Abstractive Text Summarization using Bidirectional LSTMs

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

MASTER OF TECHNOLOGY
IN
INTEGRATED SOFTWARE ENGINEERING

by

YALAMADDI ABHINAV (17MIS7077)

Under the Guidance of

Dr. D. Sumathi

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

SEPTEMBER 2021

CERTIFICATE

This is to certify that the Capstone Project work titled "**Abstractive Text Summarization**" that is being submitted by **YALAMADDI ABHINAV (17MIS7077)** is in partial fulfillment of the requirements for the award of Master of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. D. Sumathi
Guide

The thesis is satisfactory / unsatisfactory

Approved by

PROGRAM CHAIR
M. Tech. SE

DEAN
School of Computer Science and Engineering

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. D. Sumathi, Associate Professor, School of Computer Science and Engineering, VIT-AP, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavour. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Data Structures, Cloud Computing, Artificial Intelligence, Machine Learning and NLP.

I would like to express my gratitude to Dr. G. Viswanathan, Dr. S. V. Kota Reddy, Dr. Sekar Viswanathan and Dr. Sudha S V, School of Computer Science and Engineering, for providing with an environment to work in and for her inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Reeja S R, All the teaching staff and students who serve as university members for their unfocused eagerness and timely encouragement showered me with passion, and this encouraged them to gain the required information to successfully complete my studies.

It is a joy to thank my friends for persuading me to carry out this work and encouraging me to do that. Finally, and not least, I would want to convey my gratitude and gratitude to all who have helped me to complete this project directly or indirectly.

Place: AMARAVATHI

Date: 27-08-21 YALAMADDI ABHINAV

ABSTRACT

The objective of this project is to imitate the fundamental concepts of this research of state-of-the-art abstract text repeated models to examine various processes until the work had a reasonable functional basis. This work was motivated by various research papers with several novel features that have achieved remarkable achievement. In multiple iterations, this study will enhance the adoption of words, complexity of decoders, and attentiveness. In addition, a bilinear care mechanism adds the last model, increasing the rate of loss of training.

Text Summarization is one of the most experimental subjects in natural language processing that reduces the size of a document while keeping its meaning. Summary techniques are classed as extractive or abstractive based on whether the precise phrases in the original text are produced or whether new phrases are constructed using natural language methods. Extractive summaries have been carefully examined and a developed state has been obtained. Abstractive summary is the focus of the research. Due to the ins and outs of the text, abstractive summarization is challenging.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER 1

INTRODUCTION

In recent years the summarization of the text has been noticed by the web content overflow. This information abundance enhances the need for dynamic and able text summaries. It is of immense importance, because there are many different applications including journals, books, magazines, stories about this same topic, events, scientific articles, weather forecasting, stock exchange, news, novels, music, plays, movies and speech. Thanks to its huge expansion, numerous world-class universities including Aarhus University-Denmark, NaCTeM-Manchester University and others worked hard to develop it.

As the amount of information and data available on the Internet grows every day, it is becoming constantly an open research issue to acquire and understand the information needed as soon as feasible. The collection of all the information and subsequently the production in summary form is a tiresome operation. The Internet is a database information platform. But this data still has to be processed massively. Thus, a text-based synthesis was required, which conserves the meaning and content of the document into a shorter version. A brief is therefore helpful in saving time and retrieving huge data from papers.

In general, the text resume can be classified into two categories: extractive summary and abstract summary. Extractive type selects significant sentences or phrases from source materials and organize them into a synopsis without modifying the original text. The phrases are generally followed as in the original text. However, abstraction of the original text is studied and examined using the approach of linguistic use. A comprehensive summary aims to provide information accurately that usually calls for advanced language production and strategies of compression.

Abstractive summary is an effective kind of summary in comparison with extractive summary because it collects information from several texts in order to construct an accurate summary. This has becoming more popular since new sentences may be developed to convey vital information from text documents. In a consistent and linguistically accurate form, an abstractive summarizer delivers summarized information. Readability or language quality is a vital driver to improve a summary's quality.

OBJECTIVES

The basic idea behind abstractive text summarization is to be able to extract a short subset of the most important information from a large set and present it in a human-readable format. As the amount of textual data on the internet grows, automatic text summarization methods have the potential to be very useful because more useful information can be read in a shorter amount of time.

BACKGROUND AND LITERATURE SURVEY

The summarization is about providing a brief summary that highlights the major concepts of the text of the input. For this project, the work was focused on abstract composition that creates a paraphrase summary. The extractive summary works by extracting only the input terms by comparison.

Abstraction models fall within the more profound learning category known as models that map from input sequence to target sequence. While sequences of sequences have been applied effectively for several nlp difficulties, including machine translation, the abstract summary of cutting-edge models still offers lots of chance for improvement. While state of the art models can achieve high ROUGE scores in tiny input summaries, the model often loses its capacity to sum up key points once large inputs are assessed. The abstract model's standard metric. While hierarchical models have managed to summarize large inputs, they are still a long way away. The task of summarizing papers is thus an unresolved difficulty for natural languages to be processed.

Despite the clear distinction between abstract summary and machine translation, an international standard of abstractive summary has become the attention-grabbing model of the RNN encoder-decoder as suggested [2]. The encoder generates an input representation for both activities, and the decoder generates the final output using these encoding. [6] This paper proposes, as a base model, a two-directional encoder, a one-directional decoder, an attention mechanism for the cached state of the source and a layer from SoftMax over the vocabulary. Then combines several elements to focus attention, including a huge quantity of vocabulary, feature-rich keyword coding and the use of pointer networks for modelling unusual words. [5] The vocabulary trick restricted the decoder's vocabulary to terms in batch input files, increasing convergence and addressing the bottlenecks of SoftMax computing throughout the entire vocabulary. It adds words from the objective dictionary

to the vocabulary of the decoder until a default size is reached. This approach concentrates the model on the source words, which are helpful in summary.

ORGANIZATION OF THE REPORT

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology and analysis.
- Chapter 3 discusses the results obtained after the project was implemented.
- Chapter 4 concludes the report.
- Chapter 5 consists of code snippets.
- Chapter 6 gives references.

CHAPTER 2

ABSTRACTIVE TEXT SUMMARIZATION

PROPOSED SYSTEM

1. Preprocessing and Dataset

   Within this particular project, we trained the model on Kaggle's Amazon Fine Food Reviews dataset, containing 500,000 reviews, generally used for training models for abstractive summarization from 1999 through 2012. We have drawn up 80% data and summaries during preprocessing and trained the model to predict the abstract summary following the user review.

   Ref: https://www.kaggle.com/snap/amazon-fine-food-reviews/download

2. Model

   A Three Layer Stacked LSTM Encoder-Decoder model with Global Attention Mechanism was used during implementation. On the training set, the algorithm was able to achieve an accuracy of 77.27 percent using this model (constituting 80 percent of the dataset). This model also achieved a cumulative BLEU-4 score of 0.8800 on the test set.

WORKING METHODOLOGY

As stated, before this project is implemented using 3 models as a stacked layer.

1. The first model was a basic one-way LSTM encoder decoder with randomly initialized words. As mentioned later, discovered that using preexisting word embedding would be better, and hence iterated Model 2 using this model.
2. The model has been built with a two-way LSTM encoder and one-way LSTM decoder.
3. Implementation of a two-way LSTM encoder in the last model and incorporated the LSTM decoder worldwide attention. Previously provided attention scoring functions. The final secret state (forward and reverse) of the encoder is linked and utilized as the initial hidden state of the decoder.

Model 1 randomly deploys and upgrades the Word Embedding model. Here it is assumed the words vector representations that were learned would be more easily customizable than Word2Vec or Glove vectors that were previously trained. But it soon became known that limits on machine and time would prevent the attainment of such an objective, as many iterations on huge data sets are necessary to develop accurate word embedding. Glove vectors are loaded for the model 2 and model 3 embedding matrices because it is not possible to train in words on a huge dataset.

Used the same integration and terminology to facilitate encoding and decoding. In the context in which a word appears, however, different patterns such as Rush et al. (2015) provide a variety of embedding arrays for diverse functionalities, thereby increasing the concept of vector semantics.

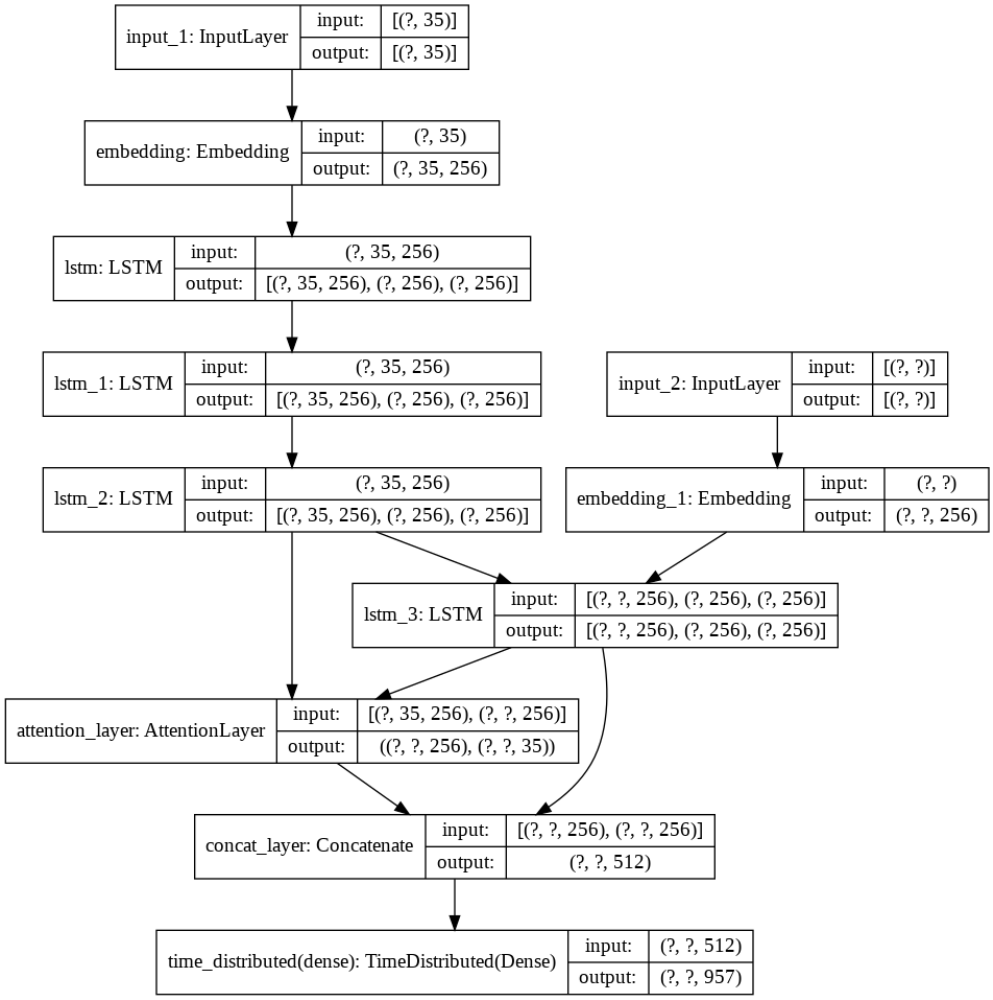

Figure 1. Flow structure of the algorithm implemented.

ANALYSIS

At training time, each step of the model decoders incorporates article truth as an input. This enables the machine to learn predictions given inputs x and past predictions, y. However, during the test, the decoder sends its result into the next decoder cell as a word embedding input. For the gradient descent technique, implemented using RNN algorithm. The reason to pick RNN because it processes sparse data well and makes greater updates for small frequency parameters and smaller updates for large frequency parameters. This is extremely beneficial in handling larger vocabulary sizes, as common words appear far more frequently than unusual terms.

The need to start exploring new attention functions, after creating and successfully testing basic model models. To boost the model's capacity to recognize critical data, allowing the encoder to offer scaled 'size values' in over-sized states. You must either update your attention scores directly or you must connect them to your covert vector encoder.

Therefore, ROUGE had to be used to examine the results, but time to construct a competitive model was not available. Unfortunately. However, a totally different strategy was tested in the last effort. In 3000 vocabulary data points and 600 epochs there were 1,000 instead of 100,000 training data points with 5,000 vocabulary-size 10 epochs during implementation. At the period of 80 we saw the summary predicted, the first word of which was based on essential truth, but the article or title was not immediately replicated by the first word.

Accuracy Graph:

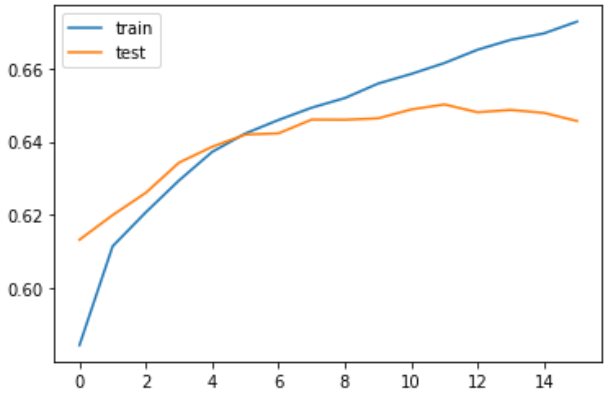

Figure 2. Accuracy Graph

Loss Graph:

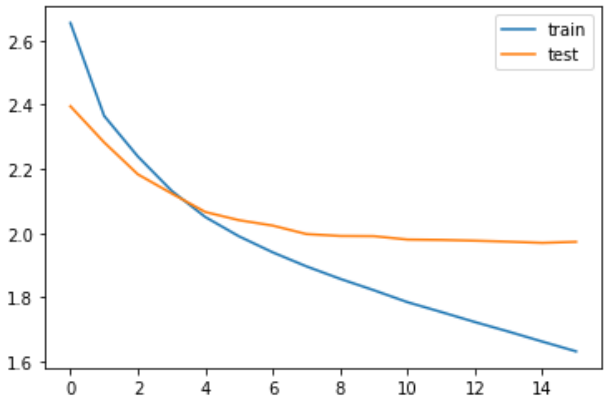

Figure 3. Loss Graph

CHAPTER 3

RESULTS AND DISCUSSIONS

Although Nallapati et al. (2016) [6] work has successfully built the RNN summarizer-decoder as the basis of the model, there was insufficient time for training, Model 3 was mostly trained on misdate, and had several challenges with the Collab GPU. For the first time 10 times, Model 3 was trained on 100,000 sentence-headline combinations. The 200 Glove vector and vocabulary are used with roughly 200 encoders and decoders. The loss decreased between 8,667 and 6,584, however the model started creating completely UNK toks after evaluation of the development forecasts, for maximum output length. Later examined the data used to train the model and the manner through which initialized the vocabulary in the model to detect the problem.

The most common V-words from the entire data package are used to construct a buffer matrix that starts with a V-size vocabulary. The V = 10,000 words most typically referred to are still significantly less than the first 100,000 single words of the complete unnoted English mega word Corpus. Many truth values have therefore been declared as UNK tokens. UNK was the largest term in truth, and UNK's recurrent output may minimize losses.

The RNN grabber model decided without sufficient time to utilize a large data set to generalize a model to unnoticed information, using a small dataset in which model vocabulary is identical to the words in learning & development. This solves the problem of repetitive UNK token impressions and even prints intelligible words but only at the 80th of the 600s the results were inspected.

CHAPTER 4

CONCLUSION AND FUTURE WORK

This has been the first experience in establishing a large-scale data project, and the work that I done is quite different from any other works that I previously done. The creation of every starting code was a significant problem. The need to grasp what each functions call accomplished, but to rely on the code of earlier.

I had never built scripts to analyze large volumes of text, thus it was incredibly fascinating to think about each design decision. Came across very bad documentation and weird constraints in contrib libraries when used TensorFlow, which had to solve in a self-manner. Then finally modified the source code numerous times to try debugging one or two problems. The most interesting thing, once I had the baselines, was to explore the model.

I can't go as far as my guide would like, But I do know how vital it is to give up time for all the variables - as seen above, I seem to be in the right direction with the final version, a very modest collection of data and many more epochs Out of 600 80 times and the results of the test and train data can be observed in the fig2 and fig3. The project is optimistic that if we run the final model across the epochs of this data set (this will happen shortly after this submission), a meaningful summary will work much better.

CHAPTER 5

APPENDIX

1. Preprocessing

#This the dictionary used for expanding contractions

contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": “because” "could've": "could have", "couldn't": "could not",

"didn't": "did not", "doesn't": "does not", "don't": "do not", "hadn't": "had not", "hasn't": "has not", "haven't": "have not",

"he'd": "he would","he'll": "he will", "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "how'll": "how will", "how's": "how is",

"I'd": "I would", "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i would",

"i'd've": "i would have", "i'll": "i will", "i'll've": "i will have","i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "it would",

"it'd've": "it would have", "it'll": "it will", "it'll've": "it will have","it's": "it is", "let's": "let us", "ma'am": "madam",

"mayn't": "may not", "might've": "might have","mightn't": "might not","mightn't've": "might not have", "must've": "must have",

"mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have","o'clock": "of the clock",

"oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not", "sha'n't": "shall not", "shan't've": "shall not have",

"she'd": "she would", "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have", "she's": "she is",

"should've": "should have", "shouldn't": "should not", "shouldn't've": "should not have", "so've": "so have","so's": "so as",

"this's": "this is","that'd": "that would", "that'd've": "that would have", "that's": "that is", "there'd": "there would",

"there'd've": "there would have", "there's": "there is", "here's": "here is","they'd": "they would", "they'd've": "they would have",

"they'll": "they will", "they'll've": "they will have", "they're": "they are", "they've": "they have", "to've": "to have",

"wasn't": "was not", "we'd": "we would", "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we're": "we are",

"we've": "we have", "weren't": "were not", "what'll": "what will", "what'll've": "what will have", "what're": "what are",

"what's": "what is", "what've": "what have", "when's": "when is", "when've": "when have", "where'd": "where did", "where's": "where is",

"where've": "where have", "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have",

"why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not", "won't've": "will not have",

"would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have", "y'all": "you all",

"y'all'd": "you all would","y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you all have",

"you'd": "you would", "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have",

"you're": "you are", "you've": "you have"}

2. Text Cleaning

```
import nltk
nltk.download('stopwords')

def text_cleaner(text,num):
    newString = text.lower()  #converts all uppercase characters in the string into lowercase characters and returns it
    newString = BeautifulSoup(newString, "lxml").text #parses the string into an lxml.html
    newString = re.sub(r'\([^)]*\)', '', newString) #used to replace a string that matches a regular expression instead of perfect match
    newString = re.sub('"','', newString)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in newString.split(" ")]) #for expanding contractions using the contraction_mapping dictionary
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
```

```
    if(num==0):
      tokens = [w for w in newString.split() if not w in stop_words]  #converting the strings into
tokens
    else :
      tokens = newString.split()
    long_words=[]
    for i in tokens:
        if len(i)>1:                 #removing short words
            long_words.append(i)
    return (" ".join(long_words)).strip()

#Calling the function
cleaned_text = []
for t in reviewsData['Text']:
    cleaned_text.append(text_cleaner(t,0))
```

3. Summary Cleaning

```
cleaned_summary = []    #Using the text_cleaner function for cleaning summary too
for t in reviewsData['Summary']:
    cleaned_summary.append(text_cleaner(t,1))
```

4. Preparing Tokenizer after splitting data into test and train

```
#Text Tokenizer

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

#preparing a tokenizer for reviews on training data
X_tokenizer = Tokenizer()
X_tokenizer.fit_on_texts(list(X_train))
```

5. Defining the Tokenizer with top most common words for reviews

```
#Preparing a Tokenizer for reviews on training data
X_tokenizer = Tokenizer(num_words=tot_cnt-cnt)   #provides top most common words
X_tokenizer.fit_on_texts(list(X_train))

#Converting text sequences into integer sequences
X_train_seq    =   X_tokenizer.texts_to_sequences(X_train)
X_test_seq   =   X_tokenizer.texts_to_sequences(X_test)

#Padding zero upto maximum length
X_train    =   pad_sequences(X_train_seq,  maxlen = max_text_len, padding = 'post')
X_test   =   pad_sequences(X_test_seq, maxlen = max_text_len, padding = 'post')
```

```
#Size of vocabulary (+1 for padding token)
X_voc   =  X_tokenizer.num_words + 1
```

6. Defining Tokenizer with the most common words in summary

```
#Preparing a tokenizer for summaries on training data
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)  #provides top most common words
y_tokenizer.fit_on_texts(list(y_train))

#Converting text sequences into integer sequences
y_train_seq    =   y_tokenizer.texts_to_sequences(y_train)
y_test_seq   =   y_tokenizer.texts_to_sequences(y_test)

#Padding zero upto maximum length
y_train    =   pad_sequences(y_train_seq, maxlen=max_summary_len, padding='post')
y_test   =   pad_sequences(y_test_seq, maxlen=max_summary_len, padding='post')

#size of vocabulary
y_voc  =   y_tokenizer.num_words +1
```

7. Model Building

```
#Adding Custom Attention layer

import tensorflow as tf
import os
from tensorflow.python.keras.layers import Layer
from tensorflow.python.keras import backend as K

class AttentionLayer(Layer):
    """
    Three sets of weights introduced W_a, U_a, and V_a
     """

    def __init__(self, **kwargs):
        super(AttentionLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        assert isinstance(input_shape, list)
        # Create a trainable weight variable for this layer.

        self.W_a = self.add_weight(name='W_a',
                                   shape=tf.TensorShape((input_shape[0][2], input_shape[0][2])),
                                   initializer='uniform',
```

```
                trainable=True)
        self.U_a = self.add_weight(name='U_a',
                shape=tf.TensorShape((input_shape[1][2], input_shape[0][2])),
                initializer='uniform',
                trainable=True)
        self.V_a = self.add_weight(name='V_a',
                shape=tf.TensorShape((input_shape[0][2], 1)),
                initializer='uniform',
                trainable=True)

        super(AttentionLayer, self).build(input_shape)  # Be sure to call this at the end

    def call(self, inputs, verbose=False):
        """
        inputs: [encoder_output_sequence, decoder_output_sequence]
        """
        assert type(inputs) == list
        encoder_out_seq, decoder_out_seq = inputs
        if verbose:
            print('encoder_out_seq>', encoder_out_seq.shape)
            print('decoder_out_seq>', decoder_out_seq.shape)

            reshaped_enc_outputs = K.reshape(encoder_out_seq, (-1, en_hidden))
            # <= batch_size*en_seq_len, latent_dim
            W_a_dot_s = K.reshape(K.dot(reshaped_enc_outputs, self.W_a), (-
1, en_seq_len, en_hidden))
            if verbose:
                print('wa.s>',W_a_dot_s.shape)

            """ Computing hj.Ua """
            U_a_dot_h = K.expand_dims(K.dot(inputs, self.U_a), 1)  # <= batch_size, 1, latent_dim
            if verbose:
                print('Ua.h>',U_a_dot_h.shape)

            """ tanh(S.Wa + hj.Ua) """
            # <= batch_size*en_seq_len, latent_dim
            reshaped_Ws_plus_Uh = K.tanh(K.reshape(W_a_dot_s + U_a_dot_h, (-1, en_hidden)))
            if verbose:
                print('Ws+Uh>', reshaped_Ws_plus_Uh.shape)

            """ softmax(va.tanh(S.Wa + hj.Ua)) """
            # <= batch_size, en_seq_len
            e_i = K.reshape(K.dot(reshaped_Ws_plus_Uh, self.V_a), (-1, en_seq_len))
            # <= batch_size, en_seq_len
            e_i = K.softmax(e_i)
```

```
            if verbose:
                print('ei>', e_i.shape)

            return e_i, [e_i]

        def context_step(inputs, states):
            """ Step function for computing ci using ei """
            # <= batch_size, hidden_size
            c_i = K.sum(encoder_out_seq * K.expand_dims(inputs, -1), axis=1)
            if verbose:
                print('ci>', c_i.shape)
            return c_i, [c_i]

        def create_inital_state(inputs, hidden_size):
            # We are not using initial states, but need to pass something to K.rnn funciton
            fake_state = K.zeros_like(inputs)  # <= (batch_size, enc_seq_len, latent_dim
            fake_state = K.sum(fake_state, axis=[1, 2])  # <= (batch_size)
            fake_state = K.expand_dims(fake_state)  # <= (batch_size, 1)
            fake_state = K.tile(fake_state, [1, hidden_size])  # <= (batch_size, latent_dim
            return fake_state

        fake_state_c = create_inital_state(encoder_out_seq, encoder_out_seq.shape[-1])
        fake_state_e = create_inital_state(encoder_out_seq, encoder_out_seq.shape[1])  # <= (batch
_size, enc_seq_len, latent_dim

        """ Computing energy outputs """
        # e_outputs => (batch_size, de_seq_len, en_seq_len)
        last_out, e_outputs, _ = K.rnn(
            energy_step, decoder_out_seq, [fake_state_e],
        )

        """ Computing context vectors """
        last_out, c_outputs, _ = K.rnn(
            context_step, e_outputs, [fake_state_c],
        )

        return c_outputs, e_outputs

    def compute_output_shape(self, input_shape):
        """ Outputs produced by the layer """
        return [
            tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[1][2])),
            tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[0][1]))
        ]
```

8. Summary Generation

```
for i in range(0,20):
    print("Review:",seq2text(X_train[i]))
    print("Original summary:",seq2summary(y_train[i]))
    print("Predicted summary:",decode_sequence(X_train[i].reshape(1,max_text_len)))
    print("\n")
```

9. Accuracy validation

```
#BLEU Score of Training set
from nltk.translate.bleu_score import sentence_bleu
for i in range(0,1000):
  reference = seq2summary(y_train[i])
  candidate = decode_sequence(X_train[i].reshape(1, max_text_len))
```

CHAPTER 6

REFERENCES

Aref, I. Moawad and M. 2012. "Semantic graph reduction approach for abstractive text summarization ." *Computer Engineering System (ICCES), 2012 Seventh International Conferenc.*

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. "Neural Machine Translation by Jointly Learning to Align and Translate." 1409.0473.

Harabagiu, S. H. Finley and S. M. 2002. "Generating single and ." *In U. Hahn & D. Harman, Proceedings of the workshop on automatic summarization* 30-38.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2018. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv*.

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

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

S, N. Munot and P.-N. P. I. Sharvari. 2015. "Conceptual framework for abstractive text summarization." *International Journal on NaturaL Language Computing (IJNLC).*

BIO DATA

Name	: Yalamaddi Abhinav
Mobile Number	: 9182364034
E-mail	: abhinav.yalamaddi@vitap.ac.in
Permanent Address	: Lotus Heaven Apartment, Khammam, Telangana – 507002.

All the documents related to the project can be found in the link - GDrive link