Text Summerizer with Deep Learning

- @Nasir Uddin
- 20 Jan 2020

In this notebook, we will build an abstractive based text summarizer using deep learning from the scratch in python using keras

Please read throug here (https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/) to cover all the concepts which is required to build our own summarizer

Understanding the Problem Statement

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.

We will be working on a really cool dataset. Our objective here is to generate a summary for the Amazon Fine Food reviews using the abstraction-based approach we learned about above. You can download the dataset from here (https://www.kaggle.com/snap/amazon-fine-food-reviews)

Custom Attention Layer

Keras does not officially support attention layer. So, we can either implement our own attention layer or use a third-party implementation. We will go with the latter option for this article. You can download the attention layer from here (https://github.com/thushv89/attention_keras/blob/master/layers/attention.py) and copy it in a different file called attention.py.

Let’s import it into our environment:

In [1]:
```
from attention import AttentionLayer
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWar
ning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/anaconda3/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWar
ning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/anaconda3/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWar
ning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/anaconda3/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWar
ning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/anaconda3/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: FutureWar
ning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/anaconda3/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWar
ning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

Import the Libraries

In [2]:
```
import numpy as np
import pandas as pd
import re
from bs4 import BeautifulSoup
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistributed
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Embedding
import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
```

```
Using TensorFlow backend.
```

Read the 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. It also includes reviews from all other Amazon categories.

We'll take a sample of 100,000 reviews to reduce the training time of our model. Feel free to use the entire dataset for training your model if your machine has that kind of computational power.

In [3]:
```
data=pd.read_csv("./Reviews.csv",nrows=100000)
```

Drop Duplicates and NA values

In [4]:
```
data.drop_duplicates(subset=['Text'],inplace=True)#dropping duplicates
data.dropna(axis=0,inplace=True)#dropping na
```

Information about dataset

Let us look at datatypes and shape of the dataset

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88421 entries, 0 to 99999
Data columns (total 10 columns):
Id                        88421 non-null int64
ProductId                 88421 non-null object
UserId                    88421 non-null object
ProfileName               88421 non-null object
HelpfulnessNumerator      88421 non-null int64
HelpfulnessDenominator    88421 non-null int64
Score                     88421 non-null int64
Time                      88421 non-null int64
Summary                   88421 non-null object
Text                      88421 non-null object
dtypes: int64(5), object(5)
memory usage: 7.4+ MB
```

Preprocessing

Performing basic preprocessing steps is very important before we get to the model building part. Using messy and uncleaned text data is a potentially disastrous move. So in this step, we will drop all the unwanted symbols, characters, etc. from the text that do not affect the objective of our problem.

Here is the dictionary that we will use for expanding the contractions:

In [6]:
```
contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because", "coul
d've": "could have", "couldn't": "could not",
                           "didn't": "did not",  "doesn't": "does not", "don't": "do not", "hadn't": "had no
t", "hasn't": "has not", "haven't": "have not",
                           "he'd": "he would","he'll": "he will", "he's": "he is", "how'd": "how did", "ho
w'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 h
ave","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 a
m", "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'v
e": "might not have", "must've": "must have",
                           "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "need
n'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'l
l'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", "tha
t's": "that is", "there'd": "there would",
                           "there'd've": "there would have", "there's": "there is", "here's": "here is","the
y'd": "they would", "they'd've": "they would have",
                           "they'll": "they will", "they'll've": "they will have", "they're": "they are", "th
ey'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", "wh
o's": "who is", "who've": "who have",
                           "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will no
t", "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'l
l've": "you will have",
                           "you're": "you are", "you've": "you have"}
```

We will perform the below preprocessing tasks for our data:

1.Convert everything to lowercase

2.Remove HTML tags

3.Contraction mapping

4.Remove (‘s)

5.Remove any text inside the parenthesis ( )

6.Eliminate punctuations and special characters

7.Remove stopwords

8.Remove short words

Let’s define the function:

In [7]:
```
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def text_cleaner(text,num):
    newString = text.lower()
    newString = BeautifulSoup(newString, "lxml").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)
    newString = re.sub('[m]{2,}', 'mm', newString)
    if(num==0):
        tokens = [w for w in newString.split() if not w in stop_words]
    else:
        tokens=newString.split()
    long_words=[]
    for i in tokens:
        if len(i)>1:                                                 #removing short word
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

```
[nltk_data] Downloading package stopwords to /Users/nasir/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [8]:
```
#call the function
cleaned_text = []
for t in data['Text']:
    cleaned_text.append(text_cleaner(t,0))
```

Let us look at the first five preprocessed reviews

In [9]:
```
cleaned_text[:5]
```

Out[9]:
```
['bought several vitality canned dog food products found good quality product looks like stew processed meat
smells better labrador finicky appreciates product better',
 'product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor inten
ded represent product jumbo',
 'confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares liberally coa
ted powdered sugar tiny mouthful heaven chewy flavorful highly recommend yummy treat familiar story lewis li
on witch wardrobe treat seduces edmund selling brother sisters witch',
 'looking secret ingredient robitussin believe found got addition root beer extract ordered made cherry soda
flavor medicinal',
 'great taffy great price wide assortment yummy taffy delivery quick taffy lover deal']
```

In [10]:
```
#call the function
cleaned_summary = []
for t in data['Summary']:
    cleaned_summary.append(text_cleaner(t,1))
```

Let us look at the first 10 preprocessed summaries

In [11]:
```
cleaned_summary[:10]
```

Out[11]:
```
['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']
```

In [12]:
```
data['cleaned_text']=cleaned_text
data['cleaned_summary']=cleaned_summary
```

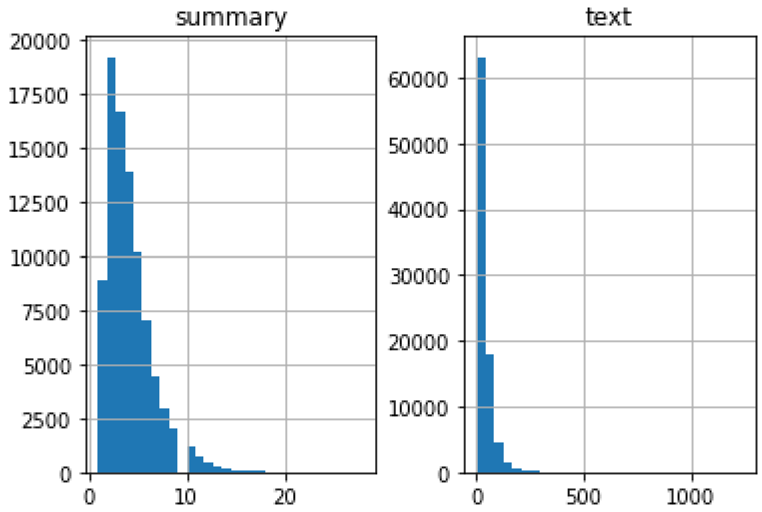
Drop empty rows

In [13]:
```
data.replace('', np.nan, inplace=True)
data.dropna(axis=0,inplace=True)
```

Understanding the distribution of the sequences

Here, we will analyze the length of the reviews and the summary to get an overall idea about the distribution of length of the text. This will help us fix the maximum length of the sequence:

In [14]:
```
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()
```



Interesting. We can fix the maximum length of the summary to 8 since that seems to be the majority summary length.

Let us understand the proportion of the length of summaries below 8

In [15]:
```
cnt=0
for i in data['cleaned_summary']:
    if(len(i.split())<=8):
        cnt=cnt+1
print(cnt/len(data['cleaned_summary']))
```

```
0.9424907471335922
```

We observe that 94% of the summaries have length below 8. So, we can fix maximum length of summary to 8.

Let us fix the maximum length of review to 30

In [16]:
```
max_text_len=30
max_summary_len=8
```

Let us select the reviews and summaries whose length falls below or equal to **max_text_len** and **max_summary_len**

In [17]:
```
cleaned_text =np.array(data['cleaned_text'])
cleaned_summary=np.array(data['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].split())<=max_text_len):
        short_text.append(cleaned_text[i])
        short_summary.append(cleaned_summary[i])

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

Remember to add the **START** and **END** special tokens at the beginning and end of the summary. Here, I have chosen **sostok** and **eostok** as START and END tokens

Note: Be sure that the chosen special tokens never appear in the summary

In [18]:
```
df['summary'] = df['summary'].apply(lambda x : 'sostok '+ x + ' eostok')
```

We are getting closer to the model building part. Before that, we need to split our dataset into a training and validation set. We'll use 90% of the dataset as the training data and evaluate the performance on the remaining 10% (holdout set):

In [19]:
```
from sklearn.model_selection import train_test_split
x_tr,x_val,y_tr,y_val=train_test_split(np.array(df['text']),np.array(df['summary']),test_size=0.1,random_stat
e=0,shuffle=True)
```

Preparing the Tokenizer

A tokenizer builds the vocabulary and converts a word sequence to an integer sequence. Go ahead and build tokenizers for text and summary:

Text Tokenizer

In [20]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_tr))
```

Rarewords and its Coverage

Let us look at the proportion rare words and its total coverage in the entire text

Here, I am defining the threshold to be 4 which means word whose count is below 4 is considered as a rare word

In [21]:
```
thresh=4

cnt=0
tot_cnt=0
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: 66.12339930151339
Total Coverage of rare words: 2.953684513790566
```

Remember:

- **tot_cnt** gives the size of vocabulary (which means every unique words in the text)
- **cnt** gives me the no. of rare words whose count falls below threshold
- **tot_cnt - cnt** gives me the top most common words

Let us define the tokenizer with top most common words for reviews.

In [22]:
```
#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
x_tokenizer.fit_on_texts(list(x_tr))

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

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

#size of vocabulary ( +1 for padding token)
x_voc   =  x_tokenizer.num_words + 1
```

In [23]:
```
x_voc
```

Out[23]: 8440

Summary Tokenizer

In [24]:
```
#prepare a tokenizer for reviews on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_tr))
```

Rarewords and its Coverage

Let us look at the proportion rare words and its total coverage in the entire summary

Here, I am defining the threshold to be 6 which means word whose count is below 6 is considered as a rare word

In [25]:
```
thresh=6

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: 78.12740675541863
Total Coverage of rare words: 5.3921899389571895
```

Let us define the tokenizer with top most common words for summary.

In [26]:
```
#prepare a tokenizer for reviews on training data
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
y_tokenizer.fit_on_texts(list(y_tr))

#convert text sequences into integer sequences
y_tr_seq    =   y_tokenizer.texts_to_sequences(y_tr)
y_val_seq   =   y_tokenizer.texts_to_sequences(y_val)

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

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

Let us check whether word count of start token is equal to length of the training data

In [27]:
```
y_tokenizer.word_counts['sostok'],len(y_tr)
```

Out[27]:
```
(42453, 42453)
```

Here, I am deleting the rows that contain only **START** and **END** tokens

In [28]:
```
ind=[]
for i in range(len(y_tr)):
    cnt=0
    for j in y_tr[i]:
        if j!=0:
            cnt=cnt+1
    if(cnt==2):
        ind.append(i)

y_tr=np.delete(y_tr,ind, axis=0)
x_tr=np.delete(x_tr,ind, axis=0)
```

In [29]:
```
ind=[]
for i in range(len(y_val)):
    cnt=0
    for j in y_val[i]:
        if j!=0:
            cnt=cnt+1
    if(cnt==2):
        ind.append(i)

y_val=np.delete(y_val,ind, axis=0)
x_val=np.delete(x_val,ind, axis=0)
```

Model building

We are finally at the model building part. But before we do that, we need to familiarize ourselves with a few terms which are required prior to building the model.

Return Sequences = True: When the return sequences parameter is set to True, LSTM produces the hidden state and cell state for every timestep

Return State = True: When return state = True, LSTM produces the hidden state and cell state of the last timestep only

Initial State: This is used to initialize the internal states of the LSTM for the first timestep

Stacked LSTM: Stacked LSTM has multiple layers of LSTM stacked on top of each other. This leads to a better representation of the sequence. I encourage you to experiment with the multiple layers of the LSTM stacked on top of each other (it's a great way to learn this)

Here, we are building a 3 stacked LSTM for the encoder:

In [31]:
```
from keras import backend as K



#from keras import Embedding
K.clear_session()

latent_dim = 300
embedding_dim=100

# 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,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=True,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, 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, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

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

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to                     
==================================================================================================
input_1 (InputLayer)            [(None, 30)]         0                                            
__________________________________________________________________________________________________
embedding (Embedding)           (None, 30, 100)      844000      input_1[0][0]                    
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 30, 300), (N 481200      embedding[0][0]                  
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0                                            
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 30, 300), (N 721200      lstm[0][0]                       
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 100)    198900      input_2[0][0]                    
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 30, 300), (N 721200      lstm_1[0][0]                     
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 300),  481200      embedding_1[0][0]                
                                                                 lstm_2[0][1]                     
                                                                 lstm_2[0][2]                     
__________________________________________________________________________________________________
attention_layer (AttentionLayer ((None, None, 300),  180300      lstm_2[0][0]                     
                                                                 lstm_3[0][0]                     
__________________________________________________________________________________________________
concat_layer (Concatenate)      (None, None, 600)    0           lstm_3[0][0]                     
                                                                 attention_layer[0][0]            
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 1989)   1195389     concat_layer[0][0]               
==================================================================================================
Total params: 4,823,389
Trainable params: 4,823,389
Non-trainable params: 0
__________________________________________________________________________________________________
```

I am using sparse categorical cross-entropy as the loss function since it converts the integer sequence to a one-hot vector on the fly. This overcomes any memory issues.

In [32]:
```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')
```

Remember the concept of early stopping? It is used to stop training the neural network at the right time by monitoring a user-specified metric. Here, I am monitoring the validation loss (val_loss). Our model will stop training once the validation loss increases:

In [33]:
```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,patience=2)
```

We'll train the model on a batch size of 128 and validate it on the holdout set (which is 10% of our dataset):

In [34]:
```
history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)[:,1:] ,epochs=50,callbacks
=[es],batch_size=128, validation_data=([x_val,y_val[:,:-1]], y_val.reshape(y_val.shape[0],y_val.shape[1], 1)
[:,1:]))
```

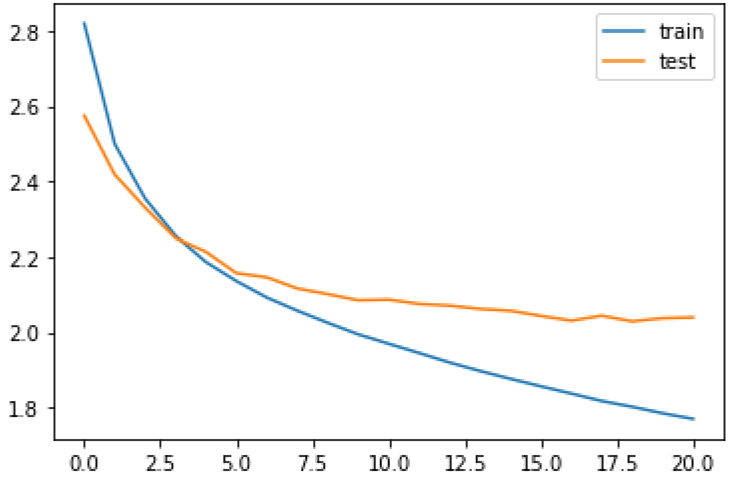
```
Train on 41346 samples, validate on 4588 samples
Epoch 1/50
41346/41346 [==============================] - 179s 4ms/sample - loss: 2.8191 - val_loss: 2.5743
Epoch 2/50
41346/41346 [==============================] - 168s 4ms/sample - loss: 2.4993 - val_loss: 2.4178
Epoch 3/50
41346/41346 [==============================] - 170s 4ms/sample - loss: 2.3543 - val_loss: 2.3301
Epoch 4/50
41346/41346 [==============================] - 169s 4ms/sample - loss: 2.2557 - val_loss: 2.2495
Epoch 5/50
41346/41346 [==============================] - 167s 4ms/sample - loss: 2.1866 - val_loss: 2.2137
Epoch 6/50
41346/41346 [==============================] - 167s 4ms/sample - loss: 2.1362 - val_loss: 2.1572
Epoch 7/50
41346/41346 [==============================] - 177s 4ms/sample - loss: 2.0925 - val_loss: 2.1462
Epoch 8/50
41346/41346 [==============================] - 178s 4ms/sample - loss: 2.0577 - val_loss: 2.1166
Epoch 9/50
41346/41346 [==============================] - 175s 4ms/sample - loss: 2.0255 - val_loss: 2.1015
Epoch 10/50
41346/41346 [==============================] - 172s 4ms/sample - loss: 1.9948 - val_loss: 2.0853
Epoch 11/50
41346/41346 [==============================] - 179s 4ms/sample - loss: 1.9694 - val_loss: 2.0868
Epoch 12/50
41346/41346 [==============================] - 172s 4ms/sample - loss: 1.9449 - val_loss: 2.0755
Epoch 13/50
41346/41346 [==============================] - 176s 4ms/sample - loss: 1.9197 - val_loss: 2.0709
Epoch 14/50
41346/41346 [==============================] - 178s 4ms/sample - loss: 1.8973 - val_loss: 2.0622
Epoch 15/50
41346/41346 [==============================] - 173s 4ms/sample - loss: 1.8765 - val_loss: 2.0572
Epoch 16/50
41346/41346 [==============================] - 177s 4ms/sample - loss: 1.8566 - val_loss: 2.0436
Epoch 17/50
41346/41346 [==============================] - 175s 4ms/sample - loss: 1.8374 - val_loss: 2.0311
Epoch 18/50
41346/41346 [==============================] - 173s 4ms/sample - loss: 1.8176 - val_loss: 2.0442
Epoch 19/50
41346/41346 [==============================] - 176s 4ms/sample - loss: 1.8023 - val_loss: 2.0292
Epoch 20/50
41346/41346 [==============================] - 172s 4ms/sample - loss: 1.7850 - val_loss: 2.0376
```

```
Epoch 21/50
41346/41346 [==============================] - 173s 4ms/sample - loss: 1.7701 - val_loss: 2.0396
Epoch 00021: early stopping
```

Understanding the Diagnostic plot

Now, we will plot a few diagnostic plots to understand the behavior of the model over time:

In [35]:
```
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



From the plot, we can infer that validation loss has increased after epoch 17 for 2 successive epochs. Hence, training is stopped at epoch 19.

Next, let’s build the dictionary to convert the index to word for target and source vocabulary:

In [36]:
```
reverse_target_word_index=y_tokenizer.index_word
reverse_source_word_index=x_tokenizer.index_word
target_word_index=y_tokenizer.word_index
```

Inference

Set up the inference for the encoder and decoder:

In [37]:
```
# 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_text_len,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_s
tate_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])
```

We are defining a function below which is the implementation of the inference process (which we covered [here (https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/)](https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/)):

In [38]:
```
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['sostok']

    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!='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_summary_len-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
```

Let us define the functions to convert an integer sequence to a word sequence for summary as well as the reviews:

In [39]:
```
def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['sostok']) and i!=target_word_index['eostok']):
            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
```

Here are a few summaries generated by the model:

In [40]:
```
for i in range(0,100):
    print("Review:",seq2text(x_tr[i]))
    print("Original summary:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,max_text_len)))
    print("\n")
```

```
Review: gave caffeine shakes heart anxiety attack plus tastes unbelievably bad stick coffee tea soda thanks
Original summary: hour
Predicted summary:  good product


Review: got great course good belgian chocolates better
Original summary: would like to give it stars but
Predicted summary:  great


Review: one best flavored coffees tried usually like flavored coffees one great serve company love
Original summary: delicious
Predicted summary:  great coffee


Review: salt separate area pain makes hard regulate salt putting like salt go ahead get product
Original summary: tastes ok packaging
Predicted summary:  salt


Review: really like product super easy order online delivered much cheaper buying gas station stocking good
long drives
Original summary: turkey jerky is great
Predicted summary:  great product


Review: best salad dressing delivered promptly quantities last vidalia onion dressing compares made oak hill
farms sometimes find costco order front door want even orders cut shipping costs
Original summary: my favorite salad dressing
Predicted summary:  best dressing ever


Review: think sitting around warehouse long time took long time send got tea tasted like cardboard red rasbe
rry leaf tea know supposed taste like
Original summary: stale
Predicted summary:  not for me


Review: year old cat special diet digestive problems also diabetes stopped eating usual special formula food
tried different kinds catfood one liked easy digestion diabetes thank newman
Original summary: wonderful
```

```
Predicted summary:  great cat food


Review: always perfect snack dog loves knows exactly starts ask time evening gets greenie snack thank excell
ent product fast delivery
Original summary: greenies buddy treat
Predicted summary:  great treats


Review: dog loves tiny treats keep one car one house
Original summary: dog loves them
Predicted summary:  dog treats


Review: liked coffee much subscribing dark rich smooth
Original summary: makes great cup of java
Predicted summary:  great coffee


Review: far dog tried chicken peanut butter flavor absolutely loves love natural makes happy giving dog some
thing healthy treats small soft big plus calories
Original summary: love zuke mini naturals
Predicted summary:  my dog loves these


Review: absolutely delicious satisfy something sweet really filling great early morning time make breakfast
great afternoon snack work feeling sluggish
Original summary: protein bar
Predicted summary:  yummy


Review: aware decaf coffee although showed search decaf cups intended purchase gift kept recipient drink caf
feine favorite means
Original summary: not decaf
Predicted summary:  not bad


Review: wonderful wrote perfect iced cookie one pen writing cookies names happy ca
Original summary: cookie
Predicted summary:  great cookies
```

```
Review: truffle oil quite good prefer brand france urbani italy expensive oh delicious tried black white goo
d black bit stronger pungent event healthy alternative butter enjoy
Original summary: delicious but not the best
Predicted summary:  great product


Review: enjoy coffee office split right middle loving think worth try order regularly
Original summary: hit or miss
Predicted summary:  great coffee


Review: husband gluten free food several years tried several different bread mixes first actually enjoys buy
ing amazon saves loaf
Original summary: really good gluten free bread
Predicted summary:  great gluten free bread mix


Review: hubby eats says good snacks morning done apple flavor
Original summary: really good nice snack
Predicted summary:  great snack


Review: waste money disgusting product chocolate taste tastes like plastic lining paper carton using milk tr
eated ultra high temperatures like fresh milk go get fresh milk hershey syrup want chocolate milk
Original summary: please do not waste your money
Predicted summary:  nasty


Review: absolutely loves apple chicken happy hips looks forward one morning one night gets soooo excited wou
ld eat allowed
Original summary: healthy treats
Predicted summary:  my dog loves these


Review: strong much flavor little aroma tried purchase another time similiar brands met standards expected
Original summary: no flavor
Predicted summary:  good coffee


Review: company wanted chose order anyway
Original summary: water
Predicted summary:  good product
```

```
Review: introduced number people hooked best sour gummy ever great flavors got great price
Original summary: new favorite
Predicted summary:  best gummi bears


Review: new price attractive however tastes horrible maybe old zico coconut water brands might find acceptab
le
Original summary: do not be by the price
Predicted summary:  bad taste


Review: sure ever going buy product way expensive market price
Original summary: too expensive
Predicted summary:  good product


Review: flavor normally find local stores plus buy bulk things take savings add veggies even stir egg noodle
s cook add nutrition quick meals lot extra
Original summary: good value
Predicted summary:  great product


Review: order tea labeled decaff must caffeine residue levels tested tea caffeine decaff non decaff tea anyw
here caffeine caffeine caffeinated tea caffeine slightly less naturally present tea leaf
Original summary: caffeine is not
Predicted summary:  tea


Review: excellent babies toddler really best offer little one delicious rich vitamins calcium protein low fa
t sorry products available website
Original summary: excellent product for babies and toddler
Predicted summary:  great product


Review: purchased item dented would bet run dented product clearing ship ones
Original summary: sometimes dented
Predicted summary:  dented cans


Review: almost tastes like mini blueberry pie love one favorite thoroughly fallen love
```

```
Original summary: excellent love the blueberry pecan
Predicted summary:  great taste


Review: dog loves keeps busy minutes long time chew hound
Original summary: chew away
Predicted summary:  dog loves them


Review: plant came quickly looks great office nice pot plant thriving well
Original summary: very nice office plant
Predicted summary:  great gift


Review: dog loves lickety stik bacon flavor since likes much plan getting flavors great liquid treat dog hig
hly recommend lickety stik
Original summary: great dog treat
Predicted summary:  dog loves them


Review: great toy dogs chew everything else little literally eats toys one toys yet destroy loves carries ar
ound everywhere got rex cutest thing
Original summary: good for chewers
Predicted summary:  dogs love it


Review: really search good deals tea tea great price tea amazon almost cup price cup coffee herbal varieties
low caffine good option wife used dinner coffe
Original summary: great price for great tea
Predicted summary:  great tea


Review: pricey essentially small bag hard crumbs maybe dog spoiled treats like third class treats definitely
bottom doggie treat often simply walk away glad people like buying
Original summary: waste of money
Predicted summary:  not as pictured


Review: little pricey consider sugar low cal caffine really rich flavor best chai ever found
Original summary: fabulous product
Predicted summary:  great product
```

```
Review: loves taste beef freeze dried dog treats use training really works
Original summary: dog lover
Predicted summary:  great treat


Review: three dogs cairn terriers year old border collie proud greenies like taste helps keep gums teeth goo
d shape
Original summary: our dogs love greenies
Predicted summary:  great product


Review: good soft drink smooth strawberry cream soda tasty
Original summary: good stuff
Predicted summary:  refreshing


Review: item arrived sugar free shipped regular version caramel syrup small internal sticker bottle stated s
ugar free although company label bottle stated regular version
Original summary: wrong item
Predicted summary:  not as good as the


Review: like strong coffee coffee rated found weak sickening taste
Original summary: disapointed
Predicted summary:  weak coffee


Review: saw peanut butter chocolate cereal knew try pleased eat chocolate breakfast feel guilty two kids lov
e cereal well great eat alone favorite milk product yogurt mix homemade granola well
Original summary: the yummy
Predicted summary:  great snack


Review: begging time loves used buy small bottle buying every weeks since saw oz buying last lot longer gas
money cheaper buy online
Original summary: my dog loves it
Predicted summary:  great product


Review: true also need decent scale tried caviar recipe everything worked perfectly first try fun easy make
kit comes large enough samples looks like good uses
```

```
Original summary: great to
Predicted summary:  great product


Review: dog really likes treats like buy run mill treats loaded fat fillers continue buy
Original summary: buddy biscuits
Predicted summary:  dog treats


Review: tulsi green tea great good iced tea well
Original summary: green tea
Predicted summary:  great tea


Review: always put something market couple poof gone best tasting product pepsi
Original summary: best taste
Predicted summary:  great taste


Review: like tomatoes fresh flavorful also come carton welcome alternative metal cans impart flavor sometime
s lined plastic containing
Original summary: yummy tomatoes good packaging
Predicted summary:  great product


Review: great get habit forming careful bought whole case save overall versus going supermarket rich dark ch
ocolate crisp cookie worth every penny oreo eat heart
Original summary: delicious
Predicted summary:  delicious


Review: else say arrived promptly perhaps time expected expiration date like next day good go
Original summary: baby loves it
Predicted summary:  not what expected


Review: bought local recently advertised cheesy flavor detectable product even salt flavor avoid product
Original summary: no cheese flavor
Predicted summary:  not too


Review: big volume coffee morning one great
```

```
Original summary: great morning coffee
Predicted summary:  great coffee


Review: drank try keep awake fell asleep minutes drinking feel anything
Original summary: it made me fall
Predicted summary:  not the best


Review: drink cups day verona italian french roast coffee wanted try lower acid version brand coffee smells
tastes like vinegar totally unpalatable better drinking water acid coffee bothers
Original summary: single worst coffee ever
Predicted summary:  not bad


Review: getting price however afraid stocking anymore reduced price think one trying eat crackers low calori
e string cheese breakfast every total calories put breakfast baggie go
Original summary: am addicted to these
Predicted summary:  yummy


Review: first time using fondarific fondant general one really easy use baby shower cake worked indicated al
so colored made two tier cake final product looked great greasy
Original summary: easy to use
Predicted summary:  great product


Review: work home drink cups cup coffee day good tasting coffee lowest price cup market
Original summary: great coffee great price
Predicted summary:  great coffee


Review: guys say natural really tastes great pleasantly surprised stand flavor carbonated think would even b
etter product time come fed sweet juices aftertaste make obvious really natural switch really gets vote
Original summary: great taste all natural
Predicted summary:  not bad


Review: product good goes long way quite good one dd good product less
Original summary: very good
Predicted summary:  good product
```

```
Review: tea wonderful soothing even soothing get shipped house found hard find decaffeinated tea grocery sto
re much easier
Original summary: decaffeinated french vanilla tea yummy
Predicted summary:  great tea


Review: wow little calorie espresso sugar serve cold delicious little shot espresso sugar overly sweet sugar
helps offset taste espresso caffe bitter sweet tastes good really gave afternoon kick pants
Original summary: nice little pick me up
Predicted summary:  great taste


Review: mayonnaise delicious side side taste test would give hellman edge hellman richer taste
Original summary: excellent but
Predicted summary:  great product


Review: love medium full flavored roast smooth taste bitter acidic taste excellent coffee good value also tr
y timothy kona good also
Original summary: wonderful coffee
Predicted summary:  great coffee


Review: nice item chunks meat good gravy cat fond varieties nice little treat nonetheless think item bit pri
cy per ounce
Original summary: nice but pricey
Predicted summary:  good stuff


Review: bought cookies gifts open last long good make great gifts would definitely buy
Original summary: mouth watery cookies
Predicted summary:  great cookies


Review: great price fast shipping best chips better ingredients less calories snack foods plus taste like re
al chips
Original summary: pop chips are the best
Predicted summary:  great chips


Review: taco bell chipotle sauce bold flavorful tried chicken wings tacos salad made dish extremely tasty gl
```

```
ad sampled new sauce staple condiment
Original summary: bold flavor
Predicted summary:  great hot cocoa


Review: bought seeds make centerpieces really surprised fast grow planted seeds potting soil without ny prep
aration anything kept watering days super tall ready displayed centerpieces perfect
Original summary: perfect for in days
Predicted summary:  great gift


Review: every time need sun dried tomatoes local grocery stores conveniently small pouches ensure always han
d called recipe
Original summary: sun dried tomato bliss
Predicted summary:  great gift


Review: love soup eat plain use recipe cannot find area glad amazon
Original summary: soup chicken cheese
Predicted summary:  soup soup soup soup soup


Review: size quite good dog training smell strong cannot put open bag must seal everytime gave treat otherwi
se dog stand trying fetch believe taste great puppy purchase sure
Original summary: strong smell and my puppy loves it
Predicted summary:  dog treats


Review: love chips auto order every months taste great whole bag calories bag every day sure helped weight l
oss little bags eat huge amount
Original summary: great purchase
Predicted summary:  great chips


Review: many kit wines cost three four times made many kits find fine table wine recommend adding water five
gallon mark flavor
Original summary: good wine
Predicted summary:  great product


Review: sooo much pepper heavy salt reminds adams trick food cannot eat seriously fresh nuts seasoned
Original summary: over the top seasoning
```

```
Predicted summary:  great salt


Review: loved brand best vanilla flavor others tried would buy better price
Original summary: wolfgang puck coffee vanilla
Predicted summary:  great coffee


Review: another brand cinammon carried amazon much better tasting brand maybe packaging part problem simple
plastic bag tie amazon brand comes carefully set plastic box
Original summary: edible have had much better
Predicted summary:  not the best


Review: throw pack one actually taste bad especially compared orange tangerine like carbonation adds juice f
lavors need work switch drinks best worst watermelon strawberry kiwi berry black cherry orange tangerine
Original summary: my favorite of the four tried
Predicted summary:  not bad


Review: daughter drinking since months old months old still loves snack time healthy delicious great additio
n menu
Original summary: great snack
Predicted summary:  great snack


Review: live guinea africa order products delivered boat every months sometimes disappointed time zero calor
ies zero carbs taste great price zero delivery costs prime ordered different flavors one favorite love
Original summary: love it
Predicted summary:  great product


Review: purchased larger size love size perfect keep purse snack especially times others dessert snack canno
t eat must gluten free spouse touch diet food loves
Original summary: cannot get enough
Predicted summary:  great snack


Review: always house drink favorite mix sprite oh good every day mind larger bottles use much bring
Original summary: am an adult still love this
Predicted summary:  great product
```

```
Review: ginger snaps overpowering ginger go great milk really enjoyed house great buy affordable compared al
ternative diet foods last least week store well
Original summary: you can eat ginger again
Predicted summary:  ginger ginger drink


Review: give squid one star use might thoroughly disappointed quite possibly call crazy
Original summary: can for your
Predicted summary:  great product


Review: quality seeds excellent begin germinate hours days ready use never sprouted seeds results good easil
y recommend sprouter whether human consumption four legged friends
Original summary: wheat grass seeds
Predicted summary:  great product


Review: love stuff great store bought homemade baked goods kicking things professional level works colored d
ark light frosting also used dusting powdered sugar pretty fine texture
Original summary: fun like dust
Predicted summary:  great product


Review: bought jumbo greenies black lab loved way expensive regular use notice difference breath primary rea
son buying
Original summary: jumbo greenies good but very expensive
Predicted summary:  greenies


Review: also bought costco per box included bags oz kids fighting remaining bags good buying due price high
price prevent product reaching mass distribution
Original summary: very good but too pricey
Predicted summary:  great product


Review: originally found mints whole foods taste superb get lot money plus comes cute little tin uses dog lo
ves go organic
Original summary: wonderful
Predicted summary:  great product
```

```
Review: regular spam awful almost inedible would give tastes like animal know mean fellow spam turkey spam p
retty good great would give worth try
Original summary: better than regular
Predicted summary:  not bad


Review: really need know many cans also whitefish tuna buffet canned cat food thanks
Original summary: need to know how many in case
Predicted summary:  great product


Review: great tasting rich flavor perfect making nice hot cup mocha bought test hershey syrup mocha incredib
le distinct taste difference noticeable much richer tastes like chocolate less sugary hershey syrup
Original summary: great taste
Predicted summary:  great taste


Review: number one japan number one great save get shipped automatically every month lugging car
Original summary: great tea
Predicted summary:  great product


Review: bought item read best mayo sold yes even better worlds favorite hellman well review good bit better
hellman fact put empty hellman jar said nothing family never knew difference
Original summary: blue mayo
Predicted summary:  not bad


Review: gum great makes car smell good leave refreshing sweet tart smooth
Original summary: love the gum and the price
Predicted summary:  great gum


Review: flavorful smells like heaven great price compared stores arrived fast
Original summary: divine
Predicted summary:  great product


Review: love low calorie organic doctors recommend grams fiber daily smart bran grams per serving fruits veg
gies set day eat dry vanilla frozen yogurt cinnamon
Original summary: yes to smart bran
Predicted summary:  great product
```

```
Review: found spice blend dallas years back tell restaurant using grilled shrimp like cajun spice grilling f
ish recommend store dry place replace every year least lose flavor
Original summary: good stuff
Predicted summary:  great product


Review: plain riceselect couscous delicious easy quick prepare great side item base main course far found ba
d product riceselect
Original summary: yummy
Predicted summary:  great
```

This is really cool stuff. Even though the actual summary and the summary generated by our model do not match in terms of words, both of them are conveying the same meaning. Our model is able to generate a legible summary based on the context present in the text.

This is how we can perform text summarization using deep learning concepts in Python.

How can we Improve the Model’s Performance Even Further?

increase the training dataset size and build the model. The generalization capability of a deep learning model enhances with an increase in the training dataset size

Bi-Directional LSTM which is capable of capturing the context from both the directions and results in a better context vector

beam search strategy for decoding the test sequence instead of using the greedy approach (argmax)

Evaluate the performance of model based on the **BLEU score**

pointer-generator networks and **coverage mechanisms**

In [ ]:

In [ ]: