NLP Assignment - 2

Text Summarization using RNNs

Name: Chepuri Naga Venkata Varsha
RollNo: 21CS30014

1] Data Exploration with CNN/Daily Mail Dataset

Overview:

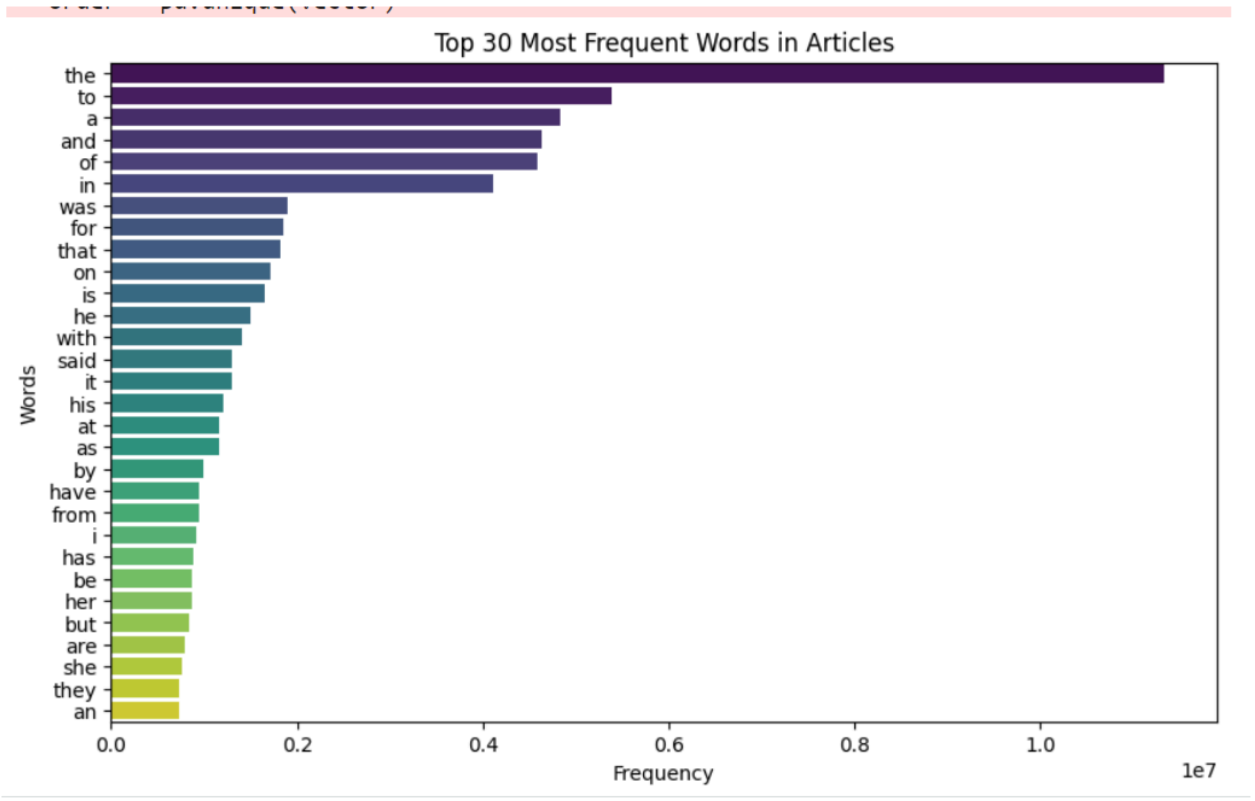
- The task involves conducting **Exploratory Data Analysis (EDA)** to gain insights into the dataset and prepare it for model building.
- Analyzing the dataset's size and components, including articles and corresponding summaries.
    - Dataset({
    -     features: ['article', 'highlights', 'id'],
    -     num_rows: 287113
    - })
    - Dataset({
    -     features: ['article', 'highlights', 'id'],
    -     num_rows: 13368
    - })
    - Dataset({
    -     features: ['article', 'highlights', 'id'],
    -     num_rows: 11490
    - })
    - {'article': Value(dtype='string', id=None), 'highlights': Value(dtype='string', id=None), 'id': Value(dtype='string', id=None)}
- Preprocessing the text by lowercasing, removing special characters, and tokenizing the articles and summaries.
- Identifying the most frequently occurring words(top 30) in the dataset using Counter class by getting the frequency of each word in the articles and summaries.
- For **sentence length analysis**, we measured the number of words (tokens) in each article and summary to understand their distribution. A subset of 500 articles and summaries was selected for the analysis to reduce computation time and provide a representative sample.
- KDE (Kernel Density Estimation) was added to visualize the probability density of sentence lengths.

Word Frequency Analysis:

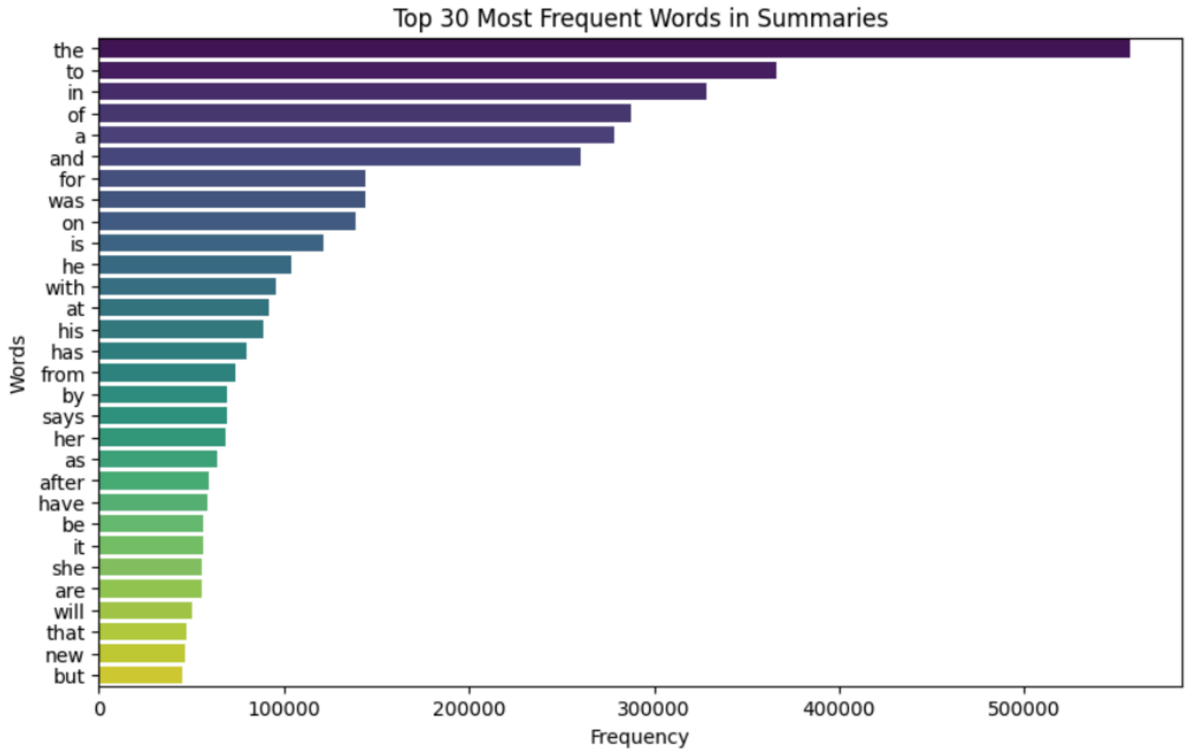
- By this we get an understanding of which words are most commonly used in both the articles and summaries. Common stopwords such as "the", "to", "a", etc., are expected to dominate the list.
- High frequency of common stopwords suggests the need for **stopword removal** during model training.

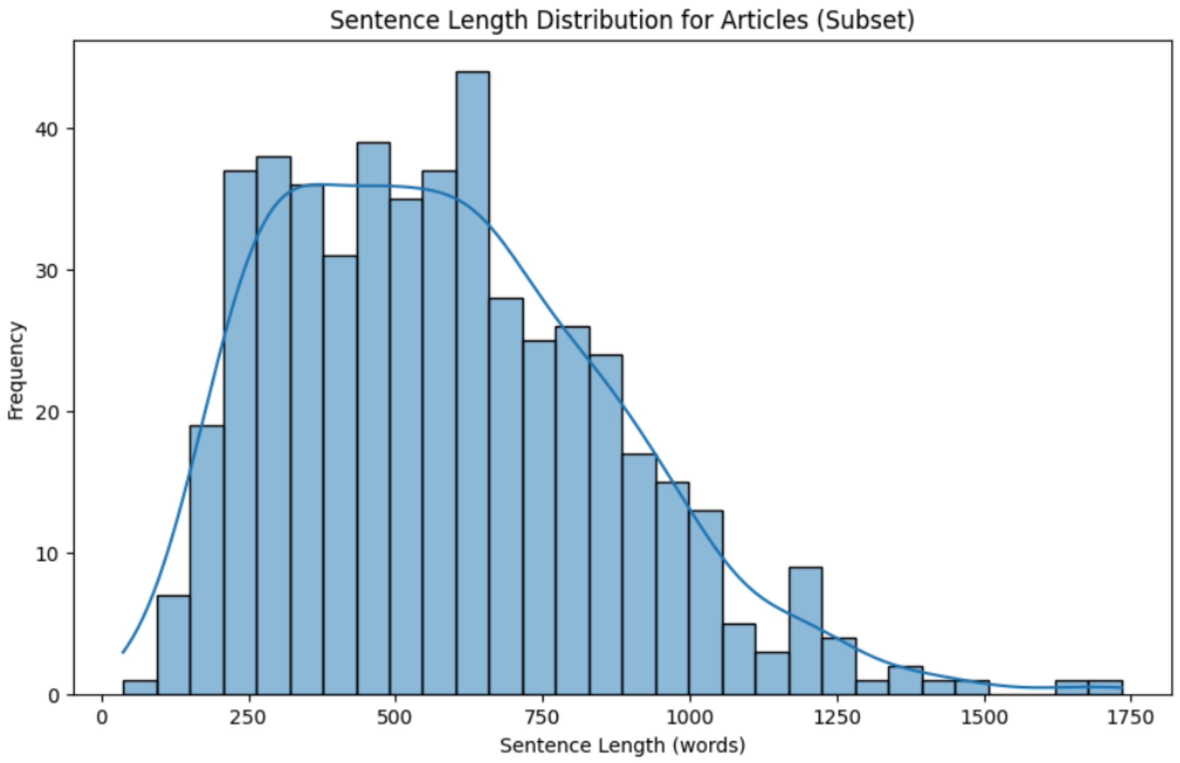
Sentence Length Analysis:

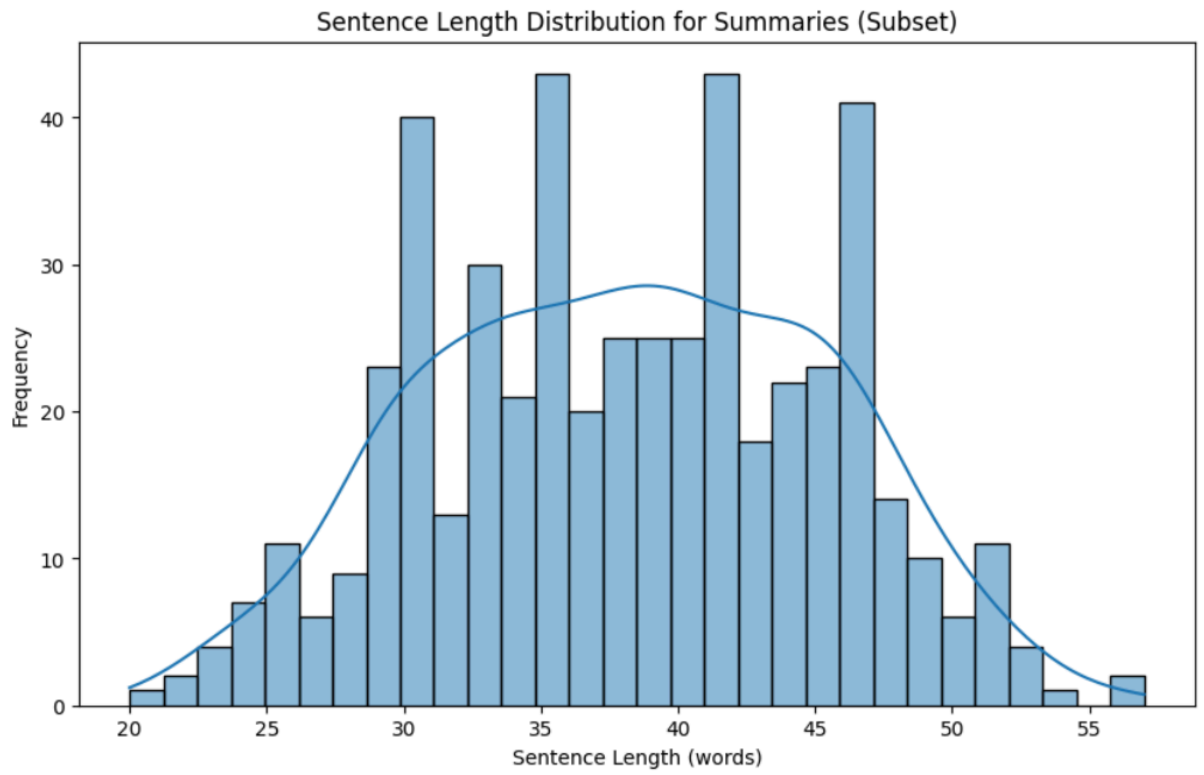
- By analyzing the sentence lengths, we observed differences between articles and summaries. Articles are generally longer, while summaries tend to be more concise.
- Handling of longer sequences, Padding or truncation strategies for text processing, also we need have balance of article and summary length



```
order = pd.unique(vector)
```







2] Building and Training a Seq2Seq Model for Text Summarization

Introduction:

- Text summarization aims to condense a large body of text into a shorter version while preserving essential information and meaning. Here, we implemented a Sequence-to-Sequence (Seq2Seq) model using Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units to summarize articles from the CNN/Daily Mail dataset. The model's performance was evaluated using ROUGE scores, specifically ROUGE-2 and ROUGE-L.

Data-Preprocessing:

- **Text Cleaning**: The articles and summaries were cleaned by converting text to lowercase, removing punctuation, numbers, and stop words, and eliminating any irrelevant content, such as text inside parentheses. [Tried with removing stop words also but model didn't generalized well]
- Identified the maximum article length (for the encoder) and maximum summary length (for the decoder) in order to define the architecture of the LSTM model.

These lengths determine the number of LSTM units (or hidden states) needed in both the encoder and decoder components of the Seq2Seq architecture.

- **Tokenization and Vocabulary Creation**: Later Tokenized the cleaned text to build a vocabulary of the most common words. A vocabulary size of 16,000 was set, including special tokens like `<pad>`, `<start>`, `<end>`, and `<unknown>`.
    - Vocabulary size: 16000
    - Sample tokens from vocabulary: ['<pad>', '<start>', '<end>', '<unknown>', 'the', 'to', 'a', 'and', 'of', 'in']
- **Sequence Conversion**: The cleaned articles and summaries were converted into sequences of indices based on the vocabulary, allowing them to be input into the LSTM model.
    - First 20 Word to Index Mappings:
    - <pad>: 0
    - <start>: 1
    - <end>: 2
    - <unknown>: 3
    - the: 4
    - to: 5
    - a: 6
    - and: 7
    - of: 8
    - in: 9
    - was: 10
    - for: 11
    - that: 12
    - on: 13
    - is: 14
    - he: 15
    - it: 16
    - with: 17
    - said: 18
    - his: 19
- **Padding**:Sequences were padded to ensure uniform input sizes. The maximum article length was determined to be 2097 tokens, and the maximum summary length was 1194 tokens.
    - Also padded <start> at the start of the train_summaries for training the decoder.

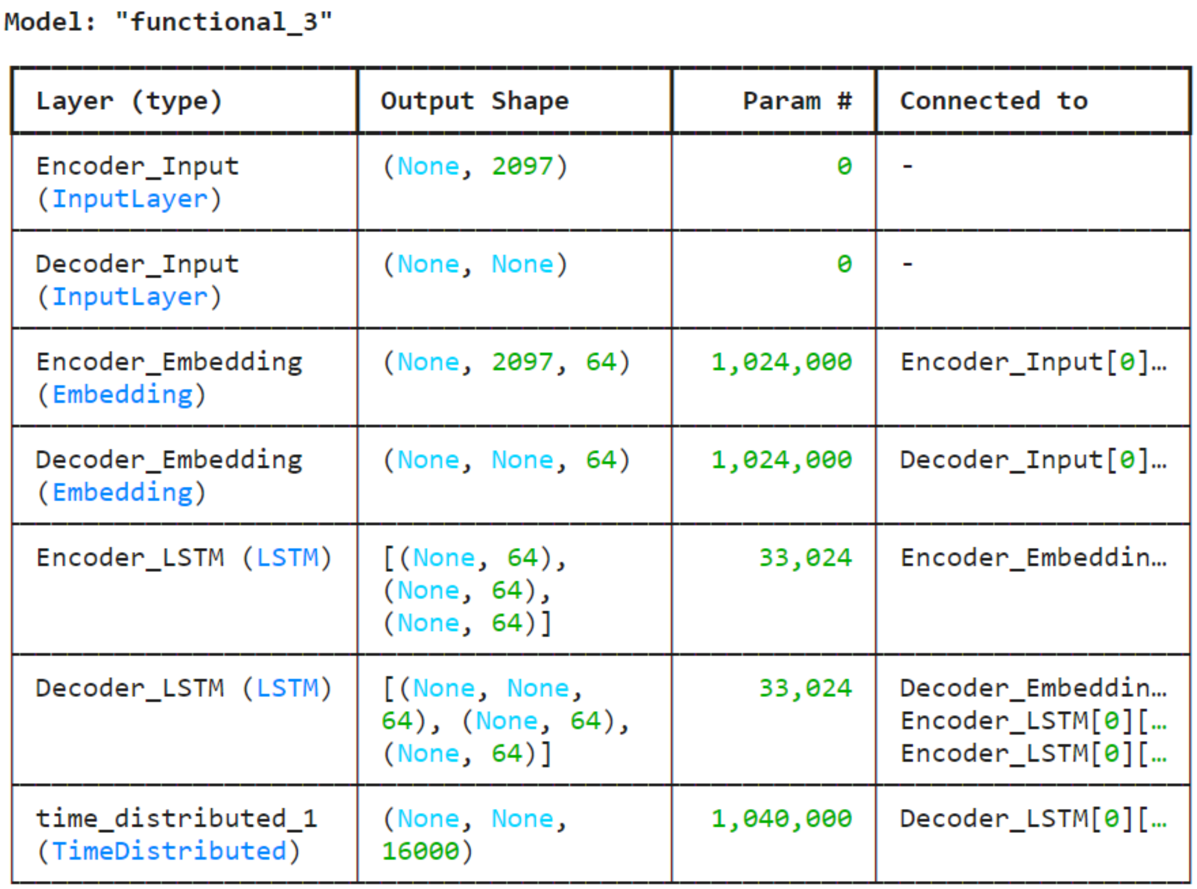
Model Architecture Overview

The Seq2Seq model consists of an encoder-decoder architecture implemented using LSTM layers. Here's a breakdown of its components:

1. **Encoder**:
   - **Input Layer**: Takes articles as input with a shape defined by `max_article_length`.
   - **Embedding Layer**: Converts input tokens into dense vectors of fixed size (`embedding_dim`). This layer is trainable, meaning the embeddings will be optimized during training.
   - **LSTM Layer**: Processes the embedded sequences and outputs the final hidden and cell states, which will be used to initialize the decoder.
2. **Decoder**:
   - **Input Layer**: Accepts the summary inputs with a variable length (shape `(None,)`).
   - **Embedding Layer**: Similar to the encoder, this layer converts summary tokens into dense vectors.
   - **LSTM Layer**: The decoder's LSTM processes the embeddings while receiving the final states from the encoder as its initial states.
   - **TimeDistributed Layer**: A dense layer applied to each time step of the LSTM outputs, generating a probability distribution over the vocabulary for each timestep.
3. **Model Compilation**:
   - The model is compiled using the **Adam optimizer** with a learning rate of `1e-4`, and it employs **sparse categorical cross entropy** as the loss function since the target output is in a sparse format.
4. **Training Process**:
   - The model is trained using the encoder-decoder setup where the decoder is provided with the previous target output during training (teacher forcing).
5. **Hyperparameters**:
   - **Latent Dimension (latent_dim)**: Set to **64**, this hyperparameter defines the size of the hidden state in the LSTM units. A larger latent dimension can capture more complex patterns and relationships within the data, but it may also increase the risk of overfitting, especially if the training data is limited.(checked with higher dimensions but it is taking huge time)
   - **Embedding Dimension (embedding_dim)**: Also set to **64**, this defines the size of the embedding vector for each token in the vocabulary. The embedding layer transforms the input tokens into dense vectors of fixed size, allowing the model to learn semantic relationships between words.
   - **Vocabulary Sizes (article_vocab_size and summary_vocab_size)**: These sizes are derived from the vocabulary of the articles and

summaries, respectively. It's crucial that the vocabulary size is sufficient to cover all unique tokens in the training data. Using the same vocabulary size for both the encoder and decoder can simplify the architecture and improve performance by allowing shared embeddings.

- **Dropout Rates in Decoder**: The dropout rates for the LSTM in the decoder are set to **0.4** (for input dropout) and **0.2** (for recurrent dropout). Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of the inputs to zero during training. Higher dropout rates can enhance generalization.
- **Learning Rate**: The learning rate is specified as **1e-4** in the Adam optimizer. This parameter controls how much to adjust the model weights during training in response to the computed gradients. A smaller learning rate can lead to more stable training and convergence but may require more epochs, while a larger learning rate can speed up training but may overshoot the optimal solution.

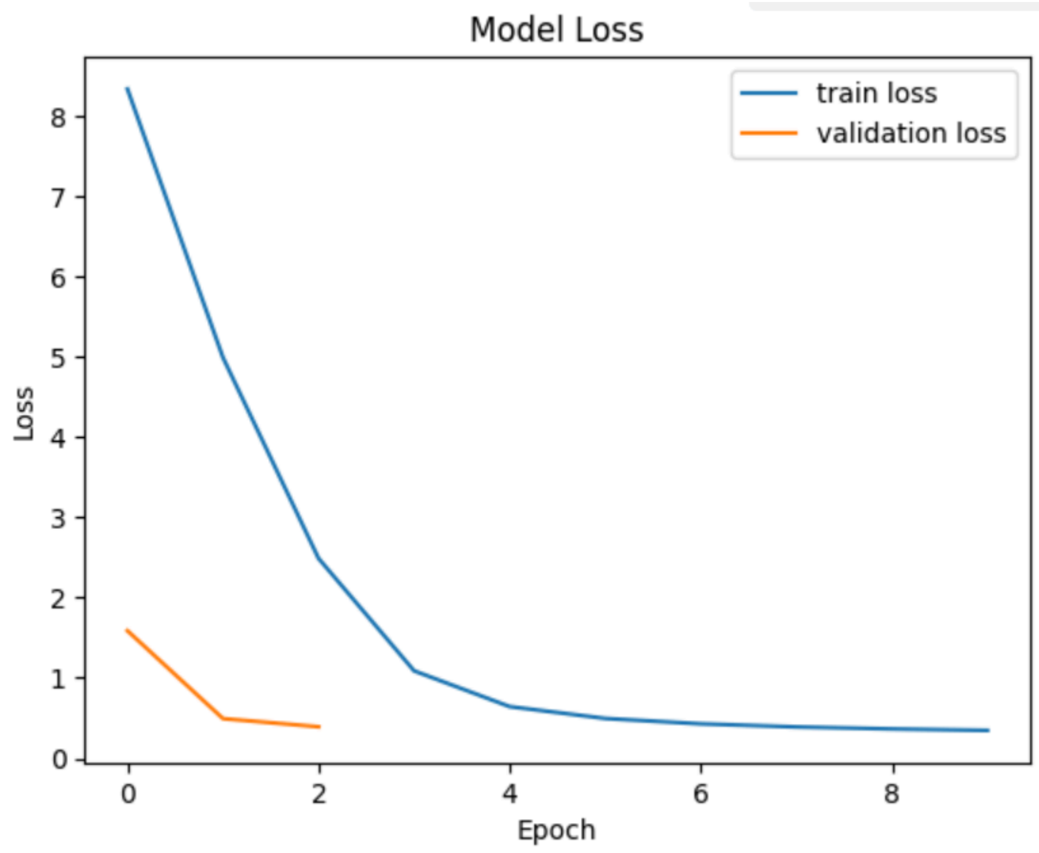


```
Model: "functional_3"

┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ Encoder_Input       │ (None, 2097)      │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ Decoder_Input       │ (None, None)      │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ Encoder_Embedding   │ (None, 2097, 64)  │  1,024,000 │ Encoder_Input[0]… │
│ (Embedding)         │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ Decoder_Embedding   │ (None, None, 64)  │  1,024,000 │ Decoder_Input[0]… │
│ (Embedding)         │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ Encoder_LSTM (LSTM) │ [(None, 64),      │     33,024 │ Encoder_Embeddin… │
│                     │ (None, 64),       │            │                   │
│                     │ (None, 64)]       │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ Decoder_LSTM (LSTM) │ [(None, None,     │     33,024 │ Decoder_Embeddin… │
│                     │ 64), (None, 64),  │            │ Encoder_LSTM[0][… │
│                     │ (None, 64)]       │            │ Encoder_LSTM[0][… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed_1  │ (None, None,      │  1,040,000 │ Decoder_LSTM[0][… │
│ (TimeDistributed)   │ 16000)            │            │                   │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘
```

Important Features of the Implementation

- **Teacher Forcing**: During training, the decoder uses the true output sequences from the previous time step instead of its own previous predictions. This helps the model learn faster and more accurately by providing the correct context at each step, which mitigates the problem of exposure bias where the model learns based on its predictions rather than the actual target outputs.
- **Dropout and Recurrent Dropout**: The decoder LSTM includes dropout (0.4) and recurrent dropout (0.2) to prevent overfitting, ensuring the model generalizes better on unseen data.
- **Batch Size and Epochs**: The model is trained with a batch size of 16 for 10 epochs, allowing for effective training on a reasonable amount of data at each iteration.
- **Validation**: The model employs a validation set to monitor performance during training, using early stopping to prevent overfitting. The training process halts if the loss does not improve for 3 consecutive epochs, ensuring efficient use of resources.
- **Model Visualization**: The training loss and validation loss are plotted to visualize the model's performance over epochs, helping in diagnosing issues like overfitting or underfitting.



ROUGE-2 Score: 0.2780

ROUGE-L Score: 0.3891

Tasks Explored

1. **Pre-trained Word2Vec Embeddings**:
    - Explored the use of pre-trained Word2Vec embeddings to enhance the model's semantic understanding.
    - Resulted in poor performance metrics compared to simpler tokenizations and using Embedding layer.
2. **Initial Training on 5% of the Dataset**:
    - Started training the model using 5% of the dataset to assess capabilities and performance.took 8hr for training of 10 epochs
3. **Reduction to 1% of the Dataset**:

    - Due to session completion in between model not saved, later the training was later adjusted to 1% of the dataset.
    - This change impacted the training dynamics and overall model performance.

2] Evaluate the Model on the Wikipedia Summary Dataset

Dataset Loading and Preprocessing:

- Utilized the Hugging Face `load_dataset` function to fetch the Wikipedia summary dataset, specifically targeting the 'jordiclive/wikipedia-summary-dataset'.
- Implemented a cleaning function (`text_cleaner`) to preprocess the articles and summaries, ensuring consistent and high-quality input data.

Sequence Conversion:

- Converted the cleaned articles and summaries into sequences of indices using a mapping dictionary (`word_to_index`). This transformation facilitates the integration of text data into the model.

Padding:

- Applied padding to both the article and summary sequences to ensure uniform length across all samples, using `pad_sequences`. This step is crucial for batch processing in LSTM models and avoids shape mismatches during training and evaluation.

Model Evaluation:

- Created a dedicated evaluation function (`evaluate_model_on_wikipedia`) to assess the model's performance on the test set derived from the Wikipedia dataset.
- Generated summaries from the model and compared them with reference summaries using ROUGE metrics (ROUGE-2 and ROUGE-L). These metrics are standard for evaluating the quality of generated text, particularly in summarization tasks.

Performance Metrics:

- Calculated and printed the ROUGE scores, providing insights into the model's summarization capabilities and effectiveness on real-world data.

ROUGE-2 Score: 0.0190

ROUGE-L Score: 0.0350