Text Summarization with Seq2Seq model and Transformers

Abstract:

There has been an exponential growth in the amount of text data available online, which is why text summarization has become important. The majority of modern recommender and text classification systems necessitate the analysis of massive amounts of data. It is extremely laborious and time-consuming to manually create accurate and fluid summaries of lengthy articles. As a result, creating automated summaries for the data and using them to train machine learning models will save space and time. There are two distinct processes for creating summaries: extractive and abstractive. The extractive technique selects only the pertinent sentences from the source text and extracts them. On the other hand, abstractive summarization techniques generate the summary after interpreting the original text, which makes it more difficult. In our project, we worked on two different experiments. First, we used the CNN news dataset to implement a seq2seq model and produce summaries for the articles in the dataset. The summaries, in our views, are composed of a significant combination of key phrases and words. They lacked coherence and had no meaning when taken together. Consequently, we applied hugging face transformer models like Pegasus and Bart to a text input and discovered that Pegasus produced the best summary that was both meaningful and abstract in comparison to the other models. We came to this conclusion by contrasting it with a manually crafted summary of the text.

Introduction:

We are surrounded by information every day. Daily, we read a wide range of articles. As a result, there is a lot of data moving around, most of which is text. If there is information in an article that we need to learn, we must read the entire article to comprehend it. However, many times these articles grow to be overly lengthy, such as a 5000-word article, which is time-consuming. This means that to get the 1000 words worth of information, we must read the entire 5000 words of the article, which is a complete waste of time. If we need to read several articles like that for work, it will take a long time and cost us an hour of work. Hence there is a need for summarization methods to deal with this data. Text Summarization is the process of extracting summaries from the original large text without losing important information. The summary must be abstract, continuous, and accurately convey the significance. The summarization of documents has applications across almost all Internet domains. As part of the search experience, search engines offer query- and context-specific summary snippets; news websites use them to condense the articles; social media use them for content targeting; and e-commerce websites use them to improve the browsing experience by highlighting items or products.

Based on their output, text summarization methods can be divided into two broad categories: Abstractive and extractive methods.The extractive method entails highlighting the most important phrases and lines in the documents. It then combines all the important lines to form the summary. Therefore, every word and line of the summary in this instance comes from the source material that is being summarized. The sentences chosen from the original text passage are not used to build the abstractive summaries. Instead, they generate a paraphrase of the main points of the given text, employing a vocabulary set distinct from that of the original document and this is very similar to what humans do. In our minds, we encode the document with semantic information. Then, we choose words from our general vocabulary (the words we frequently use) that are semantically appropriate to produce a summary that captures the essence of the original document.

In our project, we implement a seq2seq model on CNN news dataset and apply hugging face transformers on input text which in this case is the plot of the movie “Turning Red”. We found training a seq2seq model with an attention layer to be more challenging on devices without a powerful GPU support, hence we implemented a simple seq2seq model with LSTM based encoder-decoder architecture with standard parameters. We faced a similar challenge while applying transformer models like Bart and Pegasus on the CNN dataset, hence we implemented pre-trained

transformer models that are provided by hugging face on a simple text input to understand the different architectures of Pegasus and Bart models. We found that the Bart model extracted the best keywords, combined them into a summary and generated it, whereas Pegasus produced a more fluid, natural sounding and a meaningful summary.

Related Work:

The task of summarization using NLP was first introduced in 1958. In the beginning, statistical methods were used to calculate scores for each sentence and then choose the ones with the best scores. Many methods, including TF-IDF, Bayesian models, etc., were used to calculate this score. Even though all these methods were extractive and merely trimmed the original text, they were all capable of computing a reliable summary by key phrase extraction. The use of machine learning summarization algorithms, like Bayesian Learning Models as done in the paper, came into focus next. These machine learning methods have proven effective at finding patterns in texts.

The LSTM network, one of the RNN variants, uses connected nodes to retain sequential information by keeping important information and discarding irrelevant information that assisted in producing summaries. The encoder-decoder model was created using the LSTM network's methodology. The encoder-decoder framework used to implement Seq2seq models for NLP task solving produced excellent results, but parallelization remained an issue. Although the sequential information is kept in the encoder-decoder model, processing is done by taking one input at a time because LSTM only accepts one input at a time. This is a problem because, even though this model produces better results, it fails in every scenario that could possibly arise, negating the goal of creating machine perception.

As a result, an attention layer was added to the encoder-decoder model, which examines the input sequence at each step and assigns a weight to it based on the sequences that have come before. For one input, the attention layer generates vector matrices by considering every word in the sentence. The attention layer compels the computer to read the entire text as one input rather than each sequence as a separate input. For the abstractive approach, this mechanism proved to be very useful and was widely adopted.

Several pre-trained language models, such as BERT, PEGASUS, UNiLM, and GPT, have revolutionized the field of natural language processing. Their strong results motivated us to use these models for summarization. Most of the pre-trained models utilized are built using Google's potent Transformer architecture, which was created in 2017. It is modeled after the encoder-decoder framework and is like RNN. The transformer model was developed to solve natural language processing tasks that involved transforming an input sequence into an output sequence. In our project, we used the Hugging face library's pre-trained language models that can be applied to solve a variety of NLP problems.

Methodology:

Seq2Seq Model:

Sequence to Sequence models is a unique class of recurrent neural network architectures that are frequently employed (though not exclusively) to address challenging language-related issues like machine translation, question answering, chat-bot creation, text summarization, etc. In our first experiment, we used a CNN news dataset to train a seq2seq model.

The Encoder-Decoder architecture, which consists of the following steps, is the one most often used to create Seq2Seq models - 1. LSTM models are commonly used in both the encoder and the decoder (or sometimes GRU models). 2.After reading the input sequence, the encoder compiles the data into what are known as internal state

vectors (in the case of LSTM these are called the hidden state and cell state vectors). The encoder's outputs are discarded, and only the internal states are kept. 3. An LSTM is used as the decoder, and its initial states are set to match those of the encoder's LSTM. The decoder begins generating the output sequence using these initial states. 4. Throughout the training and inference processes, the decoder acts somewhat differently. We employ a method known as "teacher force" during training to speed up the decoder's learning process. Each time a time step in inference occurs, the output from the previous time step is fed. 5. The encoder condenses the input sequence into state vectors, which are sometimes referred to as "thought vectors," and feeds them to the decoder, which begins producing the output sequence using the input sequence as a starting point. The decoder is nothing more than a language model.

An overview of the encoder - We'll go through the input sequence (English sentence) word for word. Then retain the LSTM network's internal states that were generated after the last time step hk, ck (assuming the sentence contains 'k' words). The encoding of the input sequence is what these vectors (states hk and ck) are referred to as because they vectorize (encode) the entire input. After reading the entire sequence, generate the output; the outputs (Yi) from the encoder at each time step are discarded.

Inference Algorithm: a) As a result during the inference, the decoder LSTM is called in a loop, processing only one step at a time. b) The decoder's initial states are set to the encoder's final states. C) The START_ token is always used as the decoder's initial input. D) We keep track of the decoder's states at each time step and set them as the starting points for the subsequent time step. E) The predicted output from each time step is fed into the next time step's input. f) The loop is broken when the decoder predicts the END_ token.

Limitations: Seq-2-seq models have some drawbacks despite being extremely effective at what they do: 1) Long-term dependencies are still difficult to manage. 2) Parallelization is not possible due to the sequential nature of the model architecture. The Transformer concept from Google Brain addresses these issues.

Transformers:
To understand natural language, machines must be able to recognize the relationships between the words in sentences. This is where the Transformer concept comes into play.

Transformer is a deep learning model that is also an encoder-decoder-based architecture that employs the Attention mechanism but does not rely on RNNs to accelerate the model. It consists of an encoder—which can be thought of as a stack of encoders or encoding layers (six layers)—and an equal number of decoders. The decoder block is made up of three sub-layers, including two attention layers and one FFNN layer, while each encoder/encoding layer has two sub-layers, including an Attention layer and a Feed-Forward Neural Network (FFNN) layer.

Google proposed Transformer in the paper "Attention Is All You Need" where it is defined as -
“The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution.”

In this context, "transduction" refers to the transformation of input sequences into output sequences. Transformer's concept is to completely handle the dependencies between input and output with attention and recurrence.
How does this encoder and decoder stack structure perform?

- The first encoder receives the word embeddings of the input sequence.
- These are then transformed and passed on to the following encoder.
- The output of the last encoder in the encoder-stack is distributed to all decoders in the decoder-stack.

In addition to the self-attention and feed-forward layers, the decoders also have an additional layer of encoder-decoder attention. This allows the decoder to focus on the relevant parts of the input sequence.

A transformer model is used mainly for abstractive text summarization. The conventional method and the cosine similarity method of text summarization are of the extractive type, whereas the Hugging Face Transformer is of the abstractive type.

Experiments:
Experiment 1: Seq2Seq model with CNN/ daily mail data:

The CNN/Daily Mail dataset is what we are most interested in for our first part of the project. This dataset is useful because each article is accompanied by a concise list of bullet points representing the important "highlights" of the text. As a result, we can choose very carefully whether to use headlines or human-generated summaries for our summarization task.

To achieve cutting-edge language understanding results, the CNN/Daily Mail corpus has also been thoroughly examined in the question-answering/reading comprehension task. But the potential for text summarization in this dataset has not been fully investigated. Therefore, we decided to train our seq2seq model on this dataset.

There are three fields in the dataset: id, article, and highlights. Instances for the training and testing split are 287,113 and 11,490, respectively. For simpler computation, we initially dropped the id column. Following that, we concentrated on preprocessing the data by removing extraneous characters and spaces, replacing empty strings with null, removing null values from data frames, and changing the text's case to lowercase. Today's news uses a lot of acronyms, so we only mentioned the ones that frequently contradict one another.

The articles use a lot of stop words as well. By eliminating these words, we attempted to focus more attention on the important information by eliminating the low-level information from our text. We can state that eliminating these words has no negative effects on the model that we train. As an added benefit, removing stop-words reduces the dataset size, which reduces training time due to the fewer tokens included in the training.

We used this dataset to train the seq2seq model, which was discussed in the methodology section. Prior to the LSTM layer, an embedding layer was used. Several word embedding methods convert a word into a fixed-length vector. The requirement for embeddings stems from the fundamental concept of turning textual information into numerical data. The primary function of embedding layers is to change each word from being represented as only o's and 1's to a fixed-length vector with real values. With fewer dimensions, these fixed-length vectors aid in improving the representation of words.

These summaries are then saved, and they are compared to the original summaries. In comparison to the original summaries, we discovered that the summaries produced by the model performed poorly. We believe that by adding an attention layer to the current model, we will be able to produce summaries that are comparable to the original summary.

Experiment2 - Text Summarization with Transformers

Here, we first extracted summaries from plain text using the Python library Sumy. In addition to a few frameworks for evaluation, Sumy offers a number of algorithms for text summarization. Among the other algorithms available, we implemented LexRank and LSA summarizers. We used the “sumy.summarizers” module to access various summarizers.

LexRank: This text summarizer is based on graphs. The summary is made up of sentences with a high priority/importance. A sentence that is similar to many other sentences in the input text is more likely to be important and thus ranks higher. The higher-ranking sentences are then combined to form the summary.

LSA : Latent Semantic Analysis (LSA) is an unsupervised learning algorithm for extractive text summarization. By using singular value decomposition (SVD) on the term-document frequency matrix, it extracts sentences with semantically important words. These sentences are then combined to form the summary of the input text.

The abstractive method is currently the best method for summarizing texts because it creates new sentences that best capture the meaning of the entire text. This is preferred to extractive methods, that merely select sentences from the original text for the summary.

The Huggingface transformers library offers a quick and efficient way to implement abstractive summarization using transformers. HuggingFace supports cutting-edge models for tasks like summarization, classification, etc. GPT-2, GPT-3, BERT, OpenAI, GPT, and T5 are some popular models.

We have used transformer pipelines. The transformers library's pipelines serve as an abstraction for the complicated code that powers it; using pre-trained models for inference is the best choice. The pre-trained model must be imported into pipeline functions, and raw data must be passed. With just a function call, the Pipeline will perform all necessary processing steps for the algorithm's input, including tokenization, padding, and all other pertinent processing steps, on our data in the backend before returning the results.

Here we passed 4 models in the pipeline along with our input text. And found that for our input-text, the pegasus-xsum model generates the best summary among the others.

Steps:

1. Install and import the necessary libraries like sumy, nltk, pprint
2. Load the input text
3. Parse the text, apply LexRank summarizer followed by LSA summarizer
4. Now install transformers and import pipelines.
5. Apply default summarizer, pegasus-xsum, bart-large and mbart-large-50 models.
6. Tabulate the summaries to compare.

See references, for all the documentations of the pipelines, models and libraries that we used.

Conclusions:

We implemented a seq2seq LSTM model on the CNN daily mail news dataset and realized that due to its dependencies, training the model takes a long time and becomes more difficult without the support of GPUs. The fundamental issue of "vanishing gradient" persists, as the gradient signal from the objective from which the recurrent neural network learns vanishes as it travels backward. In comparison to the expected summaries, the output summaries are not the best. As a result, we attempted to implement transformer models. We were

confronted with the previously mentioned issue of training the model with large amounts of data. So we decided to use the hugging face transformers - distillbart, pegasus-xsum, bart-large, and mabrt - with the movie plot "Turning Red." In comparison to the others, we discovered that the pegasus-xsum and distillbart models generate a substantial summary.

Future Work:

In the future, we plan to extend experiment 1 to models with attention, as well as apply transformer models and compare the results. We can also work on developing more robust models at the same time. In Experiment 2, we can extend our algorithm to create variable-length summaries and apply them to multi-document summarization. We would also like to build the Pegasus model from the bottom up and use it to improve the accuracy, fluency, and coherence of the summaries.

Ethical considerations:

For our first experiment, we used the CNN Daily Mail dataset, which is an English-language dataset containing just over 300,000 unique news articles written by CNN and Daily Mail journalists. We obtained this information from a git repository (mention it here). The data is a compilation of articles written by CNN News and Daily Mail News journalists, and it was created with their permission. There are no special annotations in the data. Individual names can be found in the dataset, but no information about the original author is included. The goal of this dataset is to aid in the development of models that can summarize lengthy paragraphs of text in one or two sentences.

This task helps present information effectively when there is a lot of text present. It should be noted that any summarizations generated by models trained on this dataset are not reflective of the language used in the articles but are instead generated automatically. Version 1.0.0 of the CNN / Daily Mail dataset is licensed under the Apache-2.0 License.

In their study of the CNN/Daily mail dataset, the Penn Treebank, and WikiText-2, Bordia, and Bowman (2019) examine gender bias measurement and debiasing methods. According to their metric, the CNN / Daily mail dataset has a slightly lower gender bias than the other datasets, but it still shows evidence of gender bias when looking at words like 'fragile.' Since the articles were written by and for readers in the US and the UK, they are likely to feature events that were thought to be pertinent to those populations.

Then, in accordance with all Wikipedia licenses and agreements, we used the movie "Turning Red's plot, which was cited from its Wikipedia page for our second experiment as the input text.

References:

1. Ilya Sutskever, Oriol Vinyals, Quoc V. Le - "Sequence to Sequence Learning with Neural Networks"
2. Sawan Saxena - "Understanding Embedding Layer in Keras"
3. Andrew Tucker -" Contradictions"
4. Pradeep Dhote - "Seq2Seq-Encoder-Decoder-LSTM-Model"
5. Anushka Gupta, Diksha Chugh, Anjum, Rahul Katarya - "Automated News Summarization Using Transformers"
6. Vincet Chen, Liezl Puzon, Eduardo Torres Montaño -"An Examination of the CNN/DailyMail Neural Summarization Task"
7. Chandra Khatri, Gyanit Singh, Nish Parikh - "Abstractive and Extractive Text Summarization using Document Context Vector and Recurrent Neural Networks"
8. Baotian Hu Qingcai Chen Fangze Zhu -"LCSTS: A Large Scale Chinese Short Text Summarization Dataset"

9. Ramesh Nallapati - “Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond”
10. Aravindpai Pai - “Comprehensive Guide to Text Summarization using Deep Learning in Python”
11. Mengli Zhang, Gang Zhou, Wanting Yu, Ningbo Huang, and Wenfen Liu -”A Comprehensive Survey of Abstractive Text Summarization Based on Deep Learning”
12. SNEHA CHAUDHARI, ARUN MITHAL -”An Attentive Survey of Attention Models”
13. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin - “ Attention is All you need”
14. Christian H, Agus M, Suhartono D (2016) Single Document Automatic Text Summarization using Term Frequency-Inverse Document Frequency (TF- IDF). ComTech: Computer, Mathematics and Engineering Applications 7:285 . https://doi.org/10.21512/comtech.v7i4.3746
15. Nomoto T (2005) Bayesian Learning in Text Summarization
16. Graves A (2013) Generating Sequences With Recurrent Neural Networks. CoRR abs/1308.0850:
17. Nallapati R, Xiang B, Zhou B (2016) Sequence-to-Sequence RNNs for Text Summarization. CoRR abs/1602.06023:
18. Hochreiter S, Schmidhuber J (1997) Long Short-Term Memory. Neural Comput 9:1735–1780 . https://doi.org/10.1162/neco.1997.9.8.1735
19. Shi T, Keneshloo Y, Ramakrishnan N, Reddy CK (2018) Neural Abstractive Text Summarization with Sequence-to-Sequence Models. CoRR abs/1812.02303:
20. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is All you Need. ArXiv abs/1706.03762: Devlin J, Chang M-W, Lee K, Toutanova K (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. CoRR abs/1810.04805:
21. Radford A (2018) Improving Language Understanding by Generative Pre- Training
22. Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, Cistac P, Rault T, Louf R, Funtowicz M, Brew J (2019) HuggingFace’s Transformers: State-of- the-art Natural Language Processing. CoRR abs/1910.03771
23. Hugging Face - models , pipelines.
24. Dzmitry Bahdanau -”NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE”
25. Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer -”BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”
26. Jingqing Zhang, Yao Zhao, Mohammad Saleh, Peter J. Liu - “PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization”