Artificial Intelligence.

Project.

14/11/2022

Submitted by

2020BCS0019, Roshin Nishad

Implementation of Text Summarization using Python and Deep Learning.

Github - https://github.com/GetPsyched6/Text_Summarizer_Keras/tree/main

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.

Let us first understand what text summarization is before we look at how it works. Here is a succinct definition to get us started:

“Automatic text summarization is the task of producing a concise and fluent summary while preserving key information content and overall meaning”

There are broadly two different approaches that are used for text summarization:

- Extractive Summarization
- Abstractive Summarization

Extractive Summarization

The name gives away what this approach does. We identify the important sentences or phrases from the original text and extract only those from the text.

Abstractive Summarization

This is a very interesting approach. Here, we generate new sentences from the original text. This is in contrast to the extractive approach we saw earlier where we used only the sentences that were present.

Let us begin the implementation using Python and Keras

Code.

```
import tensorflow as tf
from tensorflow.python.keras import backend as K
import numpy as np
import pandas as pd
import re
from bs4 import BeautifulSoup
from tensorflow import keras
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import
pad_sequences
from nltk.corpus import stopwords
from keras.layers import Input, LSTM, Embedding,
Dense, Concatenate, TimeDistributed, Bidirectional
```

```
from keras.models import Model
from keras.callbacks import EarlyStopping
import warnings
import requests
from io import StringIO
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")




logger = tf.get_logger()

class AttentionLayer(tf.keras.layers.Layer):

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

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

        self.W_a = self.add_weight(name='W_a',

shape=tf.TensorShape((input_shape[0][2],
input_shape[0][2])),

initializer='uniform',
                                   trainable=True)
        self.U_a = self.add_weight(name='U_a',
```

```
shape=tf.TensorShape((input_shape[1][2],
input_shape[0][2])),

initializer='uniform',
                                   trainable=True)
        self.V_a = self.add_weight(name='V_a',

shape=tf.TensorShape((input_shape[0][2], 1)),

initializer='uniform',
                                   trainable=True)

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

    def call(self, inputs):

        assert type(inputs) == list
        encoder_out_seq, decoder_out_seq = inputs

        logger.debug(f"encoder_out_seq.shape =
{encoder_out_seq.shape}")
        logger.debug(f"decoder_out_seq.shape =
{decoder_out_seq.shape}")

        def energy_step(inputs, states):

            logger.debug("Running energy computation
step")

            if not isinstance(states, (list, tuple)):
```

```
                raise TypeError(f"States must be an
iterable. Got {states} of type {type(states)}")

            encoder_full_seq = states[-1]

            # <= batch size * en_seq_len * latent_dim
            W_a_dot_s = K.dot(encoder_full_seq,
self.W_a)

            U_a_dot_h = K.expand_dims(K.dot(inputs,
self.U_a), 1)  # <= batch_size, 1, latent_dim

            logger.debug(f"U_a_dot_h.shape =
{U_a_dot_h.shape}")

            # <= batch_size*en_seq_len, latent_dim
            Ws_plus_Uh = K.tanh(W_a_dot_s +
U_a_dot_h)

            logger.debug(f"Ws_plus_Uh.shape =
{Ws_plus_Uh.shape}")

            # <= batch_size, en_seq_len
            e_i = K.squeeze(K.dot(Ws_plus_Uh,
self.V_a), axis=-1)
            # <= batch_size, en_seq_len
            e_i = K.softmax(e_i)

            logger.debug(f"ei.shape = {e_i.shape}")

            return e_i, [e_i]

        def context_step(inputs, states):
```

```
            logger.debug("Running attention vector
computation step")

            if not isinstance(states, (list, tuple)):
                raise TypeError(f"States must be an
iterable. Got {states} of type {type(states)}")

            encoder_full_seq = states[-1]

            # <= batch_size, hidden_size
            c_i = K.sum(encoder_full_seq *
K.expand_dims(inputs, -1), axis=1)

            logger.debug(f"ci.shape = {c_i.shape}")

            return c_i, [c_i]

        # we don't maintain states between steps when
computing attention
        # attention is stateless, so we're passing a
fake state for RNN step function
        fake_state_c = K.sum(encoder_out_seq, axis=1)
        fake_state_e = K.sum(encoder_out_seq, axis=2)
# <= (batch_size, enc_seq_len, latent_dim

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

        """ Computing context vectors """
```

```
        last_out, c_outputs, _ = K.rnn(
            context_step, e_outputs, [fake_state_c],
constants=[encoder_out_seq]
        )

        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]))
        ]
```

```
url='https://drive.google.com/file/d/1BPU8RsBnZmoMq1I
_WqMnBx018u7czhOB/view?usp=sharing'
url='https://drive.google.com/uc?id=' +
url.split('/')[-2]
data=pd.read_csv(url,nrows=10000)
data.drop_duplicates(subset=['Text'],inplace=True)
#dropping duplicates
data.dropna(axis=0,inplace=True)   #dropping na
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
"we'll": "we will", "we'll've": "we will have",
"we're": "we are",

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

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

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

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

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

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

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

```
data['Text'][:10]
```

```
0    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              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    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    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...
4                                                                Great taffy at a great price.  There was a wide assortment of yummy taffy.  Delivery was very quick.  If your a taffy lover, this is a deal.
5    I got a wild hair for taffy and ordered this five pound bag. The taffy was all very enjoyable with many flavors: watermelon, root beer, melon, peppermint, grape, etc. My only complaint is there wa...
6    This saltwater taffy had great flavors and was very soft and chewy.  Each candy was individually wrapped well.  None of the candies were stuck together, which did happen in the expensive version, ...
7                                                                This taffy is so good.  It is very soft and chewy.  The flavors are amazing.  I would definitely recommend you buying it.  Very satisfying!!
8                                                                         Right now I'm mostly just sprouting this so my cats can eat the grass. They love it. I rotate it around with Wheatgrass and Rye too
9                                                                   This is a very healthy dog food. Good for their digestion. Also good for small puppies. My dog eats her required amount at every feeding.
Name: Text, dtype: object
```

```
stop_words = set(stopwords.words('english'))
def text_cleaner(text):
    newString = text.lower()
    newString = BeautifulSoup(newString,
"html.parser").text
    newString = re.sub(r'\([^)]*\)', '', newString)
    newString = re.sub('"','', newString)
    newString = ' '.join([contraction_mapping[t] if t
in contraction_mapping else t for t in
newString.split(" ")])
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    tokens = [w for w in newString.split() if not w
in stop_words]
    long_words=[]
    for i in tokens:
        if len(i)>=3:
```

```
#removing short word
            long_words.append(i)
    return (" ".join(long_words)).strip()

cleaned_text = []
for t in data['Text']:
    cleaned_text.append(text_cleaner(t))
```

```
data['Summary'][:10]
```

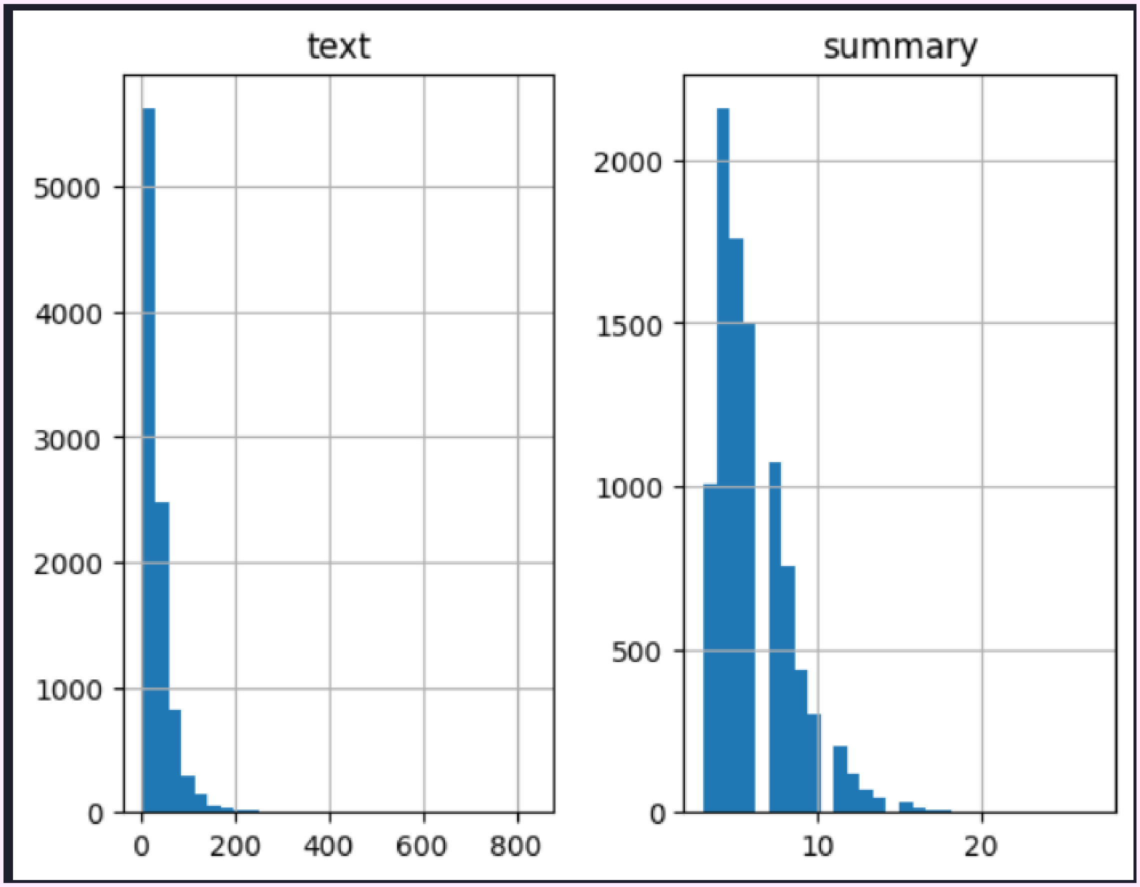
```
0                          Good Quality Dog Food
1                              Not as Advertised
2                          "Delight" says it all
3                                 Cough Medicine
4                                    Great taffy
5                                     Nice Taffy
6 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

```
def summary_cleaner(text):
    newString = re.sub('"','', text)
    newString = ' '.join([contraction_mapping[t] if t
in contraction_mapping else t for t in
newString.split(" ")])
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    newString = newString.lower()
```

```
    tokens=newString.split()
    newString=''
    for i in tokens:
        if len(i)>1:
            newString=newString+i+' '
    return newString


#Call the above function
cleaned_summary = []
for t in data['Summary']:
    cleaned_summary.append(summary_cleaner(t))


data['cleaned_text']=cleaned_text
data['cleaned_summary']=cleaned_summary
data['cleaned_summary'].replace('', np.nan,
inplace=True)
data.dropna(axis=0,inplace=True)


data['cleaned_summary'] =
data['cleaned_summary'].apply(lambda x : '_START_ '+
x + ' _END_')



for i in range(5):
    print("Review:",data['cleaned_text'][i])
    print("Summary:",data['cleaned_summary'][i])
    print("\n")
```

```
Review: bought several vitality canned dog food products found good quality product looks like stew processed meat smells better
labrador finicky appreciates product better
Summary: _START_ good quality dog food  _END_


Review: product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor intended represent
product jumbo
Summary: _START_ not as advertised  _END_


Review: confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares liberally coated powdered sugar
tiny mouthful heaven chewy flavorful highly recommend yummy treat familiar story lewis lion witch wardrobe treat seduces edmund
selling brother sisters witch
Summary: _START_ delight says it all  _END_


Review: looking secret ingredient robitussin believe found got addition root beer extract ordered made cherry soda flavor medicinal
Summary: _START_ cough medicine  _END_


Review: great taffy great price wide assortment yummy taffy delivery quick taffy lover deal
Summary: _START_ great taffy  _END_
```

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

# populate the lists with sentence lengths
for i in data['cleaned_text']:
      text_word_count.append(len(i.split()))

for i in data['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()
```




```
max_len_text=80
max_len_summary=15
```

```
from sklearn.model_selection import train_test_split
x_tr,x_val,y_tr,y_val=train_test_split(data['cleaned_text'],data['cleaned_summary'],test_size=0.1,random_state=0,shuffle=True)
```

```
#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer()
```

```
x_tokenizer.fit_on_texts(list(x_tr))

#convert text sequences into integer sequences
x_tr    =   x_tokenizer.texts_to_sequences(x_tr)
x_val   =   x_tokenizer.texts_to_sequences(x_val)

#padding zero upto maximum length
x_tr    =   pad_sequences(x_tr,  maxlen=max_len_text,
padding='post')
x_val   =   pad_sequences(x_val, maxlen=max_len_text,
padding='post')

x_voc_size   =  len(x_tokenizer.word_index) +1
```

```
#preparing a tokenizer for summary on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_tr))

#convert summary sequences into integer sequences
y_tr    =   y_tokenizer.texts_to_sequences(y_tr)
y_val   =   y_tokenizer.texts_to_sequences(y_val)

#padding zero upto maximum length
y_tr    =   pad_sequences(y_tr,
maxlen=max_len_summary, padding='post')
y_val   =   pad_sequences(y_val,
maxlen=max_len_summary, padding='post')

y_voc_size  =   len(y_tokenizer.word_index) +1
```

```
K.clear_session()

latent_dim = 300
embedding_dim=100

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

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

#encoder lstm 1
encoder_lstm1 =
LSTM(latent_dim,return_sequences=True,return_state=Tr
ue,dropout=0.4,recurrent_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=Tr
ue,dropout=0.4,recurrent_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=0.4,recurrent_dropout=0
.4)
encoder_outputs, state_h, state_c=
encoder_lstm3(encoder_output2)
```

```
# Set up the decoder, using `encoder_states` as
initial state.
decoder_inputs = Input(shape=(None,))


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

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


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


# Concat attention input 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()
```

```
Output exceeds the size limit. Open the full output data in a text editor
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 80)]         0           []

 embedding (Embedding)          (None, 80, 100)      1691000     ['input_1[0][0]']

 lstm (LSTM)                    [(None, 80, 300),    481200      ['embedding[0][0]']
                                 (None, 300),
                                 (None, 300)]

 input_2 (InputLayer)           [(None, None)]       0           []

 lstm_1 (LSTM)                  [(None, 80, 300),    721200      ['lstm[0][0]']
                                 (None, 300),
                                 (None, 300)]

 embedding_1 (Embedding)        (None, None, 100)    425000      ['input_2[0][0]']

 lstm_2 (LSTM)                  [(None, 80, 300),    721200      ['lstm_1[0][0]']
                                 (None, 300),
                                 (None, 300)]
```

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

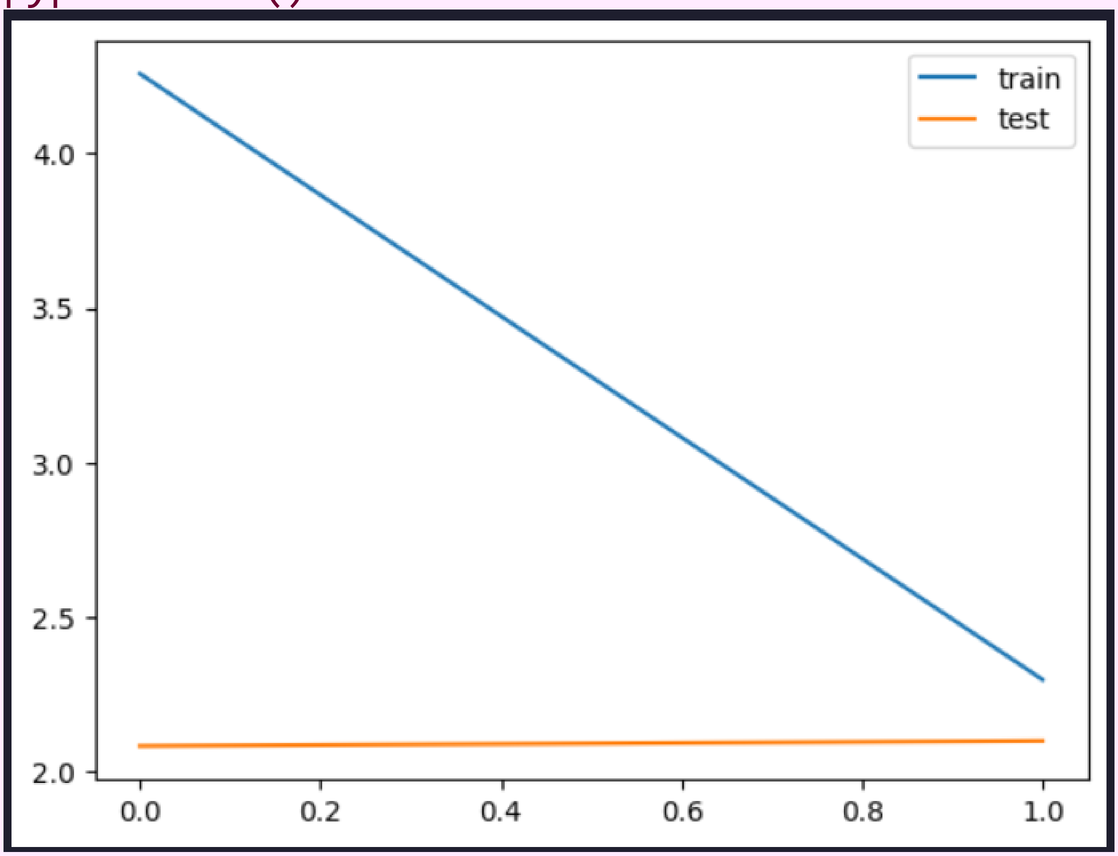
```
es = EarlyStopping(monitor='val_loss', mode='min',
verbose=1,patience=2)
```

```
history=model.fit([x_tr,y_tr[:,:-1]],
y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)[:,1:]
```

```
,epochs=2,callbacks=[es],batch_size=1000,
validation_data=([x_val,y_val[:,:-1]],
y_val.reshape(y_val.shape[0],y_val.shape[1],
1)[:,1:]))
```

```
Epoch 1/2
9/9 [==============================] - 179s 19s/step - loss: 4.2573 - val_loss: 2.0835
Epoch 2/2
9/9 [==============================] - 176s 20s/step - loss: 2.2982 - val_loss: 2.1002
```

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



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

```
# Encode the input sequence to get the feature vector
encoder_model =
Model(inputs=encoder_inputs,outputs=[encoder_outputs,
state_h, state_c])

# Decoder setup
# 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])

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))

    # Populate the first word of target sequence with
the start word.
    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



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(30,100):
    print("Original Human-made
Review:",seq2text(x_tr[i]))
    print("-------------Summary Below-------------")
    print("Predicted summary:",seq2summary(y_tr[i]))
    print("\n\n")
```

```
Output exceeds the size limit. Open the full output data in a text editor
Original Human-made Review: hot chocolate tastes like hot sugar water pinch coco although even sugary taste weak tainted flavor artificial sweetener sucralose consistency watery bland made cup directed directions box recommend product
-------------Summary Below-------------
Predicted summary: tastes like hot sugar water



Original Human-made Review: really looking forward dulce leche cheerios honey nut flavor favorite since came thought caramel quite appealing first spoonfuls seemed quite yummy ate less enjoyed caramel flavor seemed bit maybe corn oat flavor different well things considered stick honey nut tasted pretty good plain box think save want slightly sweet snack sugary pour handful eat like toddler expected sugary honey nut grams sugar per serving compared grams honey nut total carbohydrates grams per serving much difference affects body bottom line buy
-------------Summary Below-------------
Predicted summary: flavor did not work for me back to honey nut



Original Human-made Review: picked box discount grocery store paso robles cake heavenly liked much found another store sold mix well icing even pink lemonade cookie mix bought tried yet taste course lemon reminds bit flavored mix country time drank kid less tangy light refreshing simply loved hope seasonal flavor
-------------Summary Below-------------
Predicted summary: tasty
```

#--End--

End

2020BCS0019

Roshin Nishad.