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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
%tensorflow_version 1.x
import numpy as np   #Package for scientific computing and dealing with arrays
import pandas as pd  #Package providing fast, flexible and expressive data structu
import re            #re stands for RegularExpression providing full support for Pe
from bs4 import BeautifulSoup   #Package for pulling data out of HTML and XML files
from keras.preprocessing.text import Tokenizer  #For tokenizing the input sequences
from keras.preprocessing.sequence import pad_sequences  #For Padding the seqences t
from nltk.corpus import stopwords   #For removing filler words
from tensorflow.keras.layers import Input, LSTM, Attention, Embedding, Dense, Conca
from tensorflow.keras.models import Model  #Helps in grouping the layers into an ob
from tensorflow.keras.callbacks import EarlyStopping  #Allows training the model or
import warnings  #shows warning message that may arise

pd.set_option("display.max_colwidth", 200) #Setting the data sructure display lengt
warnings.filterwarnings("ignore")
```

```
TensorFlow 1.x selected.
Using TensorFlow backend.
```

```
reviewsData=pd.read_csv("/content/drive/My Drive/Reviews.csv",nrows=30000) #Taking
print(reviewsData.shape) #Analyzing the shape of the dataset
reviewsData.head(n=10)
```

```
(3000, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help

```
DATASET_COLUMNS = ["Id", "ProductId", "UserId", "ProfileName", "HelpfulnessNumerat
reviewsData.columns = DATASET_COLUMNS
reviewsData.head(n=10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	He
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

```
reviewsData.drop(['Id', 'ProductId', 'UserId', 'ProfileName' , 'HelpfulnessNumerat
reviewsData.head(n=10)
```

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned dog food products and have found them all to be of good quality. The product looks more like a stew than a processed meat and it smells better. My Labr...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small sized unsalted. Not sure if this was an error or if the vendor intended to represent the product as "Jumbo".
2	"Delight" says it all	This is a confection that has been around a few centuries. It is a light, pillowy citrus gelatin with nuts - in this case Filberts. And it is cut into tiny squares and then liberally coated with ...
3	Cough Medicine	If you are looking for the secret ingredient in Robitussin I believe I have found it. I got this in addition to the Root Beer Extract I ordered (which was good) and made some cherry soda. The fl...

```
#Reducing the length of dataset for better training and performance
reviewsData.drop_duplicates(subset=['Text'],inplace=True) #Dropping the rows with I
reviewsData.dropna(axis=0,inplace=True) #Dropping the rows with Missing values


reviewsData.info() #Getting more info on datatypes and shape of Dataset
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2989 entries, 0 to 2999
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Summary  2989 non-null   object
 1   Text     2989 non-null   object
dtypes: object(2)
memory usage: 70.1+ KB
```

```
#Preprocessing

#This the dictionary used for expanding contractions
contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot", "

                           "didn't": "did not", "doesn't": "does not", "don't": "d

                           "he'd": "he would","he'll": "he will", "he's": "he is",

                           "I'd": "I would", "I'd've": "I would have", "I'll": "I \

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

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

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

                           "mustn't": "must not", "mustn't've": "must not have", "

                           "oughtn't": "ought not", "oughtn't've": "ought not have

                           "she'd": "she would", "she'd've": "she would have", "sh
```

```
                           "should've": "should have", "shouldn't": "should not",

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

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

                           "they'll": "they will", "they'll've": "they will have",

                           "wasn't": "was not", "we'd": "we would", "we'd've": "we

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

                           "what's": "what is", "what've": "what have", "when's":

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

                           "why's": "why is", "why've": "why have", "will've": "wi

                           "would've": "would have", "wouldn't": "would not", "wou

                           "y'all'd": "you all would","y'all'd've": "you all would

                           "you'd": "you would", "you'd've": "you would have", "yo

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


    #Text Cleaning
    import nltk
    nltk.download('stopwords')

    stop_words = set(stopwords.words('english'))
    def text_cleaner(text,num):
        newString = text.lower()  #converts all uppercase characters in the string int
        newString = BeautifulSoup(newString, "lxml").text #parses the string into an l
        newString = re.sub(r'\([^)]*\)', '', newString) #used to replace a string that
        newString = re.sub('"','', newString)
        newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else
        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
        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))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
```

```
reviewsData['Text'][:10] #Looking at the 'Text' column of the dataset
```

```
0    I have bought several of the Vitality canned dog food products and have
1              Product arrived labeled as Jumbo Salted Peanuts...the peanuts w
2    This is a confection that has been around a few centuries.  It is a ligh
3    If you are looking for the secret ingredient in Robitussin I believe I h
4                                                              Great taffy a
5    I got a wild hair for taffy and ordered this five pound bag. The taffy w
6    This saltwater taffy had great flavors and was very soft and chewy.  Eac
7                                                              This taffy is
8                                                                      Righ
9                                                                 This is a
Name: Text, dtype: object
```

```
cleaned_text[:10] #Looking at the Text after removing stop words, special characte
```

```
['bought several vitality canned dog food products found good quality product
 'product arrived labeled jumbo salted peanuts peanuts actually small sized u
 'confection around centuries light pillowy citrus gelatin nuts case filberts
 'looking secret ingredient robitussin believe found got addition root beer e
 'great taffy great price wide assortment yummy taffy delivery quick taffy lo
 'got wild hair taffy ordered five pound bag taffy enjoyable many flavors wat
 'saltwater taffy great flavors soft chewy candy individually wrapped well no
 'taffy good soft chewy flavors amazing would definitely recommend buying sat
 'right mostly sprouting cats eat grass love rotate around wheatgrass rye',
 'healthy dog food good digestion also good small puppies dog eats required a
```

```
#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))
```

```
reviewsData['Summary'][:10]
```

```
0                             Good Quality Dog Food
1                                 Not as Advertised
2                             "Delight" says it all
3                                    Cough Medicine
4                                       Great taffy
5                                        Nice Taffy
6    Great!  Just as good as the expensive brands!
7                            Wonderful, tasty taffy
8                                        Yay Barley
9                                 Healthy Dog Food
Name: Summary, dtype: object
```

```
cleaned_summary[:10]
```

```
['good quality dog food',
 'not as advertised',
```

```
        'delight says it all',
        'cough medicine',
        'great taffy',
        'nice taffy',
        'great just as good as the expensive brands',
        'wonderful tasty taffy',
        'yay barley',
        'healthy dog food']
```

```
reviewsData['Cleaned_Text'] = cleaned_text  #Adding cleaned text to the dataset
reviewsData['Cleaned_Summary'] = cleaned_summary  #Adding cleaned summary to the da
#Dropping Empty Rows
reviewsData['Cleaned_Summary'].replace('', np.nan, inplace=True)
#Dropping rows with Missing values
reviewsData.dropna(axis=0,inplace=True)
```

```
#Before Cleaning
print("Before Preprocessing:\n")
for i in range(5):
    print("Review:",reviewsData['Text'][i])
    print("Summary:",reviewsData['Summary'][i])
    print("\n")
```

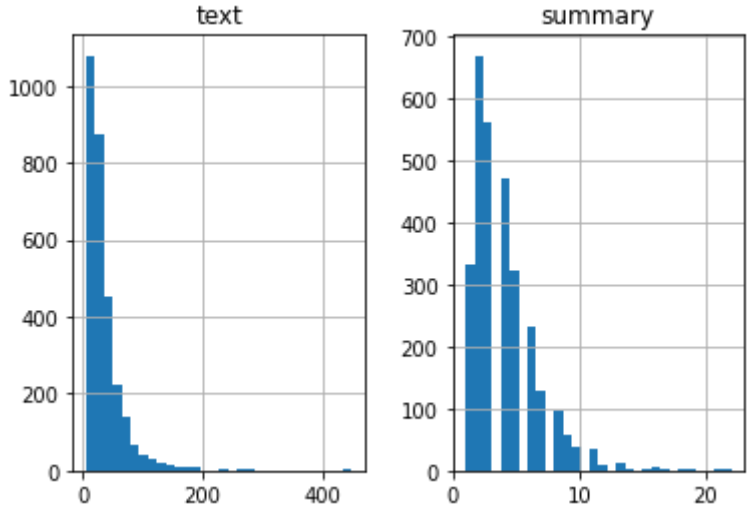
```
Before Preprocessing:

Review: I have bought several of the Vitality canned dog food products and ha
Summary: Good Quality Dog Food


Review: Product arrived labeled as Jumbo Salted Peanuts...the peanuts were ac
Summary: Not as Advertised


Review: This is a confection that has been around a few centuries.  It is a l
Summary: "Delight" says it all


Review: If you are looking for the secret ingredient in Robitussin I believe
Summary: Cough Medicine


Review: Great taffy at a great price.  There was a wide assortment of yummy t
Summary: Great taffy
```

```
#Printing the Cleaned text and summary which will work as input to the model
print("After Preprocessing:\n")
for i in range(5):
    print("Review:",reviewsData['Cleaned_Text'][i])
    print("Summary:",reviewsData['Cleaned_Summary'][i])
    print("\n")
```

```
After Preprocessing:
```

```
Review: bought several vitality canned dog food products found good quality p
Summary: good quality dog food


Review: product arrived labeled jumbo salted peanuts peanuts actually small s
Summary: not as advertised


Review: confection around centuries light pillowy citrus gelatin nuts case fi
Summary: delight says it all


Review: looking secret ingredient robitussin believe found got addition root
Summary: cough medicine


Review: great taffy great price wide assortment yummy taffy delivery quick ta
Summary: great taffy
```

```
#Data Visualization
import matplotlib.pyplot as plt
text_word_count = []
summary_word_count = []

#Populating the lists with sentence lengths
for i in reviewsData['Cleaned_Text']:
      text_word_count.append(len(i.split()))

for i in reviewsData['Cleaned_Summary']:
      summary_word_count.append(len(i.split()))

length_df = pd.DataFrame({'text':text_word_count, 'summary':summary_word_count})
length_df.hist(bins = 30)
plt.show()
```




```
#Function for getting the Maximum Review length
count=0
for i in reviewsData['Cleaned_Text']:
```

```
    if(len(i.split())<=35):
        count=count+1
print(count/len(reviewsData['Cleaned_Text']))
```

```
0.6687876758204957
```

```
#Function for getting the Maximum Summary length
count=0
for i in reviewsData['Cleaned_Summary']:
    if(len(i.split())<=8):
        count=count+1
print(count/len(reviewsData['Cleaned_Summary']))
```

```
0.942397856664434
```

```
#From the above data we got an idea about maximum lengths of review and summary
max_text_len = 35
max_summary_len = 8
```

```
#Adding START and END tags to summary for better decoding
cleaned_text =np.array(reviewsData['Cleaned_Text'])
cleaned_summary=np.array(reviewsData['Cleaned_Summary'])

short_text=[]
short_summary=[]

for i in range(len(cleaned_text)):
    if(len(cleaned_summary[i].split())<=max_summary_len and len(cleaned_text[i].sp
        short_text.append(cleaned_text[i])
        short_summary.append(cleaned_summary[i])

df=pd.DataFrame({'text':short_text,'summary':short_summary})
```

```
df['summary'] = df['summary'].apply(lambda x : 'sostok '+ x + ' eostok')
```

```
#Splitting the Dataset
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(np.array(df['text']),np.array(df['s
```

```
#Preparing Tokenizer

#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))
```

```
#Rarewords and their coverage in review
```

```
thresh = 4  #If a word whose count is less than threshold i.e 4, then it's conside

cnt = 0      #denotes no. of rare words whose count falls below threshold
tot_cnt = 0  #denotes size of unique words in the text
freq = 0
tot_freq = 0

for key,value in X_tokenizer.word_counts.items():
    tot_cnt=tot_cnt+1
    tot_freq=tot_freq+value
    if(value<thresh):
        cnt=cnt+1
        freq=freq+value

print("% of rare words in vocabulary:",(cnt/tot_cnt)*100)
print("Total Coverage of rare words:",(freq/tot_freq)*100)
```

```
% of rare words in vocabulary: 71.35983263598327
Total Coverage of rare words: 15.894039735099339
```

```
#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


X_voc
```

```
1370
```

```
#Summary Tokenizer

#Preparing a Tokenizer for summaries on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))


#Rarewords and their coverage in summary

thresh = 6  ##If a word whose count is less than threshold i.e 6, then it's consid

cnt = 0
tot_cnt = 0
```

```
freq = 0
tot_freq = 0

for key,value in y_tokenizer.word_counts.items():
    tot_cnt = tot_cnt+1
    tot_freq = tot_freq+value
    if(value<thresh):
        cnt = cnt+1
        freq = freq+value

print("% of rare words in vocabulary:",(cnt/tot_cnt)*100)
print("Total Coverage of rare words:",(freq/tot_freq)*100)
```

```
% of rare words in vocabulary: 88.28483920367535
Total Coverage of rare words: 22.488570369455086
```

```
#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


y_voc
```

```
154
```

```
#Checking the length of training data
y_tokenizer.word_counts['sostok'],len(y_train)
```

```
(1542, 1542)
```

```
#Deleting rows containing START and END tokens
#For Training set
ind=[]
for i in range(len(y_train)):
    cnt=0
    for j in y_train[i]:
        if j!=0:
            cnt=cnt+1
    if(cnt==2):
        ind.append(i)
```

```
y_train=np.delete(y_train,ind, axis=0)
X_train=np.delete(X_train,ind, axis=0)


#For Validation set
ind=[]
for i in range(len(y_test)):
    cnt=0
    for j in y_test[i]:
        if j!=0:
            cnt=cnt+1
    if(cnt==2):
        ind.append(i)

y_test=np.delete(y_test,ind, axis=0)
X_test=np.delete(X_test,ind, axis=0)


#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):
    """
    This class implements Bahdanau attention (https://arxiv.org/pdf/1409.0473.pdf)
    There are 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_s
                                   initializer='uniform',
                                   trainable=True)
        self.U_a = self.add_weight(name='U_a',
                                   shape=tf.TensorShape((input_shape[1][2], input_s
                                   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 

    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)

        def energy_step(inputs, states):
            """ Step function for computing energy for a single decoder state """

            assert_msg = "States must be a list. However states {} is of type {}".
            assert isinstance(states, list) or isinstance(states, tuple), assert_ms

            """ Some parameters required for shaping tensors"""
            en_seq_len, en_hidden = encoder_out_seq.shape[1], encoder_out_seq.shape
            de_hidden = inputs.shape[-1]

            """ Computing S.Wa where S=[s0, s1, ..., si]"""
            # <= batch_size*en_seq_len, latent_dim
            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_se
            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
            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_l
            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
```

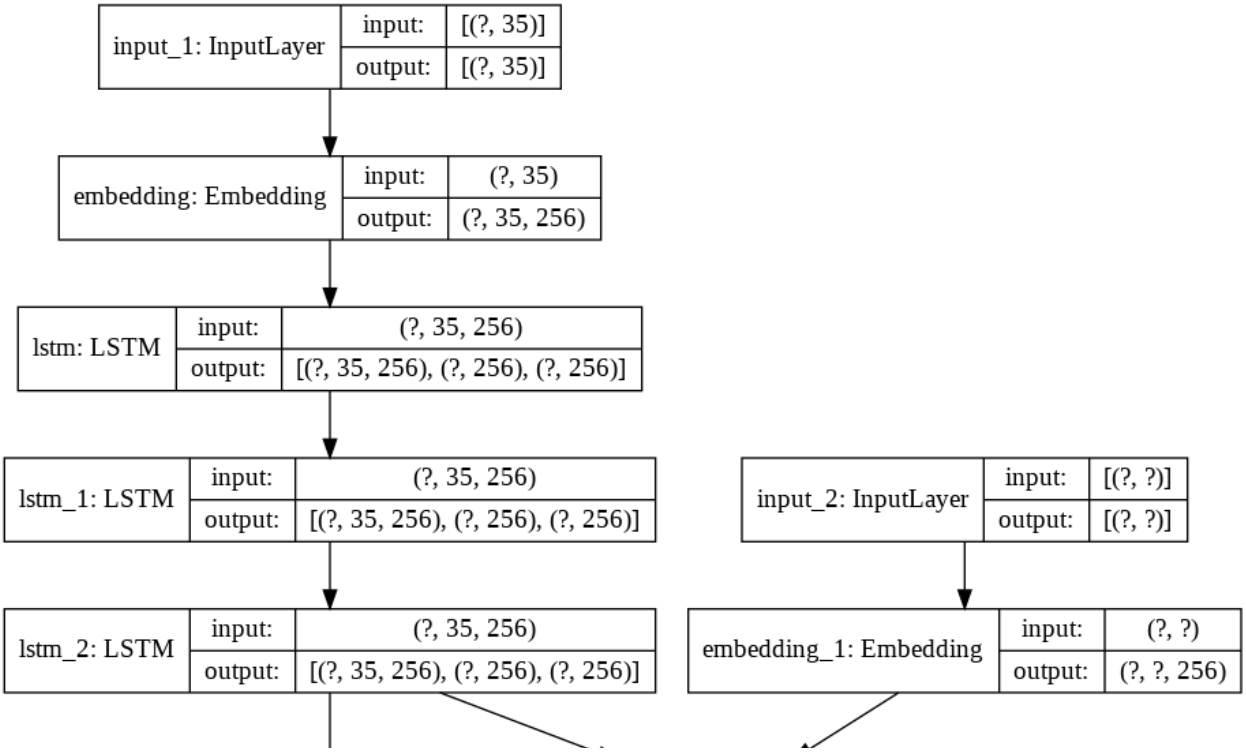
```
            c_i = K.sum(encoder_out_seq * K.expand_dims(inputs, -1), axis=1)
            if verbose:
                print('ci>', c_i.shape)
            return c_i, [c_i]

        def create_inital_state(inputs, hidden_size):
            # We are not using initial states, but need to pass something to K.rnn
            fake_state = K.zeros_like(inputs)  # <= (batch_size, enc_seq_len, later
            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, la
            return fake_state

        fake_state_c = create_inital_state(encoder_out_seq, encoder_out_seq.shape[
        fake_state_e = create_inital_state(encoder_out_seq, encoder_out_seq.shape[

        """ 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
        ]


from keras import backend as K
K.clear_session()  #Resets all state generated by Keras

latent_dim = 256
embedding_dim = 256

# Encoder
encoder_inputs = Input(shape=(max_text_len,))

#embedding layer
enc_emb =  Embedding(X_voc, embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
```

```
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3= LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

#Setting up the Decoder using 'encoder_states' as initial state
decoder_inputs = Input(shape=(None,))

#Embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initia

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

#Concating Attention input and Decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs,

#Dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

#Defining the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.summary()
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/k
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to th
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/o
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:Entity <bound method AttentionLayer.call of <__main__.Atte
WARNING: Entity <bound method AttentionLayer.call of <__main__.AttentionLayer
Model: "model"
__________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================
input_1 (InputLayer)            [(None, 35)]         0
__________________________________________________________________________
embedding (Embedding)           (None, 35, 256)      350720      input_1[0][0
__________________________________________________________________________
lstm (LSTM)                     [(None, 35, 256), (N 525312      embedding[0]
__________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________
lstm_1 (LSTM)                   [(None, 35, 256), (N 525312      lstm[0][0]
__________________________________________________________________________
embedding_1 (Embedding)         (None, None, 256)    39424       input_2[0][0
__________________________________________________________________________
```

```
lstm_2 (LSTM)                   [(None, 35, 256), (N 525312      lstm_1[0][0]
__________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 256),  525312      embedding_1[
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________
attention_layer (AttentionLayer ((None, None, 256),  131328      lstm_2[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________________
concat_layer (Concatenate)      (None, None, 512)    0           lstm_3[0][0]
                                                                 attention_la
__________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 154)    79002       concat_layer
==========================================================================================
Total params: 2,701,722
Trainable params: 2,701,722
Non-trainable params: 0
__________________________________________________________________________________________
```

```
#Visualize the Model
from tensorflow.keras.utils import plot_model

plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=Tru
```



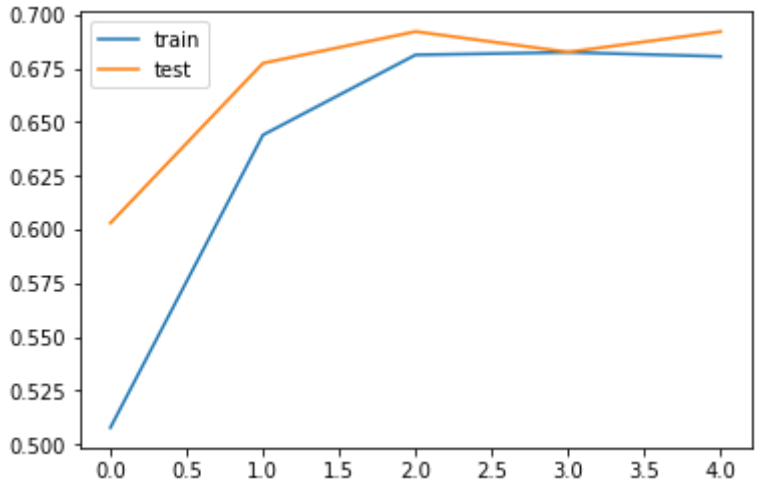

```
#Adding Metrics
model.compile(optimizer='rmsprop' , loss='sparse_categorical_crossentropy' , metri
```

```
#Adding Callback
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)
```

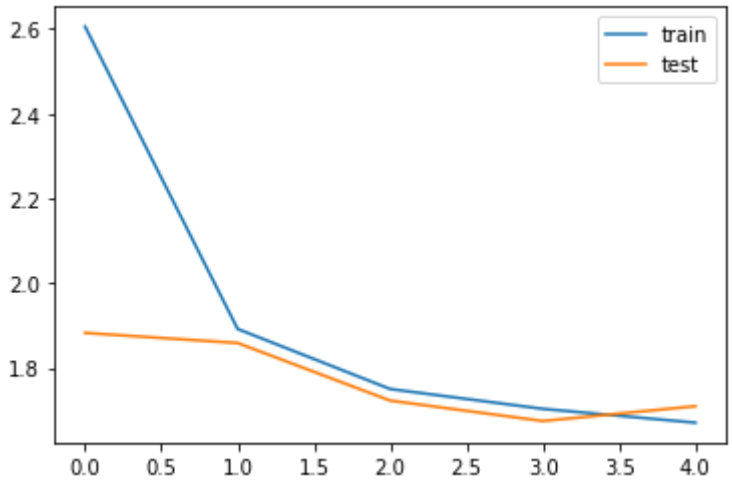
```
#Training the Model
%tensorflow_version 1.x
history = model.fit([X_train,y_train[:,:-1]], y_train.reshape(y_train.shape[0],y_t
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/o
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 1345 samples, validate on 330 samples
Epoch 1/50
1345/1345 [==============================] - 44s 33ms/sample - loss: 2.6055 -
Epoch 2/50
1345/1345 [==============================] - 40s 30ms/sample - loss: 1.8919 -
Epoch 3/50
1345/1345 [==============================] - 41s 30ms/sample - loss: 1.7501 -
Epoch 4/50
1345/1345 [==============================] - 41s 30ms/sample - loss: 1.7034 -
Epoch 5/50
1345/1345 [==============================] - 43s 32ms/sample - loss: 1.6708 -
Epoch 00005: early stopping
```

```
#Visualizing Accuracy
from matplotlib import pyplot
pyplot.plot(history.history['acc'], label='train')
pyplot.plot(history.history['val_acc'], label='test')
pyplot.legend()
pyplot.show()
```




```
#Visualizing Loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```




```
#Building Dictionary for 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


#Inference/Validation Phase
#Encoding the input sequence to get the feature vector
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, sta

#Decoder setup
#These 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_text_len,latent_dim))

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

#Setting the initial states to the states from the previous time step for better p
```

```
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decod

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

#Adding Dense softmax layer to generate proability distribution over the target vo
decoder_outputs2 = decoder_dense(decoder_inf_concat)

#Final Decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_
    [decoder_outputs2] + [state_h2, state_c2])
```

```
WARNING:tensorflow:Entity <bound method AttentionLayer.call of <__main__.Atte
WARNING: Entity <bound method AttentionLayer.call of <__main__.AttentionLayer
```

```
#Function defining the implementation of inference process
def decode_sequence(input_seq):
    #Encoding the input as state vectors
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    #Generating empty target sequence of length 1
    target_seq = np.zeros((1,1))

    #Populating the first word of target sequence with the start word
    target_seq[0, 0] = target_word_index['sostok']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

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

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

        if(sampled_token!='eostok'):
            decoded_sentence += ' '+sampled_token

        #Exit condition: either hit max length or find stop word
        if (sampled_token == 'eostok'  or len(decoded_sentence.split()) >= (max_su
            stop_condition = True

        #Updating the target sequence (of length 1)
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        #Updating internal states
        e_h, e_c = h, c

    return decoded_sentence
```

```
#Functions to convert an integer sequence to a word sequence for summary as well as
def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['sostok']) and i!=target_word_index['eost
            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


#Summaries generated by the model

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")
```

```
Original summary: yummy
Predicted summary:  great


Review: loved assortment pack everyone family loves low fat lower calorie na
Original summary: loves them
Predicted summary:  great


Review: disappointed received said large looks like something child bought l
Original summary: too small
Predicted summary:  great


Review: also baking ingredient put bread along seeds organic wheat flour als
Original summary: have for healthy
Predicted summary:


Review: favorite hot sauce use give anything extra flavor loved going favor
Original summary: great hot sauce
Predicted summary:  great


Review: actually saves basil extra week two thats enough reason buy basil we
Original summary: great
Predicted summary:


Review: use vita coco coconut water pineapple mix meal nutritional mixes we
Original summary: and refreshing
Predicted summary:  great
```

```
Review: every week make waffles pancakes wife daughter opinion waffles taste
Original summary: no better than
Predicted summary:  great


Review: super tasty quick meal bit small perfect lunch go definitely going b
Original summary: love it
Predicted summary:


Review: made vanilla cake mix ok bad flavor however disappointed vanilla fla
Original summary: have better
Predicted summary:  great


Review: like spicy like spicy enjoyed also crunchy kettle cooked chips
Original summary: spicy but good
Predicted summary:  great



Review: matter mess recipe mix still makes family happy sunday morning
Original summary: love it
Predicted summary:
```

```
#BLEU Score of Training set
#n-gram individual BLEU

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))

a=sentence_bleu(reference, candidate, weights=(1, 0, 0, 0))/3
b=sentence_bleu(reference, candidate, weights=(0, 1, 0, 0))/3
c=sentence_bleu(reference, candidate, weights=(0, 0, 1, 0))/3
d=sentence_bleu(reference, candidate, weights=(0, 0, 0, 1))/3


print('Individual 1-gram: %f' % a)
print('Individual 2-gram: %f' % b)
print('Individual 3-gram: %f' % c)
print('Individual 4-gram: %f' % d)
```

```
Individual 1-gram: 0.333333
Individual 2-gram: 0.333333
Individual 3-gram: 0.333333
Individual 4-gram: 0.333333
```

```
#4-gram cumulative BLEU
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))
```

```
score = sentence_bleu(reference, candidate, weights=(0.25, 0.25, 0.25, 0.25))
score=score/3
print(score)
```

```
0.3333333333333333
```

```
#cumulative BLEU scores
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))


a=sentence_bleu(reference, candidate, weights=(1, 0, 0, 0))/3
b=sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))/3
c=sentence_bleu(reference, candidate, weights=(0.33, 0.33, 0.33, 0))/3
d=sentence_bleu(reference, candidate, weights=(0.25, 0.25, 0.25, 0.25))/3


print('Cumulative 1-gram: %f' % a)
print('Cumulative 2-gram: %f' % b)
print('Cumulative 3-gram: %f' % c)
print('Cumulative 4-gram: %f' % d)
```

```
Cumulative 1-gram: 0.333333
Cumulative 2-gram: 0.333333
Cumulative 3-gram: 0.333333
Cumulative 4-gram: 0.333333
```

✓ 0s completed at 8:54 AM