Text Summarization using Deep Learning

1st Nisarg Patel
CSE Department
ITNU
Ahmedabad, Gujarat
21bce211@nirmauni.ac.in

2nd Isha Patel
CSE Department
ITNU
Ahmedabad, Gujarat
21bce196@nirmauni.ac.in

3rd Mulya Patel
CSE Department
ITNU
Ahmedabad, Gujarat
21bce208@nirmauni.ac.in

***Abstract*—These days, a better system must be in place for efficiently and rapidly extracting information from the vast amount of information that is available online. Manually removing the summary from a lengthy text document is very challenging for humans. There is a lot of text content available online. Because of this, finding pertinent documents among the many available and extracting pertinent information from them can be difficult. Automatic text summarization is crucial to resolving the two issues mentioned earlier. Text summarization is the process of removing the most crucial and helpful information from a document or a group of related documents, and condensing it into a condensed version without compromising the text's main ideas.**

***Index Terms*—text summarization, LSTM, Attention Mechanism**

I. Introduction

We must first understand what a summary is before moving on to the text summarization. A summary is a text that is condensed from one or more texts that still communicates the key ideas from the original text. The objective of automatic text summarization is to condense the original text into a more manageable and semantically sound version. Utilizing a summary has the primary benefit of cutting down on reading time. Extractive and abstractive summarization are two categories into which text summarization techniques can be divided. Choosing key phrases, paragraphs, etc. from the source material and concatenating them into a shorter form is the process of extractive summarization.

Text summaries can be divided into two categories: indicative and informative. The user is only presented with the primary idea of the text through inductive summarization. This kind of summary usually takes up five to ten percent of the original text. Conversely, the informative summarization systems provide a succinct overview of the primary text. An informative summary should not exceed twenty to thirty percent of the original text.

II. Related Work

By emphasizing the key aspects of the text, a lengthy text document can be condensed into a clear, concise summary that highlights the key details and important concepts of the original text. This process is known as text summarising.

For text summarising, two distinct methods are employed:

- Extractive Summarization
- Abstractive Summarization

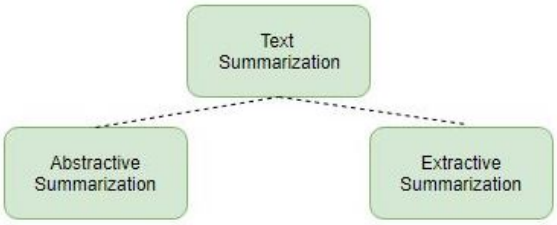


Fig. 1. Text Summarization Classification

A. Extractive Summarization

The name suggests what this method accomplishes. We extract only those significant sentences or phrases from the source text after identifying them. These selected phrases would serve as our synopsis. The graphic that follows shows extractive summarization: [5]

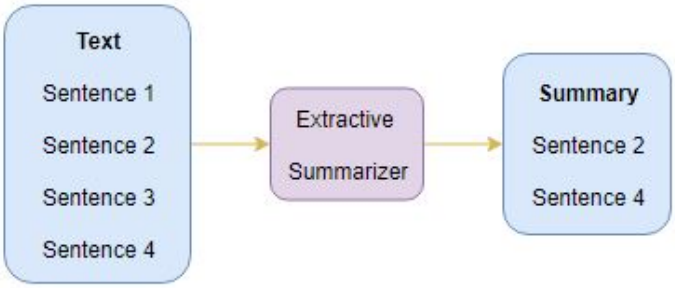


Fig. 2. Extractive Summarization

B. Abstractive Summarization

This strategy is intriguing. Here, we use the original content to create new sentences. Earlier, we saw an extractive technique where we used only the sentences that were present. This is different. The following sentences from the abstractive summarization may be absent from the original text:

III. Overview of preprocessing techniques

In Natural language processing, we cannot just use raw inputs as it is and expect to get a good model out of them. Text preprocessing is, therefore, a very crucial step whenever we are concerned about training an NLP model. There are

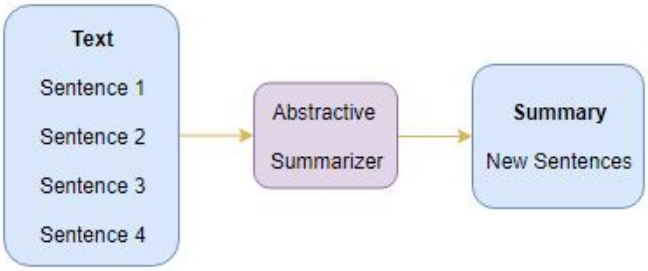


Fig. 3. Abstractive Summarization

various preprocessing techniques in the field of NLP. In our model we have used the following techniques:

- **Dropping Null and duplicate values**: The first very important step is to remove all the redundant data points from the dataset. Redundant data points can lead to misinterpretation by the model and hence reduce the effectiveness of the model. Therefore it is a good practice to remove all such values beforehand. In our project, we performed the same by writing the below-mentioned lines.

```
data.drop_duplicates(subset=['Text'],inplace=True)#dropping duplicates
data.dropna(axis=0,inplace=True)#dropping na
```

- **Converting to lowercase**: We do this in order to avoid considering two words with different cases as different. For eg: if we do not convert all the words to lowercase, the words ”Natural” and ”natural” will be considered two different words with different vocabulary indexes. Hence, we convert all the words to lowercase.
- **Remove HTML tags**: Here, removing HTML tags won't make much of a difference as this is a food reviews dataset. But as this is a good practice we did that too.
- **Contraction mapping**: Contraction mapping means converting words that are slang or not exactly in their root form to their original form. For eg: converting the word ”Isn't” to ”is not”.
- **Removing's and parenthesis text**: apostrophe(') symbols usually don't contribute anything to the meaning of the text and removing them does not necessarily change the meaning of the sentence. Therefore, it stands of no use to us and hence removing it is justified. Similarly, text within parenthesis is also removed for similar reasons.
- **Eliminate punctuations and special characters**: Removing punctuations and special characters helps in avoiding noise in the dataset, allowing the model to focus on useful text and perform effectively.
- **Removing stopwords and short words**: Stop words like a, an, the, is, a, etc. are those types of words that contribute absolutely nothing to the meaning of the sentence, and removing them also reduces the length of the texts by a significant amount. This is a major preprocessing step and is always done to develop a good model.
- **Understanding the distribution of data and limiting the size of documents**: In our dataset, the reviews posted vary greatly in length. Hence, to limit our dataset to a known range, we analyzed the distribution of data concerning its length and only considered those documents whose lengths were in the range that we specified to handle our data effectively.
- **Preparing a tokenizer**: Even after all these preprocessing techniques, the main problem remains which is, how to feed the words to a neural network that only understands numbers. For that, we prepare a tokenizer to convert these words to numbers. If you want to learn more about tokenizers and their working, do check out.
- **Rarewords and its coverage**: To avoid making too large of a vocabulary, we only considered the top common words that were present in the dataset. To implement this, we calculated several rare words and its coverage and then excluded them from the vocabulary.

IV. Seq2Seq (Sequence-to-Sequence) Modeling

To understand the seq2seq model, we first need to understand why a simple neural network is not good enough for this project. In NLP, we usually get a sentence as input or as output. And in a sentence, the order in which words occur matters. [3] For eg: consider the sentences

`"I am going to the village to sell my land along with my wife"`

and

`"I am going to the village along with my wife to sell my land"`

Now, It is visible that these two sentences have the same words but the meanings of both sentences are quite different. Therefore, we must develop such a model which understands the order in which the words are given which is not possible in ANN (artificial neural network) as even if we rearrange the order of neurons in the input layer the output won't differ. Hence, we use Seq2Seq model for such purpose.

In a Seq2Seq model, we usually provide words as an input, one at a time, and let it progress through the network and then we input another word and the feedback of the previous timestamp, i.e., stuff learned up till the previous word. In this way, we make sure that the model understands the order in which the words are given. [1]In our problem statement, we desire to make a text summarize, so we give multiple words as input and get multiple words as output as well. Hence, we will use the Many to Many Seq2Seq model. Such a model follows the Encoder-Decoder architecture. We will understand this architecture in detail in the next section.

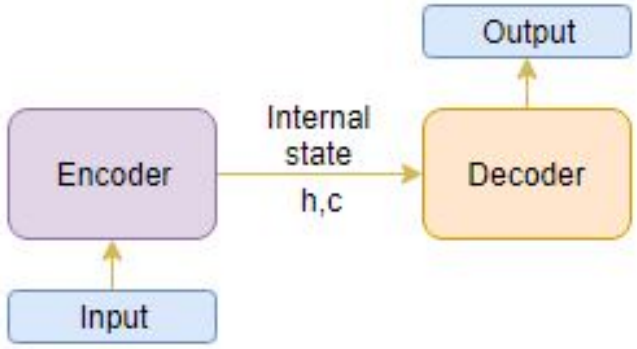
V. Encoder-Decoder Architecture

Many Applications like Machine Translation and Text Summarization have variable length inputs and outputs, in these

types of problem Statements we mostly use encoder-decoder architecture.

Enocoder-Decoder uses a latent space representation which consists of all the data that is learned by the model, the architecture just takes the input and encodes it to the latent space representation and then it will decode it to the original format which is inputted to the Model.

Concerning text summarization, let's analyze this. A condensed version of the lengthy word sequence that makes up the input will be produced. [3]



Mostly in the encoder and decoder side instead of using RNN (recurrent Neural Network), we used mainly LSTM(Long short-term memory) or GRU(Gated Recurrent Unit). The main advantage of using these two is they can retain long-term dependencies because they can overcome the vanishing gradient problem.

We can set up the Encoder-Decoder in 2 phases:

- Training phase
- Inference phase

Let's understand these concepts through the lens of an LSTM model.

VI. Training phase

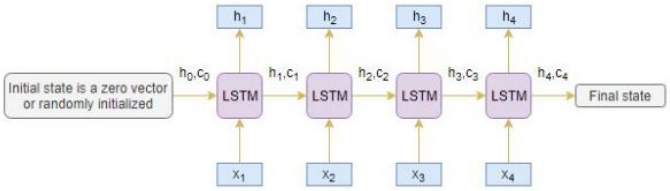
Here in the training phase, we'll talk about how to encoder and set up, we will train the model in such a way that it will take one word at a time, taking one word is technically known as one timestamp, Let see the working of encoder and decoder in more depth

A. Encoder

The complete input sequence is read by an encoder long short-term memory model (LSTM), which receives one word at a timestep. After that, it analyses the data at each timestep and extracts the contextual data from the input sequence.

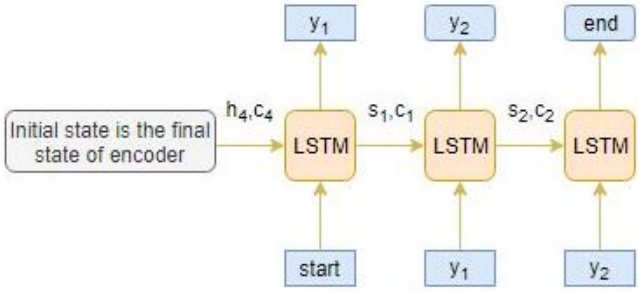
I created the diagram that follows to show this procedure:

The decoder is initialized using the cell state (ci) and hidden state (hi) of the previous time step. Recall that the reason for this is that the LSTM architecture consists of two distinct sets: the encoder and the decoder. [3]



B. Decoder

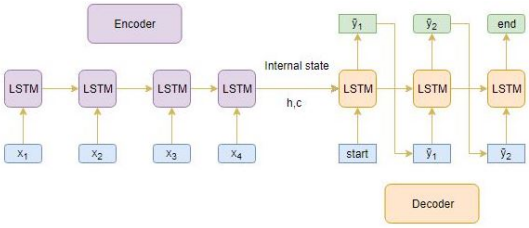
Additionally, the decoder is an LSTM network that predicts the same sequence offset by one timestep after reading the target sequence word-for-word. Given the preceding word in the sequence, the decoder is trained to anticipate the word that comes after it.



<start> and **<end>** are start and end tokens respectively. They mark the start and end of a sentence. Whenever the model is trying to predict a summary, it starts with the start token and tries to predict the next word after that. It keeps repeating this procedure till it receives the end token as the next word. Whenever an end token is encountered in such a way, it means that the sentence is ended and no more words need to be predicted. In simpler words, the summary is ready.

VII. Inference Phase

In the training phase, we were providing the output to the model for it to learn. But, after training, we need our model to run accurately on unseen data whose output is not provided to it. That's where the inference model comes into role. An inference model is very similar to training model, just that now it aims to predict the correct output instead of learning based on the given input-output sequences. Now, let's dive deep into the working of the inference model. [2]



A. Working of Inference model

The inference model works slightly differently than the training model. It follows the following steps to predict the output.

- The encoder part of both phases is the same. Hence, the first step is to receive the input words one by one just like in training model.
- After encoding is done, the model starts with <start>token. This means that now the model will start decoding the summary sentence.
- Given the start token, it will now try to predict the next word with the help of trained weights which were a result of the training model. [4]
- Therefore, in one timestamp, it predicts the probabilities of the next word and then using the softmax function, decides the next word to consider in the summary.
- Then this sampled word is passed as an input in the next timestamp along with the activation and cell states and then based on that, the next word is predicted.
- Now these steps are repeated until it gets the end token as an output. That's where it stops and the summary prediction is complete.

Consider this example to understand this thoroughly. Let's say, the input sequence is [x1,x2,x3,x4]. So now, till 4 timestamps, the encoder component will work as it is and h4 and c4 will be generated which will be provided as an input to the first layer of the decoder component. Hence, the first layer will take the start token and h4,c4 as inputs and then it will try to predict the first word from it and also provide the updated hidden state (s1) and cell state (c1) as output. Now, this predicted word will be given as an input in the next timestamp along with s1 and c1. Fig-1, Fig-2, and Fig-3 show 3 timestamps of the decoder component of the inference phase. Observe how at each timestamp, it predicts the output word. [2]

VIII. Limitations of the Encoder-Decoder Architecture

The whole concept of the encoder-decoder architecture is to first understand the input thoroughly and then try to predict the output from whatever it understood. But this requires a lot of "remembering". That is, the model should be able to understand and remember the words that it has previously seen. Now, since we are using LSTM, it solves this problem to an extent. However, the problem persists when the length of the input or output sequences becomes large. [5]

The effectiveness of this architecture, therefore, collapses exponentially when the length of the data is increased. So, to overcome this problem, we use something called the Attention mechanism in our model. The attention mechanism, as the name suggests, helps the model to figure out which parts of the sequence to focus on in order to predict the next output. Hence, it eases the process by letting the model focus on only useful stuff rather than making it consider everything.

Timestamp: t0

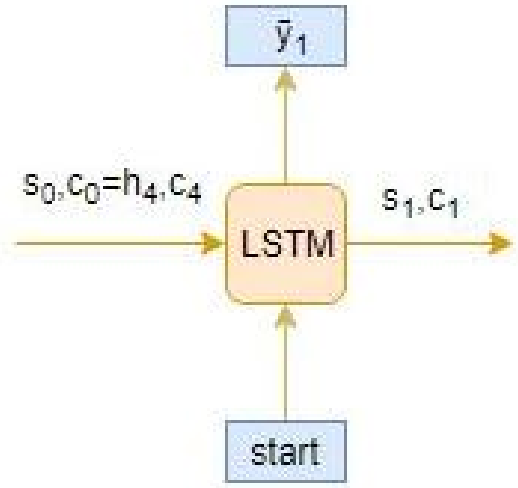


Fig-1

Timestamp: t1

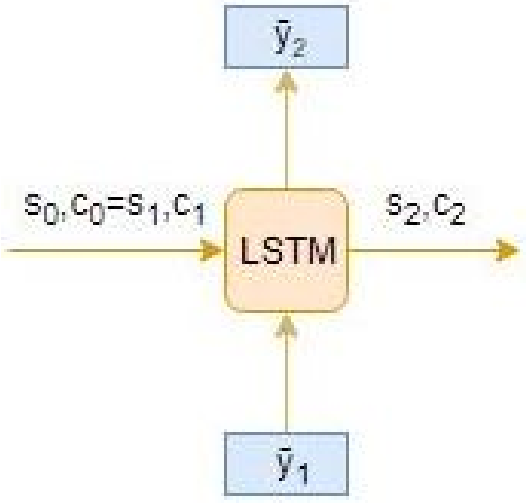


Fig-2

IX. The Attention mechanism

The attention mechanism helps the model decide how much "attention" needs to be provided to which parts of the input sequence.

for example: if I consider a sentence "Don't eat the delicious looking and smelling pizza" and try to convert it into Hindi language, then the start of the sentence in Hindi will be from the phrase which is the direct translation of "the delicious looking" phrase in English. Therefore, while predicting the first word in the output sequence, the model needs to pay more attention to the "delicious looking" phrase

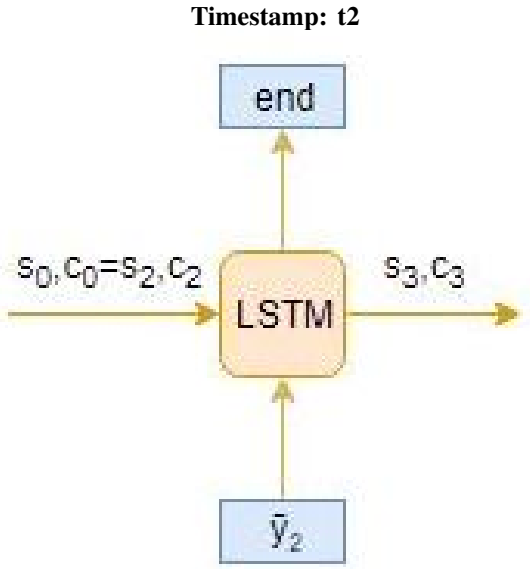


Fig-3

rather than the first word ”Don’t”. So, if we deploy the attention mechanism, the model will understand that it needs to pay more attention to that particular phrase.

The attention mechanism works in the following way:

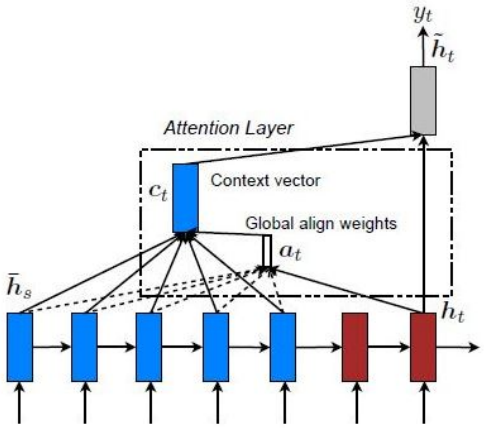
- It first calculates the similarity scores of the hidden state of the decoder timestamp and the hidden state of each of the encoder timestamps.
- Then, it uses the softmax activation function to scale all of those scores in the range of 0 to 1.
- Then, it multiplies these softmax scores to the corresponding hidden state values of the encoder.
- Lastly, it feeds all these values and the hidden state of the decoder into an FFNN, and then, using the softmax function again, it decides which word to use next.

Therefore, the attention mechanism has the following advantages:

- Firstly, it passes more data from the encoder to the decoder. Without the attention mechanism, it only passes a single hidden and cell state, whereas, by using the attention mechanism we pass the hidden and cell state of each of the encoder timestamps.
- And secondly, it introduces an additional step before producing the output of the decoder.

X. Conclusion

Hence Text summarizes can be useful tools in many domains like Question-answering chatbots and in other related fields like information retrieval systems to get short and crisp data having a good amount of accurate information in short text summaries, can be used in data extraction for various research papers and can be used for educational purposes. The use of LSTM and GRUs makes this model more efficient and



helps to retain long-term dependencies such that important information doesn’t get ignored while giving the output.

References

[1] Ishitva Awasthi, Kuntal Gupta, Prabjot Singh Bhogal, Sahejpreet Singh Anand, and Piyush Kumar Soni. Natural language processing (nlp) based text summarization - a survey. In *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pages 1310–1317, 2021.

[2] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[3] Oguzhan Tas and Farzad Kiyani. A survey automatic text summarization. *PressAcademia Procedia*, 5(1):205–213, 2007.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[5] Adhika Pramita Widyassari, Supriadi Rustad, Guruh Fajar Shidik, Edi Noersasongko, Abdul Syukur, Affandy Affandy, et al. Review of automatic text summarization techniques & methods. *Journal of King Saud University-Computer and Information Sciences*, 34(4):1029–1046, 2022.