# Transformers for Summarization of Short Documents

Gurpreet Singh

## Introduction

The Internet is filled with interesting literature and documents that would take single human hundreds if not thousands of years to decipher. Yet not everything online is something that we’d want to spend our time reading. Whether you are a researcher, a student, or an industry professional, it would be useful to have a summarization of a document before reading it to see if it gauges our interest. This could save countless hours and help those that use this service to create better work. Thus, the purpose of this project is to create models such as a word frequency model or a pre-trained Transformer model that could accurately summarize texts while maintaining its original meaning.

Due to the system requirements to summarize larger texts, this model will focus on smaller texts that are several sentences long. This challenge will only be attempted by one individual in the group. There is existing literature on it, such as Transformer models from Google, OpenAI, and HuggingFace, and different frequency models on the Internet that can be of use. I will be using HuggingFace’s pretrained BART and T5 models to summarize the inputs and create a word frequency model as a simpler method for comparison. Although it would be interesting to train a transformer model from scratch, this was deemed out of the scope of this project because of the system requirements. One of the main issues there, and with the pre-trained models from HuggingFace, is the fact that there are out-of-memory errors when using the GPU. As such, an A10 24GB Tensor Core GPU from LambdaLabs was rented at $0.60/hour was rented. In total, the GPU was rented for around 24 hours.

## Problem Formulation

There are two main methods to summarize documents: extractive and abstractive. Extraction is selecting key words of a text document to formulate a summary. This is essentially like taking the main points of the document and putting it together using methods such as word frequency models or TF-IDF. The method used for this project is word frequency model. The key concept to understand with this method is that weighted frequencies must be calculated by dividing word frequencies by the maximum frequency. In plain English, this shows you how important the words are, or how repeated they are, in the whole document.

=

*Formula 1: Formula for Weighted Frequency (WF)*

From here, to get the sentence scores, the weighted frequencies for each word of the sentences must be added together. Using algorithms like heapq, the top sentences can be selected as a summary output. Finally, using ROUGE F1 scoring metrics, a ROUGE-1, ROUGE-2, and ROUGE-L score will be outputted.

On the other hand, using abstraction is like talking to a human being in the sense that you generate words to summarize the document. It involves more advanced algorithms, such as an encoder-decoder model, and in this case, transformers were used. More specifically, pre-trained BART and T5 transformers from HuggingFace. BART is a sequence-to-sequence model that acts as a denoising autoencoder and uses bi-directional encoders and left-to-right decoders. Since the model denoises, or derives the original document from a corrupted one, it turns out to be quite effective. The T5 model is a text-to-text transformer where the input and output are text strings. This makes it a very effective encoder-decoder model for summarization. The transformer method will involve tokenizing a dataset and preprocessing using already made HuggingFace functions. From there preparing the data for training by batching and preparing the training and validating datasets. Finally, training and generating the texts. ROUGE-N F1 scores will also be used to evaluate these models.

## Methods

A computer screen shot of a code

Description automatically generatedA screenshot of a computer code

Description automatically generatedAs previously mentioned, the extractive method will be doing text summarization using the word frequency method. Before this method could be applied, however, data-preprocessing needed to be done. Punctuation, numbers, special characters, and stopwords were removed. Contractions were replaced with the full words and strings were lowercased. Finally, using the nltk library, the inputs were tokenized. Weighted frequencies were calculated using the formula in the previous section by dividing each word frequency by the maximum frequency. Using the weighted frequency, we iterate through the sentences to generate sentence scores. Referring to the weighted frequency dictionary, each word’s weighted frequency was added up and that result was the score. In this case, to prevent small sentences, no sentences below two words were considered. Likewise, to prevent long run-off sentences that would skew the sentence scores, sentences above 40 also weren’t considered. Another alternative would be to have a cut-off at 40 words to still consider those inputs and not discard them. After the scores are calculated, the top 5 sentences are selected for the summary using the heapq function.

*Figures 1 & 2: Sentence Scores (left) and Data Preprocessing (right)*

For the transformer methodology, HuggingFace’s pretrained models were used, specifically BART and T5. Both models were implemented in separate ways when it comes to training. They started off the same way in the sense that the data is tokenized and the model is imported from HuggingFace. The data is batched using the DataCollator function and prepared for training. For both models, the same data preprocessing as the extractive model was not applied, so the inputs for the tokenizer were texts. The assumption was that the tokenizers for both models would handle the preprocessing.

For BART, the training parameters were inputted inside the TrainingArguments and Trainer functions from HuggingFace. This could only be used with PyTorch. Below shows how the functions were implemented. This model was trained with the parameters: batch\_size = 4, learning\_rate = 4e-5, weight\_decay = 0.01, epochs = 3, warm\_up=500, and training\_steps = 7500.

A screenshot of a computer code

Description automatically generated

*Figure 3: TrainingArguments and Trainer Function Implementation Example*

The T5 implementation was done using TensorFlow and the compile/fit methods that the library has. One of the implementations has batch\_size = 8, learning\_rate = 4e-6, weight\_decay = 0.01, and epochs = 10. This model was ran multiple times in different ways as it was proving to be the best, and that will be shown in the next section.

A screenshot of a computer code

Description automatically generated

*Figure 4: T5-Base Implementation Example*

## Dataset and Experiments

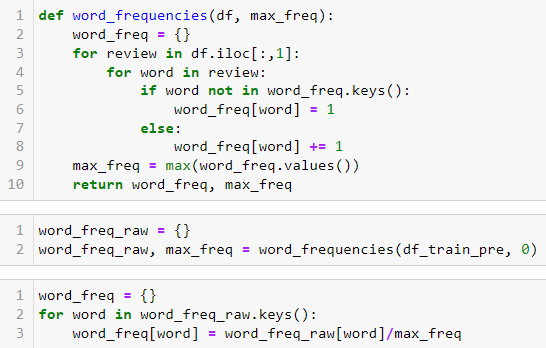
The dataset used is the CNN/DailyMail dataset downloaded directly from HuggingFace datasets. This is an excellent dataset for summarization as one column gives the article, and the other column gives a summarization. However, there are hundreds of thousands of rows, so there will be a lot of computational resources used. Depending on the model, over 75% of the data was dropped from the dataset to save time and resources.

A white screen with black text

Description automatically generated

*Figure 5: Dataset Description*

For the word frequency method, only 5% of the dataset was used. The data preprocessing was applied. Before data preprocessing, the data can go from “LOS ANGELES, California (CNN) – The Transportation…” to “[los, angeles, california, cnn, transportation]”. After the preprocessing, the word frequencies and weighted frequencies are calculated.



*Figure 6: Weight Frequencies Code*

The result is a dictionary of words that have keys with values between 0 and 1. Using the nltk library, the words are tokenized and the sentence score is calculated. Finally, the rouge score is calculated, where we compare the reference summary to the generated summaries.

A screen shot of a computer code

Description automatically generated

*Figure 7: Rouge Scores for Word Frequencies*

This resulted in scores: rouge1: 0.246, rouge2: 0.085, rougel: 0.148. It makes sense that rouge1 score is quite high since the most frequent words are going to be in the generated summaries, so if rouge1 looks for 1-gram matches, it will find quite a few.

Next the BART model was implemented. Both the transformer models were inspired by the great resources available from HuggingFace directly, but BART more so than T5 since I also implemented the Trainer function with it. The training parameters for BART are: batch\_size = 4, learning\_rate = 4e-5, weight\_decay = 0.01, epochs = 3, warm\_up=500, and training\_steps = 7500. Around 15% of the data was tokenized used BartTokenizer and the model was created using BartForConditionalGeneration. This BART model is specifically from the Facebook/Bart-Base model which has 139M parameters. This model was trained for 1.5 hours. The trainer function specifically was already shown above, but after the model was trained, it generated summaries and compared it to the actual summaries in the original database, like the word frequency method, and ROUGE scores were calculated. 100 summaries were generated, and this is also the same way the T5 models were evaluated.

A screenshot of a computer program

Description automatically generated

*Figure 8: Generating Summaries Implementation*

This is an example of some of the text generated: “'NEW: The "We\'re\'re "The "I"""\nHe says he says he was found in the death.\nHe was a "I\'m"\n"', ‘NEW: New York's death of the death death of death.\nHe says he says he was killed in the death.’”

Although some words are comprehensible, it doesn’t make much sense. Perhaps the implementation was not the best or there are better training parameters to use. Below is how the ROUGE scores were calculated for the transformer models.

A screen shot of a computer code

Description automatically generated

*Figure 9: ROUGE Score Calculations for Transformers*

The BART Rouge Scores are: rouge1 = 0.1316, rouge2 = 0.0162, rougel = 0.1267.

All the T5 transformers were implemented the same way, only the training parameters differed. Since this transformer was clearly better than all other experiments, multiple implementations were created to understand the best approach. It seems that the T5-Base model performs the best compared to T5-Small, which is no surprise, and running it at more epochs made a difference. Anywhere from 10% of the dataset was used for T5-Base (3 epochs), 15% for T5-Base (10 epochs), and 25% for T5-Small (3 epochs).

Like with BART, the T5 implementation required me to tokenize the data using AutoTokenizer, and implement T5-Base or T5-Small using the model TFAutoModelForSeq2SeqLM. The data was batched using DataCollator and the method “prepare\_tf\_dataset” that allowed the model to process data. Below are the results from the ROUGE calculations. Additionally, there are plots from the best T5-Base model that provide useful information about model accuracy and loss. It seems that if I trained the model for more than 10 epochs, maybe 12-15, it would have converged. It will also be interesting to see how the training parameters individually affect the scores, but there was not enough time for this in this experiment.

A screenshot of a table

Description automatically generated

*Figure 10: ROUGE scores for all experiments*

A graph of a model loss

Description automatically generatedA graph with blue line and orange line

Description automatically generated

*Figures 11 & 12: Model Accuracy and Model Loss Graphs*

Actual T5-Small Summary vs T5-Base summary:

“Actual Summary: Theia, a bully breed mix, was apparently hit by a car, whacked with a hammer and buried in a field .\n"She\'s a true miracle dog and she deserves a good life," says Sara Mellado, who is looking for a home for Theia.”

“Prediction Summary: Theia, a friendly white-and-black bully breed mix now named Theia, is receiving care at the Veterinary Teaching Hospital. Theia was found emaciated and dislocated in a field, but survived. Theia is a friendly white-and-black bully breed mix in Washington State. She has been receiving care at the Veterinary Teaching Hospital.”

As we can see, both summaries provide coherent sentences, and looking at the original context, you could say both summaries summarize specific parts of the inputs. It’s hard to judge the best summary models using only ROUGE scores, there must be some human element to it as well.

## Project Management

Members: Gurpreet Singh

Responsible for the research, implementation, and creating the project reports/presentations.

## Conclusion

In conclusion, the methods that were used to complete this project involve both extractive and abstractive methods such as word frequency model, BART and T5 transformer. The goal of the project was to create a summarization model that is able to coherently summarize an input text while maintaining its original meanings. This was best accomplished using the T5 Transformer, likely due to it being an encoder-decoder model that focuses on text-to-text transformations. The main challenge I faced was out-of-memory errors that took up a lot of time and space, but luckily LambdaLabs was a convenient solution for my problem. Overall, I’ve learned a ton about the different parts of a transformer, how to implement a transformer using pretrained models, and also the different techniques that exist for summarization. In the future, implementing TF-IDF for extractive summarization or GPT/BERT pretrained models for abstractive summarization would be an interesting experiment. Additionally, fine-tuning the current models would be important to understand what would make the model better.

## Key references

Attention is all you need:

* <https://arxiv.org/abs/1706.03762>

HuggingFace summarization:

* <https://huggingface.co/docs/transformers/tasks/summarization>

Datasets:

* <https://huggingface.co/datasets/cnn_dailymail>

Abstraction vs Extraction:

* [Blog.paperspace.com](https://blog.paperspace.com/extractive-and-abstractive-summarization-techniques/#:~:text=The%20process%20of%20extractive%20summarizing,verbatim%20from%20the%20source%20material.)
* [TowardDataScience](https://towardsdatascience.com/understanding-automatic-text-summarization-2-abstractive-methods-7099fa8656fe#:~:text=Abstractive%20summarizers%20are%20so%2Dcalled,different%20from%20the%20original%20document.)
* <https://towardsdatascience.com/build-your-own-transformer-from-scratch-using-pytorch-84c850470dcb>

Rouge:

* <https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/>

Word Frequency Model:

* <https://medium.com/analytics-vidhya/simple-text-summarization-using-nltk-eedc36ebaaf8>
* <https://ceur-ws.org/Vol-3395/T6-4.pdf>
* <https://medium.com/analytics-vidhya/term-frequency-text-summarization-cc4e6381254c>

Pre-Trained Models:

* <https://huggingface.co/docs/transformers/model_doc/t5>
* <https://huggingface.co/docs/transformers/model_doc/bart>
* <https://pylessons.com/transformers-training>
* <https://github.com/Moeinh77/Transformers-for-abstractive-summarization/blob/master/main-github.ipynb>
* [https://www.tensorflow.org/text/tutorials/transformer#the\_transformer](https://www.tensorflow.org/text/tutorials/transformer%23the_transformer)
* [https://colab.research.google.com/github/arminnorouzi/machine\_learning\_course\_UofA\_MECE610/blob/main/L07\_Generative\_AI/L07b\_Transformer.ipynb#scrollTo=MvX44eGv6GQl](https://colab.research.google.com/github/arminnorouzi/machine_learning_course_UofA_MECE610/blob/main/L07_Generative_AI/L07b_Transformer.ipynb%23scrollTo=MvX44eGv6GQl)
* <https://www.youtube.com/watch?v=ISNdQcPhsts&t=3538s&ab_channel=UmarJamil>
* <https://github.com/laxmimerit/NLP-Tutorials-with-HuggingFace/blob/main/NLP_with_HuggingFace_Tutorial_4_Summarization.ipynb>

Presentation:

* <https://techcommunity.microsoft.com/t5/ai-customer-engineering-team/bootstrap-your-text-summarization-solution-with-the-latest/ba-p/1268809>
* <https://www.turing.com/kb/brief-introduction-to-transformers-and-their-power>
* <https://kikaben.com/transformers-encoder-decoder/>