CSE1901
Technical Answers to Real-World Problems

Review - 3

Text Summarization of Fine Food Reviews using Stacked LSTM with Attention Mechanism

Submitted to: Prof. Sasikala R.
Slot: TB1 + TB2

Team No.: 2

Team Members

Name	Registration Number
Manas Bhardwaj	19BCE0317
Aryan Blouria	19BCE0330
Samarth Arora	19BCE0334

DECLARATION BY THE CANDIDATE

We hereby declare that the project report entitled "**Text Summarization of Fine Food Reviews using Stacked LSTM with Attention Mechanism"** submitted by us to Vellore Institute of Technology, Vellore in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** is a record of bonafide project work undertaken by us under the supervision of **Prof. Sasikala R., SCOPE.** We further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Manas Bhardwaj – 19BCE0317

Aryan Blouria – 19BCE0330

Samarth Arora – 19BCE0334

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**Text Summarization of Fine Food Reviews using Stacked LSTM with Attention Mechanism"** submitted by **MANAS BHARDWAJ (19BCE0317), ARYAN BLOURIA (19BCE0330)** and **SAMARTH ARORA (19BCE0334)** to Vellore Institute of Technology, Vellore in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** is a record of bonafide project work undertaken by they/them under my supervision. The project fulfils the requirements as per the regulations of this Institute and in my opinion, meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Signature of the Supervisor

Table of Contents

1. Abstract

Customer reviews can often be long and descriptive. Analyzing these reviews manually, as you can imagine, is really time-consuming. This is where the brilliance of Natural Language Processing can be applied to generate a summary for long reviews, making it easier for the user to conclude their search.

Abstractive Text Summarization is the task of generating a short and concise summary that captures the salient ideas of the source text. The generated summaries potentially contain new phrases and sentences that may not appear in the source text.

Our objective here is to generate a summary for Amazon Fine Food reviews by implementing the abstractive summarization using a stacked LSTM with 3 LSTM layers stacked on top of each other. This approach is better than a single-layered model since stacking multiple LSTM layers allows for a better model architecture and accuracy. In addition, we will also be using the Attention mechanism to further tackle weaknesses of the traditional LSTM model and allow it to account for differences in the importance of the input words.

Thus, our approach will enable the model to accept text in the English language and correspondingly produce a succinct summary while retaining the original meaning.

2. Introduction

With the rise of the Internet today, we have huge amounts of data in various natural languages like news articles, reviews etc. Text Summarization is a technique that is used to summarize language data so that we can understand the gist of the entire article very easily. It is difficult to train a computer to perform tasks that involve Natural Language because producing a summary of the original text is a difficult task even for humans. However, the recent advancements in NLP like seq2seq Architecture, Attention Mechanism, Word Embeddings helps us in developing complex models.

A. Text Summarization Approaches: Text Summarization can be classified into two types:

- Extractive Summarization: Extractive Summarization finds important sentences, passages from the given original text and then creates the summary using those phrases. The summarized version is part of the original text, new phrases in the summarized version cannot be found.
- Abstractive Summarization: Abstractive Summarization first understands the context of the original text and then forms the summary. It is different compared to Extractive Summarization as this produces new sentences in the summary which were not present in the original text. It is an advanced approach and is a difficult task but with the help of seq2seq model architecture, an Abstractive Summarization Model can be developed.

B. Introduction to Seq2seq Architecture: Sequence-to-Sequence or “seq2seq” model is made up of Recurrent Neural Networks. It is also called an encoder-decoder model. The encoder takes input

as a sequence of words and encodes it into corresponding hidden vectors called "context vectors". The decoder will decode the context vectors and produce output as a sequence of words. Both the Encoder and the Decoder use an RNN cell to read input cells. In Decoder, an End of Sentence token is appended at the end of the input sentence. Both Encoder and Decoder networks are trained simultaneously so that both learn the same context vector representation. Seq2seq Model doesn't require inputs and outputs to be of equal length. It has revolutionized the way of Natural Language Processing with the help of Deep Neural Networks.

C. Stacked LSTMs: Long Short Term Memory networks or simply "LSTMs" – are a special kind of Recurrent neural network which is able to learn long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem and hence they are able to remember information for a longer duration. The Stacked LSTM adds to this model by having multiple hidden LSTM layers in which each layer contains multiple memory cells. The main reason for stacking LSTM is that it allows a greater model complexity.

D. Attention Mechanism: Attention Mechanism is an attempt to implement the action of concentrating on only a few important things while ignoring others in deep neural networks. Instead of reading all the words in the source sequence, there is an increased importance of specific parts of the source sequence that result in the target sequence. There are 2 different attention mechanisms: Global Attention and Local Attention. We'll be using global attention with stacked LSTMs to develop an abstractive summarization model.

3. Dataset

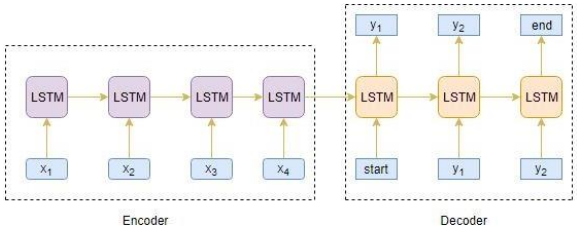
We worked on the Amazon Fine Food reviews dataset. This dataset consists of reviews of fine foods from Amazon. The data spans a period of more than 10 years, including all ~500,000 reviews up to October 2012. These reviews include product and user information, ratings, plain text review, and summary.

Columns: ProductID, UserID, Profile Name, No. of Helpful Votes, No. of Total Votes, Score, Timestamp for Review, Summary, Text of Review

Link: [Amazon Fine Food Reviews]

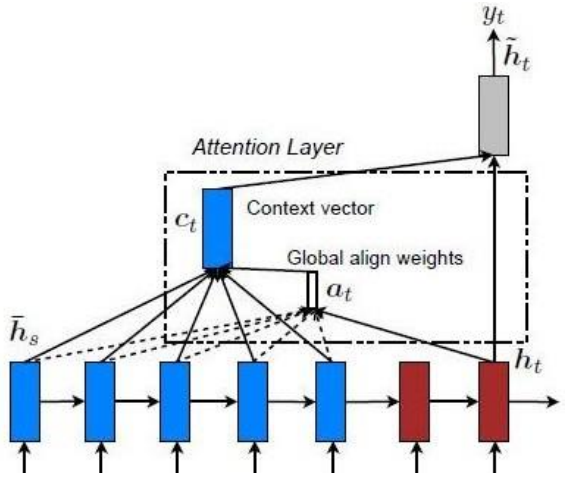
4. Methodology

Our model is built using the Encoder-decoder sequence to sequence architecture. Our model will be a stacked LSTM with 3 LSTM layers stacked on top of each other. This approach is better than a single-layered model since stacking multiple LSTM layers allows for a better model architecture and accuracy.



In addition, we will also be using the Attention mechanism to further tackle weaknesses of the traditional LSTM model

and allow it to account for differences in the importance of the input words.



The following are the steps taken to build the abstractive summarizer model:

Importing and reading the dataset

Firstly we need to load the 'Amazon Fine Food reviews' dataset, which is available on Kaggle, into Colab Notebook. In this dataset, there are 10 columns out of which we only need the summary and text columns. We drop the rest of the 8 columns and rename the 'text' to review and interchange the two columns. We then remove all the duplicate entries in the dataset.

Preprocessing and text cleaning

The next step is to preprocess the data. There are various steps in preprocessing like stopword removal, lowercasing, contraction mapping, short words removal, stemming etc. We have defined several functions which carry out each step of preprocessing. For stopword removal, we are using the 'Natural Language Toolkit' library to get a set of English stopwords. We have also built a contraction dictionary, which contains all the contractions of the English language and their expansions. Both the reviews and summaries are then preprocessed.

Analyzing the data

We then analyze the length of the reviews and the summary to get an overall idea about the distribution of length of the text, using matplotlib library, we plot two charts that show the number of reviews/summaries with the number of words present in them. From the graph, we can find the majority review and summary length.

Preparing the tokenizer

We then build a tokenizer that helps to build the vocabulary and converts word sequences into integer sequences. We first create an instance of tokenizer for the reviews in the training data. Then we define the threshold to be 4 which means a word whose count is below 4 is considered as a rare word. From this we find the total number of rare words and subtract it from the size of vocabulary to get the number of most common words. We then create another tokenizer instance with the top most common words for reviews. We then convert the words in the reviews into its corresponding integer sequence for both the training and the testing sets. We then pass these integer sequences with 0's to make all the inputs of the same size. We then do the same procedure for the original summaries in the dataset.

Model creation and training

The next step is to build the model. Our model consists of a 3-layer stacked LSTM along with an attention mechanism. The model consists of input layers, embedding layers, LSTM layers and the attention layer in the encoder architecture. Concatenate and

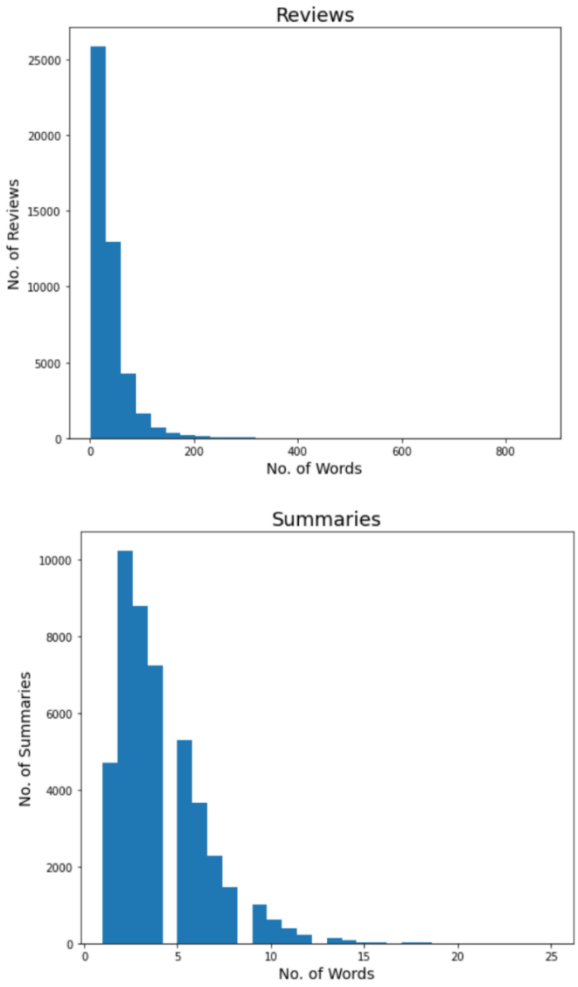
Time Distributed layers are used in the decoder architecture. The model is then trained with early stopping in case the validation loss starts increasing. We also plot a diagnostic plot to understand how our model is behaving.

Generating summaries

To generate the summaries, we need to convert the integer sequences generated by the model back to word sequences. The input sequence is encoded as the internal states of the encoder. We generate an empty target sequence of length 1, and the first word of the target sequence is set as the start token sumstart. Now the decoder is initialized with the target sequence and the encoder states. The output of this is the probability of the next word. The word with the maximum probability is selected. This sampled word is added to the decoded sentence variable and is passed as input to the decoder in the next timestep and the internal states are updated with the current time step. This process will repeat until we hit the end token sumend, or we reach the maximum length of the target sequence. We finally return the decoded sentence. In the end, each review is then printed with its original and predicted summary.

5. Results and Discussion

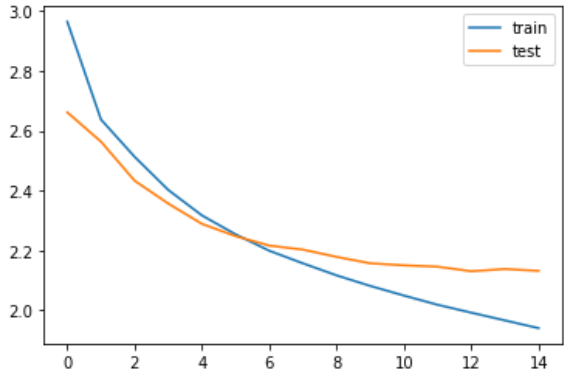
We visualized our dataset into histograms showing the total numbers of reviews and summaries and the number of words contained in them. Using this information, we deduced that the majority of the reviews were under 60 words and the majority of the summaries were under 8 words. As the LSTM model only accepts fixed length inputs, we specified the input review and summary lengths to be 60 and 8 words respectively.



As we wanted to generate the vocabulary of the model using only the common words present in the reviews and summaries, we defined a rarity threshold of 4 occurrences per word for reviews and 6 occurrences per word for summaries. Based on these thresholds, we calculated the rare word coverage to be only 3 percent of the total word occurrences and thus the size of the review vocabulary generated using the most common words came out to be 8692 words. Similarly, we calculated the size of the summary vocabulary to be 1703 words.

After creating the model by stacking 3 LSTM layers along with a global attention mechanism, we trained and tested it on the basis of validation loss with the aim to minimize it as much as possible. Our model was compiled using the rmsprop optimiser and sparse categorical crossentropy as the loss function. We also used an early stopping mechanism which stops the model training if the loss increases for two consecutive epochs.

After training the model, we visualized the loss and validation loss of the model during training across the number of epochs and observed that the negative gradient became negligible after the 14th epoch and thus, the early stopping mechanism stopped the training process midway.



Finally, after converting the integer sequences produced by the model back to textual data, we obtained the generated summaries and concluded that they were meaningful, concise and captured the essence of the review using words that were not even present in the original review, hence verifying that the summarization was indeed abstractive in nature.

Sample output:

```
Review: drops remember ones expecting powdered sugar disappointed
Original summary: lemon drops leave sour taste
Predicted summary:  not as good as expected
```

6. Conclusion

We have successfully loaded, analysed and described the Fine Food Reviews dataset and gained useful insights about the data. Next, we have accordingly pre-processed the data, including lowercase conversion, stopword removal etc. to make the data suitable for training and validating the 3-layer Stacked LSTM model that we have implemented. We have also successfully integrated the attention mechanism with our stacked LSTM architecture to improve its performance. After training the model, we are able to generate short summaries that also include words that were not present in the original review text. Hence, we conclude that we have successfully created an abstractive text summarizer for food reviews.

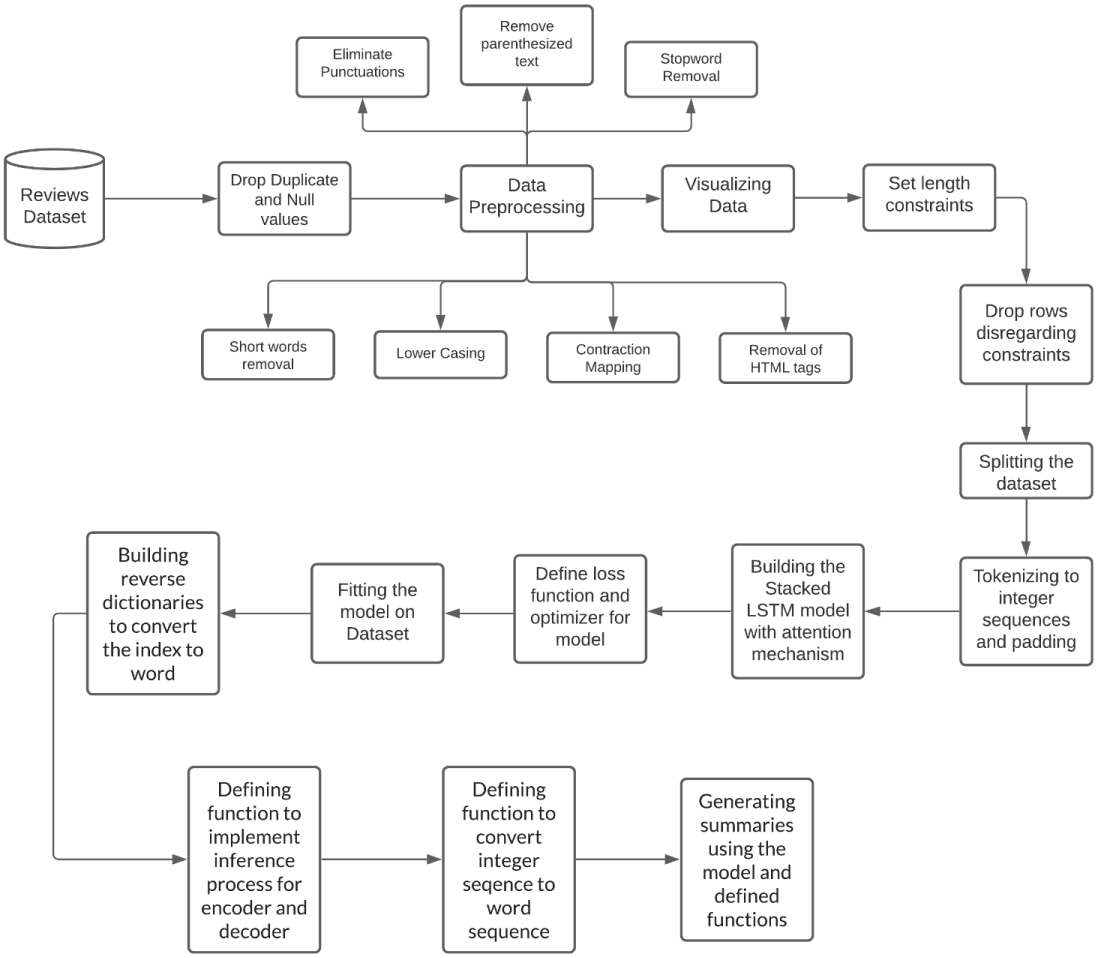
This project can be further implemented in the form of a web application in which the user can enter some text and get their summaries or in the form of a browser extension in which the contents of a page can be summarized. Also, right now the model has been trained on a dataset of only 500,000 reviews. By increasing the size of the dataset along with using powerful GPUs for it, we can improve the performance of the model.

7. References

1. Suleiman, D. and Awajan, A., 2020. Deep learning based abstractive text summarization: Approaches, datasets, evaluation measures, and challenges. Mathematical Problems in Engineering, 2020.
2. Song, S., Huang, H. and Ruan, T., 2019. Abstractive text summarization using LSTM-CNN based deep learning. Multimedia Tools and Applications, 78(1), pp.857-875.
3. Masum, A.K.M., A Bengali Text Generation Approach in Context of Abstractive Text Summarization using RNN.
4. Zhang, Y., Li, D., Wang, Y., Fang, Y. and Xiao, W., 2019. Abstract text summarization with a convolutional Seq2seq model. Applied Sciences, 9(8), p.1665.
5. Yang, M., Qu, Q., Shen, Y., Liu, Q., Zhao, W. and Zhu, J., 2018, August. Aspect and sentiment aware abstractive review summarization. In Proceedings of the 27th international conference on computational linguistics (pp. 1110-1120).
6. Masum, A.K.M., Abujar, S., Talukder, M.A.I., Rabby, A.S.A. and Hossain, S.A., 2019, July. Abstractive method of text summarization with sequence to sequence RNNs. In 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-5). IEEE.
7. Jiang, J., Zhang, H., Dai, C., Zhao, Q., Feng, H., Ji, Z. and Ganchev, I., 2021. Enhancements of Attention-Based Bidirectional LSTM for Hybrid Automatic Text Summarization. IEEE Access, 9, pp.123660-123671.
8. Hanunggul, P.M. and Suyanto, S., 2019, December. The impact of local attention in lstm for abstractive text summarization. In 2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI) (pp. 54-57). IEEE.
9. Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H. and Xu, B., 2016, August. Attention-based bidirectional long short-term memory networks for relation classification. In Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers) (pp. 207-212).
10. Roul, R.K., Joshi, P.M. and Sahoo, J.K., 2019, December. Abstractive Text Summarization Using Enhanced Attention Model. In International Conference on Intelligent Human Computer Interaction (pp. 63-76). Springer, Cham.
11. Li, P., Lam, W., Bing, L. and Wang, Z., 2017. Deep recurrent generative decoder for abstractive text summarization. arXiv preprint arXiv:1708.00625.
12. Nallapati, R., Zhou, B., Gulcehre, C. and Xiang, B., 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. arXiv preprint arXiv:1602.06023.

8. Appendix

Architecture Diagram:



Code:

```
import numpy as np
import pandas as pd
from warnings import filterwarnings
filterwarnings('ignore')

from google.colab import drive
drive.mount('/content/drive')

data = pd.read_csv("/content/drive/My Drive/TARP Project/Reviews.csv", nrows=50000)

data.head()

data.shape

data.columns

data.isnull().sum()

data.info()

df = data.drop(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
'HelpfulnessDenominator', 'Score', 'Time'], axis = 1)
df.rename(columns={'Text': 'Review'}, inplace = True)

column_titles = ["Review", "Summary"]
df = df.reindex(columns = column_titles)

df.drop_duplicates(subset = ['Review'], inplace = True)
df.dropna(axis = 0, inplace = True)

df.head()

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = stopwords.words('english')

contraction_map = {
 "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", …….. }
```

```
import re
from bs4 import BeautifulSoup

def make_lowercase(text):
  return text.lower()

def remove_HTML(text):
  return BeautifulSoup(text, 'lxml').text

def map_contractions(text):
  return ' '.join([contraction_map[word] if word in contraction_map else word for word in
text.split(" ")])

def remove_extras(text):
  text = re.sub("", '', text)
  text = re.sub(r"'s\b", "", text)
  text = re.sub(r'\([^)]*\)', '', text)
  text = re.sub("[^a-zA-Z]", " ", text)
  return text

def remove_stopwords(text, type):
  if type == 0:
    tokens = [word for word in text.split() if not word in stop_words]
  else:
    tokens = text.split()
  return tokens

def remove_shortwords(tokens):
  long_words = []
  for token in tokens:
    if len(token) > 1:
      long_words.append(token)
  return (" ".join(long_words)).strip()

processed_reviews = []
processed_summaries = []

for review, summary in zip(df['Review'], df['Summary']):
  review, summary = make_lowercase(review), make_lowercase(summary)
  review, summary = remove_HTML(review), remove_HTML(summary)
  review, summary = map_contractions(review), map_contractions(summary)
  review, summary = remove_extras(review), remove_extras(summary)
  review, summary = remove_stopwords(review, 0), remove_stopwords(summary, 1)
  review, summary = remove_shortwords(review), remove_shortwords(summary)
  processed_reviews.append(review)
  processed_summaries.append(summary)
```

```
processed_reviews[:10]

processed_summaries[:10]

df['Review'] = processed_reviews
df['Summary'] = processed_summaries

df.replace('', np.nan, inplace = True)
df.dropna(axis = 0, inplace = True)

review_length = []
summary_length = []

for review, summary in zip(df['Review'], df['Summary']):
  review_length.append(len(review.split()))
  summary_length.append(len(summary.split()))

import matplotlib.pyplot as plt

plt.figure(figsize = (18, 7))
plt.subplot(1, 2, 1)
plt.title('Reviews', fontsize = 18)
plt.xlabel('No. of Words', fontsize = 14)
plt.ylabel('No. of Reviews', fontsize = 14)
plt.hist(review_length, bins = 30)

plt.subplot(1, 2, 2)
plt.title('Summaries', fontsize = 18)
plt.xlabel('No. of Words', fontsize = 14)
plt.ylabel('No. of Summaries', fontsize = 14)
plt.hist(summary_length, bins = 30)
plt.show()

review_count = 0
summary_count = 0

for review, summary in zip(df['Review'], df['Summary']):
  if len(review.split()) <= 50:
    review_count = review_count + 1
  if len(summary.split()) <= 8:
    summary_count = summary_count + 1

print("% of reviews having less than 80 words: ", (review_count / len(df['Review'])) * 100)
print("% of summaries having less than 8 words: ", (summary_count / len(df['Summary'])) *
100)
```

```
max_review_length = 50
max_summary_length = 8

reviews = np.array(df['Review'])
summaries = np.array(df['Summary'])

short_reviews = []
short_summaries = []

for index in range(len(reviews)):
  if len(reviews[index].split()) <= max_review_length and len(summaries[index].split()) <=
max_summary_length:
    short_reviews.append(reviews[index])
    short_summaries.append(summaries[index])

df = pd.DataFrame({'Review': short_reviews, 'Summary': short_summaries})
df['Summary'] = df['Summary'].apply(lambda summary: 'sumstart ' + summary + ' sumend')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(np.array(df['Review']),
np.array(df['Summary']), test_size = 0.1, random_state = 10, shuffle = True)

from keras_preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences

X_tokenizer = Tokenizer()
X_tokenizer.fit_on_texts(list(X_train))

review_rare_threshold = 4

review_rare_words = 0
review_total_words = 0
review_rare_frequency = 0
review_total_frequency = 0

for frequency in X_tokenizer.word_counts.values():
  review_total_words = review_total_words + 1
  review_total_frequency = review_total_frequency + frequency
  if (frequency < review_rare_threshold):
    review_rare_words = review_rare_words + 1
    review_rare_frequency = review_rare_frequency + frequency

print("% of rare words in vocabulary:", (review_rare_words/review_total_words) * 100)
print("Total coverage of rare words:", (review_rare_frequency/review_total_frequency) * 100)

review_common_words = review_total_words - review_rare_words
```

```
X_tokenizer = Tokenizer(num_words = review_common_words)
X_tokenizer.fit_on_texts(list(X_train))

X_train_sequence = X_tokenizer.texts_to_sequences(X_train)
X_test_sequence = X_tokenizer.texts_to_sequences(X_test)

X_train = pad_sequences(X_train_sequence, maxlen = max_review_length, padding = 'post')
X_test = pad_sequences(X_test_sequence, maxlen = max_review_length, padding = 'post')

X_vocabulary_size = X_tokenizer.num_words + 1
print("Size of Review Vocabulary: ", X_vocabulary_size)

y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))

summary_rare_threshold = 6

summary_rare_words = 0
summary_total_words = 0
summary_rare_frequency = 0
summary_total_frequency = 0

for frequency in y_tokenizer.word_counts.values():
  summary_total_words = summary_total_words + 1
  summary_total_frequency = summary_total_frequency + frequency
  if (frequency < summary_rare_threshold):
    summary_rare_words = summary_rare_words + 1
    summary_rare_frequency = summary_rare_frequency + frequency

print("% of rare words in vocabulary:", (summary_rare_words/summary_total_words) * 100)
print("Total coverage of rare words:", (summary_rare_frequency/summary_total_frequency) *
100)

summary_common_words = summary_total_words - summary_rare_words

y_tokenizer = Tokenizer(num_words = summary_common_words)
y_tokenizer.fit_on_texts(list(y_train))

y_train_sequence = y_tokenizer.texts_to_sequences(y_train)
y_test_sequence = y_tokenizer.texts_to_sequences(y_test)

y_train = pad_sequences(y_train_sequence, maxlen = max_summary_length, padding = 'post')
y_test = pad_sequences(y_test_sequence, maxlen = max_summary_length, padding = 'post')

y_vocabulary_size = y_tokenizer.num_words + 1
```

```
print("Size of Summary Vocabulary: ", y_vocabulary_size)

empty = []
for summary in range(len(y_train)):
  token_count = 0
  for token in y_train[summary]:
    if token != 0:
      token_count = token_count + 1
  if token_count == 2:
    empty.append(summary)

X_train = np.delete(X_train, empty, axis = 0)
print(X_train)

y_train = np.delete(y_train, empty, axis = 0)
print(y_train)

empty = []
for summary in range(len(y_test)):
  token_count = 0
  for token in y_test[summary]:
    if token != 0:
      token_count = token_count + 1
  if token_count == 2:
    empty.append(summary)

X_test = np.delete(X_test, empty, axis = 0)
print(X_test)

y_test = np.delete(y_test, empty, axis = 0)
print(y_test)

from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate,
TimeDistributed
from tensorflow.keras.models import Model

from attention import AttentionLayer
from keras import backend as K

latent_dimension = 300
embedding_dimension = 100

encoder_inputs = Input(shape=(max_review_length,))
encoder_embed_layer =  Embedding(X_vocabulary_size, embedding_dimension,
trainable=True)
encoder_embedding = encoder_embed_layer(encoder_inputs)
```

```
encoder_lstm1 = LSTM(latent_dimension, return_sequences=True, return_state=True,
dropout=0.4, recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(encoder_embedding)

encoder_lstm2 = LSTM(latent_dimension, return_sequences=True, return_state=True,
dropout=0.4, recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

encoder_lstm3 = LSTM(latent_dimension, return_sequences=True, return_state=True,
dropout=0.4, recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

decoder_inputs = Input(shape=(None,))
decoder_embed_layer = Embedding(y_vocabulary_size, embedding_dimension, trainable=True)
decoder_embedding = decoder_embed_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dimension, return_sequences=True, return_state=True,
dropout=0.4, recurrent_dropout=0.2)
decoder_outputs, decoder_fwd_state, decoder_back_state = decoder_lstm(decoder_embedding,
initial_state=[state_h, state_c])

attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs, attn_out])
decoder_dense =  TimeDistributed(Dense(y_vocabulary_size, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.summary()

model.compile(optimizer = 'rmsprop', loss = 'sparse_categorical_crossentropy',
experimental_run_tf_function = False)

from tensorflow.keras.callbacks import EarlyStopping

es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience = 2)

history = model.fit([X_train, y_train[:, :-1]], y_train.reshape(y_train.shape[0], y_train.shape[1],
1)[:, 1:], epochs = 20, callbacks = [es], batch_size = 128, validation_data=([X_test, y_test[:, :-1]],
y_test.reshape(y_test.shape[0], y_test.shape[1], 1)[:, 1:]))

plt.plot(history.history['loss'], label = 'train')
plt.plot(history.history['val_loss'], label = 'test')
plt.legend()
```

```
plt.show()

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

encoder_model = Model(inputs = encoder_inputs, outputs = [encoder_outputs, state_h, state_c])

decoder_state_input_h = Input(shape = (latent_dimension,))
decoder_state_input_c = Input(shape = (latent_dimension,))
decoder_hidden_state_input = Input(shape = (max_review_length, latent_dimension))

decoder_embedding2 = decoder_embed_layer(decoder_inputs)
decoder_outputs2, state_h2, state_c2 = decoder_lstm(decoder_embedding2, initial_state =
[decoder_state_input_h, decoder_state_input_c])

attn_out_inference, attn_states_inference = attn_layer([decoder_hidden_state_input,
decoder_outputs2])
decoder_inference_concat = Concatenate(axis = -1, name = 'concat')([decoder_outputs2,
attn_out_inference])

decoder_outputs2 = decoder_dense(decoder_inference_concat)

decoder_model = Model(
   [decoder_inputs] + [decoder_hidden_state_input, decoder_state_input_h,
decoder_state_input_c],
   [decoder_outputs2] + [state_h2, state_c2])

def decode_sequence(input_seq):
   e_out, e_h, e_c = encoder_model.predict(input_seq)

   target_seq = np.zeros((1, 1))
   target_seq[0, 0] = target_word_index['sumstart']

   stop_condition = False
   decoded_sentence = ''

   while not stop_condition:

      output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

      sampled_token_index = np.argmax(output_tokens[0, -1, :])
      sampled_token = reverse_target_word_index[sampled_token_index]

      if(sampled_token != 'sumend'):
         decoded_sentence += ' ' + sampled_token
```

```
        if (sampled_token == 'sumend' or len(decoded_sentence.split()) >= (max_summary_length -
1)):
            stop_condition = True

        target_seq = np.zeros((1, 1))
        target_seq[0, 0] = sampled_token_index

        e_h, e_c = h, c

    return decoded_sentence

def seq2summary(input_seq):
    newString = ''
    for i in input_seq:
        if((i != 0 and i != target_word_index['sumstart']) and i != target_word_index['sumend']):
            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(0, 75):
    print("Review:", seq2text(X_train[i]))
    print("Original summary:", seq2summary(y_train[i]))
    print("Predicted summary:", decode_sequence(X_train[i].reshape(1, max_review_length)))
    print("\n")
```