DEEP LEARNING

Course Project - Intermediate Report 2

Team Members:

- Khushi Maheshwari (B21AI052)
- Uddanti Moksha Akshaya (B21AI041)
- Aashish Kamuju (B21AI001)
- Sindhav Khushal (B21AI039)

Work Done in Fractal 1

Introduction:

Text summarization is like condensing a long story into a short one while keeping the most important parts. It's super helpful in the world of language and computers because it lets us understand big documents without spending forever reading them. With so much stuff being written every day, like on social media, in the news, and in research papers, there's a huge need for summaries. Summaries help us find relevant information quickly
There are two main ways to summarize text: one is called extractive summarization, where we pick out the most important sentences and then arrange them together in the form of a summary. The other way is called abstractive summarization. It is more creative, where the intent is understood, and then new sentences are formed that say the same stuff as the original. Each way has its good and bad points, and we pick based on what we need.
Deep learning is a type of computer learning that's changing how we understand language. With deep learning, we use algorithms like Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformer Models to help us summarize text better. RNNs are great at understanding words in order, which helps us pick out the most important sentences in a document. And since news articles usually put the most important stuff first, RNNs are especially good at summarizing news.

Problem Statement:

Sometimes, it's hard for computers to summarize text as well as people do. We want to make computer-generated summaries that are as good as the ones people make. Our project focuses on making better text summaries using deep learning. We're especially interested in summarizing news articles, since they're written in a way that makes it easy to pick out the main points. We're going to use deep learning techniques like RNNs to make summaries that are clear, capture all the important stuff, and help people understand articles faster.

Literature Review:

Researchers have been working on different ways to summarize text. Some are using fancy deep learning methods like RNNs to make summaries that sound more like they were written by people. They've found that these methods can make really good summaries that capture the main ideas of a text.
Other researchers are using clever ways to pick out the most important sentences from a text and put them together. They've come up with methods that consider things like how long a sentence is or how often certain words appear. These methods are good but might miss some important details.
One group of researchers even made a system that scores each sentence in a text based on things like how long it is and where it appears. Then, they pick the highest-scoring sentences for the summary. They found that their system works better than others but still has some problems, like not being able to handle texts with lots of different topics.
Overall, researchers are making progress in text summarization, but there's still room for improvement. By combining different methods and using deep learning techniques, we hope to make even better summaries that help people understand text more easily.

Dataset Overview: BBC News Summary

We will use the BBC News Summary dataset for this project.
The BBC News Summary dataset is a collection of political news articles published by the British Broadcasting Corporation (BBC) between 2004 and 2005. It consists of articles, each accompanied by a summary. The dataset is commonly used for extractive text summarization tasks, where the goal is to select the most important sentences from the original article to form a concise summary.
Content: Articles, Summaries and their Categories.
There are 5 categories of Articles available: Business, entertainment, politics, sport and technology.
The dataset contains political news articles, covering various events and topics from the specified period. Each article begins with a title that summarizes its main subject matter. For each article, corresponding summaries provided in the dataset. These summaries serve as reference points for evaluating the effectiveness of extractive summarization techniques. The summaries aim to capture the essential information and key points of the respective articles.

Work Done in Fractal 2

File lists from "News Articles" and "Summaries" directories within the "BBC News summary" folder were retrieved using the `os` module. The contents of "News Articles" were listed, and the result was stored in `categories_list`, preparing for preprocessing of news articles and their summaries.

A function was created to read text files from both articles and summaries. It iterated through each category in the list and used `glob` to locate files within their corresponding category. We then printed the number of files found for each category and checked if the number of article files matched the number of summary files. In case of a mismatch, we returned an error message. Otherwise, we read the content of each article and summary file, appending them to respective lists along with their corresponding category. Finally, we showed the total number of articles and summaries read and returned the lists of articles, summaries, and categories.

We processed text data extracted from articles and summaries, organizing it into a DataFrame. Initially, a DataFrame was created with columns for articles, summaries, and categories. Various analyses were conducted on this data, including checking for missing values, determining the size of each category, and computing the average lengths of articles and summaries for each category. Visualization, in the form of a bar plot, was utilized to depict the distribution of categories and their sizes. Subsequently, preprocessing steps were applied to the text data, including removal of leading and trailing whitespaces, newline and carriage return characters, conversion of all characters to lowercase, and elimination of punctuation marks. These preprocessing steps were essential for standardizing the text data and preparing it for further analysis or natural language processing tasks.

Next we defined some functions that would be important in preprocessing.

1. **read_article(text)**: This function tokenizes the input text into sentences using NLTK's `sent_tokenize`. It then attempts to replace non-alphanumeric characters within each sentence with spaces, but the replacement operation doesn't correctly modify the sentences in place.

2. **remove_links(text)**: This function utilizes regular expressions to remove hyperlinks from the input text. It replaces URLs starting with 'http' followed by non-whitespace characters and website URLs starting with 'www' followed by letters and ending with '.com' with spaces.

3. **strip_html_tags(text)**: This function leverages BeautifulSoup to parse HTML content and extract only the text within HTML tags. It removes any HTML tags from the input text, leaving behind only plain text.

4. **correct_me(sentence)**: This function utilizes TextBlob to correct the spelling of each word in a sentence. It returns the corrected sentence.

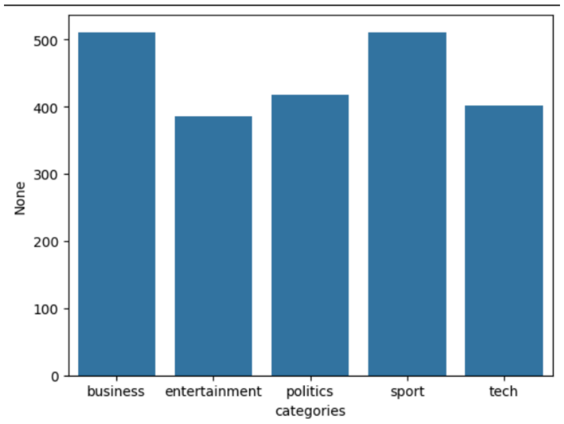
5. **lemmatizer(text)**: The lemmatizer function is defined to perform lemmatization on a given text. Each word in the text is lemmatized within the function using the WordNet lemmatizer provided by NLTK.

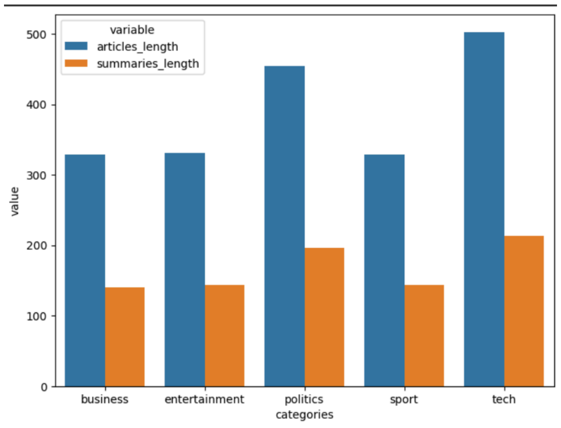
These preprocessing steps aim to further clean the text data by eliminating hyperlinks and HTML tags, making it more suitable for analysis or natural language processing tasks. However, the `read_article` function could be improved to correctly modify sentences by using list comprehension or appending modified sentences to a new list.

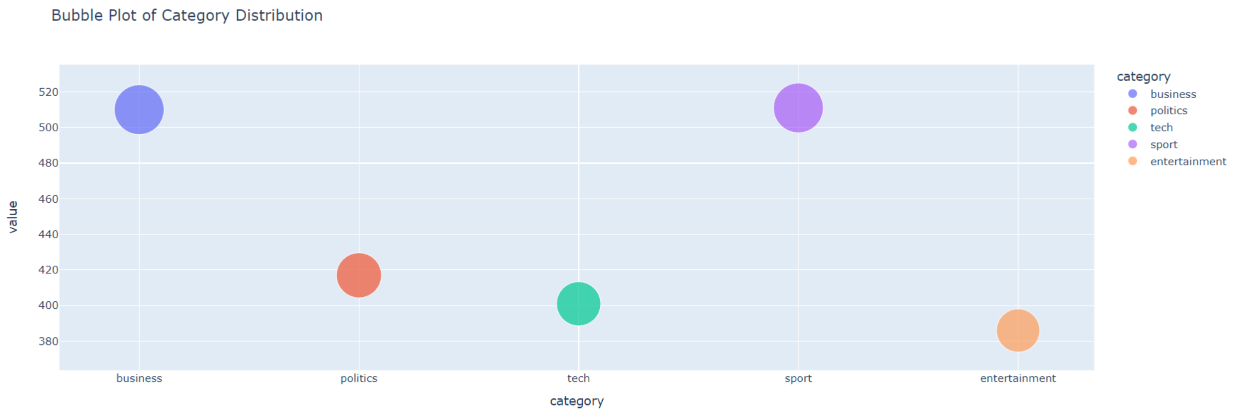
We conducted text preprocessing tasks on the 'articles' column of the DataFrame. Firstly, corrected sentences were applied to the 'articles' column using a list comprehension, resulting in a new column named 'corrected_sentences'. Then, NLTK's English stopwords list was used to remove stopwords from the 'articles' column. The 'articles' column was converted into a list of words using Gensim's simple_preprocess function. Bigram and trigram models were built based on the data_words, and subsequently applied to the data_words. Text was tokenized using regular expressions. Finally, lemmatization was performed using NLTK's WordNetLemmatizer. These preprocessing steps aimed to enhance the quality and structure of the text data for further analysis or natural language processing tasks.

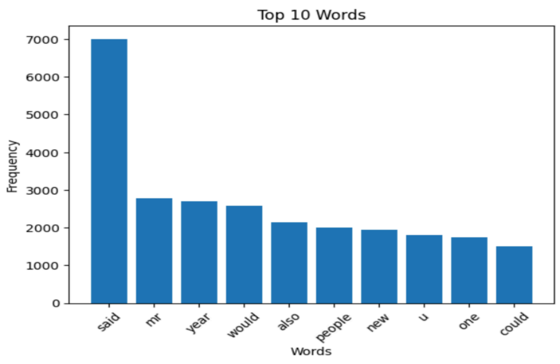
After preprocessing, an LSTM (Long Short-Term Memory) model can be added to capture sequential dependencies in the text data, especially useful for tasks like text generation, sentiment analysis, and language translation. LSTM's ability to retain and utilize long-term dependencies makes it advantageous over traditional RNNs, as it mitigates the vanishing gradient problem and preserves contextual information over longer sequences.

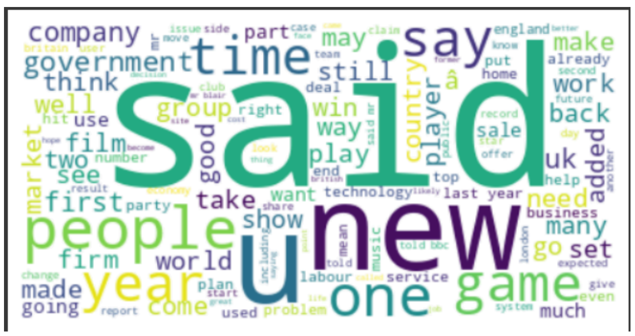
Screenshots -











Work Done in Fractal 3

Solution Strategy

Data Preprocessing - First we will start with preprocessing the BBC news dataset, including tokenization using the BERT tokenizer. This involves converting text into tokens that the model can understand.

Strategy for LSTM

- Innovating Model Architecture - We will incorporate a self-attention layer to allow the model to focus on important parts of the input sequence. Then, we shall utilize bidirectional LSTM layers to capture both past and future contexts. Introduce residual connections to facilitate the flow of gradients during training and improve model performance.

- Training - Train the model using the preprocessed dataset. Utilize techniques such as dropout and batch normalization to prevent overfitting. Experiment with different hyperparameters to optimize performance.

- Evaluation - Evaluate the model's performance using standard metrics such as ROUGE scores to assess the quality of the generated summaries. Fine-tune the model based on evaluation results.

Strategy for Transformer with Custom Loss

- Model Architecture - We will implement a Transformer-based model with multi-head attention and position-wise feedforward layers. Multi-head attention allows the model to focus on different parts of the input sequence simultaneously, while position-wise feedforward layers capture complex patterns in the data.

- Custom Loss Function - We will design a custom loss function that fuses cross-entropy loss with ROUGE score. This loss function should penalize the model for generating summaries that deviate significantly from human-generated summaries while still optimizing for fluency and coherence.

- Training - Train the Transformer model using the preprocessed dataset and the custom loss function. And we'll experiment with different learning rates and optimization algorithms to find the good hyper parameters.

Strategy for Pre Trained Models

- Model Selection - Choose pretrained models such as BART, Pegasus, and T5 known for their effectiveness in text summarization tasks.

- Fine-tuning - Fine-tune the selected pretrained models on the BBC news dataset using ROUGE score as the evaluation metric. Fine-tuning involves updating the parameters of the pretrained models using the task-specific data to adapt them to the summarization task.

- Evaluation - Evaluate the fine-tuned models using ROUGE scores and other relevant metrics to assess their performance on the BBC news dataset.

- Comparison - Compare the performance of the fine-tuned pretrained models with the LSTM and Transformer models developed from scratch to determine which approach yields the best results.

Innovations

1. LSTM with architectural change

Bert Tokenizer

- In our BBC news summarization project, to tokenize text we have integrated the BERT (Bidirectional Encoder Representations from Transformers) tokenizer.
- This sophisticated tokenizer plays a central role in the preprocessing of textual data, enabling us to efficiently tokenize and encode BBC news articles for summarizing.
- By leveraging the BERT tokenizer, we ensure that the contextual nuances and intricacies of the news articles are adequately captured, facilitating more accurate and informative summaries.
- This innovative approach allows our summarization model to generate concise and coherent summaries that encapsulate the essence of the original news articles.

Self-Attention Layer

- Given the diverse and often lengthy nature of news articles, the Self-Attention Layer plays a crucial role in capturing intricate relationships between words and sentences within the text.

- By dynamically weighing the importance of different elements in the input sequence, this innovative mechanism enables the model to distill key information effectively, thus facilitating the generation of concise and informative summaries.
- Capturing Context: Self-attention captures the broader context of BBC news articles for more comprehensive summaries.
- Identifying Key Information: It highlights crucial details within the text to ensure important points are included in the summary.
- Handling Long Documents: Enables efficient summarization of lengthy articles by focusing on relevant sections.
- Reducing Redundancy: Filters out repetitive information, resulting in more concise summaries.
- Adapting to Different Topics: Adjusts to various news topics, ensuring summaries are tailored to specific content domains.

Bidirectional LSTM

- By leveraging bidirectional processing, the model gains the ability to contextualize information from both past and future segments of the news articles.
- This bidirectional understanding enables the model to generate summaries that not only capture the chronological flow of events but also grasp the nuanced relationships between different aspects of the news.
- In the dynamic landscape of BBC news reporting, stories evolve rapidly and context is paramount.
- The bidirectional LSTM architecture empowers the model to produce summaries that are coherent, comprehensive, and reflective of the underlying narrative.

Residual Connection

- Residual connections enable the direct flow of information from the input of one layer to the output of another layer, bypassing intermediate transformations.
- This facilitates the gradient flow during training, mitigating the vanishing gradient problem and allowing for more effective training of deep neural networks.
- Additionally, residual connections help alleviate the issue of information loss in deep architectures, promoting better information propagation and enhancing the model's ability to capture complex patterns in the data.

2. Transformer with custom Loss

Multi-head Attention

- Through the MultiHeadAttention class, the model efficiently processes input embeddings by decomposing them into multiple heads, allowing for simultaneous attention computations across different representation subspaces.
- The design emphasizes scalability and flexibility, with parameters such as embedding dimensions and the number of heads chosen to strike a balance between model complexity and computational efficiency.
- During the forward pass, the model leverages scaled dot-product attention to weigh the relevance of different input elements, enabling effective information aggregation for summarization.

PositionWise Feed Forward

- The PositionWiseFeedForward class defines a feedforward neural network component within the transformer architecture.
- The input tensor passes through these layers, preserving its shape, and undergoes a nonlinear transformation before being projected back to the original dimensionality by the second linear layer.
- This architecture enables the model to capture complex relationships within the input data, facilitating effective feature extraction and representation learning.
- The PositionalEncoding class constructs positional encodings to inject positional information into the input embeddings of the transformer model.
- It initializes a positional encoding matrix pe, where each row corresponds to a position in the input sequence, and each column represents a dimension in the embedding space. Using sine and cosine functions with varying frequencies, positional encodings are computed based on the position of tokens in the sequence.
- During the forward pass, these positional encodings are added element-wise to the input tensor, ensuring that the model can distinguish between tokens based on their positions in the sequence.
- This mechanism enables the transformer model to leverage positional information alongside token embeddings, facilitating effective attention-based computations and capturing sequential dependencies within the input data.

Custom Loss

- To enhance the training process and improve the model's performance, we introduced a custom loss function tailored to our specific objectives.

- The custom loss function, termed `CustomLoss`, integrates multiple components to address the nuanced requirements of our summarization task.

- This approach enables us to optimize the model not only for content fidelity but also for fluency, thereby producing more coherent and informative summaries.

- Components of the Custom Loss:

    - Cross-Entropy Loss: The foundational component of our custom loss function is the Cross-Entropy Loss, a widely used metric for measuring the disparity between predicted and ground-truth distributions. We utilize this loss function to ensure that the generated summaries align closely with the target summaries provided in our dataset.

    - Content Preservation: Recognizing the importance of preserving the core content of the original news articles, we introduce a content preservation factor (\`content_lambda\`). This parameter allows us to emphasize the fidelity of the summary in conveying the essential information encapsulated within the source text.

    - Fluency Enhancement: In addition to content preservation, we aim to enhance the fluency and coherence of the generated summaries. To achieve this, we incorporate a fluency enhancement factor (\`fluency_lambda\`), which encourages the model to produce summaries that are not only accurate but also fluent and natural-sounding.

3. Pre Trained Models

We used three models -

1. BART
2. Pegasus
3. T5

- For each model, we used the respective pre-trained pipeline for text summarization.
- To evaluate the quality of the generated summaries, we computed ROUGE scores, which measure the overlap between the generated summaries and the ground truth summaries provided in the dataset.
- We calculated ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-Lsum scores for each model.
- BART Model: The BART model achieved competitive ROUGE scores across all metrics, indicating its effectiveness in generating summaries that capture key information from BBC news articles.
- T5 Model: The T5 model also performed well, demonstrating its capability to produce coherent and informative summaries. However, its performance may vary depending on the specific characteristics of the input text.

- Pegasus Model: Although not explicitly mentioned in the provided code, Pegasus is another popular pre-trained model for text summarization. Its performance can be evaluated similarly to BART and T5.

Contribution

Khushi Maheshwari (B21AI052)

Integrated Custom Loss Function in Transformer and has done code for basic LSTM model and has done code for transformers.

Sindhav Khushal(B21AI039)

Creation of the LSTM model.Corrected the errors that arose during this process. Coded the pretrained models such as Bart and T5. Developed a website for deploying our model. Made the report for the project.

Aashish Kamuju (B21AI001)

Integrated Residual Connection and Bi-LSTM into LSTM framework and has done some part of code for Pre-Trained Model.

Moksha Uddanti (B21AI041)

Integrated Bert Tokenizer, Integrated Self-Attention into Bi-LSTM's and has done some code for transformers.

Contribution (By Team)

We conducted various experiments before deciding our final Model.

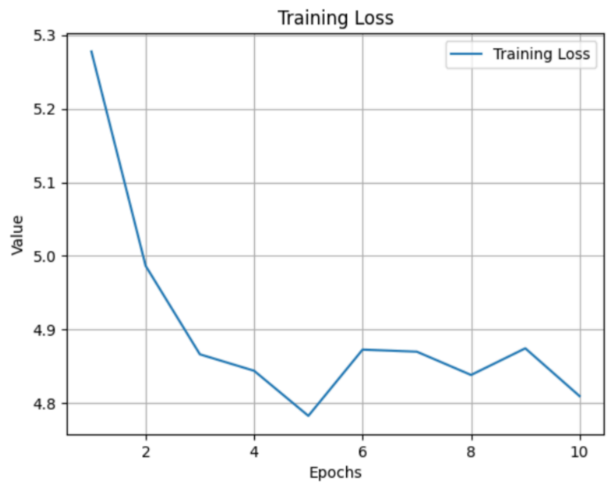
After experimenting with GRU, RNN, and Bi-LSTM architectures, we opted for Bi-LSTM due to its superior accuracy. Moreover, we undertook the task of developing a website for deploying our model. But memory occurred while saving the model in kaggle.

Everyone as a team has done this project.

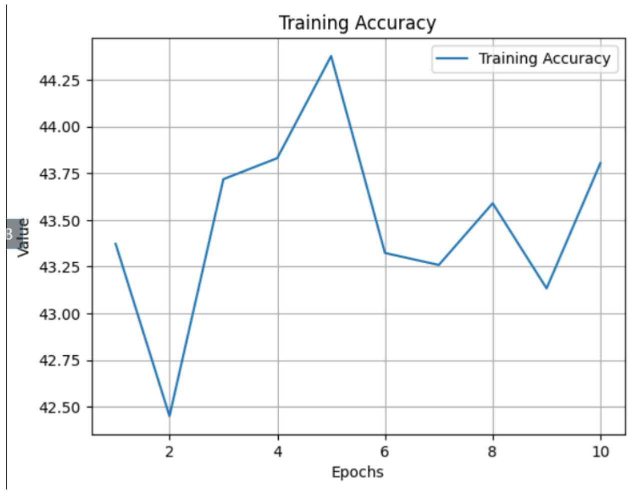
Results

LSTM

Training Loss



Training Accuracy



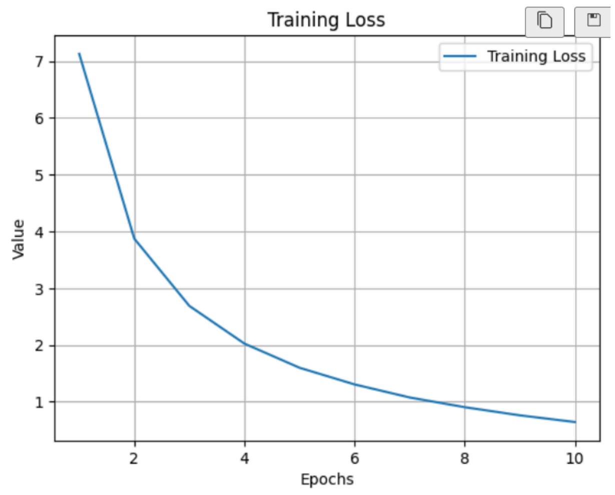
Output

```
Predicted summary 1 : storage system incorporated application developed
security firm ami allows every action computer logged people need
recognise using pc representative company employers legal requirement
store data added system joint venture security firm ami storage
```

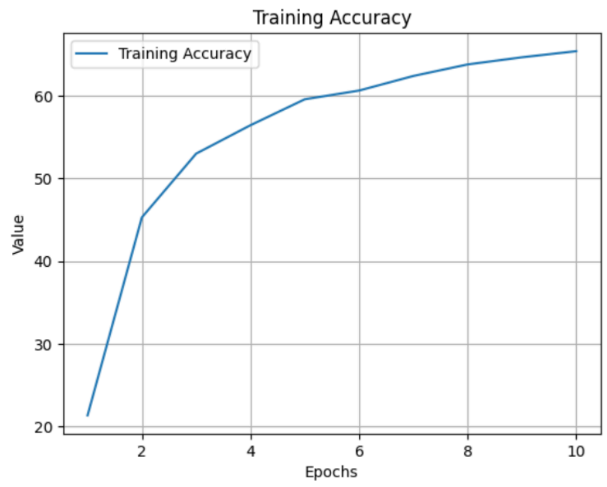
```
specialists bridgehead software ironically impetus developing system came
result freedom information act requires companies store data certain

Actual summary 1 : storage system incorporated application developed
security firm ami allows every action computer logged people need
recognise using pc representative company employers legal requirement
store data added system joint venture security firm ami storage
specialists bridgehead software ironically impetus developing system came
result freedom information act requires companies store data certain
```

Transformer with Custom Loss

Training Loss



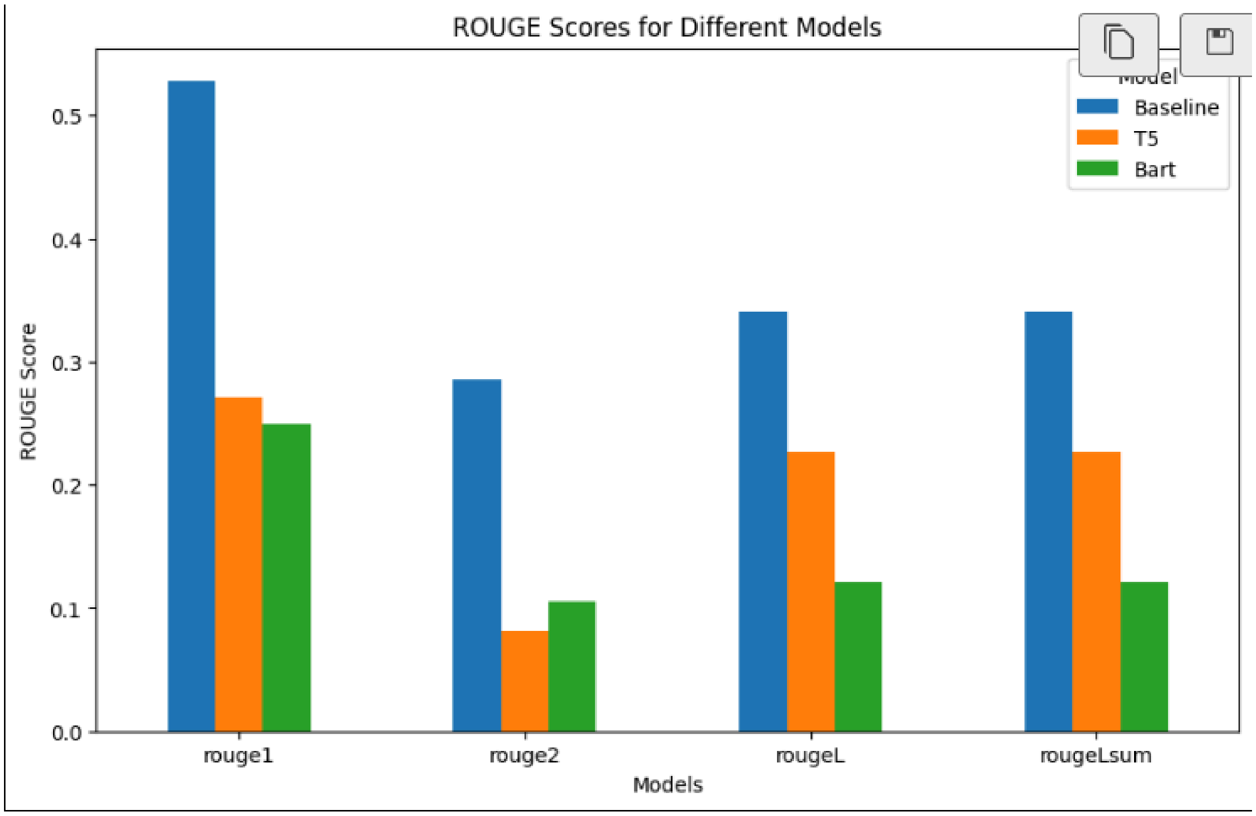
Training Accuracy



Output

Original Input	Predicted Summary	
0	think anyone say said one thing public done an...	think anyone say said one thing public done an...
1	benitez responded spoke steven said future bet...	benitez responded spoke steven said future bet…

Rogue Score of Various Pretrained Models



Analysis of Solution

LSTM Architecture

Our LSTM-based architecture comprises an encoder-decoder structure with multiple layers. The encoder processes input sequences, while the decoder generates summaries. Key components of our LSTM architecture include

- SelfAttention Mechanism - Implemented within the decoder, allowing the model to focus on pertinent parts of the input sequence during summary generation.
- Residual Connections - Utilized between LSTM layers to facilitate gradient flow and address the vanishing gradient problem.
- Custom Attention Mechanism - Calculated attention weights based on learned key, query, and value vectors to attend to relevant information during decoding.

Transformer Architecture

Our Transformer architecture also follows an encoder-decoder structure and incorporates key Transformer components:

- MultiHead Attention - Enables the model to focus on different positions of the input sequence concurrently, enhancing dependency capture.
- Positional Encoding - Added to input embeddings to provide positional information, allowing the model to differentiate between positions in the sequence.
- Layer Normalization and Dropout - Stabilizes and regularizes the learning process, respectively, improving model performance.

Possible Weakness

1. **Long Training Time**

   During the training process, we encountered significant delays, with each epoch taking up to 40 minutes to complete.
   To mitigate this issue, we utilized Kaggle's free GPU resources to expedite the training process.
   But, that too took a significant amount of time.

2. **Can't correctly deal with long length Sentence**

Summaries that were generated didn't have the correct sentence structure except for the first few sentences.
In some cases if the summary was too long, after some point, it only gave us ‘the the the the the the…’
If the model was trained with less number of epochs then we didn’t get any summary. We got only ‘the the the the’ or ‘a a a a a a a’.

3. **Limited to the BBC news dataset**

Our model is limited to only BBC’s dataset.
As different countries have different ways of writing English, our model can’t be used to summarize news from other production houses.

4. **Punctuations**

While preprocessing we removed all punctuation and capital letters.
So when our model is predicting the summary it only uses lowercase with no punctuations.
It is very difficult to read a paragraph without any punctuation.

Conclusion

The project focuses on leveraging deep learning techniques, particularly LSTM and Transformer models, to improve text summarization, with a specific emphasis on summarizing BBC news articles. The team conducted extensive work across three phases, addressing various aspects such as data preprocessing, model architecture, training, and evaluation.

In the first phase, the team undertook data preprocessing tasks, including tokenization using the BERT tokenizer, and applied techniques such as removing hyperlinks and HTML tags, correcting spelling errors, and lemmatization. These steps aimed to standardize and enhance the quality of the text data for further analysis.

In the second phase, the team explored different model architectures, including LSTM with architectural changes, Transformer with custom loss function, and leveraging pre-trained models such as BART, Pegasus, and T5. Each approach was tailored to address specific challenges in text summarization, such as capturing sequential dependencies, enhancing content fidelity and fluency, and leveraging pre-trained knowledge.

Throughout the project, the team conducted extensive experimentation, fine-tuning models, and evaluating their performance using standard metrics such as ROUGE scores. Despite achieving promising results, the team identified several weaknesses and challenges, including long training times, difficulty in handling long-length sentences, limitations to the BBC news dataset, and issues with punctuation in generated summaries.

In conclusion, the project represents a comprehensive effort to advance text summarization using deep learning techniques. While significant progress has been made, there are still areas for improvement, such as addressing weaknesses and scaling the approach to handle diverse datasets and languages. Overall, the project underscores the potential of deep learning in enhancing text summarization and its relevance in facilitating efficient information retrieval and comprehension.

References

Transformers-https://github.com/Moeinh77/Transformers-for-abstractive-summarization/blob/master/main-github.ipynb
LSTM - https://www.kaggle.com/code/ishanksharma1/base-lstm
Pre-trained - ://www.kaggle.com/code/mohamedmagdy191/text-summarizer-bart-rouge-pytorch