the first 2 (and last sentence) of the source text didn't capture the gist of the stories hence, for such test instances, the summaries turned out to be very different from the actual summaries. Our model would have produced better summaries in such cases if given more data along with high computational power.

The abstractive summarization using BART (with repetition penalty), although produce summaries closer to human-level summarization compared to extractive summarization techniques, has two issues: they tend to produce factual (numbers) details inaccurately, and cannot handle the out-of-vocabulary (OOV) words. The "Pointer-Generator" mechanism proposed by Abigail See [13] can tackle the factual details and OOV words. This approach combines both the extractive and abstractive summarization techniques. The pointer copies the words from the source text while the generator generates new words simultaneously. It uses the generation probability pgen ([0,1]) to select a word from the copied and generated words. We can use the Transformers network as the generator and BERTScore as the evaluation metric.

References

[1] [n.d.]. Seq2Seq Model - Simple Transformers. https://simpletransformers.ai/docs/seq2seq-model/. (Accessed on 12/08/2020).

[2] [n.d.]. Transformer Neural Network Definition | DeepAI. https://deepai.org/machine-learning-glossary-and-terms/transformer-neural-network. (Accessed on 12/08/2020).

[3] Jason Brownlee. 2019. How to Prepare News Articles for Text Summarization. https://machinelearningmastery.com/prepare-news-articles-text-summarization/

[4] Sumit Chopra, Michael Auli, and Alexander M Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 93–98.

[5] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research* 12, ARTICLE (2011), 2493–2537.

[6] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems* 28 (2015), 1693–1701.

[7] Baotian Hu, Qingcai Chen, and Fangze Zhu. 2015. Lcsts: A large scale chinese short text summarization dataset. *arXiv preprint arXiv:1506.05865* (2015).

[8] Urvashi Khandelwal, Kevin Clark, Dan Jurafsky, and Lukasz Kaiser. 2019. Sample efficient text summarization using a single pre-trained transformer. *arXiv preprint arXiv:1905.08836* (2019).

[9] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *CoRR* abs/1910.13461 (2019). arXiv:1910.13461 http://arxiv.org/abs/1910.13461

[10] 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* (2016).

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

[12] Horacio Saggion and Thierry Poibeau. 2013. Automatic text summarization: Past, present and future. In *Multi-source, multilingual information extraction and summarization*. Springer, 3–21.

[13] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. *CoRR* abs/1704.04368 (2017). arXiv:1704.04368 http://arxiv.org/abs/1704.04368

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[15] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[16] David Zajic, Bonnie Dorr, and Richard Schwartz. 2004. Bbn/umd at duc-2004: Topiary. In *Proceedings of the HLT-NAACL 2004 Document Understanding Workshop, Boston*. 112–119.

TEAMWORK DIVISION AND OVERALL EXPERIENCE

1. Kalyan Kumar P: Preprocessing, BART Transformer models, and Report
2. Yash Mahajan: Seq2Seq LSTM baseline Model, preprocessing and Report
3. Harikrishna Nagarajan: Pointer-Generator Transformer, Data Collection And Report

   Initially we split the work three ways so that all three of us can work simultaneously. We were successful with the Transformer models and seq2seq Baseline model but couldn't achieve any form of success with the Pointer Generator Transformer model due to its complexity and computational requirements. If provided with more computational power we would have run our models on complete data instead of using only a section of the data. We would have also tried building Pointer-Generator models with both Transformers and RNNs as the generator networks. Overall working with each other was fun, but it would have been more productive if COVID wasn't a thing.

INSTRUCTIONS TO RUN THE CODE

1. Baseline model file name:
   a. LSTM_baseline_model.ipynb
2. BART model file name:
   a. Simple_Transformers_2 Test Evaluation.ipynb &
   b. Simple_Transformers_3 Test Evaluation.ipynb

To run the files just follow the instructions below:

1. It's better to run the code in Google Colab as it requires GPU and for the BART model using simple Transformers you need high ram (that's why I subscribed for Google Colab pro).
2. Make sure you have the data we have submitted
3. Check the data path for the models
4. Then, just click on restart and run all cells.