International Journal of Innovative Research in Engineering & Management (IJIREM)
ISSN: 2350-0557, Volume-9, Issue-2, April 2022
https://doi.org/10.55524/ijirem.2022.9.2.3
Article ID IJIR-1183, Pages 18-28
www.ijirem.org

A Study of Implementation of Deep Learning Techniques for Text Summarization

Bilkeesa Akhter[1], and Dr. Monika Mehra[2]

[1] M.Tech Scholar, Department of Electronics and Communication Engineering, RIMT University Mandi Gobingrah, Punjab India

[2] Professor, Department of Electronics and Communication Engineering, RIMT University Mandi Gobingrah Punjab India

Correspondence should be addressed to Bilkeesa Akhter; sneharajput9393@gmail.com

ABSTRACT- To automatically summarize a piece of material, the length of the original text must be reduced while the content's important informative parts and significance are preserved. As a result, automating manual text summarizing, which is a time-consuming and labor-intensive procedure, is gaining popularity, and is therefore a major motivator for academic study. In today's age of data overload, abstracting and summarizing huge texts is critical. Over time, a variety of approaches for summarizing text have been created. Traditional approaches construct a summary directly as a result of the duplication and omission of the document summary connection. Deep learning algorithms have been demonstrated to be useful in creating summaries. We concentrate on deep learning-based text summarizing algorithms that have been developed throughout time.

KEYWORDS- Summarize text, Deep Learning Techniques, Effective, Automatization

I. INTRODUCTION

To describe a piece of content autonomously, the original text's length must be decreased while the content's main informative elements and relevance are kept. As a result, automating manual text summarization, which is a time-consuming and labor-intensive technique, is gaining traction and becoming a significant incentive for academic study. The volume of text data from various sources has exploded in the age of big data. These kind of text volumes are a treasure of information that must be expertly presented in order to be useful. Humans usually study a book in its entirety to obtain a complete understanding of it before creating a summary that emphasizes the most significant aspects[1]. Because computers lack human understanding and language ability, summarizing material is difficult.

Natural language processing techniques may be used to summarize a piece of text by using algorithms like page rank algorithms. These algorithms are wonderful for text summaries, but they can't come up with new terms that aren't in the document, and they can't catch grammatical errors. Therefore, we may rely on Deep Learning, a text summarization model that incorporates new terms. As a result, we use deep learning algorithms to construct grammatically and phraseological correct summaries.

A. Deep Learning

Several nonlinear processing units are utilized in a cascade while performing transformations and feature extractions, such that the output of one layer is provided as an input to the next layer. Using a sequence of feature layers, deep learning algorithms may learn from inputs in both an unsupervised and supervised manner.[2] The features layers are not clarified and evolved by humans as a result of a generalized learning process but are automatically learned from generalized learning.

In order to construct the summary, text summarizing methods entail extracting words directly from the textual content. Eliminating stop words and finding noun groups are both part of the lemmatization process.[3] Using traditional methods, on the other hand, has the major disadvantage of producing a summary that is not accurate. Given the lack of a record of the keywords chosen before, it's possible that certain words will occur in the summary as well as in the main text. Furthermore, the created summary and the document have a low correlation using standard methodologies. As a result, the condensed information makes it more difficult for customers to comprehend the paper. Deep learning approaches are utilized for summarization in this way to overcome difficulties.[4]

B. Need for text summarization

When it comes to text summary, it's important to extract words directly from a text. Stop words are removed from noun groups during normalization. To be sure, using conventional procedures has drawbacks, such as the inability to develop unique content. Because there is no record of the keywords that were previously picked, it's likely that certain terms will appear in both the summary and the main text. Furthermore, the link between the summary and the document produced by traditional methods is relatively weak.[5] Customers will find it more difficult to comprehend a summarized content as a result. Therefore, text summarizing is performed automatically.,

C. Approaches used for automatic text summarization

There are two basic types of NLP strategies for summarizing text. Each has its own set of drawbacks, such as the inability to produce new text. Because there is no record of the keywords that were previously picked, it's likely that certain terms will appear in both the summary and the main text.[6] Furthermore, the link between the summary and the document produced by traditional methods is relatively weak. Customers will find it more difficult to comprehend a summarized content as a result. Therefore, text summarizing is performed automatically.[7].

D. Approaches used for automatic text summarization:

There are two main types of how to summarize text in NLP:

- Extractive Summarization
- Abstractive Summarization

- **Extractive Text Summarization: Extracting** Extractive text summarizing (ETS) is the process of extracting key phrases from a source material and utilizing them in a summary. During the extraction procedure, the texts are not changed in any manner. Figure 1 depicts this procedure.[7]
- **Abstractive Text Summarization:** Sections of the quality content are paraphrased and condensed as part of the applied external, as seen in Figure 2. When employed in deep learning, text summarization utilizing abstraction helps overcome the grammatical flaws of the extractive technique. They, like people, create new phrases and sentences to communicate the most useful information from the original material.

As a result, abstraction outperforms extraction.

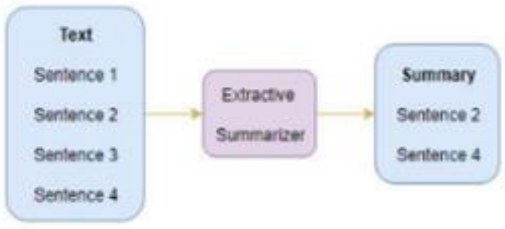


Figure 1: Extractive Summarization

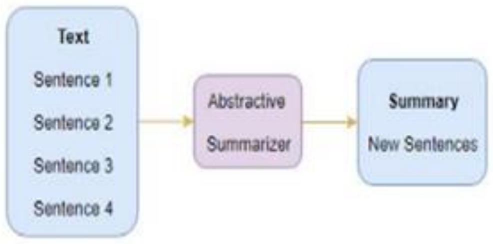


Figure 2: Abstractive Summarization

II. LITERATURE REVIEW

Automatic summary of legal content is challenging due to a variety of writing styles and different features of legal themes included in the text. The writers of Legal Text Summarization provide a detailed examination of the approaches used in legal text summarization [4]. An asymmetric weighted graph, in which sentences are portrayed as nodes in a graph, to summarize legal documents. Only sentences with high node values are chosen for retention in the summary table for the overview. A document is represented as a collection of phrases that belong to almost the same linked constituent as a number of related graphs. This strategy encourages variety and, therefore, provides a steady flow of information. The authors utilize both keyword/key phrase matching and case-based approaches, according [3] to capture information diversity, discriminant analysis is presented for multi-document summarization of Arabic text [8].

To cluster court judgments, hierarchical Latent Dirichlet Allocation (hLDA) might be employed [4]. The similarity measure between topics and documents is used to run hLDA and discover the summary of each document using the same topics. The significance score may be derived by adding the TF-IDF scores for each word in each sentence and normalizing by the sentence length, according to [3]. The summarizing task is separated into two parts, according to [4], segmentation of the document using artificial potential field to identify rhetorical roles, and development of a summary from the segments so discovered.[8]

For text summarization, a variety of models have been suggested, ranging from simple multi-layer networks to complex neural network designs [11]. Deep learning techniques, on the other hand, have rarely been used to create legal document summaries, as far as we know. We provide a deep learning-based approach for summarizing legal documents using computerized sentence labelling.[9]

III. OBJECTIVES

The main purpose of this project is to:

- Use transfer learning to extract the most important meaning from text and provide it to the user.
- Use data mining algorithms that can be designed to read documents and find crucial information.

IV. METHODOLOGY

A. Natural Language Processing

Natural language processing (NLP) is an expert system (AI) area that helps computers comprehend and analyze natural language. Natural language processing (NLP) may be used to organize and rearrange content in order to complete tasks like localization and summaries, according to researchers.[10]

1. ***Components of NLP*** Five main Component of Natural Language processing are:

- Morphological and Lexical Analysis
- Syntactic Analysis
- Semantic Analysis
- Discourse Integration
- Pragmatic Analysis

Morphological and Lexical Analysis: If you want to learn more about lexical analysis, you'll discover how to identify the structure of words by studying, recognizing, and analyzing them. and it entails breaking down a document into paragraphs, words, and phrases.[11]
Each word is broken down into its component pieces.
Analytical Semantics: Semantic Analysis is the process of a syntactic analyzer assigning interpretations to a sentence. This component, as its name says, turns a sequence of Its purpose is to show how the words connect to one another.
Syntax analysis: Words are usually considered as the smallest units of syntax. If you want to learn more about syntax, it is the set of principles and rules that govern how sentences are constructed in every language.[12]

Pragmatic Analysis: Prag focuses on the analysis of variety of communication content, as well as its effect on interpretation. The process of deriving or abstracting the meaningful usage of language is referred to as this. As a consequence of this study, the focus is always on what was said and how it was communicated.
"Close the window?" for example, should be viewed as a request rather than a command.

B. Discourse Integration

If you want to know what a single phrase means, you can look it up on the internet. It also considers the significance of the following sentence.
In the phrase "He wished that," for example, "that" is a function of what has occurred before in the sentence.

V. SIMULATION METHODOLOGY

A. Extractive Approach

Extractive methods identify the most important words from a list of keywords to summarize publications.
The most important portions of sentences are weighted in summary sentences. Sentences are rated using a number of algorithms and methodologies based on their relevance and similarity. Because this approach cannot produce text on its own, the outcome will always include a piece of the original text. There are several approaches for extractive summarization. Essentially said, we'll utilise unsupervised learning to find, and rank connected phrases. We won't have to train and build a model before we can utilise it for our project this way.

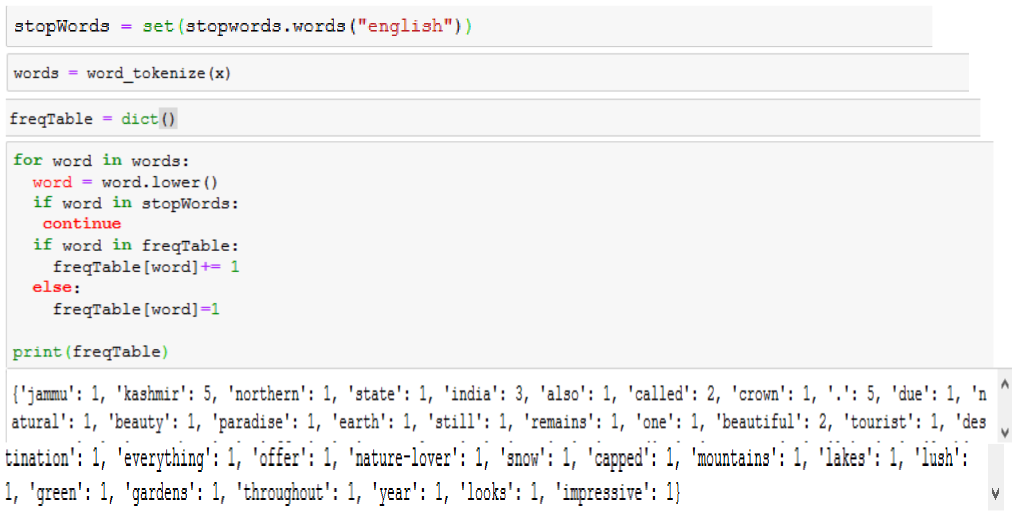
1. ***Unsupervised approach*:** The robots are taught to use data that hasn't been classified or labelled without supervision. In essence, this means that no training data may be provided, and the computer is forced to learn on its own. The computer should not require any prior knowledge of the data to classify it. The machine must be programmed for it to learn on its own. The computer must be able to comprehend and evaluate both structured and unstructured data. Here's a real-life example of unsupervised classification:[13]

Python's libraries include a lot of support for natural language processing. To summarize the subject, the NLTK (natural language toolkit) will be utilized.

Step 1: Importing the necessary libraries
Creating effective feedback exhibited great will need the use of two NLTK libraries.
imported stop words from nltk.corpus
import word tokenize, sent tokenize from nltk
The terms used in this article include
:Corpus: Corpus is a term that refers to a collection of things. Texts, such as poetry by a single poet or an author's whole body of work, may be utilized as data sets. In this circumstance, a set of pre-determined stop words will be utilized.[14]

- Tokenizers: It deconstructs a text into tokens. Word, phrase, and regex tokenizers are examples of tokenizers, with the former being the most common.[15] As a result, we'll only employ the word and term tokenizers.

Step 2: Stop Words are removed and stored in a separate array of words.
Stop Words: There are various words in a statement that aren't essential, such as (is, a, an, the, for). Consider the following sentence as an illustration.
Jammu and Kashmir, sometimes known as the Crown of India, is India's northernmost state.
Following the removal of stop words, we may reduce the number of words while maintaining the meaning as follows:
:'Jammu','and','Kashmir','northern','most','state','India','also','called','crown','India'
Step 3: Create a word frequency table.
Figure 3 shows how a python dictionary keeps track of how many times each word appears after stop words are deleted.[16] We can run each sentence through the dictionary to see which sentences contain the most important information in the overall text.
Step 4: Assign score to each phrase subject to the terms it includes and the frequency table
To produce the array of phrases displayed in Figure 4, we may use the sent tokenize () function. Second, we'll need a lexicon to keep track of the sentences' scores.



```
stopWords = set(stopwords.words("english"))
```

```
words = word_tokenize(x)
```

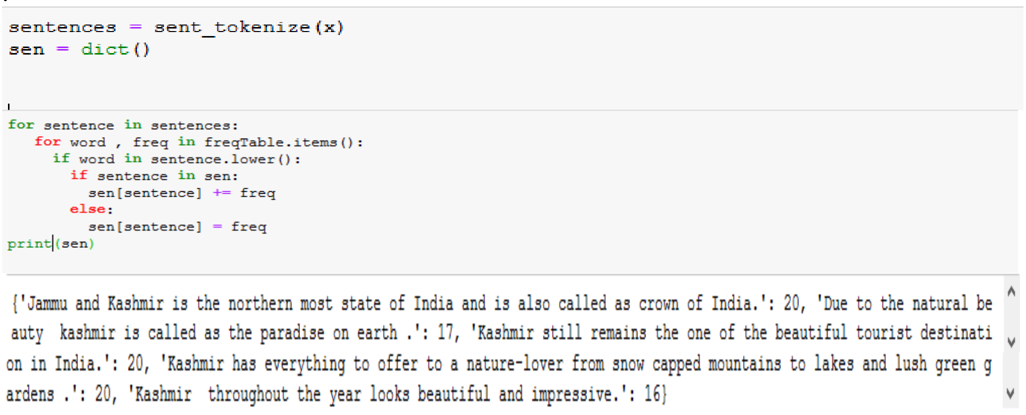
```
freqTable = dict()
```

```
for word in words:
  word = word.lower()
  if word in stopWords:
   continue
  if word in freqTable:
    freqTable[word]+= 1
  else:
    freqTable[word]=1

print(freqTable)
```

```
{'jammu': 1, 'kashmir': 5, 'northern': 1, 'state': 1, 'india': 3, 'also': 1, 'called': 2, 'crown': 1, '.': 5, 'due': 1, 'n
atural': 1, 'beauty': 1, 'paradise': 1, 'earth': 1, 'still': 1, 'remains': 1, 'one': 1, 'beautiful': 2, 'tourist': 1, 'des
tination': 1, 'everything': 1, 'offer': 1, 'nature-lover': 1, 'snow': 1, 'capped': 1, 'mountains': 1, 'lakes': 1, 'lush':
1, 'green': 1, 'gardens': 1, 'throughout': 1, 'year': 1, 'looks': 1, 'impressive': 1}
```

Figure 3: Creating a Frequency Table

.



```
sentences = sent_tokenize(x)
sen = dict()
```

```
for sentence in sentences:
   for word , freq in freqTable.items():
     if word in sentence.lower():
       if sentence in sen:
         sen[sentence] += freq
       else:
         sen[sentence] = freq
print(sen)
```

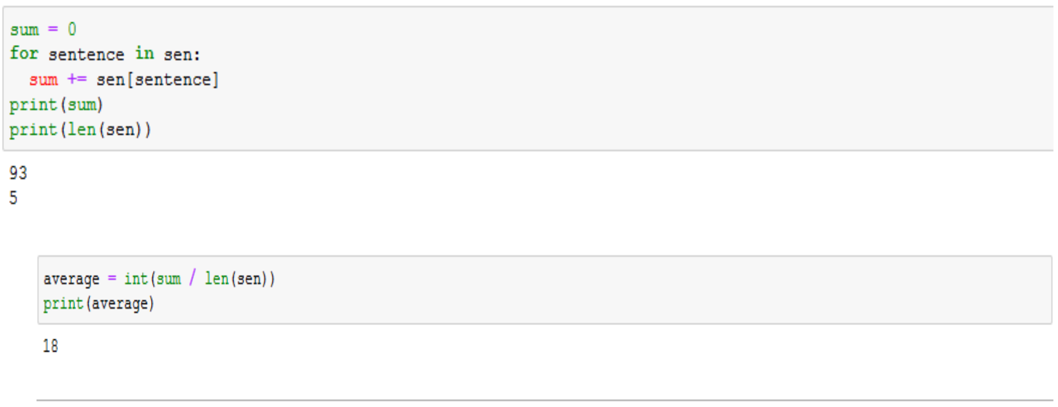
```
{'Jammu and Kashmir is the northern most state of India and is also called as crown of India.': 20, 'Due to the natural be
auty  kashmir is called as the paradise on earth .': 17, 'Kashmir still remains the one of the beautiful tourist destinati
on in India.': 20, 'Kashmir has everything to offer to a nature-lover from snow capped mountains to lakes and lush green g
ardens .': 20, 'Kashmir  throughout the year looks beautiful and impressive.': 16}
```

Figure 4: Assign scores to sentences

Step 5: To compare the sentences in the feedback, provide a score.
Finding the average score of a phrase, as shown in Figure 5, is a straightforward way to compare our scores. The average can be a useful criterion in and of itself.

As illustrated in Figure 6, apply the threshold value and store phrases in order in the summary.[18]

.



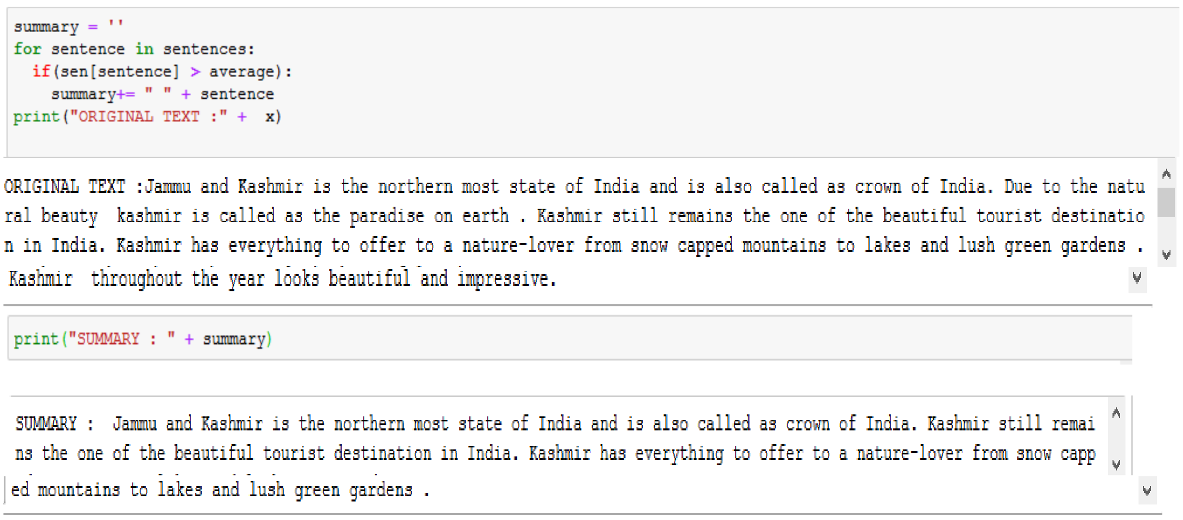
```
sum = 0
for sentence in sen:
  sum += sen[sentence]
print(sum)
print(len(sen))
```

```
93
5
```

```
average = int(sum / len(sen))
print(average)
```

```
18
```

Figure 5: Assign Scores to compare the sentences



```
summary = ''
for sentence in sentences:
  if(sen[sentence] > average):
    summary+= " " + sentence
print("ORIGINAL TEXT :" +  x)
```

```
ORIGINAL TEXT :Jammu and Kashmir is the northern most state of India and is also called as crown of India. Due to the natu
ral beauty  kashmir is called as the paradise on earth . Kashmir still remains the one of the beautiful tourist destinatio
n in India. Kashmir has everything to offer to a nature-lover from snow capped mountains to lakes and lush green gardens .
 Kashmir  throughout the year looks beautiful and impressive.
```

```
print("SUMMARY : " + summary)
```

```
SUMMARY :  Jammu and Kashmir is the northern most state of India and is also called as crown of India. Kashmir still remai
ns the one of the beautiful tourist destination in India. Kashmir has everything to offer to a nature-lover from snow capp
ed mountains to lakes and lush green gardens .
```

Figure 6: Storing the sentences

B. *Abstractive Summarization*

We produce new terms from the original information. This contrasts with the extractive strategy, in which we only used the sentences that were already in the database. Certain sentences may be missing from the original text because of the abstractive summarizing procedure.
Before we go into the implementation, let's go through the principles that are required to develop a Text Summarizer model.

1. ***Recurrent Neural Network:*** The term "recurrent neural networks" refers to a type of neural network. RNNs that are very good at modelling genetic sequences, such as time series. This layout is based on the sequential information concept. Many the most frequently occurring words are sent into the RNN network. The computer searches the data for words that appear often in order to predict the following word in a sentence. As a result, do you realize how critical the RNN is in our daily lives? Among reality, it has bred laziness in us [17]. Figures 7 and 8 demonstrate the RNN's fundamental structure as well as a visual depicting the RNN's basic equations**.**

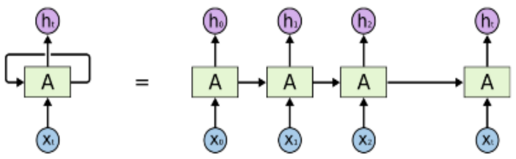
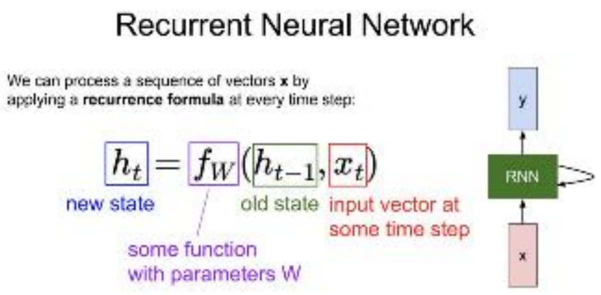


Figure 6: Basic Architecture of RNN



Recurrent Neural Network
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:
$h_t = f_W(h_{t-1}, x_t)$
new state
old state
input vector at some time step
some function with parameters W
y
RNN
x

Figure 7: Basic Equations of RNN

***2. Types of RNN architectures*: —** RNN architectures at a basic level
One-on-one:- When dealing with simple machine learning difficulties, this method is often known as a vanilla neural network. It features a single input and many outputs. Application: used in picture captioning, such as the dog catching the ball in the air we saw before.
— From many to one: It has a lot of inputs and just one output. This is mostly utilized in sentiment analysis, where we provide a statement as an input and receive sentiment about it as an output.
— Many to Many—It accepts a set of inputs and produces a set of outputs. Machine Translation (Application)
Issues while training a RNN: —

- Vanishing Gradient Problem
- Exploding Gradient Problem

Gradients return to the initial layer as a deep neural network is trained. The gradients must flow via a continuous matrix multiplication because of the chain rule. If their values are little, they will quickly dwindle to the point where they will vanish (1). The vanishing gradient problem is the name for this phenomenon. As a result, over time, data got lost. When inclinations have large values (>1), this is known as the growing gradient issue.

- Issues due to these problems:
- Long training time
- Poor Performance
- Bad Accuracy

C. *Introduction to Sequence-to-Sequence (Seq2Seq) Modeling*

We may use the Seq2Seq paradigm to analyse and solve any problem involving sequential data. Sentiment classification, Neural Phonetic Transcription, and Named Entity Recognizing are some of the most common applications of sequential data.
Machine Translation takes a text in one language and produces a text in that other.

I love playing sports $\longrightarrow$ Me encanta hacer deporte

A set of words is sent into Named Entity Recognition, which produces a list of tags for each of the words in the succession.

Andrew ng founded coursera $\longrightarrow$ B-PER, I-PER, O, O

Our objective is to develop a text summarizer that takes a big list of words (from a text body) as input and produces a concise summary of the content (which is a sequence as well). We may represent the problem as a Seq2Seq issue with many-to-many components. Consider the following Seq2Seq model design as an example:
A Seq2Seq model is made up of two major parts.:

- Encoder
- Decoder.

Understanding the Architecture of Encoders and Decoders Encoder-decoder architecture is used to solve the Seq2Seq problem.
Let's take a look at it from a textual aspect to better understand it. You offer a large string of words as an input, and you provide a condensed version of the original signal as an output.[3]
The encoder-decoder can be set up in two stages:

- Training phase
- Inference phase

1. ***Training phase***: We'll move on to the training phase about the after building up the coder and receiver. In this step, we'll train our data to simulate the complementary strand with a the setup of the encoder and decoder will be described in depth.[11]

Encoder An Encoder Long Short-Term Memory Model feeds one word into the encoder at each timestep (LSTM). Each timestep is analyzed, and the input sequence's environmental data is recorded. Figure 9 demonstrates this. The hidden state (hi) and cell state (ci) of the previous time step are used to initialize the decoder. This is because the encoder and decoder are two independent components of the LSM design.

Decoder The decoder, like the encoder, is an LSTM network that reads the whole object of the class work and expects so same sequence, but the code is trained to forecast the next word using the prior word as a cue. Figure 10 depicts the basic construction of a decoder.

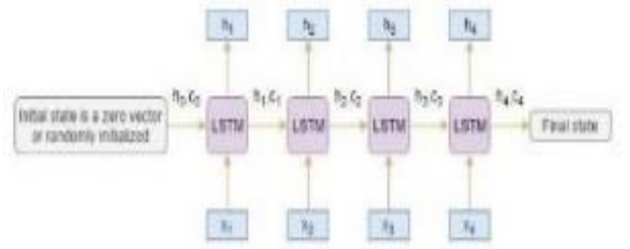


Figure 8: Basic Structure of Encoder

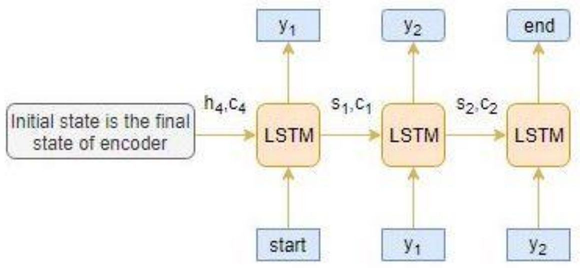


Figure 9: Basic structure of Decoder

Special tokens called <start> and <finish> must be attached to the target DNA in order to decode it. The promoter region is unknown during decoding the test sequence. The decoder is given the first word, which is always <start>, to predict the target sequence. It's also worth remembering that end> denotes the statement's end.

2. ***Inference Phase***: After training, the model is put to the test on new source sequencing with unknown target sequences. To decode an iteration process, we must first build up the inference architecture.:

D. *Working of Inference*

The steps to decoding the measurements in order are as follows:

- Encrypt the whole input stream first, then utilize the encoded data to initialize the decoder.
- The <start> token should be passed to Decoding as an input.
- Then, for a single timestep, execute the decoder.
- The probability of the next word being returned. The phrase that is most likely to be picked will be chosen.[9]
- In the next timestep, provide the sampled word to the decoder and adjust the internal states with the current linear interpolation. 6.
- Tokenize the target sequence by repeating the test 3–5 until an end> token is formed, or the target sequence is reached.

As an example, consider the test sequence [x1, x2, x3, x4]. How will the inference method work for this test sequence?

- Create internal state arrays from the test sequence.
- At each keyframe, see how the parser forecasts the target sequence:

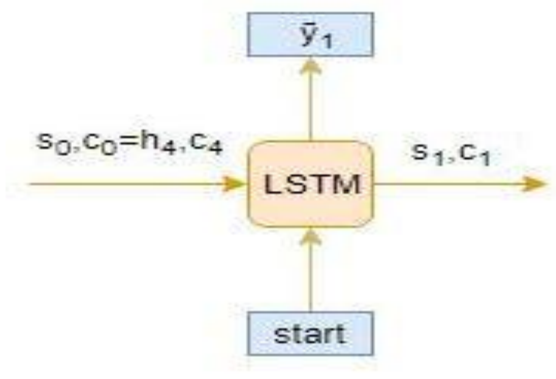


Figure: 10 Timestep t=1

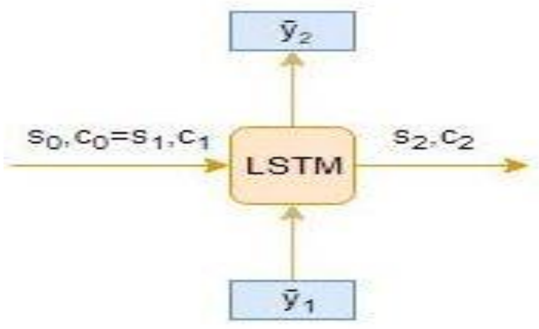


Figure: 11 Timestep t=2

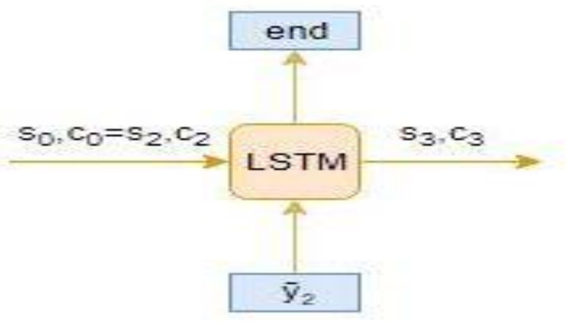
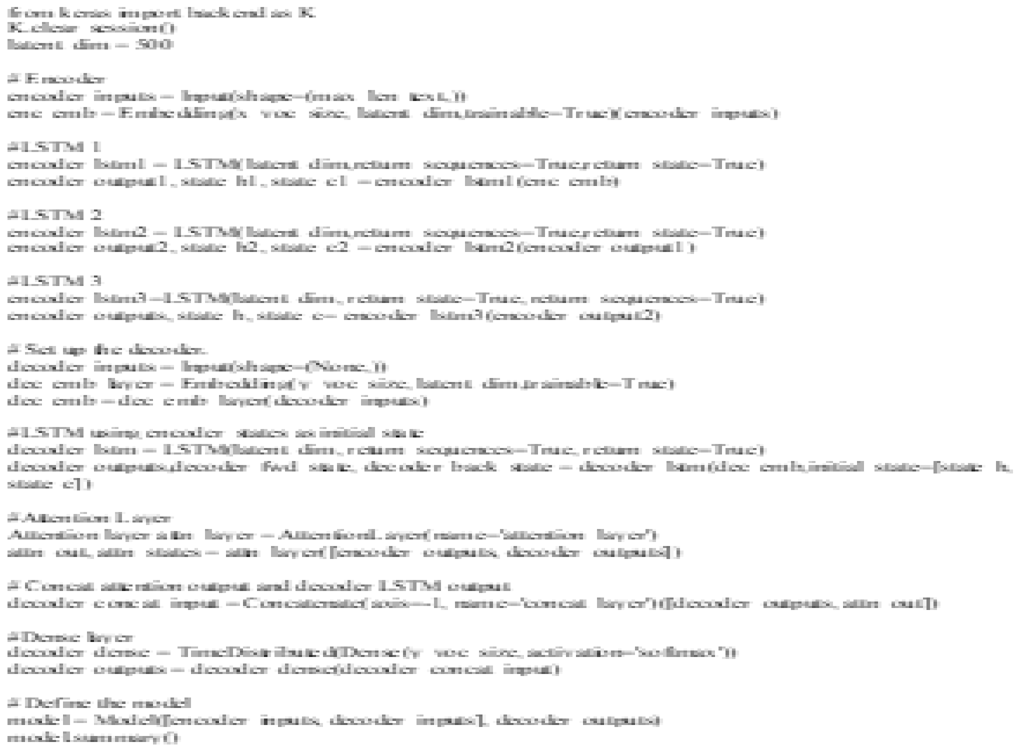


Figure:12 timestep t=3

VI. RESULTS

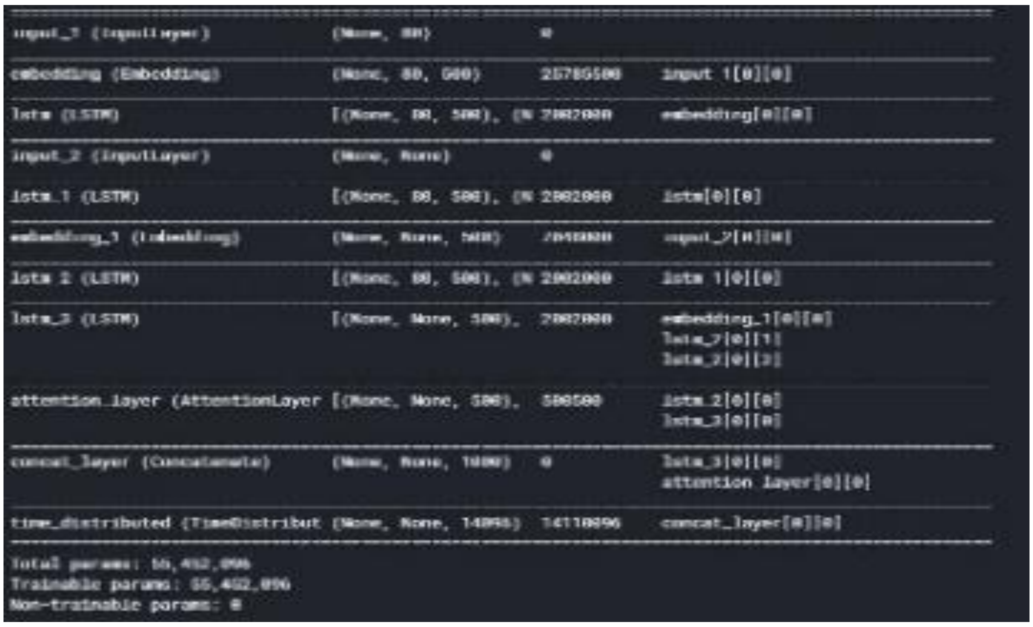
We've finally arrived to the stage when, in order to create the model, we'll need to become acquainted with a few terminology.

- Return Sequences = True: When this option is selected, LSTM creates hidden and cell states for each timestep.
- Return State = True: When return state = True, LSTM creates just the prior timestep's covert data and cell state, as the name implies.
- Initial State: Sets the LSTM's emotions and opinions for the first timestep.
- Layered LSTM: This sort of LSTM consists of many layers stacked on top of each other. As a result, the sequence is more accurately shown. It's an excellent technique to learn to stack a bunch of LSTMs on top of each other.
- Figure 14 shows how a three-stacked LSTM generator is built, and Figure 15 shows the encoder's output.:



```
from keras import backend as K
K.clear session()
latent dim = 500

# Encoder
encoder inputs = Input(shape=(max len text,))
enc emb = Embedding(x voc size, latent dim,trainable=True)(encoder inputs)

#LSTM 1
encoder lstm1 = LSTM(latent dim,return sequences=True,return state=True)
encoder output1, state h1, state c1 =encoder lstm1(enc emb)

#LSTM 2
encoder lstm2 = LSTM(latent dim,return sequences=True,return state=True)
encoder output2, state h2, state c2 =encoder lstm2(encoder output1)

#LSTM 3
encoder lstm3=LSTM(latent dim, return state=True, return sequences=True)
encoder outputs, state h, state c= encoder lstm3(encoder output2)

# Set up the decoder.
decoder inputs = Input(shape=(None,))
dec emb layer = Embedding(y voc size, latent dim,trainable=True)
dec emb =dec emb layer(decoder inputs)

#LSTM using encoder states as initial state
decoder lstm = LSTM(latent dim, return sequences=True, return state=True)
decoder outputs,decoder fwd state, decoder back state = decoder lstm(dec emb,initial state=[state h,
state c])

# Attention Layer
Attention layer attn layer = AttentionLayer(name='attention layer')
attn out, attn states = attn layer([encoder outputs, decoder outputs])

# Concat attention output and decoder LSTM output
decoder concat input =Concatenate(axis=-1, name='concat layer')([decoder outputs, attn out])

#Dense layer
decoder dense = TimeDistributed(Dense(y voc size, activation='softmax'))
decoder outputs = decoder dense(decoder concat input)

# Define the model
model = Model([encoder inputs, decoder inputs], decoder outputs)
model.summary()
```

Figure 13: Three stack LSTM for encoder



```
==================================================================================================
input_1 (InputLayer)           (None, 80)           0
__________________________________________________________________________________________________
embedding (Embedding)          (None, 80, 500)      25785500    input 1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                    [(None, 80, 500), (N 2002000     embedding[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)           (None, None)         0

lstm_1 (LSTM)                  [(None, 80, 500), (N 2002000     lstm[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)        (None, None, 500)    7048000     input_2[0][0]
__________________________________________________________________________________________________
lstm 2 (LSTM)                  [(None, 80, 500), (N 2002000     lstm 1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                  [(None, None, 500),  2002000     embedding_1[0][0]
                                                                lstm_2[0][1]
                                                                lstm_2[0][2]
__________________________________________________________________________________________________
attention_layer (AttentionLayer [(None, None, 500),  500500      lstm_2[0][0]
                                                                lstm_3[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)     (None, None, 1000)   0           lstm_3[0][0]
                                                                attention layer[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 14096)  14110096    concat_layer[0][0]
==================================================================================================
Total params: 55,452,096
Trainable params: 55,452,096
Non-trainable params: 0
```

Figure 14: Output for 2 stack LSM for encoder

Sparse categorizing cross-entropy is the loss function, which turns an integer string into a one-hot vector on the fly. This takes care of any memory issues.

model. Compile (optimizer='rmsprop', loss='sparse_categorical_crossentropy')

Based on a user-specified measure, it decides when to cease training the neural network. The validity loss is something we're keeping an eye on (if it becomes too high, our model will cease training).:

es = Early Stopping (monitor='val_loss', mode='min', verbose=1)

We'll test the model on the holdout set after training it with 512 batches (which makes up 10 percent of our dataset):

y tr.reshape(y tr.shape[0],y tr.shape[1], 1)[:,1:] history=model.fit([x tr,y tr[:,:-1]], y tr.reshape(y tr.shape[0],y tr.shape[1], 1)[:,1:] ,epochs=50,callbacks=[es],batch size=512, validation data=,epochs=50,callbacks=[es],batch size=512, validation data= y val.reshape(y val.shape[0],y val.shape[1], 1)[:,1:] ([x val,y val[:,:-1]), y val.reshape(y val.shape[0],y val.shape[1], 1)[:,1:]))

A. Understanding the Diagnostic plot

Now we'll make a few diagnostic graphs to see how the model behaves over time.:

from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend() pyplot.show()

Output:

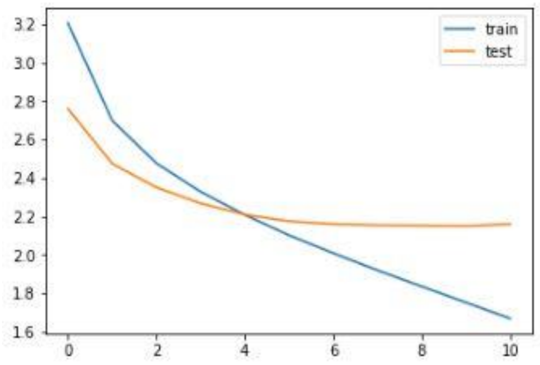


Figure 15: Output of Cross Entropy

We may conclude that the prediction error has grown somewhat after epoch 10. As a result, after this epoch, we will no longer train the model.

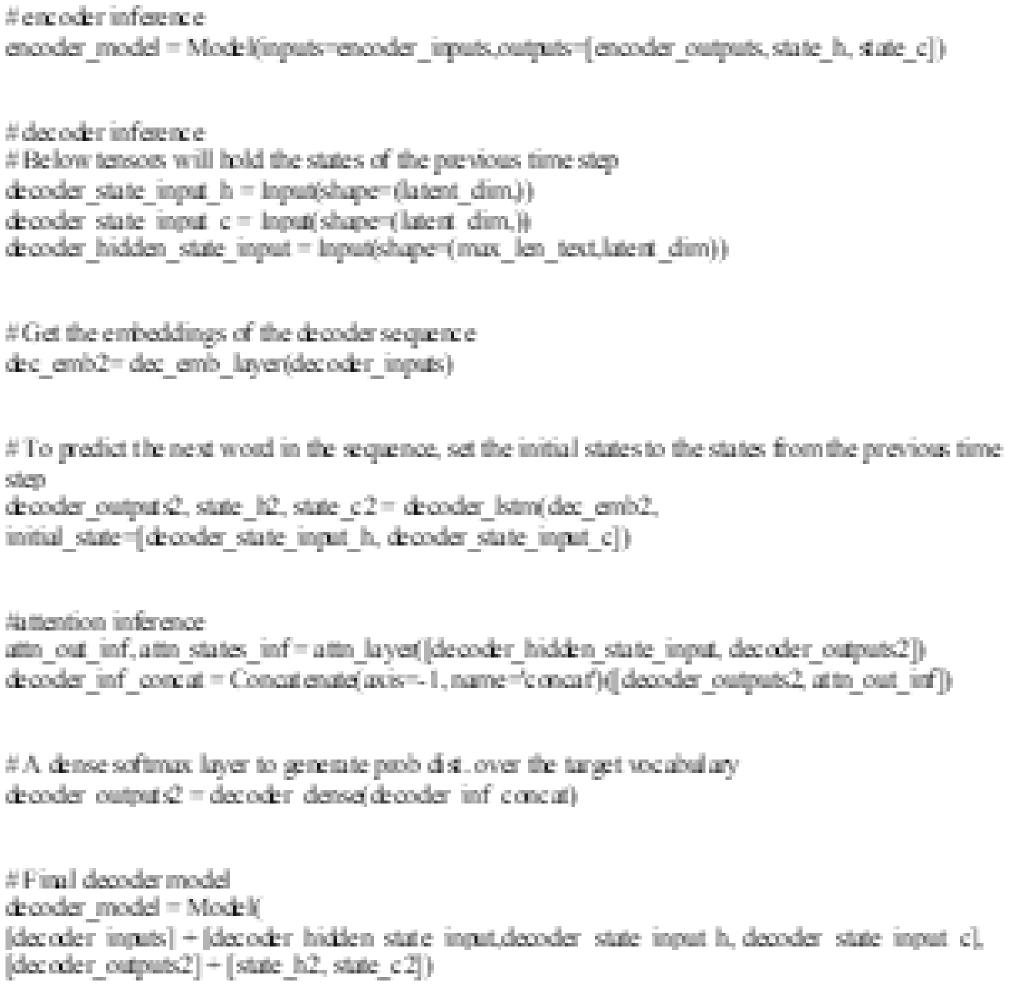
Let's now create a dictionary to translate the index to words for both the target and source vocabulary:

reverse_target_word_index=y_tokenizer.index_word
reverse_source_word_index=x_tokenizer.index_word
target_word_index=y_tokenizer.word_index

B. Inference

The various steps involved are

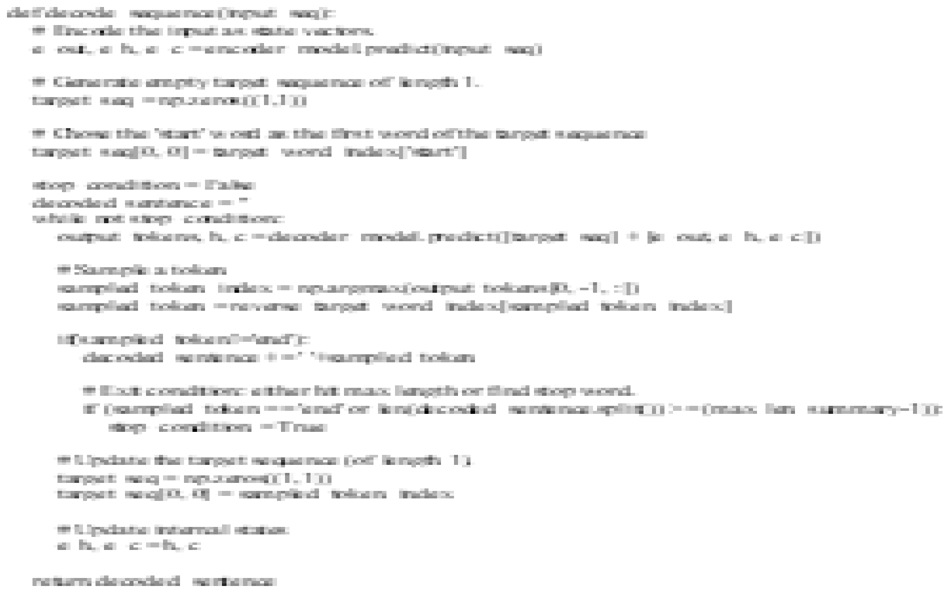
Step 1: Set up the inference for the encoder and decoder as shown in Figure 17



```
#encoder inference
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, state_c])


#decoder inference
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder state input c = Input(shape=(latent dim,))
decoder_hidden_state_input = Input(shape=(max_len_text,latent_dim))


# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)


# To predict the next word in the sequence, set the initial states to the states from the previous time
step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,
initial_state=[decoder_state_input_h, decoder_state_input_c])


#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_out_inf])


# A dense softmax layer to generate prob dist. over the target vocabulary
decoder outputs2 = decoder dense(decoder inf concat)


# Final decoder model
decoder_model = Model(
[decoder inputs] + [decoder hidden state input,decoder state input h, decoder state input c],
[decoder_outputs2] + [state_h2, state_c2])
```

Figure 16: Inference setup for Encoder and Decoder

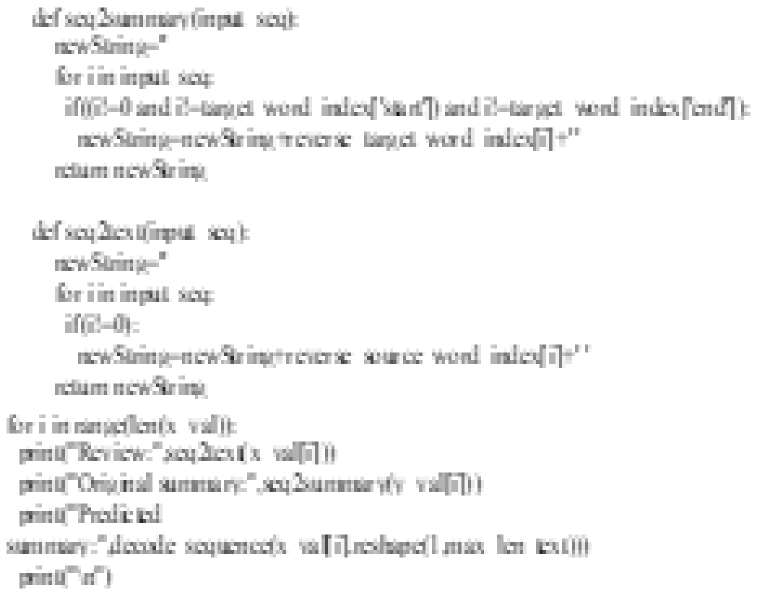
Step 2 Figure 18 illustrates how this function conducts the inference technique.



```
def decode sequence(input seq):
    # Encode the input as state vectors.
    e out, e h, e c = encoder model.predict(input seq)

    # Generate empty target sequence of length 1.
    target seq = np.zeros((1,1))

    # Chose the 'start' word as the first word of the target sequence
    target seq[0, 0] = target word index['start']

    stop condition = False
    decoded sentence = ''
    while not stop condition:
        output tokens, h, c = decoder model.predict([target seq] + [e out, e h, e c])

        # Sample a token
        sampled token index = np.argmax(output tokens[0, -1, :])
        sampled token = reverse target word index[sampled token index]

        if(sampled token!='end'):
            decoded sentence += ' '+sampled token

            # Exit condition: either hit max length or find stop word.
            if (sampled token == 'end' or len(decoded sentence.split()) >= (max len summary-1)):
                stop condition = True

        # Update the target sequence (of length 1).
        target seq = np.zeros((1,1))
        target seq[0, 0] = sampled token index

        # Update internal states
        e h, e c = h, c

    return decoded sentence
```

Figure 17: The interference procedure implementation

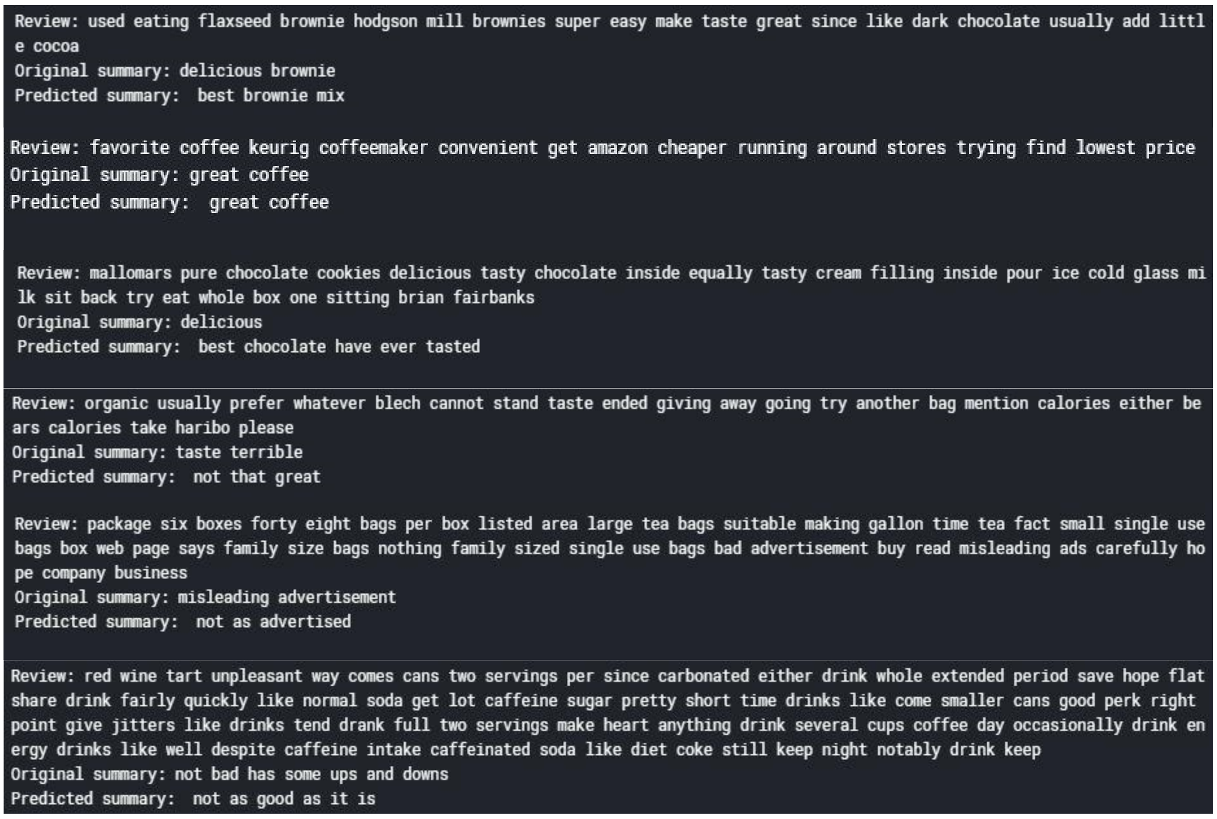
Step 3 As illustrated in Figure 19, we now construct the routines to transform a numeric sequence to a word sequence for the summary and reviews.



```
  def seq2summary(input_seq):
    newString=''
    for i in input_seq:
      if((i!=0 and i!=target_word_index['start']) and i!=target_word_index['end']):
        newString=newString+reverse_target_word_index[i]+' '
    return newString

  def seq2text(input_seq):
    newString=''
    for i in input_seq:
      if(i!=0):
        newString=newString+reverse_source_word_index[i]+' '
    return newString
for i in range(len(x_val)):
  print("Review:",seq2text(x_val[i]))
  print("Original summary:",seq2summary(y_val[i]))
  print("Predicted
summary:",decode_sequence(x_val[i].reshape(1,max_len_text)))
  print("\n")
```

Figure 18: Defining functions and converting into a word sequence

Step 4 A few summaries generated by the model are shown in Figure 20.:



```
Review: used eating flaxseed brownie hodgson mill brownies super easy make taste great since like dark chocolate usually add littl
e cocoa
Original summary: delicious brownie
Predicted summary:  best brownie mix

Review: favorite coffee keurig coffeemaker convenient get amazon cheaper running around stores trying find lowest price
Original summary: great coffee
Predicted summary:  great coffee


Review: mallomars pure chocolate cookies delicious tasty chocolate inside equally tasty cream filling inside pour ice cold glass mi
lk sit back try eat whole box one sitting brian fairbanks
Original summary: delicious
Predicted summary:  best chocolate have ever tasted

Review: organic usually prefer whatever blech cannot stand taste ended giving away going try another bag mention calories either be
ars calories take haribo please
Original summary: taste terrible
Predicted summary:  not that great

Review: package six boxes forty eight bags per box listed area large tea bags suitable making gallon time tea fact small single use
bags box web page says family size bags nothing family sized single use bags bad advertisement buy read misleading ads carefully ho
pe company business
Original summary: misleading advertisement
Predicted summary:  not as advertised

Review: red wine tart unpleasant way comes cans two servings per since carbonated either drink whole extended period save hope flat
share drink fairly quickly like normal soda get lot caffeine sugar pretty short time drinks like come smaller cans good perk right
point give jitters like drinks tend drank full two servings make heart anything drink several cups coffee day occasionally drink en
ergy drinks like well despite caffeine intake caffeinated soda like diet coke still keep night notably drink keep
Original summary: not bad has some ups and downs
Predicted summary:  not as good as it is
```

Figure 19: Examples of the Output Summaries

VII. CONCLUSION

Despite the fact that our model's summary and the real one are not the same length, they both communicate the same idea. My model can provide a comprehensible summary of the information using context from the text.
This is how deep learning methods in Python may be used to summarize text.

- Build the model by expanding the training dataset. As the amount of the training dataset expands, a deep learning model's ability to generalize improves.
- Use a Bi-Directional LSTM to capture information from both directions and generate a better context vector.
- Use the beam search strategy to decode the test sequence instead of the greedy method (argmax)
- assess your effectiveness of the algorithm using the BLEU score.
- Set up pointer-generating networks and techniques of inclusion.

REFERENCES

[1] Amjad Abu-Jbara and Dragomir Radev.. "Coherent citation-based summarization of scientific papers". -Volume 1. Association for Computational Linguistics, 500–509. 2011

[2] Rasim M Alguliev, Ramiz M Aliguliyev, Makrufa S Hajirahimova, and ChingizAMehdiyev. 2011. MCMR: "Maximum coverage and minimum redundant text summarization model." Expert Systems with Applications 38, 12 (2011), 14514–14522.

[3] Rasim M Alguliev, Ramiz M Aliguliyev, and Nijat R Isazade.. "Multiple documents summarization based on evolutionary optimization algorithm." 2013

[4] Mehdi Allahyari and KrysKochut.. "Automatic topic labeling using ontology-based topic models. In Machine Learning and Applications (ICMLA)", 2015 IEEE 14th International Conference on. IEEE, 259–264.

[5] Mehdi Allahyari and KrysKochut. 2016. "Discovering Coherent Topics with Entity Topic Models. In Web Intelligence (WI)", 2016 IEEE/WIC/ACM International Conference on. IEEE, 26–33.

.

[6] Mehdi Allahyari and KrysKochut. "Semantic Context-Aware Recommendation via Topic Models Leveraging Linked Open Data. In International Conference on Web Information Systems Engineering. Springer" , 263–277. 2016.

[7] Mehdi Allahyari and KrysKochut. "Semantic Tagging Using Topic Models Exploiting Wikipedia Category Network. In Semantic Computing (ICSC)", 2016.

[8] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut. 2017. A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. ArXiv e-prints (2017). arXiv:1707.02919

[9] Einat Amitay and Cécile Paris.." Automatically summarising web sites: is there a way around it?." 2000

[10] Elena Baralis, Luca Cagliero, Saima Jabeen, Alessandro Fiori, and Sajid Shah.". Multi-document summarization based on the Yago ontology. Expert Systems with Applications" 40, 17 (2013), 6976–6984. 2013

[11] Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. . "Jointly learning to extract and compress. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-", 481–490. 2011

[12] David M Blei, Andrew Y Ng, and Michael I Jordan.. "Latent dirichlet allocation. the Journal of machine Learning research" (2003), 993–1022.

[13] Asli Celikyilmaz and DilekHakkani-Tur." A hybrid hierarchical model for multi-document summarization." 2010

[14] YlliasChali and Shafiq R Joty.. "Improving the performance of the random walk model for answering complex questions" 2008.

[15] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, and others."Semi supervised learning. Vol. 2. MIT press Cambridge" 2006.

[16] Ping Chen and Rakesh Verma. 2006." A query-based medical information summarization system using ontology knowledge" 2006.

[17] Freddy Chong Tat Chua and Sitaram Asur. 2013. "Automatic Summarization of Events from Social Media" 2008

[18] John M Conroy and Dianne P O'leary.. "Text summarization via hidden markov models". In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 406–407. 2001