

Abstractive Text Summarization

Manchikanti Vikas

*Undergraduate Scholar
Department of Information Technology
Anurag University*

Goli Anusha

*Undergraduate Scholar
Department of Information Technology
Anurag University*

Nittala Aishwarya

*Undergraduate Scholar
Department of Information Technology
Anurag University*

Mr.G.L.Anand Babu

*Assistant Professor
Department of Information Technology
Anurag University*

ABSTRACT- *In this more data-filled world, where people are becoming too sluggish to read it all, we have taken a step to create a summary from the data, which includes news stories, massive document files, reviews, and lengthy paragraphs, using an upgraded technique to extract the information as quickly and effectively as possible. Although it can be tiresome and time-consuming, humans find it highly challenging to manually extract the summary of huge text volumes. On the Internet, there is a wealth of textual content. As a result, finding relevant documents among the many documents available and learning useful information from them is a challenge. Automatic text summarization is crucial for resolving issues of this nature. These issues should be resolved by automatic text summarising, which enables you to quickly and simply identify the main concepts in a given text. Text summarising is a strategy for producing a brief, accurate summary of lengthy texts while concentrating on the passages that provide relevant information and keeping the overall meaning intact.*

Keywords-*Abstractive Summarization, Self Attention, Multi-headed Attention, NLP, RNN encoder-decoder, LSTM, Bi-Directional RNN, GRU, Seq2Seq Model.*

Date of Submission: 05-03-2023

Date of acceptance: 18-03-2023

I. INTRODUCTION

Text summarization is the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings by creating a coherent and fluent summary of the text document and involving the outlining of the text's major points. Automatic text summarization is a common problem in machine learning and natural language processing (NLP). We can broadly classify text summarization into two types:

Extractive Summarization: This technique involves the extraction of important words/phrases from the input sentence. The underlying idea is to create a summary by selecting the most important words from the input sentence.

Abstractive Summarization: This technique involves the generation of entirely new phrases that capture the meaning of the input sentence. The underlying idea is to put a strong emphasis on the form — aiming to generate a grammatical summary thereby requiring advanced language modeling techniques.

Abstractive summarization is a Natural Language Processing (NLP) task that aims to generate a concise summary of a source text. Unlike extractive summarization, abstractive summarization does not simply copy important phrases from the source text but also potentially comes up with new phrases that are relevant, which can be seen as paraphrasing.

II. BACKGROUND

Abstractive summarization has gone through several stages of development, with each stage building on the previous one. Here is a brief overview of the key developments in the field:

RNN Encoder-Decoder Summarization: In the early days of abstractive summarization, researchers used recurrent neural networks (RNNs) to encode the input text and generate the summary. The basic idea behind this approach was to use the RNN to build a fixed-length representation of the input text, which was then fed to another RNN to generate the summary. However, this approach had several limitations, such as the difficulty of handling long input sequences and the tendency to generate generic and repetitive summaries.

Bi-Directional RNN: To address the limitations of the RNN encoder-decoder approach, researchers introduced bi-directional RNNs (BRNNs), which allowed the model to capture the context of the input text more effectively. BRNNs work by processing the input text in both directions (forward and backward) and concatenating the hidden states of the two directions to obtain a more comprehensive representation of the input text.

Gated Recurrent Neural Networks (LSTM and GRU): Another improvement to the RNN architecture was the introduction of gated recurrent neural networks (GRNNs), which include long short-term memory (LSTM) and gated recurrent unit (GRU) networks. These models were designed to address the vanishing gradient problem that occurs when training deep RNNs. LSTMs and GRUs use special gates to control the flow of information in and out of the hidden state, allowing the model to store and retrieve information for longer periods of time.

Attention Mechanism: The introduction of the attention mechanism was a major breakthrough in abstractive summarization. The attention mechanism allows the model to focus on specific parts of the input text, based on the relevance to the summary. In other words, attention allows the model to dynamically adjust its focus to the most important parts of the input, rather than relying on a fixed-length representation. This approach has led to significant improvements in the quality and fluency of the generated summaries.

In summary, abstractive summarization has evolved from simple RNN-based models to more advanced architectures like bi-directional RNNs, LSTMs, GRUs, and attention mechanisms. These developments have led to significant improvements in the quality of abstractive summaries and opened up new possibilities for natural language processing applications.

III. LITERATURE SURVEY

Seq2Seq models are successfully used in a variety of natural language processing-related applications, including text summarization, image captioning, and machine translation. Rush et al.'s study introduced the neural attention Seq2Seq model and used an attention-based encoder and neural network language model (NNLM) decoder. By substituting the feed-forward NNLM with RNN, Chopra et al. improved this model even further. Nallapati et al. added some fresh ideas to the RNN encoder-decoder architecture to address various issues with abstractive text summarization. It featured: 1) A unique encoder to record keywords. 2) A switching generator-pointer to identify and manage words that have run out of vocabulary. Additionally, numerous models attempted to condense brief documents into a single-line summary. When it comes to condensing lengthy documents into summaries of several sentences, these approaches have various drawbacks: 1) They are unable to faithfully replicate the key details of the original papers. 2) Unable to effectively handle OOV terms. 3) Has trouble producing non-repetitive summaries and comes off as unnatural. See et al. suggested a pointer-generator network approach to address the first two problems. It can use a pointer to copy a word from the source text and then use a generator to find a new word to replace it. This method allows for the accurate reproduction of information as well as efficient handling of OOV words. Numerous approaches, including coverage mechanism, intra-temporal, and intra-decoder attention mechanisms, are utilized to overcome the third issue.

IV. EXPERIMENTAL

A. Dataset

1) Standard Dataset:

Dataset Description:

In this project, we are using Inshorts News dataset which is provided by Kaggle. Here you will be provided with news articles and their summary.

Size of Dataset: 55104 records

B. Experimental Environment

Windows 10 operating system as a test platform,

- CPU is Intel Core I7 6500u, which has dual cores running on 2.4GHZ.

- RAM 16GB.
- GPU is Nvidia GTX 970, CUDA Cores 1664, 4GB GDDR5

C. Methodology

Self-Attention

Suppose we have the following input document: "The COVID-19 pandemic has led to many changes in the way people live and work. Many countries have imposed lockdowns and social distancing measures to slow the spread of the virus. This has resulted in a shift towards remote work and online learning. Additionally, the pandemic has highlighted the importance of healthcare and the need for better preparedness for future pandemics."

To generate a summary of this document, we can use a self-attention mechanism in the following way:

Input Representation-We encode the input document into a sequence of hidden states using a pre-trained language model such as BERT or RoBERTa.

Self-Attention Layer- Next, we apply a self-attention layer to the hidden states to calculate a set of attention weights that indicate how much attention should be paid to each word in the input sequence when generating each word in the output sequence. The attention weights are computed using a dot-product attention function, which calculates the dot product of each word's hidden state with every other word's hidden state.

For example, suppose we want to generate the first word in the summary, which is "The". We first calculate the attention weights for this word by taking the dot product of its hidden state with the hidden states of every other word in the input sequence. This gives us a vector of attention weights, which we can normalize using the softmax function to obtain a set of normalized attention weights that sum to 1.

Context Vector- Using the attention weights, we can calculate a context vector, which is a weighted sum of the hidden states. The context vector represents a summary of the most important information in the input sequence. For example, suppose the attention weights for the first word "The" are [0.2, 0.1, 0.1, 0.2, 0.2, 0.2]. This indicates that the first word should pay the most attention to the first and last words in the input sequence. We can use these attention weights to calculate a context vector by taking a weighted sum of the hidden states, like this:

$$\text{context} = 0.2 * \text{hidden_state_1} + 0.1 * \text{hidden_state_2} + 0.1 * \text{hidden_state_3} + 0.2 * \text{hidden_state_4} + 0.2 * \text{hidden_state_5} + 0.2 * \text{hidden_state_6}$$

Decoding- Finally, we use the context vector to generate the output summary one token at a time. At each step, the decoder uses the context vector and the previously generated tokens to predict the next token in the summary.

For example, using the context vector calculated above, the decoder might generate the first word of the summary as "The". Then, it would use the context vector and the first generated word to predict the second word in the summary, and so on, until the complete summary is generated.

In this way, the self-attention mechanism allows the model to selectively focus on the most relevant parts of the input document when generating the summary, which can improve the quality of the final summary.

Query, Key, and Values

Self-attention is a mechanism used in Transformer-based models to learn the importance of each word in a sequence concerning all other words in the sequence. The basic idea behind self-attention is to represent each word as a combination of weighted sums of all the other words in the sequence. To accomplish this, self-attention uses three types of vectors: key vectors, query vectors, and value vectors.

In the context of abstractive summarization, the input document is first encoded into a sequence of embeddings, and each of these embeddings is treated as a query, key, and value vector for self-attention. The three vectors are derived from the same embedding vector, but each has a different learned weight matrix.

Here's an example to illustrate the concept of self-attention with key, query, and value vectors:

Input Document: "The COVID-19 pandemic has affected many aspects of people's lives. One of the most significant changes has been the move towards remote work and online learning. With many schools and

businesses closed or operating at reduced capacity, people have had to adapt to new ways of working and studying. In addition, the pandemic has highlighted the importance of healthcare and the need for better preparedness for future outbreaks.”

Step 1: Embedding Layer

The input document is first encoded into a sequence of embeddings using a pre-trained transformer model.

Step 2: Key, Query, and Value Vectors

Next, the embeddings are used to generate key, query, and value vectors for each word in the sequence. The key, query, and value vectors are learned weight matrices of the embedding vector. Here is an example of how the key, query, and value vectors are generated for the first word "The" in the input document:

Query Vector: The query vector for "The" is a learned linear transformation of the embedding for "The".

Key Vector: The key vector for "The" is a learned linear transformation of the embedding for "The".

Value Vector: The value vector for "The" is a learned linear transformation of the embedding for "The".

Step 3: Calculating Attention Weights

Once the key, query, and value vectors have been generated for each word in the sequence, the self-attention mechanism calculates attention weights for each word in the sequence. The attention weight for each word is calculated by taking the dot product of the query vector for that word with the key vectors for all the other words in the sequence. The result is a set of scores that indicate how much each word contributes to the representation of the current word.

Step 4: Weighted Sum of Values

Finally, the self-attention mechanism computes a weighted sum of the value vectors for all the words in the sequence, where the weights are given by the attention scores calculated in step 3. This weighted sum represents the output of the self-attention mechanism for the current word in the sequence.

By applying self-attention with key, query, and value vectors to the input document, the transformer-based model can learn to attend to different parts of the document to generate a more informative summary.

Neural network representation of Attention

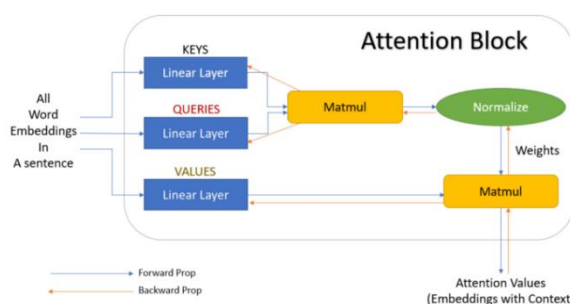


Figure 1. Neural network representation of Attention block

Figure 1 shows the neural network representation of an attention block. The word embeddings are first passed into some linear layers. These linear layers do not have a ‘bias’ term, and hence are nothing but matrix multiplications. One of these layers is denoted as ‘keys’, the other as ‘queries’, and the last one as ‘values’. If a matrix multiplication is performed between the keys and the queries and is then normalized, we get the weights. These weights are then multiplied by the values, and summed up, to get the final attention vector. This block can now be used in a neural network and is known as the Attention block. Multiple such attention blocks can be added to provide more context. And the best part is, we can get a gradient backpropagating to update the attention block (weights of keys, queries, values).

Multi-Head Attention

Abstractive summarization using a multi-headed attention mechanism is a technique that combines the self-attention mechanism with multiple heads to generate a more informative summary. Here's how it works, along with an example:

Input Document: “The COVID-19 pandemic has affected many aspects of people's lives. One of the most significant changes has been the move towards remote work and online learning. With many schools and businesses closed or operating at reduced capacity, people have had to adapt to new ways of working and studying. In addition, the pandemic has highlighted the importance of healthcare and the need for better preparedness for future outbreaks.”

Step 1: Embedding Layer

The input document is first encoded into a sequence of embeddings using a pre-trained transformer model.

Step 2: Multi-Headed Attention

Next, the multi-headed attention mechanism is applied to the sequence of embeddings. Let's assume that we are using four attention heads, each of which generates its own set of key, query, and value vectors. Here's how the multi-headed attention works for the first attention head:

Query Vector: The query vector for "The" is a learned linear transformation of the embedding for "The".

Key Vector: The key vector for "The" is a learned linear transformation of the embedding for all the other words in the sequence.

Value Vector: The value vector for "The" is a learned linear transformation of the embedding for all the other words in the sequence.

The second, third, and fourth attention heads operate similarly but attend to different parts of the input sequence.

Step 3: Concatenation and Linear Projection

The output vectors from each attention head are concatenated into a single vector, which is then projected into a lower-dimensional space using a learned weight matrix.

Step 4: Decoding and Text Generation

The summary vector is fed into a decoder network that generates the abstractive summary. The decoder network is typically a language model that generates text word by word, based on the summary vector and the previously generated words. The output of the decoder at each time step is used as input for the next time step until the end-of-sequence token is generated.

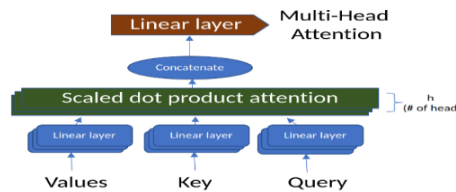


Figure 2. Multi-Head attention with ‘h’ layers

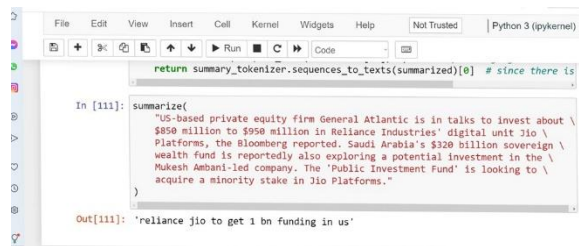
By applying multi-headed attention to the input document, the transformer-based model can learn to attend to different parts of the document and generate a more informative summary that incorporates information from all attention heads. The simpler version of figure 2, but with h number of heads is shown in figure 2.

C. Experiment's Evaluation Factors:

The three algorithms are evaluated in accordance with the proposed model according to the following Factors:

- Accuracy
- Performance
- Execution Time

VI. EXPERIMENTAL RESULT & DISCUSSION



```

return summary_tokenizer.sequences_to_texts(summarized)[0] # since there is

In [111]: summarize(
    "US-based private equity firm General Atlantic is in talks to invest about \
    $850 million to $950 million in Reliance Industries' digital unit Jio \
    Platforms, the Bloomberg reported. Saudi Arabia's $320 billion sovereign \
    wealth fund is reportedly also exploring a potential investment in the \
    Mukesh Ambani-led company. The 'Public Investment Fund' is looking to \
    acquire a minority stake in Jio Platforms."
)

Out[111]: 'reliance jio to get 1 bn funding in us'

```

Figure 3: Abstractive Summarization Result

VII CONCLUSION AND FUTURE WORK

Text summarization can be used by personal or specialized assistants, that would be the ultimate use case. Apart from that it can be used for many personalized devices or applications. Mail clients, report generation, a news feed, etc. The critical point is that NLP makes Business Intelligence more accessible and diverse. Suppose you need to find out the current trend of the industry from various new articles. But you have barely enough time to even glimpse through the headlines, let alone read all of those to reach the crux of their ideas. In that case, a text summarization can even help you extract the summary of multiple news articles. Financial and investment decisions require meticulous research and sorting through a large amount of content. It is both true for individual investors and investment corporations. Automatic text summarization designed for analyzing and condensing financial documents can help you out in this regard. For instance, people who are not familiar with extracting data from financial graphs or data can use the help of automatic text summarization to create a well-rounded summary of those financial visualizations.

Nearly every website has some built-in chatbots that pop up on the website's homepage. The idea behind these chatbots is to provide users with uninterrupted services without human intervention. These chatbots rely on natural language processing (NLP) and machine learning for auto-text summarization.

Businesses develop various resource materials such as e-books, newsletters, blogs, or white papers to reach out to their audience. However, these lengthy documents will get more chances to reach their target audience if they are broken down into short summaries by automatic text summarization. That way, a user can readily understand if the content will be worth reading. Besides, these short excerpts are more suitable for circulating through various social media platforms. Text summarization doesn't only work to create text-to-text summaries; it can also be used for speech-to-text summarization. And in this post-COVID remote working culture, that's a huge relief for employees around the globe. For instance, if you need meeting notes from a virtual conference, you can get a speech-to-text summarization application to do that for you. It will be more efficient than having to listen to that audio record over and over to develop a summary manually.

REFERENCES

- [1] HaoranLi, "Self-Attention Guided Copy Mechanism for Abstractive Summarization", 2020
- [2] Roshan Sharma, "End-to-end speech summarization using restricted self-attention", 2022
- [3] Jen-TzungChien, "Attention in Sequential Learning for Summarization", 2019
- [4] YujiaXie, "Conditional self-attention for query-based summarization", 2020
- [5] N Moratanch, "A survey on abstractive text summarization", 2016