Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report

On

"Text Summarizer"

For partial fulfilment of the course *"***COMP 206***"*
CE Second Year/ First Semester in Computer Science/Engineering

Submitted by

Utsav Paudel(39)
Santosh Shrestha(46)
Santosh Subedi(50)
Achyut Thapa(52)

Submitted to

Nabin Ghimire
Lecturer
Department of Computer Science & Engineering (DoCSE)
10th March 2020

Bona fide Certificate

This project work on "**Text Summarization**" titled as "**Text Summarizer**" is the bona fide work of Utsav Paudel, Santosh Shrestha, Santosh Subedi and Achyut Thapa, who carried out the project work under my supervision**.**

Project Supervisor:

_____

Dr. Prakash Poudyal
Lecturer
Department of Computer Science and Engineering
Kathmandu University

ACKNOWLEDGEMENT

First and foremost, with our sincere regards, we would like to thank the Department of Computer Science and Engineering (DOCSE), Kathmandu University (KU), School of Engineering (SOE) for including the project COMP 206 in our syllabus. We are also heartily thankful to our Project Supervisor Dr. Prakash Poudyal for giving us this wonderful opportunity to explore our abilities and knowledge in the field of computing. It would not have been possible without his kind support and help in every process. Also, we would like to thank Mr. Nabin Ghimire for providing all the necessary logistics and proper management of the program so we could do this program without the burden of any logistics.

Sincerely,

Utsav Paudel(39)
Santosh Shrestha(46)
Santosh Subedi(50)
Achyut Thapa(52)

ABSTRACT

Text Summarization is the process of extracting salient information from the source text and to present that information to the user in the form of summary. Reading large documents of text & manually summarizing it will require more time & effort, so the proposed system will provide extractive as well as abstractive summary of a document automatically within some amount of time. Thus, it improves two factors namely, time consumption & efficiency. The technique involves conversion of input document to vectors, then we assign weight to every sentence according to features such as length of sentence, position of sentence, count of that word in the given document. From the above information, the system will return an extractive summary. The output of the extracted summary will be given as input to the abstractive module which will return abstracted summary. We will use encoder -decoder model, the encoder RNN first read the source string word-by-word encoding the information in the hidden state & passing the context forward on complete pass the encoder of the string which captures all the information & context of the input text. The decoder is another RNN which learns to decode the vectors into output sequence. Now the attention model which calculates the importance of each input encoding for the current state by doing a similarity check between decoder output at this step & input encoding. Thus, the model will produce the abstractive summary.

Keywords —Extractive, Abstractive, Natural language processing, Attention Mechanism

List of Figures:

Abbreviations

seq2seq- Sequence to Sequence

LSTM- Long Short Term Memory

CN- ConceptNet

RNN- Recurrent Neural Network

UNK- Unknown

PAD- Padding

GPU- Graphics Processing Unit

Table of Contents

Chapter 1: Introduction

1.1 Background

In the field of text summarization, most follows two approaches i.e. Extractive and Abstractive. In the past, researchers used an extractive approach of text summarization where they identify the important sections of the text and generate the verbatim producing subset of the sentences from the original text. However, the actual gist of the text gets either mixed up or lost. After the introduction of deep learning concept, researchers followed an abstractive approach of text summarization where they used a pre-trained model for summarization that actually identifies the gist of the text and summarize it so that the theme or idea of the text is not lost and can be clearly understood by the user.

Nowadays, people use the internet to find information through information retrieval tools such as Google, Yahoo, Bing, Wikipedia and so on. Because of the increasing rate of data, people need to get meaningful information. So, it is not possible for users to read each document in order to find the useful one. Among all the modern technologies, text summarization has become an important and timely tool for the users to quickly understand the large volume of information. There are many text summarizers but the one developed by us helps to summarize the food reviews given by the people to various food companies. This helps the company to make its products better and fulfill the demands of the customers.

1.2 Objectives

- To make large section of text concise and meaningful
- Optimize the readability
- Summarizing the text so as to save time
- Summarization of reviews to increase the efficiency of the company

1.3 Motivation and Significance

As we know there are many long and boring texts or articles that we want to read but get carried away due to the size of the text. Due to this, the main gist or idea of the text gets lost and readers simply leave reading without understanding the content of the text. To solve this problem and make reading interesting and effective, 'Text Summarizer' makes it short and sweet without losing the main idea of the text. This will not only help the users to understand the text but also influence readers to read more articles without the consumption of more time. Reading will be fun again.

We have developed a 'Text Summarizer' app based on 'Abstractive Text Summarization' that at present summarizes the food reviews using 'Recurrent Neural Network' so as to help the food company. This helps in generating the gist of the reviews given by the consumers or say customers that will help the food companies to incorporate the ideas given by the customers as well as to enhance their product and solve the varieties of problems of their product as suggested by the customers. This will save time and helps the company to reach out to a large section of customers.

Chapter 2: Literature Review

Text Summarizing is not an easy task especially when a text is very big. However, we found from our research that most of the application was unmaintained and also the information was not valid but at the same time, we also found few applications, web as well as android done in this field that truly is good and among those applications that serve similar purpose as ours are as follows:

2.1 Related Works

a) http://textsummarization.net/text-summarizer (11 November 2019):

This is an online platform to summarize any text based on the keywords, and can summarize a text in any number of lines. However, this summarizer is not useful for those who do not know the technical size of summarized text, the user has to provide manually to summarize in a certain number of lines.

b) Text Compactor (https://www.textcompactor.com/) (11 November 2019)

This is a platform where users can write or paste text from some source. Users need to set a value here as well for how much summarized text they need.

c) MeaningCloud Summarization is language independent and extracts a summary for a given document by selecting the most relevant sentences in it. It charges some amount per summarization after 40000 requests of it.

d) Open Text summarizer (https://www.splitbrain.org/services/ots) (11 November 2019)

The tool automatically analyzes texts in various languages and tries to identify the most important parts of the text.

Chapter 3: Design and Implementation

We built a seq2seq model that can create relevant summaries for reviews written about fine foods sold on Amazon. This dataset contains above 500,000 reviews, and is hosted on Kaggle.

To build the model we used a two-layered bidirectional RNN with LSTMs on the input data and two layers, each with an LSTM using bahdanau attention on the target data. This model used Conceptnet Numberbatch's pre-trained word vectors.

The sections of this project are:

- Inspecting the Data
- Preparing the Data
- Building the Model
- Training the Model
- Making the Summaries

Preparing the Data

- Convert to lowercase.
- Replace contractions with their longer forms.
- Remove any unwanted characters (this step was done after replacing the contractions because apostrophes would be removed. Notice the backward slash before the hyphen. Without this slash, all characters that are ‘between’ the characters before and after the hyphen would be removed. This created some unwanted effects like typing “a-d” would remove a,b,c,d.).
- Stopwords were not very relevant in training the model, so by removing them we were able to train the model faster because of less data. They remain in the summaries because they were short and we preferred them to sound more like natural phrases.

Word Embeddings

We used pre-trained word vectors to improve the performance of the model. We used GloVe for this at first, but we found another set of word embeddings, called ConceptNet Numberbatch (CN) better. Based on the work of its creators, it seemed to outperform GloVe, which makes sense because CN is an ensemble of embeddings that includes GloVe.

We limited our vocabulary to words that are either in CN or occurred more than 20 times in the dataset. This allowed us to have very good embeddings for every word because the model can better understand how words are related when they see them more times.

To train the model faster, we sorted the reviews by the length of the descriptions from shortest to longest. This helped each batch to have descriptions of similar lengths, which resulted in less padding, thus less computing. We could have done a secondary sort by the length of the summaries, but this would result in quite a bit of looping to organize the data. Plus, we doubted that it would reduce much extra padding since the summaries are rather short.

Some reviews would not be included because of the number of UNK tokens that are in the description or summary. If there was more than 1 UNK in the description or any UNKs in the summary, the review would not be used. This was done to ensure that we were building the model with meaningful data. Less than 0.7% of words are UNKs, so only few reviews were removed.

Building the Model

We used a Neural Architecture called **sequence-to-sequence(seq2seq)** with Attention Mechanism.LSTMs for preservation of gradients

- **Sequence-to-sequence**

  Introduced for the first time in 2014 by Google, a sequence to sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ.

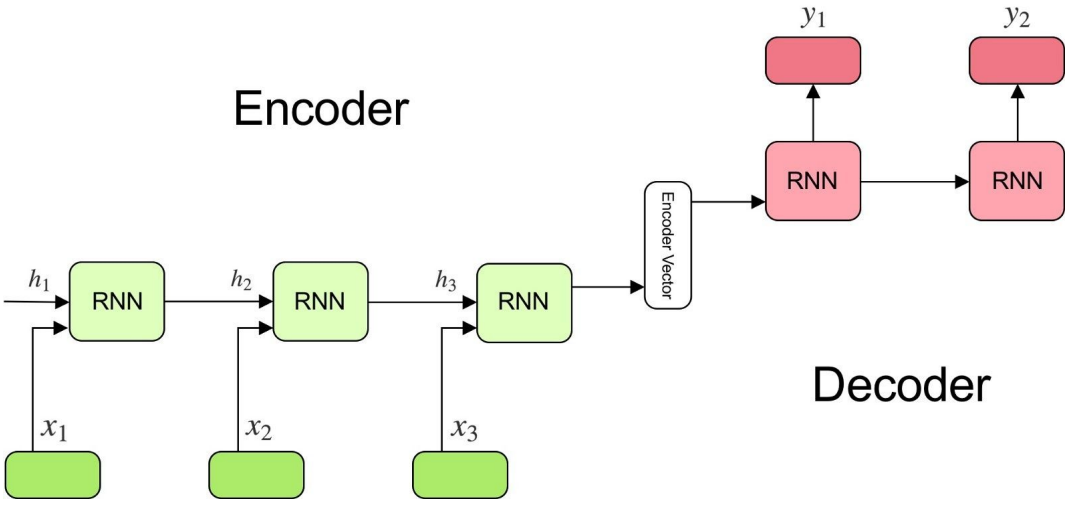


Fig: seq2seq model

- **RNN**

The invention of the Perceptron back in the 50's by Frank Rosenblatt was controversial. It was the ancestor of today's gigantic Neural Models, a single neuron inspired by neurons in the brain. That magical model was expected to be "the embryo of an electronic computer that expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." Olazaran, 1996 Despite the fact that the per- ceptron was not able to walk and talk, but Neural Network has not been abandoned forever. The later architectures achieved better performance by using a combination of perceptrons in connected layers. Among the revolutionary models, around the years of the 1980s inspired by the sequential memory mechanism of the brain, Recurrent Neural Network (RNN) was proposed. An RNN tries to relate the elements of a sequence to each-other so that a network retains the memory of the previous data. To this end, RNN re-uses its output state and feeds it back to its hidden layer (Medsker and Jain, 2001). Using the hidden layers state might cause misunderstanding. To clarify, we compare the conventional feed-forward architecture with RNN. In a feed-forward setting, the inputs which are raw data inserted in the input layer. With forward propagation, the state of next layers will change one by one. If we insert another instance of the input in the input layer, the states of the hidden layer will completely change. RNN incorporates the concept of memory, picking the state of the hidden layer combines it with the current hidden input to make the current hidden state.

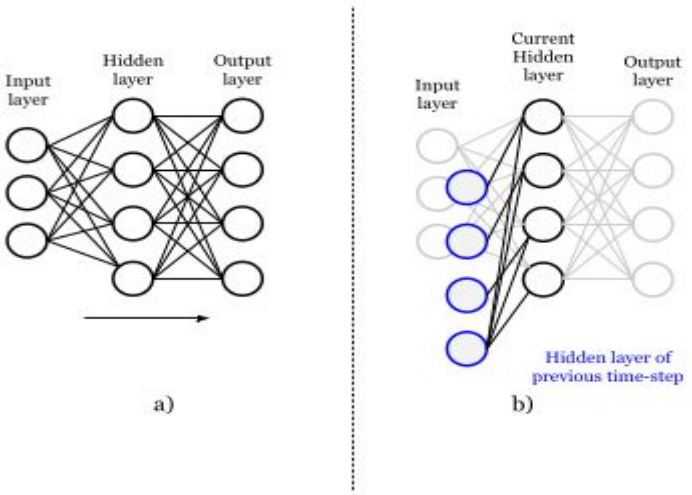


Fig:- a)Forward propagation in a Neural Network. b) Feeding previous hidden layer state to the current time-step hidden layer (for clarity just some of the connections are drawn)

Figure above illustrates how the hidden state of the previous time-step combines with the input to make the new hidden state. We repeat this recursive procedure for each and every element of the input sequence. To understand better you can imagine a network with as many hid- den layers as the elements of the sequence. This way of depicting the RNNs is called Back-propagation Through Time (BPTT).

- **Long Short Term Memory(LSTM)**

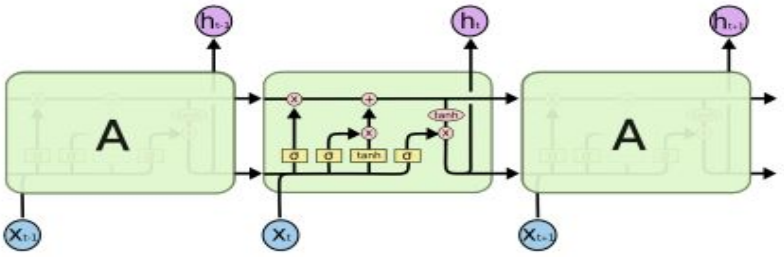


Fig:- A Basic LSTM Unit with its layers.

RNNs were a breakthrough in the field of Neural Networks. They opened the NNs to sequential data and introduced the concept of memory. But they had a few short- comings. We discussed that they are not able to learn unbalanced pairs of sequences in which the target length is different from the source. Even with the encoder- decoder configuration, they had very limited usage because of the lack of long-term memory. In each time-step an RNN loops back the state of its hidden layer. So in the next step, it has a memory of the past step. Imagine a sequence of length 10 as our input. in the 2nd step the RNN can remember the state 1. But as it progresses in the length of the sequence, each time a new input combines with the looped back state, so in step 10 the network has a memory of all the previous steps with the notion that they do not have the same strength. The first step almost dissolves after this long repetitive incoming of new inputs. Therefore an RNN can remember the immediate past better, or with a better interpretation, it has a short-term memory. Equipping these RNN architectures to long-term memory was the key idea of LSTM cells. Their benefit is that they can learn long-term memories. To understand the short and long- term memories consider this two sequences below:

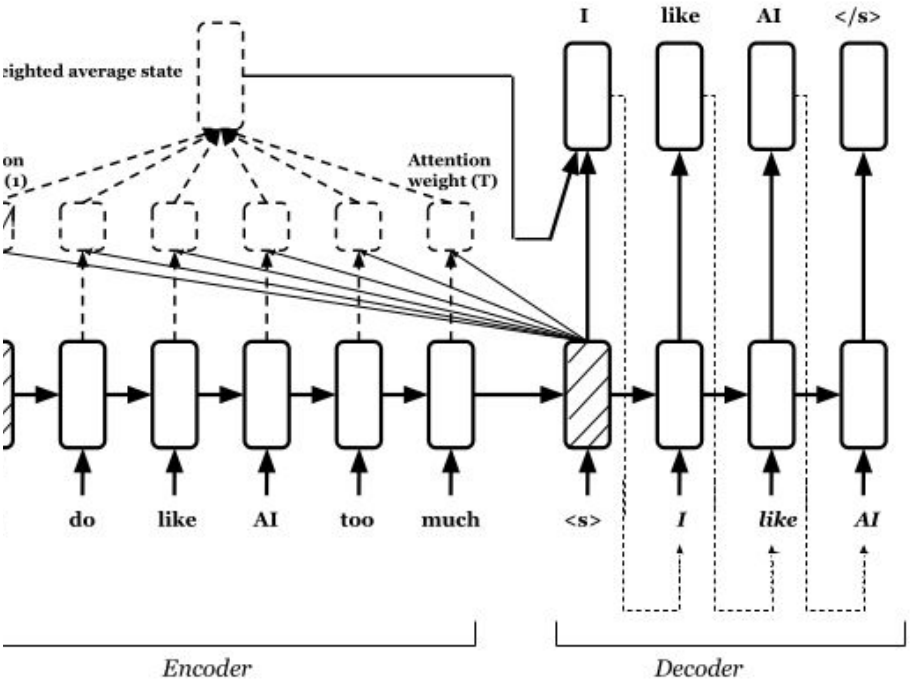
Sequence (1): x x x A B x x

Sequence (2): x A x x x B x

In sequence 1, the character "B" appears immediately after "A" (x elements could be any character). "AB" is a pattern in our sequence. By feeding more examples of data samples that include this pattern, the RNN memorizes that after "A" there must be a "B". It is a short-term memory. So if the model receives the "AB" pattern again it will remember its order and can predict the corresponding output. In sequence 2, the character "B" comes 4 time-steps after the character "A". A simple RNN directly copies the internal state of the previous state and combines it with the input. If the model receives new data during these 4 steps, they will destroy the memory of the "A" character. To learn a long pattern like "A x x x B" we need a new mechanism. LSTM cells are rather complex structures based on RNNs (Hochreiter and Schmid- huber, 1997). We just explain two main parts of them that are two handles for keeping or forgetting the memory. A line that all the states can freely pass across it till the last timestep, and a forget gate. The forget gate itself is controlled by a 1 simple feed forward network, usually one layer that is fed with the previous hidden state. By training these gates, the LSTM will learn when to keep, and when to forget. It all depends on the pattern of the input data. For any input pattern if it is more frequent in the dataset, the network will learn to keep its memory. LSTM, unlike the RNN, does it regardless of the length of pattern, long or short term memories will retain if they are more frequent.

- **Improving Seq2seq Model with Attention Mechanism**

The performance of sequential models improved to a higher extent by when they were equipped to the attention mechanism. A seq2seq model is combined with an en- coder and a decoder. The decoder receives all the contents of the encoder com- pressed in one vector and trains the target with its presence. It is a very clever idea, however, when the source sequence is too long, its content will fade in the coded vector. It makes sense to think that hypothetically the average vector of two big enough corpus are very similar to each-other. To solve this problem, attention mechanism tries to get multiple encoded vectors from the source and not only one. This way with a very long sequence, the seq2seq model can align the source and tar- get sequences, while retaining the most important contents of the source. Figure 4.1 shows how a seq2seq model focuses on the different parts of the source sequence. The procedure is as below: The encoder processes the input sequence till the last el- ement. Decoder copies the hidden state of the encoder in the last step to its first timestep. It compares this new state with all the hidden states of the encoder to find their similarity. The hidden states which are more similar will get a higher weight. The decoder then uses the weighted average states of the encoder and combines it with its current state. With the new hidden state, decoder can infer the first output. It repeats this procedure for all the target timesteps. The rest of the procedure is like a conventional seq2seq model. The alignment of source and target sequences can be illustrated with a matrix. Figure 4.2 shows a sample of this alignment; the first element (I) has the most sim- ilarity, therefore will get the highest attention weight, while the "I" and "AI" has no semantic correlation, so the comparison weight is close to zero.



Fig;-4.1 :- Seq2Seq model with attention

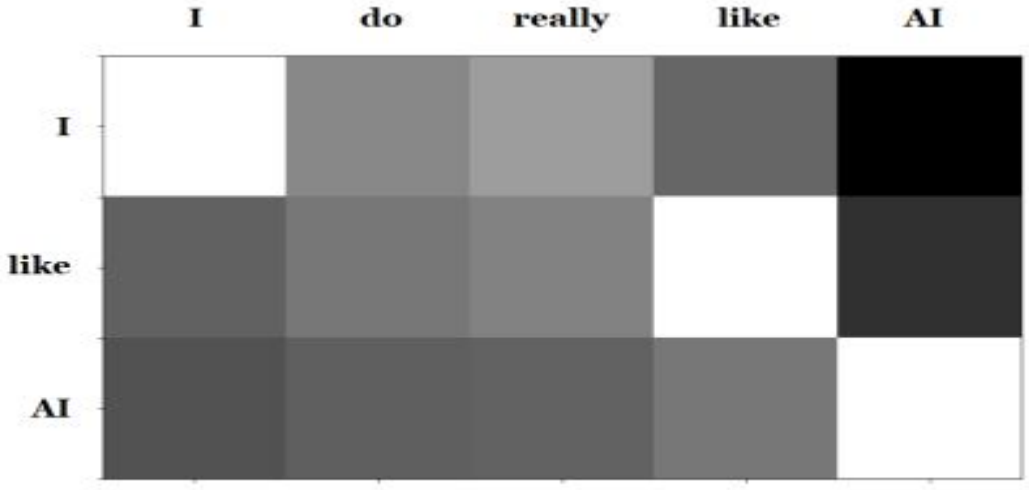


Fig:-4.2 :- Alignment of source and target sequences

- **Handling Varying Sequence Length**

Because input and output sequence lengths vary, we standardized all inputs to be some max length by padding short sentences with PAD tokens and cutting off longer sentences. This padding was taken into account in both the scoring and loss functions, preventing the model from updating parameters that should not be updated.

- **Encoding Layer**

  To build the encoding layer, we used a bidirectional RNN with LSTMs

- **Decoding Layer**

  Just a two layer LSTM with dropout.

- **Attention**

  We used Bhadanau instead of Luong for the attention style which helped the model to train faster and produce better results.

Training the Model

We Trained our model with only a subset of 50,000 reviews on ***GOOGLE COLAB.***

- **Google Colab**

  Google Colab is a free cloud service and now it supports free GPU! We can develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and OpenCV.

3.1. System Requirement Specification

3.1.1. Software Specification

3.1.1.1. Operating System: Linux, Mac, Windows (64-bit)

3.1.1.2. Python 3.7 Library Dependencies:

Python Library	**Version**
TensorFlow	1.15.0
NumPy	1.13.3
Pandas	1.14.0
NLTK	3.4.5
Pip	19.3.2

3.1.2. Hardware Specification

3.1.2.1. Training:

- **GPU:** At least 8 GB of available RAM.
- **System Memory:** Depends on the dataset.

3.1.2.1. Inference:

- **CPU:** 64-bit based processor (Modern CPU recommended).
- **RAM:** About 6 GB of RAM Recommended.
- **System Memory:** 2 GB of free space.

Chapter 4: Discussions and achievements

- **Observation**

  - Choosing the right tuning strategy is very important. Tuning a deep learning model is very tricky, but following a few rules of thumb will reduce its time . In this work we used a greedy approach.First tip is to do the fastest experiments first. Start from some small models with low complexity e.g. few number of layers and small hidden layer size. When you get a good training accuracy and overfits the dataset, and if the validation accuracy is not following the training accuracy, use a regularization technique. Having the proper model-size you can tune other hyper-parameters to achieve the best possible model.Our model ROUGE Score is shown below

```
Precision is :0.23809523809523808
Recall is :0.38461538461538464
F Score is :0.2941185998269972
Sum of ROUGE Score:  12.640978813379
Average ROUGE Score =  0.36117082323940003
Count:  35
```

Fig-Rogue Score

  - Abstractive Text Summarization is a more challenging problem compared to it's extractive counterpart. When dataset samples are too small, it carries more abstractive content. Sometimes the summaries tokens are not available in the source. It makes the prediction very difficult and as a result, our summarizer systems do not show good

per-formance measures on them. Amazon dataset is an example which gave us a very lower Rouge score in comparison with CNN DM dataset.

- Removing stop words actually reduces the input noise to the model. We got better results from the Amazon dataset without stopwords. The stopwords can be more harmful in a small dataset. As they are very frequent and the model can overfit on them.

- Rouge scores so far have been the golden standard to evaluate summarizer systems. Rouge is a statistical metric that works by comparing the frequency of tokens in source and target. Consequently, It cannot evaluate the grammatical and semantic correctness of the outputs. It also fails to deal with Out Of Dictionary Words. A better metric that can evaluate a summarizer system from all aspects is needed.

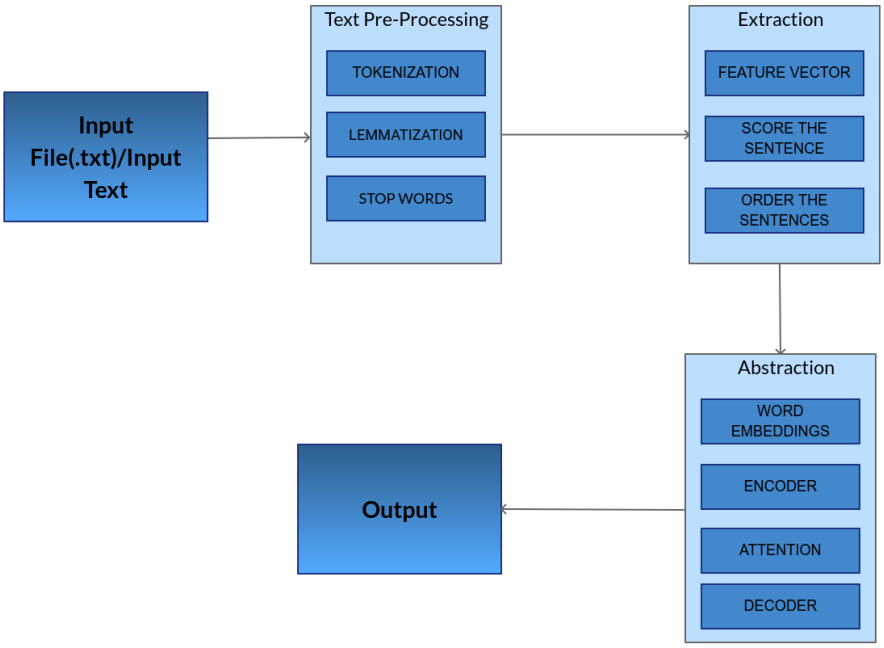


Fig:System Architecture

Chapter 5: Conclusion and Recommendation

5.1 Conclusion

Abstractive Text Summarization is a text-to-text problem, very close to translation, with a big difference that the source and target sequences are not properly aligned. Therefore, it demands sophisticated models and high-performance algorithms to deal with its complexity. We discussed that RNN networks as the first sequential models were not able to handle the non-aligned inputs, hence were not useful for summarizing tasks. Seq2seq neural architectures succeed the RNN models in many sequence-to-sequence problems including text summarization. They improved furthermore with the addition of attention mechanisms. The Transformer is a quite new model that works only with the disciplines of attention. It is relatively faster because of the removed Backpropagation Through Time that allows them to process long sequences in a shorter amount of time. Although its performance speed depends on the setting and dataset structure, so that in some cases it might perform slower. The model had shown state of the art performance on translation and summarization tasks. This architecture has a high potential to substitute previous successors. To make use of this potential, it's rather more complex architecture should be explored more in different problems and experiments with a variety of configurations. Comparing the results with the experiments on Amazon Reviews dataset showed that it's more complicated hyper-parameters should be precisely tuned depending on the characteristics of the dataset. A highly essential requirement for a summarizer model is a precise metric to direct the training process. So far Rough has been the most reliable metric for abstractive text summarization.

5.2 Limitation

Although we completed our project successfully, our project still has some faults or limitations as no system is perfect in itself. Some of the limitations of our project are listed as follows:

- ❖ This application is trained with amazon food review so that it can summarize only related to food.
- ❖ It requires better computational power like GPUs.
- ❖ It is not familiar with all types of computer users.
- ❖ The accuracy of our model is less than 90% .
- ❖ Focuses on single document summarization.

5.3 Future Enhancement:

There is always room for improvement in every project or system. Similarly, there are still some features that can be added and improved in the future. Some of them are as follows:

- This work focuses on single document summarization. It can be extended to multi-document.
- Multilingual summarization
- It can be developed for all kinds of computer users and readers.
- It can be implemented on different online platforms.
- More User Friendly interface can be implemented.

GANTT Chart

S.N.	Tasks			1	2	3	4	5	6	7	8	9	10	11	12	13
1.	Research on Different method of summarization															
2.	Research on Data processing															
3.	System Design															
4.	Coding															
5.	Testing and Debugging															
6.	Documentation															

References

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." [1409.0473] Neural Machine Translation by Jointly Learning to Align and Translate. N.p., 19 May 2016.

Jean, Sébastien, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. "On Using Very Large Target Vocabulary for Neural Machine Translation – Sampled Softmax."The Neural Perspective. N.p., 26 Oct. 2016.

Luong, Thang, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing(2015).

Nallapati, Ramesh, Bowen Zhou, Cicero Dos Santos, Caglar Gulcehre, and Bing Xiang. "Abstractive Text Summarization Using Sequence-to-sequence RNNs and Beyond."Proceedings of The 20th SIGNLL Conference on ComputationalNatural Language Learning (2016).

Rush, Alexander M., Sumit Chopra, and Jason Weston. "A Neural Attention Model for Abstractive Sentence Summarization." [1509.00685] A Neural Attention Model for Abstractive Sentence Summarization. N.p., 03 Sept. 2015.

Appendix:

```
Epoch  41/100 Batch  220/3559 - Loss:  1.543, Seconds: 2.46
Epoch  41/100 Batch  240/3559 - Loss:  1.579, Seconds: 2.72
Epoch  41/100 Batch  260/3559 - Loss:  1.537, Seconds: 2.73
Epoch  41/100 Batch  280/3559 - Loss:  1.372, Seconds: 2.21
Epoch  41/100 Batch  300/3559 - Loss:  1.436, Seconds: 2.55
Epoch  41/100 Batch  320/3559 - Loss:  1.466, Seconds: 2.90
Epoch  41/100 Batch  340/3559 - Loss:  1.650, Seconds: 2.82
Epoch  41/100 Batch  360/3559 - Loss:  1.680, Seconds: 2.87
Epoch  41/100 Batch  380/3559 - Loss:  1.556, Seconds: 2.75
Epoch  41/100 Batch  400/3559 - Loss:  1.557, Seconds: 3.01
Epoch  41/100 Batch  420/3559 - Loss:  1.388, Seconds: 2.65
Epoch  41/100 Batch  440/3559 - Loss:  1.523, Seconds: 2.55
Epoch  41/100 Batch  460/3559 - Loss:  1.457, Seconds: 2.48
Epoch  41/100 Batch  480/3559 - Loss:  1.590, Seconds: 2.70
Epoch  41/100 Batch  500/3559 - Loss:  1.572, Seconds: 3.16
Epoch  41/100 Batch  520/3559 - Loss:  1.522, Seconds: 3.06
Epoch  41/100 Batch  540/3559 - Loss:  1.506, Seconds: 2.36
Epoch  41/100 Batch  560/3559 - Loss:  1.456, Seconds: 3.00
Epoch  41/100 Batch  580/3559 - Loss:  1.565, Seconds: 2.66
Epoch  41/100 Batch  600/3559 - Loss:  1.577, Seconds: 2.85
Epoch  41/100 Batch  620/3559 - Loss:  1.815, Seconds: 3.20
Epoch  41/100 Batch  640/3559 - Loss:  1.788, Seconds: 2.35
Epoch  41/100 Batch  660/3559 - Loss:  1.644, Seconds: 2.60
Epoch  41/100 Batch  680/3559 - Loss:  1.563, Seconds: 2.72
Epoch  41/100 Batch  700/3559 - Loss:  1.475, Seconds: 3.41
Epoch  41/100 Batch  720/3559 - Loss:  1.427, Seconds: 3.00
Epoch  41/100 Batch  740/3559 - Loss:  1.601, Seconds: 2.52
Epoch  41/100 Batch  760/3559 - Loss:  1.704, Seconds: 3.08
Epoch  41/100 Batch  780/3559 - Loss:  1.625, Seconds: 2.70
Epoch  41/100 Batch  800/3559 - Loss:  1.573, Seconds: 2.40
Epoch  41/100 Batch  820/3559 - Loss:  1.395, Seconds: 2.84
Epoch  41/100 Batch  840/3559 - Loss:  1.530, Seconds: 2.76
Epoch  41/100 Batch  860/3559 - Loss:  1.571, Seconds: 3.38
Epoch  41/100 Batch  880/3559 - Loss:  1.607, Seconds: 3.13
Epoch  41/100 Batch  900/3559 - Loss:  1.551, Seconds: 2.53
Epoch  41/100 Batch  920/3559 - Loss:  1.523, Seconds: 2.46
Epoch  41/100 Batch  940/3559 - Loss:  1.479, Seconds: 2.92
Epoch  41/100 Batch  960/3559 - Loss:  1.461, Seconds: 3.01
Epoch  41/100 Batch  980/3559 - Loss:  1.565, Seconds: 2.60
Epoch  41/100 Batch 1000/3559 - Loss:  1.583, Seconds: 3.01
Epoch  41/100 Batch 1020/3559 - Loss:  1.409, Seconds: 3.15
Epoch  41/100 Batch 1040/3559 - Loss:  1.501, Seconds: 2.81
Epoch  41/100 Batch 1060/3559 - Loss:  1.416, Seconds: 2.70
Epoch  41/100 Batch 1080/3559 - Loss:  1.420, Seconds: 3.13
Epoch  41/100 Batch 1100/3559 - Loss:  1.576, Seconds: 2.67
Epoch  41/100 Batch 1120/3559 - Loss:  1.521, Seconds: 3.17
Epoch  41/100 Batch 1140/3559 - Loss:  1.439, Seconds: 3.10
Epoch  41/100 Batch 1160/3559 - Loss:  1.380, Seconds: 2.98
Epoch  41/100 Batch 1180/3559 - Loss:  1.474, Seconds: 2.27
Average loss for this update: 1.55
No Improvement.
Stopping Training.
```

Fig:- Model Training

```
[ ]     input_data = loaded_graph.get_tensor_by_name('input:0')
        logits = loaded_graph.get_tensor_by_name('predictions:0')
        text_length = loaded_graph.get_tensor_by_name('text_length:0')
        summary_length = loaded_graph.get_tensor_by_name('summary_length:0')
        keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')

        #Multiply by batch_size to match the model's input parameters
        answer_logits = sess.run(logits, {input_data: [text]*batch_size,
                                          summary_length: [np.random.randint(5,8)],
                                          text_length: [len(text)]*batch_size,
                                          keep_prob: 1.0})[0]

    # Remove the padding from the tweet
    pad = vocab_to_int["<PAD>"]

    #print('Original Text:', reviews.Text[random])
    #print('Original summary:', reviews.Summary[random])#clean_summaries[random]
    ogSum=reviews.Summary.loc[random]

    predSum=" ".join([int_to_vocab[i] for i in answer_logits if i != pad])
    print('\nText')
    print('  Word Ids:    {}'.format([i for i in text]))
    print('  Input Words: {}'.format(" ".join([int_to_vocab[i] for i in text])))

    print('\nSummary')
    print('  Word Ids:       {}'.format([i for i in answer_logits if i != pad]))
    print('  Response Words: {}'.format(" ".join([int_to_vocab[i] for i in answer_logits if i != pad])))
```

```
INFO:tensorflow:Restoring parameters from /content/drive/My Drive/latest_trained_model/best_model.ckpt

Text
  Word Ids:    [2, 495, 351, 940, 71, 355, 629, 330, 71, 157, 2948, 1154, 407, 1674, 2890, 1949, 637, 5639, 1234, 2, 2890]
  Input Words: dog absolutely loves treats price nearly 1 2 price pay pet grocery store plus worry running quite shall say dog worry

Summary
  Word Ids:       [13, 71, 13, 61]
  Response Words: great price great product
```

```
[ ] from __future__ import division
```

Fig:- Model Output

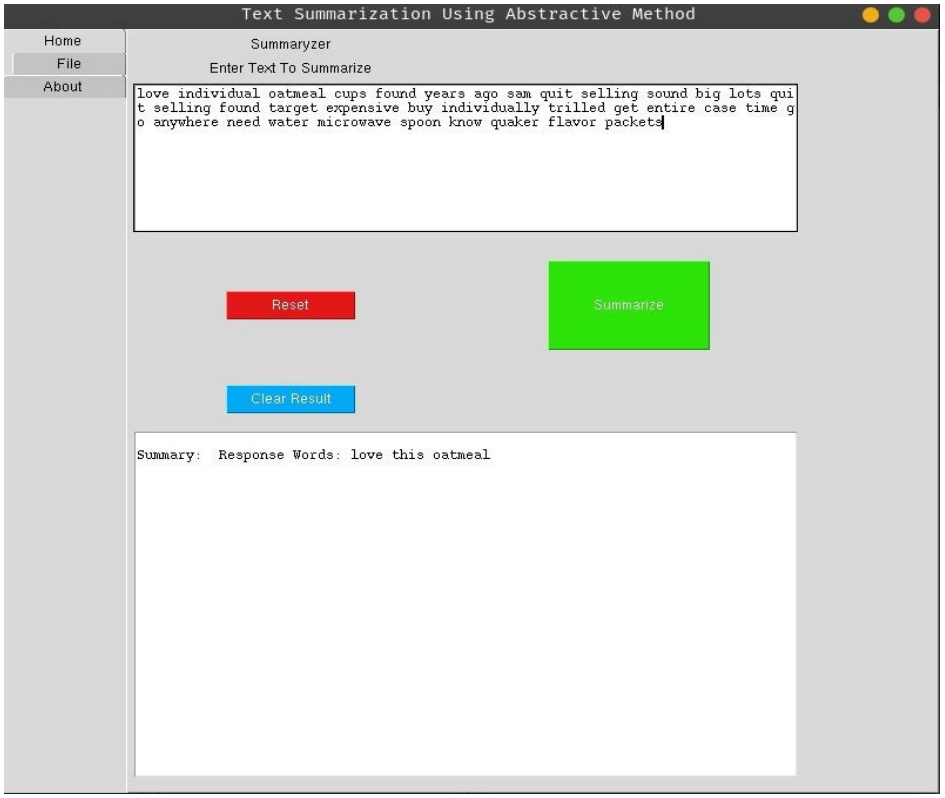


Fig:- App Interface

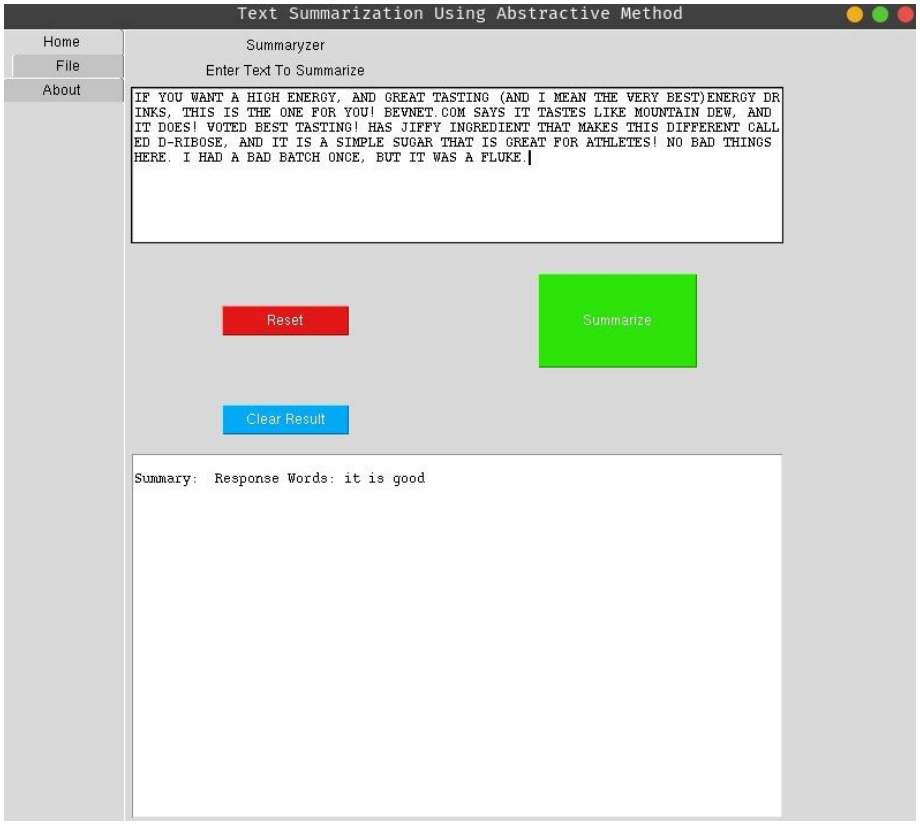


Fig:- Summarization in the App