Zohair Hashmi *UIN# 668913771*

CS 512: Advanced Machine Learning - Project Progress Update
Text Summarization using Seq2Seq Model

1. Introduction

This project focuses on automated text summarization using a deep learning approach based on sequence-to-sequence (Seq2Seq) models with attention mechanisms. By collecting and preprocessing a large dataset of articles and their summaries, we developed a Seq2Seq model that generates summaries of the input text. We evaluate the model's performance using standard metrics like ROUGE and BLEU. The project has applications in diverse domains, such as news, academic papers, and social media content.

2. Dataset

The CNN/DailyMail Dataset is a widely used dataset consisting of over 300,000 news articles in English, which have been authored by journalists from two prominent media outlets, CNN and the Daily Mail. The dataset was downloaded from Kaggle, a popular platform for hosting data science and machine learning resources. The dataset is particularly valuable for natural language processing tasks such as text summarization, as it contains both article texts and summaries.

The dataset has two primary columns, one for the article text and another for the summary. These articles are written on a variety of topics, including politics, sports, and entertainment. However, due to computational limitations, we have chosen to utilize only a subset of this data, consisting of 70,000 articles and their corresponding summaries, for training our seq2seq model.

3. Data Processing

Data Cleaning: Natural language processing involves analyzing and understanding human language by computers. Data cleaning is an important step in the preprocessing of text data as it eliminates unwanted noise to ensure that the text is in a suitable format for analysis. This study's data cleaning process involves expanding contracted words to enhance model accuracy by improving word frequency and co-occurrence. Furthermore, this process includes removing unwanted characters, converting text to lowercase, and eliminating stop words to enhance model accuracy by reducing the dimensionality of the data and eliminating noise. In general, data cleaning is a critical step that plays a crucial role in ensuring accurate and meaningful results in natural language processing.

Tokenization: Tokenization is an essential aspect of natural language processing (NLP) that involves transforming textual data into a numerical format that can be analyzed using machine learning algorithms. In this process, the stopwords corpus from the NLTK library is utilized to eliminate frequently occurring words that do not contribute significantly to the overall meaning of the text. Moreover, the calculation of the maximum number of features aids in limiting the vocabulary size and increasing computational efficiency. The Tokenizer class is employed to convert the text into an integer sequence, while the pad_sequences method ensures consistency in modeling by setting the same length for each sequence. The resulting sequences are then stored for further analysis.

Pre-trained embeddings: Pre-trained embeddings play a crucial role in enhancing the performance of natural language processing (NLP) tasks, particularly in text summarization. In our research, pre-trained embeddings are utilized to convert each word in the text into a numerical vector, which is then fed into a machine learning model. We make use of GloVe embeddings, a popular choice in NLP, which are loaded from a text file and stored in a dictionary structure. Each key in the dictionary represents a word, and the corresponding value is a vector that represents that word.

Employing pre-trained embeddings improves the accuracy of the model's word representation, as opposed to training the embeddings from scratch. Pre-trained embeddings are trained on extensive data and can capture intricate relationships between words that may be difficult to learn from a smaller dataset. Furthermore, the usage of pre-trained embeddings reduces training time and enhances the model's generalization performance.

4 Model Training & Inference

The code implements a sequence-to-sequence model for text summarization using an encoder-decoder architecture with pre-trained word embeddings. The model architecture consists of an encoder and a decoder.

Encoder

The encoder takes in the input text sequence and produces an output state vector that captures the semantic meaning of the input. The input text sequence is first passed through an embedding layer initialized with pre-trained word embeddings. This layer maps each word in the input sequence to a high-dimensional vector representation, which is then fed into a bidirectional LSTM layer. The LSTM layer generates two output states for each time step, which are concatenated to form the final output state vector.

Decoder

The decoder takes in the output state vector produced by the encoder and generates the summary sequence. The summary sequence is generated autoregressively, one word at a time. The decoder first receives an input token "<start>" and passes it through an embedding layer initialized with pre-trained word embeddings. The embedding layer maps the input token to a high-dimensional vector representation, which is then fed into an LSTM layer. The LSTM layer generates a sequence of output states, which are used to predict the next word in the summary sequence.

The decoder's output states are passed through a dense layer with a softmax activation function to obtain the probability distribution over the output vocabulary. The word with the highest probability is selected as the predicted output token. This process is repeated until the decoder generates the "<end>" token, indicating the end of the summary sequence.

The model is trained using the encoder-decoder architecture and the teacher forcing technique, where the target summary sequence is fed as input to the decoder at each time step during training. The model is compiled using the categorical cross-entropy loss function and the Root Mean Square Propagation optimizer. The model is evaluated on the validation set using the ROUGE metric, which measures the similarity between the predicted summary and the ground truth summary.

5 Evaluation

The study presented in this project report yielded the following results. Analysis of the recall, precision and F-1 scores for all three rouge metrics indicated that the model's overall performance was suboptimal, mainly due to limited computational resources. As a remedy, additional LSTM layers and an activation layer were incorporated in the model. However, the Rouge scores decreased further, indicating that each approach did not lead to significant improvement individually. We hypothesize that combining these approaches may lead to a more effective model. Therefore, further experimentation with more extensive computational resources is required to optimize the model's performance.

	R	**P**	**F**
Rouge-1	0.04962	0.30911	0.08408
Rouge-2	0.00746	0.03881	0.01229
Rouge-l	0.04806	0.30117	0.08149

Table 1. Basic Enc-Dec Approach with single LSTM layers

	R	**P**	**F**
Rouge-1	0.01392	0.04100	0.02075
Rouge-2	0.00000	0.00000	0.00000
Rouge-l	0.01392	0.04100	0.02076

Table 2. Enc-Dec Approach with multiple LSTM layers

	R	**P**	**F**
Rouge-1	0.02817	0.35000	0.05096
Rouge-2	0.00000	0.00000	0.00000
Rouge-l	0.02561	0.33166	0.04652

Table 3. Enc-Dec Approach with an Attention layer

6 Discussion

It is clear that the use of pre-trained word embeddings can significantly improve the accuracy of natural language processing models, particularly in tasks such as text summarization. The pre-trained embeddings allow the model to learn more accurate representations of words, which can capture complex relationships between words that might be difficult to learn from a smaller dataset. Furthermore, the use of pre-trained embeddings can reduce training time and improve the generalization performance of the model.

7 Conclusion

In conclusion, this project report highlights the importance of various preprocessing techniques in natural language processing. Tokenization, data cleaning, and the use of pre-trained embeddings are key components that can significantly improve the accuracy of models in tasks such as text summarization. Additionally, it is evident that the choice of modeling approach and architecture plays a crucial role in achieving accurate and meaningful results. Although there is still room for improvement, this project provides valuable insights into the preprocessing and modeling techniques that can enhance the performance of natural language processing models.

References:

[1] Gowri Shankar, P. (2021). *Newspaper Text Summarization (CNN/DailyMail)*. Kaggle. https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail

[2] S. Bird, et al., *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, O'Reilly Media, 2009.

[3] SpaCy, *Industrial-Strength Natural Language Processing in Python*, spaCy, 2023. [Online]. Available: https://spacy.io/.

CS 512: Advanced Machine Learning– 04/27/2023

Project Final Report

Guide: Sathya N. Ravi Student: Navyadhara Gogineni UIN: 660392086

In this report I will discuss the details of my project for the course CS512. My project is **Text summarization on Amazon reviews data.**

1 What is text summarizing?

Text summarizing is the process of condensing a lengthy document or text into a shorter version while retaining the most important information and ideas. The goal of text summarizing is to provide a brief and concise overview of the main points of a text, which can save time and improve comprehension for readers. Text summarizing can be done manually, but it is often automated using natural language processing techniques and algorithms, such as extractive summarization or abstractive summarization. Extractive summarization involves selecting and combining the most relevant sentences or phrases from the original text, while abstractive summarization involves generating new sentences that capture the essence of the original text.

2 Dataset

I have worked in the Amazon Fine Food Reviews dataset that is openly available to use and download.

2.1 Description

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all 500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

2.2 Data Reading and Loading

- Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.
- We'll take a sample of 100,000 reviews to reduce the training time of our model. Feel free to use the entire dataset for training your model if your machine has that kind of computational power
- Following it I performed some EDA to dig deep into the data

3 Data Processing

We will perform the below preprocessing tasks for our data: Convert everything to lowercase, Remove HTML tags, Contraction mapping, Remove (‘s), Remove any text inside the parenthesis ( ), Eliminate punctuation’s and special characters, Remove stop words, Remove short words Tokenization of sentences: Used the NLTK sentence tokenizer

4 Model Training and Inference

I have started with the basic LSTM model and later fine tuned using the BiLSTM model. Below are the details of them.

4.1 LSTM Model

- We are finally at the model building part. But before we do that, we need to familiarize ourselves with a few terms which are required prior to building the model.
- Return Sequences = True: When the return sequences parameter is set to True, LSTM produces the hidden state and cell state for every timestep
- Return State = True: When return state = True, LSTM produces the hidden state and cell state of the last timestep only
- Initial State: This is used to initialize the internal states of the LSTM for the first timestep
- Stacked LSTM: Stacked LSTM has multiple layers of LSTM stacked on top of each other. This leads to a better representation of the sequence. I encourage you to experiment with the multiple layers of the LSTM stacked on top of each other (it’s a great way to learn this)
- Here, we are building a 3 stacked LSTM for the encoder

4.2 Fine-Tuning

By incorporating a bi-directional LSTM encoder and attention mechanism, our model will be able to better understand the context and generate more coherent summaries. Furthermore, the paper provides valuable insights into the model’s architecture, training methodology, and evaluation metrics, which can guide our implementation and experimentation process.

- The improved model incorporates a bidirectional LSTM encoder to capture context from both directions in the input sequence.
- This bidirectional approach enhances the context vector representation, potentially leading to more accurate summaries.

- The decoder in this model also employs an attention mechanism, allowing it to dynamically focus on the most relevant parts of the input when generating the summary.
- This model was also trained using the same dataset and evaluated using the same metrics as the base model.

5 Results

To compare the performances of the models, I have employed BLEU scores which fit right to our models evaluation. I have calculated the average BLUE score of each model and the results are as below:

Model	BLEU Score
LSTM	0.57
BiLSTM	0.47

Table 1: BLEU scores for different models

6 Discussion

- Comparing the two models, we can observe that the BiLSTM model outperforms the base LSTM model in terms of both loss and BLEU scores.
- This improvement can be attributed to the bidirectional LSTM encoder, which captures context more effectively, and the attention mechanism that allows the decoder to focus on the most relevant input sections.
- Consequently, the BiLSTM model generates more accurate and coherent summaries.

7 Conclusion

In conclusion, this project has demonstrated the effectiveness of using deep learning techniques, specifically LSTM and BiLSTM, for the task of text summarization. Our experiments have shown that the BiLSTM model outperforms the traditional LSTM model, thanks to its ability to capture context from both directions of the input sequence. This leads to a better understanding of the text and ultimately generates higher quality summaries.

In summary, the findings of this project can serve as a valuable starting point for future research and development in the field of text summarization. By leveraging the power of deep learning models, such as BiLSTM, we can continue to enhance the quality of generated summaries and contribute to the growing body of knowledge in natural language processing.

CS 512: Advanced Machine Learning-

Long Form Question Answering Using Seq2Seq Models

Guide: Sathya N. Ravi Student: Sajal Chandra UIN: 659145709

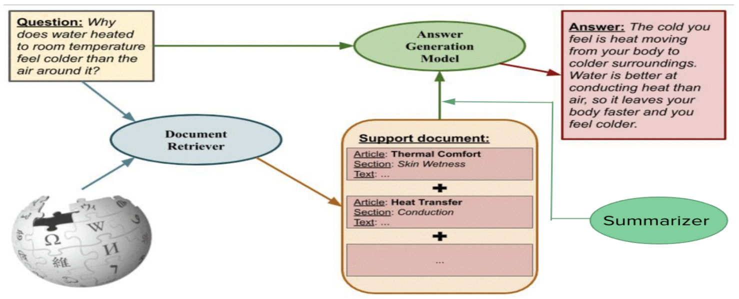
This is the final report of my project titled *Long Form Question Answering using Sequence to Sequence Models*.

1 Inspiration

The paper titled "ELI5: Long Form Question Answering"[1] was the major source of inspiration for this project. The authors introduce a large corpus(collected from Reddit) which comprises of 270,000 question-answer sets which are ideal for training a model to perform a long-form question answering task. Additionally, they introduce a procedure to improve the accuracy of the answers provided by the model- using a supporting document to answer each question.

2 Pipeline

The ELI5 authors propose a simple approach to answering the given question. As illustrated in the diagram, a document retriever is used to retrieve Wikipedia articles that potentially contain useful information pertaining to our question. The support document is then passed to the Answer Generation Model along with the original question pair.


Question: Why does water heated to room temperature feel colder than the air around it?
Answer Generation Model
Answer: The cold you feel is heat moving from your body to colder surroundings. Water is better at conducting heat than air, so it leaves your body faster and you feel colder.
Document Retriever
Support document:
Article: Thermal Comfort
Section: Skin Wetness
Text: ...
Article: Heat Transfer
Section: Conduction
Text: ...
...
Summarizer

Since Wikipedia articles are detailed and contain 3500 words on average, creating support documents using entire articles leads to slow performance. Further, articles that are not relevant to the question being asked can even have a negative impact on the prediction. To mitigate this problem, I introduced a text summarizer into this pipeline with the aim of reducing the complexity of the model and boost its performance.

3 Seq2Seq Text Summarizer

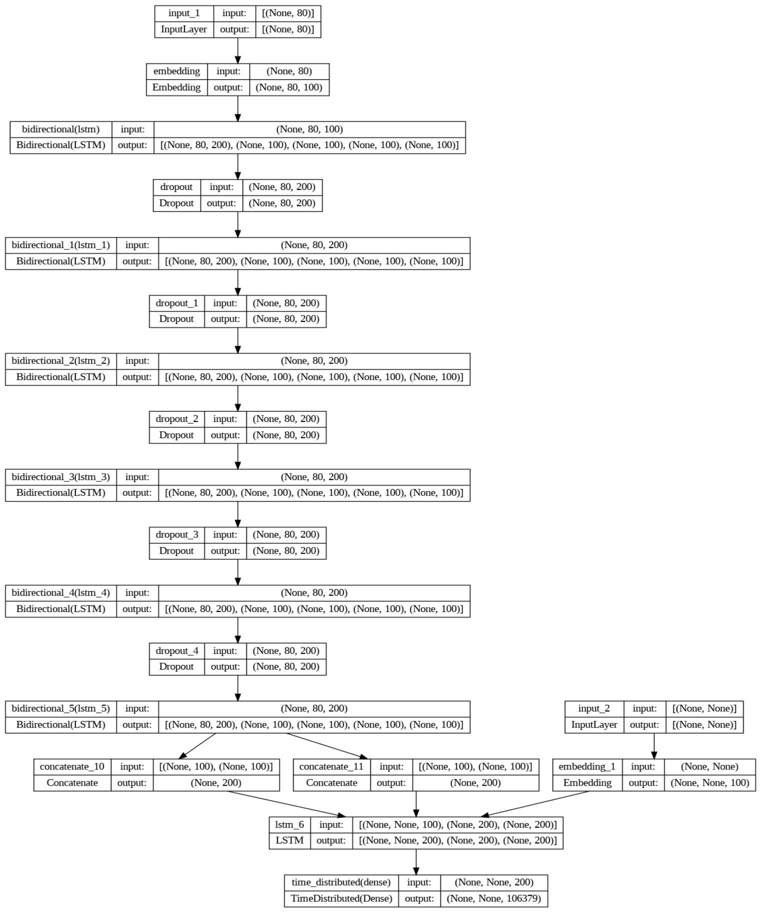
3.1 Dataset

To train a sequence-to-sequence model for summarizing Wikipedia articles, I used the wikipedia-summary dataset[2] from Hugging Face. It contains over 3M article-summary pairs. The dataset is in English, and contains extracted summaries and articles- which makes it easier to train for out specific task.

3.2 Preprocessing

The preprocessing steps are similar to the other text summarizartion tasks in this report. I performed the following operations in the raw text- removed HTML tags, converted to lowercase, removed stopwords, and eliminated punctuations. After cleaning the text, I used nltk's tokenizer to vectorize the text.

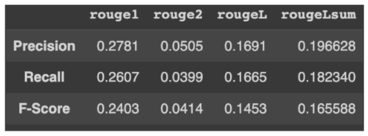
3.3 Summarization Model Architecture


input_1 | InputLayer | input: [(None, 80)] | output: [(None, 80)]
embedding | Embedding | input: (None, 80) | output: (None, 80, 100)
bidirectional(lstm) | Bidirectional(LSTM) | input: (None, 80, 100) | output: [(None, 80, 200), (None, 100), (None, 100), (None, 100), (None, 100)]
dropout | Dropout | input: (None, 80, 200) | output: (None, 80, 200)
bidirectional_1(lstm_1) | Bidirectional(LSTM) | input: (None, 80, 200) | output: [(None, 80, 200), (None, 100), (None, 100), (None, 100), (None, 100)]
dropout_1 | Dropout | input: (None, 80, 200) | output: (None, 80, 200)
bidirectional_2(lstm_2) | Bidirectional(LSTM) | input: (None, 80, 200) | output: [(None, 80, 200), (None, 100), (None, 100), (None, 100), (None, 100)]
dropout_2 | Dropout | input: (None, 80, 200) | output: (None, 80, 200)
bidirectional_3(lstm_3) | Bidirectional(LSTM) | input: (None, 80, 200) | output: [(None, 80, 200), (None, 100), (None, 100), (None, 100), (None, 100)]
dropout_3 | Dropout | input: (None, 80, 200) | output: (None, 80, 200)
bidirectional_4(lstm_4) | Bidirectional(LSTM) | input: (None, 80, 200) | output: [(None, 80, 200), (None, 100), (None, 100), (None, 100), (None, 100)]
dropout_4 | Dropout | input: (None, 80, 200) | output: (None, 80, 200)
bidirectional_5(lstm_5) | Bidirectional(LSTM) | input: (None, 80, 200) | output: [(None, 80, 200), (None, 100), (None, 100), (None, 100), (None, 100)]
input_2 | InputLayer | input: [(None, None)] | output: [(None, None)]
concatenate_10 | Concatenate | input: [(None, 100), (None, 100)] | output: (None, 200)
concatenate_11 | Concatenate | input: [(None, 100), (None, 100)] | output: (None, 200)
embedding_1 | Embedding | input: (None, None) | output: (None, None, 100)
lstm_6 | LSTM | input: [(None, None, 100), (None, 200), (None, 200)] | output: [(None, None, 200), (None, 200), (None, 200)]
time_distributed(dense) | TimeDistributed(Dense) | input: (None, None, 200) | output: (None, None, 106379)

Starting with a simple LSTM model, I trained and fine-tuned it b testing various changes such as adding hidden layers, changing the learning rate, adding dropout layers, etc. Eventually I landed on the model shown next. It is a bidirectional LSTM model with a total of 16 vertical layers. This is a sequence-to-sequence model that takes input sequences of length 80 and maps them to dense vectors of length 100 using an embedding layer. The model then processes the embeddings using six bidirectional LSTM layers to capture past and future contexts of each word. Dropout layers are included for regularization, and the target sentences are concatenated with context vectors before being fed to the decoder LSTM. The output layer applies a softmax activation to produce a variable-length output sequence.

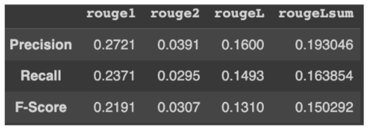
4 Evaluation

After training this model on the wikipedia-summary data, the model achieved a BLEU score of 0.6338. Plugging this model into the pipeline and evaluating the Answer Generation Model's performance, I got the following results:



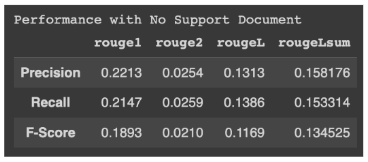
	rouge1	rouge2	rougeL	rougeLsum
Precision	0.2781	0.0505	0.1691	0.196628
Recall	0.2607	0.0399	0.1665	0.182340
F-Score	0.2403	0.0414	0.1453	0.165588

Figure 1: Full Support Document



	rouge1	rouge2	rougeL	rougeLsum
Precision	0.2721	0.0391	0.1600	0.193046
Recall	0.2371	0.0295	0.1493	0.163854
F-Score	0.2191	0.0307	0.1310	0.150292

Figure 2: Summarized Support Document



Performance with No Support Document

	rouge1	rouge2	rougeL	rougeLsum
Precision	0.2213	0.0254	0.1313	0.158176
Recall	0.2147	0.0259	0.1386	0.153314
F-Score	0.1893	0.0210	0.1169	0.134525

Figure 3: No Support Document

5 Conclusion

Upon observing the results, it is clear that summarizing the support documents did not lead to a significant loss of information- as the ROUGE scores are only slightly lower than the original model's performance. Further, when compared with the model's performance without any support document, it is confirmed that summarizing the support document indeed helped the Answer Generation Model come up with better answers.

References

[1] Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. Eli5: Long form question answering, 2019.

[2] Thijs Scheepers. Improving the compositionality of word embeddings. Master's thesis, Universiteit van Amsterdam, Science Park 904, Amsterdam, Netherlands, 11 2017.