NLP Project: Text Summarization

Ahmed Serry, Farida Helmy and Seif Maged

May 26, 2023

Abstract

This NLP project aims to develop a text summarization [Moh21] model that can generate a concise and coherent summary of a given text document while preserving the most important information and key ideas. The project will explore various approaches to extract and prioritize relevant information. The model will be trained and evaluated on a large dataset of diverse text documents, The dataset That we chose to experiment on is the CNN Daily News Dataset from Huggingface [HKG+15]. The project's ultimate goal is to improve the efficiency of information processing and facilitate effective communication by providing a quick and accurate summary of lengthy texts.

1 Introduction

Text summarization is an important task in all large language-generating models. With the outburst of Chat-GPT, our team decided to investigate one of its main tasks which is Text Summarization. There are two ways to look at this task: Constructing an Extractive and Abstractive model. Extractive techniques choose the most significant sentences from the huge bulk of text, regardless of whether they comprehend the context or not, and hence create a summary that is merely a portion of the complete document. On the other hand, Abstractive approaches require sophisticated NLP techniques, such as word embeddings, to comprehend the meaning of the text and produce a coherent summary. As a result, Abstractive methods are considerably more challenging to develop from the beginning, as they require a vast amount of data and parameters. That's why in our project, to benefit from NLP techniques that we got familiar with in this course, our team decided to focus on the Abstractive techniques when implementing this task. In this project, we used the CNN Daily news dataset to experiment the different techniques of abstractive summarization using Attention-based Transformers. In the following sections, we will propose the data preprocessing required to shape the data correctly for our models, the data analysis to have general knowledge and expectations from our data and we will propose the architecture for our project to how we will achieve the text summarization task successfully.

2 Fetching the Dataset

The dataset that we have at our hand was extracted from huggingface. Then we split the dataset into training dataframe consisting of 287113 rows, validation dataframe consisting of 13368 and testing dataframe consisting of 11490 rows.

3 Data Preprocessing

Looking at our dataset, we can see that it has 3 columns denoting the articles, their corresponding summaries and ids. The Data Preprocessing techniques will be done mainly on the 2 columns of the articles and the summaries.

3.1 Articles Preprocessing

This column is the most interesting column for our preprocessing task, because it denotes the raw text that we will experiment on to reach the desired summaries. Hence there are multiple techniques that

need to be done in order to properly handle the corpus of the articles.

1. Removing Stop words and punctuation
2. Normalisation
3. Removing Null values
4. Tokenizing each article into words.
5. Tokenizing each article into sentences.
6. Stemming

3.2 Summary Preprocessing

This column is our target column and consists of the raw text summary of the corresponding article in the row. Hence, there are multiple preprocessing techniques that should be done, which are the same techniques mentioned above:

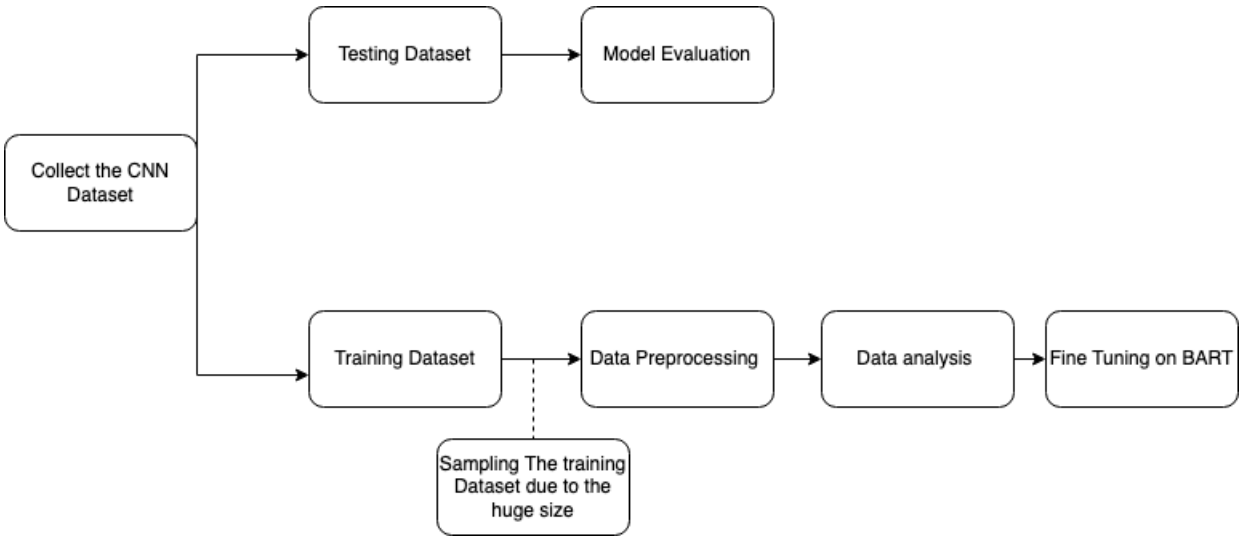
1. Removing Stop words and punctuation
2. Normalisation
3. Removing Null values
4. Tokenizing each article into words.
5. Tokenizing each article into sentences.
6. Stemming

4 Data Analysis

We want to investigate the nature of our articles, hence we studied the number of words in our corpus, the most frequent words in our corpus, and the number of unique words in our corpus. Furthermore, we wanted to study the average sentences per article in our dataset to investigate the size of the raw data that we want to summarize. This is applied on both columns: the article and the summary.

5 System Architecture

In this section, we will propose our system architecture for completing this project: After exploring the previous work done on text summarization, to the best of our knowledge, the area of applying BART [LLG+20] and GPT is the state of the art of accomplishing this task. Hence, we will try to analyze the power of this Bart after fine-tuning it for the CNN Daily News Dataset. A diagram displaying our system architecture below describes the road map for our project.



6 Methodology

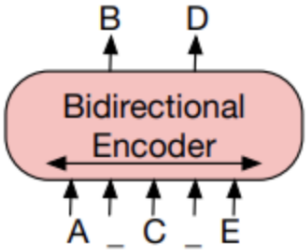


Figure 1: BERT Architecture

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained transformer-based language model that has shown remarkable performance in various natural language processing (NLP) tasks such as sentiment analysis, question answering, and text classification. The strength of BERT lies in its ability to learn contextualized word representations by leveraging the bidirectional nature of the transformer architecture, taking into account not only the words that come before but also the words that come after the target word.

However, BERT has limitations when it comes to text generation tasks. While it can understand the representation of the text really well, it may struggle to generate coherent and meaningful text sequences. This is because BERT is not designed to generate new text from scratch but rather to predict the next word or sequence of words given the context.

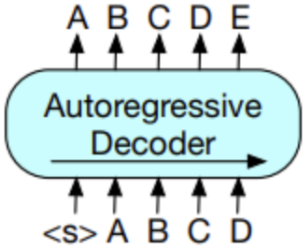


Figure 2: GPT Architecture

GPT (Generative Pre-trained Transformer) is a transformer-based language model that has shown remarkable performance in generating natural language text. Unlike BERT, GPT is designed for text

generation tasks and can generate coherent and meaningful text sequences from scratch. GPT uses a decoder architecture that depends on the past input to generate output in several tasks, such as language modeling, text completion, and text generation.

However, while GPT excels in text generation tasks, it sacrifices the full understanding of the underlying text that BERT provides. GPT generates text based on the previous input, but it does not take into account the full context of the text. This can lead to generated text that is coherent but not necessarily reflective of the overall meaning of the text.

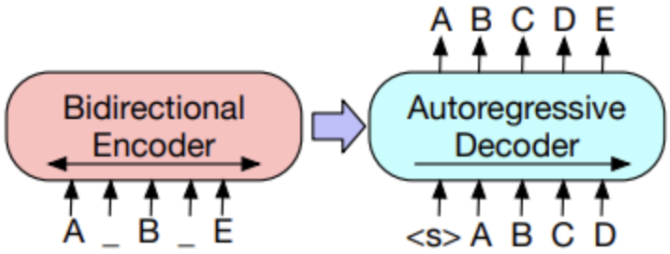


Figure 3: Combining BERT and GPT to build BART

BART (Bidirectional and Auto-Regressive Transformer) is a state-of-the-art model for text summarization that utilizes the power of both bidirectional and auto-regressive transformers. It consists of an encoder-decoder architecture, where the encoder uses self-attention mechanisms and feed-forward neural networks to process the input text, while the decoder generates the summary.

To improve text generation tasks, a potential solution is to combine the strength of BERT in understanding the representation of text with a decoder architecture. Additionally, to overcome the trade-off between generation and understanding, a combination of BERT and GPT can be used, where BERT provides a full understanding of the text and GPT generates text based on that understanding. By combining the power of these models, we can potentially achieve better performance in text generation tasks while still maintaining a full understanding of the underlying text.

The decoder takes the final hidden states from the encoder and generates the summary. It uses an auto-regressive approach, where it predicts one token at a time, conditioned on the previously generated tokens. During training, the decoder is exposed to the ground truth summary, helping it learn to generate coherent and informative summaries.

BART also incorporates several techniques to improve its performance. It includes a masked language model pretraining stage, where the model learns to predict masked tokens. This helps the model capture language patterns and generate accurate and fluent summaries. Additionally, BART utilizes a denoising objective, where it learns to reconstruct original text from noisy versions. This further enhances the model's ability to understand and summarize the input text effectively.

Overall, the BART architecture for text summarization has proven to be highly effective in producing concise and coherent summaries by leveraging the power of bidirectional and auto-regressive transformers, along with pretraining and denoising techniques. And in the following subsections, we will explain how we finetuned this model for training and how we generated the summaries, ie the desired output.

6.1 Training and Finetuning

To start our implementation, We picked up a portion of the dataset to experiment on.

We loaded a tokenizer from the BART Tokenizer, which has the pretrained model of BART Large and its tokenizer developed by facebook. Which made sense to use pytorch to fine tune the pretrained model. Afterwards, we load the model and we utilize the gpu from Colab Pro to help with the training process.

Our next step was to initialize the training arguments for the trainer of the model. These arguments define the configuration for training a model using the BART summarizer. The "num train epochs" argument specifies the number of times the training data will be iterated, we chose one epoch to reduce the time it takes to run the trainer. "per device train batch size" sets the batch size for training data, we also chose one batch after trying several batches at first and the trainer was not working properly,

however we solved it after initializing it to 1 batch. The "fp16" argument enables mixed precision training, utilizing lower precision to speed up training. "per device eval batch size" sets the batch size for evaluation data, also set to 1 in our case. "warmup steps" determines the number of steps taken to warm up the learning rate scheduler, we chose 500 for this parameter. "weight decay" specifies the amount of weight decay applied to the model's parameters during training, in our case we set it to 0.01. The "output dir" defines the directory where the trained model will be saved, we mounted a google drive to have access to the checkpoints after our training is done. "overwrite output dir" ensures that any existing model in the output directory will be overwritten. Lastly, "save steps" determines the interval at which the model checkpoints will be saved during training, and we set this parameter to 1000 steps.
Then we initialized a Trainer object for training our model. The Trainer is initialized with several parameters. The model parameter specifies the model to be trained, which is passed as an argument. The args parameter is set to training args, which contains the training configuration defined earlier. The data_collator parameter specifies the data collator object, which is responsible for batching and preprocessing the training data. It is passed as an argument. The train_dataset parameter is set to dataset_train, which represents the training dataset used for training the model, we will elaborate on the chosed trainng data in the experiments section. By initializing the Trainer with these parameters, we start the training process by calling the appropriate methods provided by the Trainer object. This includes methods for training, evaluating, and generating predictions using the trained model.
We used the trainer.train() method to finetune our model. This method is equivalent to using the traditional pytorch training loop. After training our model, we saved it to google drive to access it later in future runs.

6.2 Predictions and Output Interpretation

We proceed by testing our model to perform the task that we have at hand which is text summarization using a pre-trained model. First, we set the device to the CPU. Then, we move the model to the specified device, ensuring that the model's computations will be performed on the CPU. Next, we tested on one sample the input_text variable, which contains a piece of text to be summarized. We also define the ground_truth variable, which represents the expected summary or the ground truth for evaluation purposes.
The input text is tokenized using the tokenizer's encode() method. The resulting tokens are stored in input_ids, a PyTorch tensor. The tokenizer.encode() function handles various tokenization tasks such as splitting text into tokens, truncating to a maximum length, and encoding the tokens as input for the model.
To generate the summary, the code calls the model's generate() method, passing the input_ids tensor as input. The num_beams parameter controls the number of beams used in beam search decoding, which helps in finding better-quality summaries. The max_length parameter sets the maximum length of the generated summary, and early_stopping=True enables stopping the generation when the model predicts an end-of-sentence token.
The generated summary is decoded using the tokenizer's decode() method, removing any special tokens and converting the summary from token IDs to a readable text format. The generated summary is printed, so we can evaluate it later
Finally, we display the generated summary and the ground truth allowing for a comparison between the generated summary and the expected one for evaluation purposes.

7 Experiments

During the initial stages of our experimentation, we decided to train the model using a small subset of only 10 samples. As we proceeded with the training process, an interesting observation emerged: we noticed a clear trend indicating that as the number of training samples increased, the generated summaries became progressively shorter. This finding suggests a correlation between the size of the training dataset and the conciseness of the generated summaries. It highlights the importance of having a sufficiently large and diverse training set to ensure the generation of comprehensive and informative summaries. Further investigation and experimentation are needed to understand the underlying mechanisms and optimize the model's behavior in response to different training dataset sizes.

To ensure a comprehensive evaluation of our experiments, we opted to utilize the BLEU score, a precision-oriented metric widely employed for assessing the quality of text summarization. Given our objective of generating meaningful summaries, the BLEU score aligns well with our aim. By emphasizing precision in n-gram matching between the generated summaries and reference summaries, BLEU enables us to capture the accuracy and relevance of our summarization outputs. This choice of evaluation metric facilitates a focused assessment of the effectiveness of our models in producing concise and meaningful summaries, thereby providing valuable insights into the quality of our summarization system.
Digging deeper into the details of our model training process, we started by training the model on a subset dataset consisting of 10,000 samples. To ensure regular progress monitoring and facilitate model restoration, we implemented a checkpointing strategy, saving the model's state every 1000 samples(Already explained in the methodology section). As a result, we accumulated a total of 10 checkpoints, capturing the model's progress at different stages of the training procedure. This meticulous approach allows us to analyze and compare the model's performance across these checkpoints, enabling a comprehensive assessment of its evolution and providing valuable insights into its learning dynamics and convergence.

8 Results

8.1 BLEU Score

The BLEU[NLPnd] score, originally developed for machine translation but adapted for text summarization evaluation, is a popular metric used to assess the quality of summaries. It measures the similarity between a generated summary and reference summaries based on n-gram precision and brevity penalty. Higher BLEU scores indicate better alignment with the reference summaries, indicating higher quality and relevance. We note that BLEU might have some limitations, by assessing lower scores to some summaries that actually make sense, after conducting some research, we found out that its bias towards shorter summaries can impact the evaluation. To overcome these limitations, researchers often combine BLEU with other metrics like human evaluations to obtain a more comprehensive assessment of summarization quality. The use human judgment provides a more thorough understanding of summary quality, ensuring a more reliable evaluation of text summarization systems.

8.2 Applying BLEU on our Test Dataset

To evaluate the performance of our model at different checkpoints, we carefully selected a test set comprising 10 samples. Leveraging this test set, we systematically assessed the quality and effectiveness of each checkpoint in generating summaries.By employing the BLEU score, we were able to quantitatively measure the similarity between the generated summaries at each checkpoint and the corresponding reference summaries. This approach facilitated a rigorous evaluation of the model's summarization capabilities at various stages of training, enabling us to identify the strengths and weaknesses of different checkpoints and gain valuable insights into the model's overall performance and progress.
The following Graph demonstrates the result of our experiment on the test dataset.

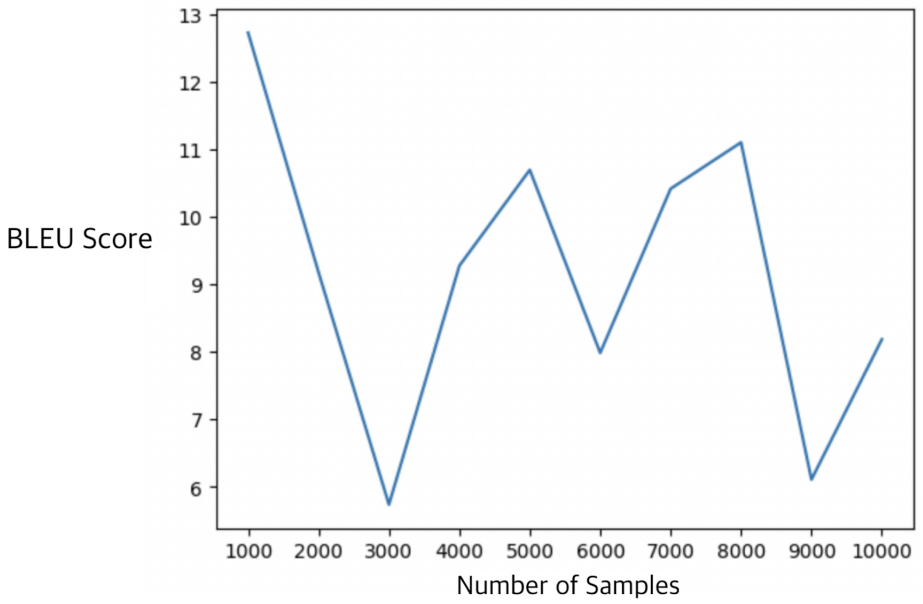


Figure 4: BLEU Scores for each sample batch.

After careful analysis, we observed that the model trained on 1000 samples outperformed the others in terms of summarization quality. One possible reason for this improved performance could be attributed to the generation of longer summaries by this particular model. As a result, there was a higher degree of overlap between the content generated by the model and the reference summaries. This increased overlap indicates a better alignment with the desired summary content, leading to higher precision and improved performance. However, further investigation and experimentation are required to fully understand the underlying factors contributing to this phenomenon and to ascertain if there are additional variables at play that influenced the model's performance on the given dataset.

9 Limitations and Future Work

The task of text summarization faces several limitations, one of which is the lack of resources for handling large datasets. Text summarization typically requires large amounts of annotated data for training effective models. The lack of sufficient training data can hinder the performance and generalization ability of text summarization models, particularly when dealing with complex or specialized content, which is well highlighted in the results section when comparing to the state-of the art.

The field of text summarization continues to evolve, and there are several potential avenues for future work. For instance, multi-document summarization poses an interesting research area, where the task involves summarizing information from multiple source documents. Developing techniques to effectively extract and synthesize information from diverse sources can contribute to creating comprehensive and concise summaries.

Furthermore, exploring the ethical and societal implications of text summarization is crucial. Research can focus on ensuring fairness, bias mitigation, and transparency in summarization models. Investigating methods to control the output to align with desired ethical standards and providing users with control over the generated summaries can help build trust and address concerns.

Not to ignore that future work in text summarization should focus on advancing the state-of-the-art models, addressing domain-specific challenges and enhancing abstractive summarization techniques. These research directions can lead to more robust, accurate, and reliable text summarization systems with broad applications in various fields.

10 Conclusion

To conclude, in this project, we explored how the BART model could be finetuned for the text summarization task, using a portion of the CNN Daily news dataset. By adding more and more data, the pre-trained BART model's language generation capabilities and fine-tuning it on domain-specific or task-specific datasets can improve, and in result we would effectively generate high-quality summaries. However, further research and development are still needed to advance the fine-tuning process for BART and other similar models. Exploring techniques to handle domain-specific challenges, optimizing hyperparameters, and improving training strategies can lead to even better performance and generalization.

References

[HKG+15] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Cnn/daily mail. https://huggingface.co/datasets/cnn_dailymail, 2015.

[LLG+20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. pages 270–278, July 2020.

[Moh21] Abhishek Mohan. Text summarization with nlp: Textrank vs seq2seq vs bart. *Towards Data Science*, June 2021.

[NLPnd] NLPlanet. Two minutes nlp: Learn the rouge metric by examples, n.d.