Text Summarization using Deep Learning Model

Developed by

- Islam Mohamed (https://github.com/islammohamedd1)
- Mostafa Yasser (https://github.com/mostafayasser)
- Eslam Mamdouh (https://github.com/EslamMamdoouh)

Problem Description and Background

Some people may not have the time to examine articles during their day because of time shortage.

Solution

The solution is to develop a learning agent to solve the problem. The agent will be a deep learning model that summarizes the output text using NLP techniques with Recurrent Neural Network to understand the context of the text and generate a summarization of the text.

Import required libraries

In [1]:

```
import numpy as np
import pandas as pd
import re
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, Conca
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import EarlyStopping

import warnings
warnings.filterwarnings("ignore")
```

```
Using TensorFlow backend.
```

Load the dataset

In [2]:

```
!wget --no-check-certificate  https://storage.googleapis.com/islamohamed

data = pd.read_csv('data.csv', nrows=150000)
data.drop_duplicates(subset=['Text'],inplace=True)
data.dropna(axis=0,inplace=True)
```

```
--2020-05-13 08:17:47--  https://storage.googleapis.com/islamohamedd1.a
ppspot.com/amazon-fine-food-reviews/Reviews.csv (https://storage.google
apis.com/islamohamedd1.appspot.com/amazon-fine-food-reviews/Reviews.cs
v)
Resolving storage.googleapis.com (storage.googleapis.com)... 64.233.18
9.128, 2404:6800:4008:c07::80
Connecting to storage.googleapis.com (storage.googleapis.com)|64.233.18
9.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 300904694 (287M) [text/csv]
Saving to: ‘data.csv’

data.csv            100%[===================>] 286.96M   195MB/s    in
 1.5s

2020-05-13 08:17:49 (195 MB/s) - ‘data.csv’ saved [300904694/300904694]
```

Extract and format the training and testing data from the dataset

In [0]:

```
# contraction_mapping source: https://gist.github.com/aravindpai/f21a286
contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "

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

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

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

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

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

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

                           "mustn't": "must not", "mustn't've": "must no

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

                           "she'd": "she would", "she'd've": "she would

                           "should've": "should have", "shouldn't": "sho

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

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

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

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

                           "we've": "we have", "weren't": "were not", "w

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

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

                           "why's": "why is", "why've": "why have", "wil

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

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

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

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

contraction_mapping['<br>'] = ""
```

In [4]:

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
nltk.download('punkt')

stop_words = set(stopwords.words('english'))

def clean_sentences(data):
  new_data = []
  for s in data:
    new_s = s.lower()
    new_s = ' '.join([w for w in s.split() if not w in stop_words])
    new_s = ' '.join([contraction_mapping[t] if t in contraction_mapping
    new_data.append(new_s)
  return new_data


def clean_labels(data):
  new_data = []
  for s in data:
    new_s = s.lower()
    new_s = ' '.join([contraction_mapping[t] if t in contraction_mapping
    new_data.append(new_s)
  return new_data

labels = clean_labels(data['Summary'])
sentences = clean_sentences(data['Text'])
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

In [0]:

```
new_labels = []
for s in labels:
  new_s = s.split()
  new_s = ['_START_'] + new_s + ['_END_']
  new_labels.append(' '.join(new_s))
labels = new_labels
```

In [6]:

```
labels[:3]
```

Out[6]:

```
['_START_ good quality dog food _END_',
 '_START_ not as advertised _END_',
 '_START_ "delight" says it all _END_']
```

In [0]:

```
from sklearn.model_selection import train_test_split
training_sentences, testing_sentences, training_labels, testing_labels =
```

Tokenize the sentences

The data is converted to be numeric tokens so the model can work with it easily

In [0]:

```
max_len = 80

x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(training_sentences)

x_training = x_tokenizer.texts_to_sequences(training_sentences)
x_validation = x_tokenizer.texts_to_sequences(testing_sentences)

x_training = pad_sequences(x_training, maxlen=max_len, padding='post')
x_validation = pad_sequences(x_validation, maxlen=max_len, padding='post

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

In [0]:

```
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(training_labels)

y_training = y_tokenizer.texts_to_sequences(training_labels)
y_validation = y_tokenizer.texts_to_sequences(testing_labels)

y_training = pad_sequences(y_training, maxlen=max_len, padding='post')
y_validation = pad_sequences(y_validation, maxlen=max_len, padding='post

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

In [10]:

```
y_tokenizer.word_counts['the']
```

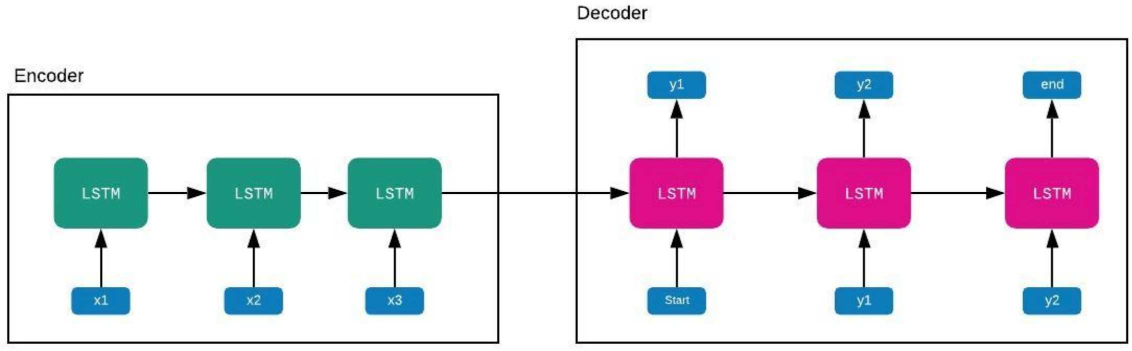
Out[10]:

```
11208
```

Create the model archeticure

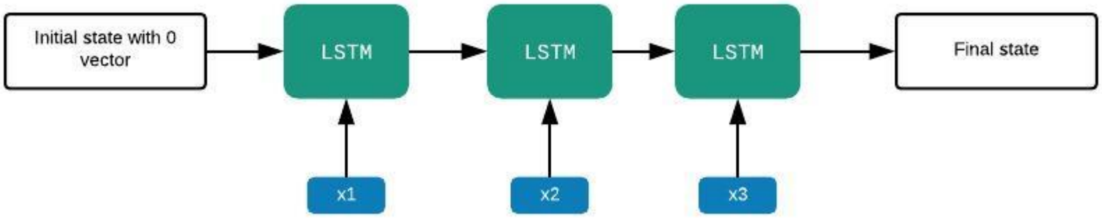
The model is usiing a LSTM (Long-Short-Term Memory) network for the encoder and another LSTM network for the decoder

The Encoder-Decoder archeticure will work as illustrated below



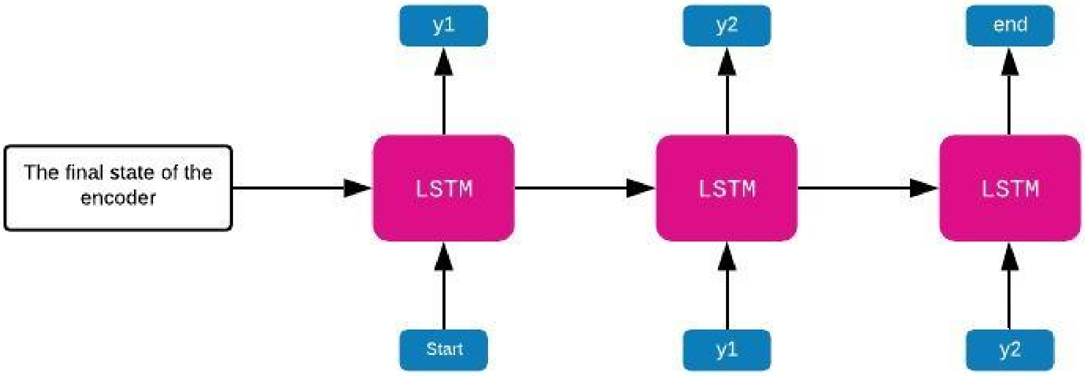
Encoder

The encoder is composed of a LSTM network. The LSTM network starts with a 0 state as it's initial state. Then, the encoder network recieves a word every timestamp. the word is passed to the first LSTM layer and processed, the LSTM layer pass the state after processing the word to the next LSTM layer. each LSTM layer recieves the a word from the input sequence and the previous state. Finally, the encoder outputs the final state.



Decoder

The decoder LSTM netword takes the final state of the encoder as it's initial state and the first word (start) as an input. Each LSTM layer in the decoder predicts the next word - y1, y2, etc - using the initial state of the encoder combined with the state of the previous LSTM state.



In [11]:

```
from keras import backend as K
K.clear_session()
embidding_dim = 500

# Encoder
encoder_inputs = Input(shape=(max_len,))
encoder_embedding = Embedding(x_vocab_size, embidding_dim, trainable=Tru

encoder_lstm1 = LSTM(embidding_dim, return_sequences=True, return_state=
encoder_output1, sate_h1, state_c1 = encoder_lstm1(encoder_embedding)

encoder_lstm2 = LSTM(embidding_dim, return_sequences=True, return_state=
encoder_output2, state_h1, state_c1 = encoder_lstm2(encoder_output1)

encoder_lstm3 = LSTM(embidding_dim, return_state=True, return_sequences=
encoder_outputs, state_h, state_c = encoder_lstm3(encoder_output2)

# Decoder
decoder_inputs = Input(shape=(None,))
decoder_embedding_layer = Embedding(y_vocab_size, embidding_dim, trainab
decoder_embedding = decoder_embedding_layer(decoder_inputs)

decoder_lstm = LSTM(embidding_dim, return_sequences=True, return_state=T
decoder_outputs, decoder_fwd_state, decoder_back_state = decoder_lstm(de

decoder_dense = TimeDistributed(Dense(y_vocab_size, activation='softmax'
decoder_outputs = decoder_dense(decoder_outputs)

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

```
Model: "model"
_____________________________________________________________________
_____________________________
Layer (type)                    Output Shape         Param #     Conn
ected to
=====================================================================
=============================
input_1 (InputLayer)            [(None, 80)]         0
_____________________________________________________________________
_____________________________
embedding (Embedding)           (None, 80, 500)      34941500    inpu
t_1[0][0]
_____________________________________________________________________
_____________________________
lstm (LSTM)                     [(None, 80, 500), (N 2002000     embe
dding[0][0]
_____________________________________________________________________
_____________________________
input_2 (InputLayer)            [(None, None)]       0
_____________________________________________________________________
_____________________________
lstm_1 (LSTM)                   [(None, 80, 500), (N 2002000     lstm
[0][0]
_____________________________________________________________________
```

```
_____________________________
embedding_1 (Embedding)         (None, None, 500)    9138500     inpu
t_2[0][0]
_____________________________________________________________________
_____________________________
lstm_2 (LSTM)                   [(None, 80, 500), (N 2002000     lstm
_1[0][0]
_____________________________________________________________________
_____________________________
lstm_3 (LSTM)                   [(None, None, 500),  2002000     embe
dding_1[0][0]
                                                                 lstm
_2[0][1]
                                                                 lstm
_2[0][2]
_____________________________________________________________________
_____________________________
time_distributed (TimeDistribut (None, None, 18277)  9156777     lstm
_3[0][0]
=====================================================================
=============================
Total params: 61,244,777
Trainable params: 61,244,777
Non-trainable params: 0
_____________________________________________________________________
_____________________________
```

In [0]:
```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy
```

Create an early stopping callback function

A callback function is called after each training epoch and stops the training of the validation loss started to increase

In [0]:
```
early_stopping_callback = EarlyStopping(monitor='val_loss', mode='min',
```

Mount google drive to the file system (Only in google colab)

In [14]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_t
ype=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.t
est%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fw
ww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.go
ogleapis.com%2fauth%2fpeopleapi.readonly (https://accounts.google.com/
o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a
pps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3a
oob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2
fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20h
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%
3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /drive
```

Create a callback function to save the weights of the model after each epoch

In [0]:

```
import os
import tensorflow as tf

checkpoint_path = "/drive/My Drive/ai_project/reviews/text_sum_model_wei
model_path = "/drive/My Drive/ai_project/reviews/text_sum_model"
checkpoint_dir = os.path.dirname(checkpoint_path)
```

In [0]:

```
# Create a callback that saves the model's weights
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkp
                                                 save_weights_only=True,
                                                 verbose=1)
```

Train the model

In [17]:

```
history=model.fit(
    [x_training,y_training[:,:-1]],
    y_training.reshape(y_training.shape[0],y_training.shape[1], 1)[:,1:]
    epochs=50,
    callbacks=[early_stopping_callback, checkpoint_callback],
    batch_size=512,
    verbose=1,
    validation_data=([x_validation,y_validation[:,:-1]], y_validation.re
    )
model.save(model_path)
```

```
Epoch 1/50
225/225 [==============================] - ETA: 0s - loss: 0.5200 - ac
c: 0.9419
Epoch 00001: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 262s 1s/step - loss: 0.5200
- acc: 0.9419 - val_loss: 0.3448 - val_acc: 0.9501
Epoch 2/50
225/225 [==============================] - ETA: 0s - loss: 0.3375 - ac
c: 0.9510
Epoch 00002: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 260s 1s/step - loss: 0.3375
- acc: 0.9510 - val_loss: 0.3152 - val_acc: 0.9527
Epoch 3/50
225/225 [==============================] - ETA: 0s - loss: 0.3151 - ac
c: 0.9529
Epoch 00003: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 260s 1s/step - loss: 0.3151
- acc: 0.9529 - val_loss: 0.3015 - val_acc: 0.9540
Epoch 4/50
225/225 [==============================] - ETA: 0s - loss: 0.3002 - ac
c: 0.9540
Epoch 00004: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 261s 1s/step - loss: 0.3002
- acc: 0.9540 - val_loss: 0.2896 - val_acc: 0.9548
Epoch 5/50
225/225 [==============================] - ETA: 0s - loss: 0.2877 - ac
c: 0.9548
Epoch 00005: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 261s 1s/step - loss: 0.2877
- acc: 0.9548 - val_loss: 0.2831 - val_acc: 0.9553
Epoch 6/50
225/225 [==============================] - ETA: 0s - loss: 0.2774 - ac
c: 0.9555
Epoch 00006: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 260s 1s/step - loss: 0.2774
- acc: 0.9555 - val_loss: 0.2776 - val_acc: 0.9556
Epoch 7/50
225/225 [==============================] - ETA: 0s - loss: 0.2680 - ac
c: 0.9563
```

```
Epoch 00007: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 261s 1s/step - loss: 0.2680
- acc: 0.9563 - val_loss: 0.2725 - val_acc: 0.9563
Epoch 8/50
225/225 [==============================] - ETA: 0s - loss: 0.2595 - ac
c: 0.9570
Epoch 00008: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 260s 1s/step - loss: 0.2595
- acc: 0.9570 - val_loss: 0.2681 - val_acc: 0.9566
Epoch 9/50
225/225 [==============================] - ETA: 0s - loss: 0.2518 - ac
c: 0.9576
Epoch 00009: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 260s 1s/step - loss: 0.2518
- acc: 0.9576 - val_loss: 0.2652 - val_acc: 0.9569
Epoch 10/50
225/225 [==============================] - ETA: 0s - loss: 0.2446 - ac
c: 0.9582
Epoch 00010: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 261s 1s/step - loss: 0.2446
- acc: 0.9582 - val_loss: 0.2630 - val_acc: 0.9571
Epoch 11/50
225/225 [==============================] - ETA: 0s - loss: 0.2375 - ac
c: 0.9588
Epoch 00011: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 259s 1s/step - loss: 0.2375
- acc: 0.9588 - val_loss: 0.2624 - val_acc: 0.9571
Epoch 12/50
225/225 [==============================] - ETA: 0s - loss: 0.2307 - ac
c: 0.9594
Epoch 00012: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 261s 1s/step - loss: 0.2307
- acc: 0.9594 - val_loss: 0.2610 - val_acc: 0.9573
Epoch 13/50
225/225 [==============================] - ETA: 0s - loss: 0.2241 - ac
c: 0.9600
Epoch 00013: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 260s 1s/step - loss: 0.2241
- acc: 0.9600 - val_loss: 0.2609 - val_acc: 0.9573
Epoch 14/50
225/225 [==============================] - ETA: 0s - loss: 0.2174 - ac
c: 0.9607
Epoch 00014: saving model to /drive/My Drive/ai_project/reviews/text_su
m_model_weights/cp.ckpt
225/225 [==============================] - 260s 1s/step - loss: 0.2174
- acc: 0.9607 - val_loss: 0.2614 - val_acc: 0.9572
Epoch 00014: early stopping
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow/python/ops/resource_variable_ops.py:1817: calling BaseResourceVariab
le.__init__ (from tensorflow.python.ops.resource_variable_ops) with con
```

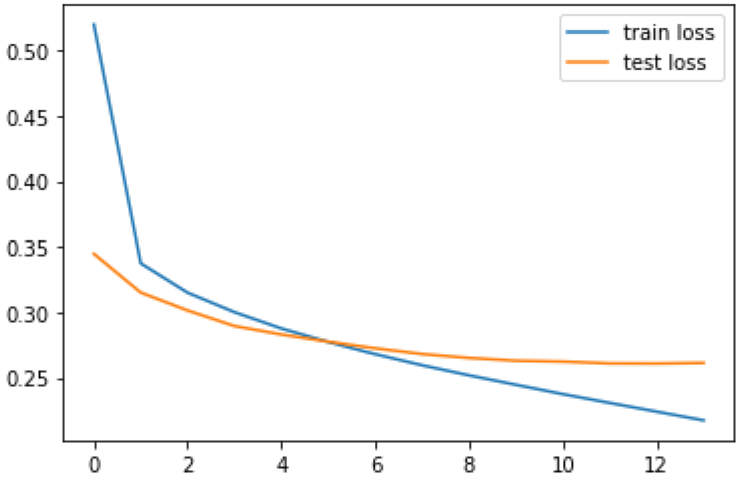
```
straint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: /drive/My Drive/ai_project/reviews/t
ext_sum_model/assets
```

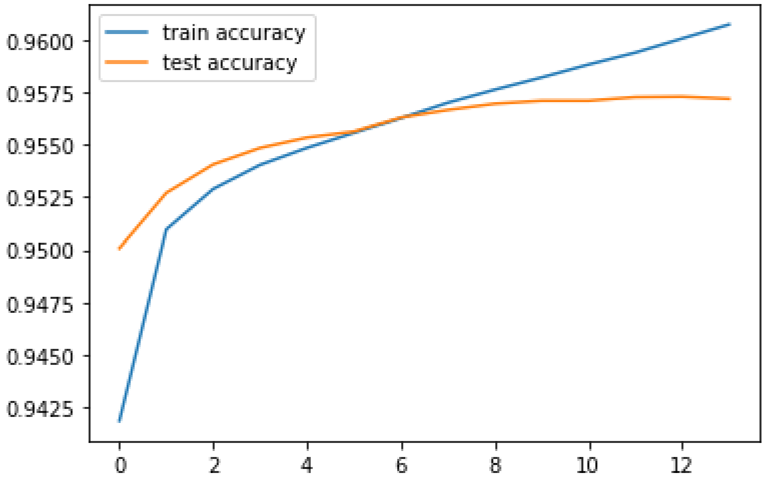
Plot the progress of the loss and the accuracy during the training

In [18]:

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

pyplot.plot(history.history['acc'], label='train accuracy')
pyplot.plot(history.history['val_acc'], label='test accuracy')
pyplot.legend()
pyplot.show()
```





In [0]:

```
# model.load_weights(checkpoint_path)
```

Create a reveresed word_index to be able to get the text of each word

using it's token

In [0]:

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

In [0]:

```
# encoder inference
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, st

#decoder inference
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(embidding_dim,))
decoder_state_input_c = Input(shape=(embidding_dim,))
decoder_hidden_state_input = Input(shape=(max_len, embidding_dim))

# Get the embeddings of the decoder sequence
decoder_embedding2 = decoder_embedding_layer(decoder_inputs)

# To predict the next word in the sequence, set the initial states to th
decoder_outputs2, state_h2, state_c2 = decoder_lstm(decoder_embedding2,

decoder_outputs2 = decoder_dense(decoder_outputs2)

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

In [0]:

```
def decode_sequence(input_sequence):
    encoder_out, encoder_h, encoder_c = encoder_model.predict(input_sequ

    output_sequence = np.zeros((1,1))

    output_sequence[0, 0] = target_word_index['start']

    stop = False
    decoded_sentence = ''
    while not stop:
        output_tokens, h, c = decoder_model.predict([output_sequence] +

        heighest_probability_token = np.argmax(output_tokens[0, -1, :])
        if (heighest_probability_token == 0):
          break
        hieghest_probability_word = reverse_target_word_index[heighest_p

        if (hieghest_probability_word != 'end'):
          decoded_sentence += ' ' + hieghest_probability_word

        if (hieghest_probability_word == 'end' or len(decoded_sentence.s
          stop = True

        # Update the target sequence (of length 1).
        output_sequence = np.zeros((1,1))
        output_sequence[0, 0] = heighest_probability_token

        # Update internal states
        encoder_h, encoder_c = h, c

    return decoded_sentence
```

Test the model

In [0]:

```
def get_summary(input_sequence):
    newString=''
    for i in input_sequence:
      if((i!=0 and i!=target_word_index['start']) and i!=target_word_ind
        newString=newString+reverse_target_word_index[i]+' '
    return newString

def get_text(input_sequence):
    newString = ''
    for i in input_sequence:
      if(i != 0):
        newString = newString + reverse_source_word_index[i] + ' '
    return newString
```

In [24]:

```
for i in range(200):
  original_summary = get_summary(y_validation[i])
  predicted_summary = decode_sequence(x_validation[i].reshape(1,max_len)
  print("Review:", get_text(x_validation[i]))
  print("Original summary:", original_summary)
  print("Predicted summary:", predicted_summary)

  print("\n")
```

```
y science diet s fish flavored food give try
Original summary: not tastey
Predicted summary:  my cats love it


Review: my cats agree really love dry food i surprised even seemed li
ke better epigen 90 i normally feed them no barf potty issues either
eating food week whew lets hope still case i try non hairball formula
Original summary: chicken is paw good
Predicted summary:  my cats love it


Review: works well power breeds heavy chewers if cheaper id buy dozen
we pitbulls rottweilers far best choice keeping busy

Original summary: great for larger dogs
Predicted summary:  great product


Review: this negative 3 star review the noodles squid ish taste textu
re i nearly gagged trying are bad or no calorie mush plant solubles r
```

In [0]: