In this notebook, You will do amazon review classification with BERT.[Download data from this link]

```
It contains 5 parts as below.  Detailed instrctions are given in the each cell. pl
ease read every comment we have written.
    1. Preprocessing
    2. Creating a BERT model from the Tensorflow HUB.
    3. Tokenization
    4. getting the pretrained embedding Vector for a given review from the BERT.
    5. Using the embedding data apply NN and classify the reviews.
    6. Creating a Data pipeline for BERT Model.
```

instructions:

```
    1. Don't change any Grader Functions. Don't manipulate any Grader functions.
    If you manipulate any, it will be considered as plagiarised.

    2. Please read the instructions on the code cells and markdown cells. We will e
xplain what to write.

    3. please return outputs in the same format what we asked. Eg. Don't return Li
st if we are asking for a numpy array.

    4. Please read the external links that we are given so that you will learn the
concept behind the code that you are writing.

    5. We are giving instructions at each section if necessary, please follow them.
```

Every Grader function has to return True.

In [2]:

```
#all imports
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.keras.models import Model
```

In [3]:

```
tf.test.gpu_device_name()
```

Out[3]:

```
'/device:GPU:0'
```

Grader function 1

In [7]:

```
def grader_tf_version():
    assert((tf.__version__)>'2')
    return True
grader_tf_version()
```

Out[7]:

```
True
```

Part-1: Preprocessing

In [4]:

```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv(r"/content/drive/My Drive/colab resources/AppliedAI/BERT/Reviews.cs
v")
#check the info of the dataset
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype 
---  ------                  --------------   ----- 
 0   Id                      568454 non-null  int64 
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568438 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64 
 5   HelpfulnessDenominator  568454 non-null  int64 
 6   Score                   568454 non-null  int64 
 7   Time                    568454 non-null  int64 
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

In [5]:

```
def score_mapper(x):
  #if score> 3, set score = 1
  if x > 3:
    return 1
  #if score<=2, set score = 0
  if x <= 2:
    return 0

#get only 2 columns - Text, Score
reviews = reviews[['Text', 'Score']]
reviews['Score'] = reviews['Score'].map(score_mapper)
#if score == 3, remove the rows.
reviews = reviews.dropna()
```

Grader function 2

In [6]:

```
def grader_reviews():
    temp_shape = (reviews.shape == (525814, 2)) and (reviews.Score.value_counts()[1]==44
3777)
    assert(temp_shape == True)
    return True
grader_reviews()
```

Out[6]:

```
True
```

In [8]:

```
def get_wordlen(x):
    return len(x.split())
reviews['len'] = reviews.Text.apply(get_wordlen)
reviews = reviews[reviews.len<50]
reviews = reviews.sample(n=100000, random_state=30)
```

In [9]:

```
import re

#remove HTML from the Text column and save in the Text column only
reviews['Text'] = reviews['Text'].map(lambda x: re.sub(r'<.*?>', '', x))
```

In [10]:

```
#print head 5
reviews.head()
```

Out[10]:

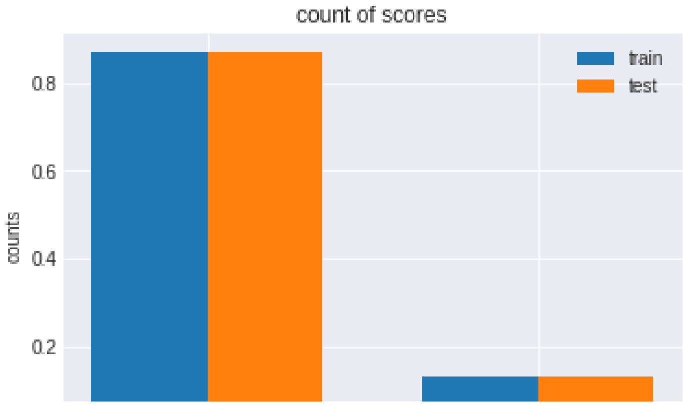
	Text	Score	len
64117	The tea was of great quality and it tasted lik...	1.0	30
418112	My cat loves this. The pellets are nice and s...	1.0	31
357829	Great product. Does not completely get rid of ...	1.0	41
175872	This gum is my favorite! I would advise every...	1.0	27
178716	I also found out about this product because of...	1.0	22

In [11]:

```
from sklearn.model_selection import train_test_split
#split the data into train and test data(20%) with Stratify sampling, random state 33,

X, y = reviews.drop('Score', axis=1), reviews['Score']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=33)
```

In [12]:

```
import matplotlib.pyplot as plt

plt.style.use('seaborn-darkgrid')
#plot bar graphs of y_train and y_test


fig, ax = plt.subplots()

width = 0.35
indices = np.arange(2)

counts = y_train.value_counts(normalize=True)
bar1 = ax.bar(indices, counts, width)

counts = y_test.value_counts(normalize=True)
bar2 = ax.bar(indices + width, counts, width)

ax.set_ylabel('counts')
ax.set_xlabel('score')
ax.set_title('count of scores')
ax.set_xticks(indices + width/2)
ax.set_xticklabels(['1', '2'])
ax.legend((bar1[0], bar2[0]), ('train', 'test'))
fig.show()
```



In [13]:

```
#saving to disk. if we need, we can load preprocessed data directly.
reviews.to_csv('preprocessed.csv', index=False)
```

Part-2: Creating BERT Model

If you want to know more about BERT, You can watch live sessions on Transformers and BERt.
we will strongly recommend you to read Transformers, BERT Paper and, This blog.

For this assignment, we are using BERT uncased Base model.
It uses L=12 hidden layers (i.e., Transformer blocks), a hidden size of H=768, and A=12 attention heads.

In [14]:

```
## Loading the Pretrained Model from tensorflow HUB
tf.keras.backend.clear_session()

# maximum length of a seq in the data we have, for now i am making it as 55. You can change this
max_seq_length = 55

#BERT takes 3 inputs

#this is input words. Sequence of words represented as integers
input_word_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input_word_ids")

#mask vector if you are padding anything
input_mask = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input_mask")

#segment vectors. If you are giving only one sentence for the classification, total seg vector is 0.
#If you are giving two sentenced with [sep] token separated, first seq segment vectors are zeros and
#second seq segment vector are 1's
segment_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="segment_ids")

#bert layer
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1", trainable=False)
pooled_output, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])

#Bert model
#We are using only pooled output not sequence out.
#If you want to know about those, please read https://www.kaggle.com/questions-and-answers/86510
bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_output)
```

In [15]:

```
bert_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________
__________
Layer (type)                    Output Shape          Param #      Connected to
```

```
==========================================================================================
=========
input_word_ids (InputLayer)     [(None, 55)]         0


__________________________________________________________________________________________
_________
input_mask (InputLayer)         [(None, 55)]         0


__________________________________________________________________________________________
_________
segment_ids (InputLayer)        [(None, 55)]         0


__________________________________________________________________________________________
_________
keras_layer (KerasLayer)        [(None, 768), (None, 109482241   input_word_ids[0][0]

                                                                 input_mask[0][0]

                                                                 segment_ids[0][0]

==========================================================================================
=========
Total params: 109,482,241
Trainable params: 0
Non-trainable params: 109,482,241
__________________________________________________________________________________________
_________
```

In [16]:

```
bert_model.output
```

Out[16]:

```
<tf.Tensor 'keras_layer/Identity:0' shape=(None, 768) dtype=float32>
```

Part-3: Tokenization

In [17]:

```
#getting Vocab file
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
```

In [20]:

```
import tokenization # - We have given tokenization.py file
```

In [21]:

```
# Create tokenizer " Instantiate FullTokenizer"
# name must be "tokenizer"
# the FullTokenizer takes two parameters 1. vocab_file and 2. do_lower_case
# we have created these in the above cell ex: FullTokenizer(vocab_file, do_lower_case )
# please check the "tokenization.py" file the complete implementation
tokenizer = tokenization.FullTokenizer(vocab_file, do_lower_case)
```

Grader function 3

In [22]:

```
#it has to give no error
def grader_tokenize(tokenizer):
    out = False
    try:
        out=('[CLS]' in tokenizer.vocab) and ('[SEP]' in tokenizer.vocab)
    except:
        out = False
    assert(out==True)
    return out
grader_tokenize(tokenizer)
```

Out[22]:

```
True
```

In [23]:

```
%%time

from tensorflow.keras.preprocessing.sequence import pad_sequences

# Create train and test tokens (X_train_tokens, X_test_tokens) from (X_train, X_test) usi
ng Tokenizer and
X_train_tokens = X_train['Text'].map(tokenizer.tokenize)
X_test_tokens = X_test['Text'].map(tokenizer.tokenize)

# maximum number of tokens is 55(We already given this to BERT layer above) so shape is (
None, 55)
# if it is less than 55, add '[PAD]' token else truncate the tokens length.(similar to pa
dding)
X_train_tokens = pad_sequences(X_train_tokens, maxlen=max_seq_length - 2,
                               value='[PAD]', dtype=object)
X_test_tokens = pad_sequences(X_test_tokens, maxlen=max_seq_length - 2,
                              value='[PAD]', dtype=object)

# add '[CLS]' at start of the Tokens and '[SEP]' at the end of the tokens.

def complete_sequence(data):
  completed_sequences = []
  for row in data:
    completed_sequences.append(['[CLS]', *row, '[SEP]'])
  return np.array(completed_sequences)

X_train_tokens = complete_sequence(X_train_tokens)
X_test_tokens = complete_sequence(X_test_tokens)

# Based on padding, create the mask for Train and Test ( 1 for real token, 0 for '[PAD]')
,
# it will also same shape as input tokens (None, 55) save those in X_train_mask, X_test_m
ask
X_train_mask = (X_train_tokens != '[PAD]')
X_test_mask = (X_test_tokens != '[PAD]')

# Create a segment input for train and test. We are using only one sentence so all zeros.
This shape will also (None, 55)
X_train_segment = np.zeros(X_train_tokens.shape)
X_test_segment = np.zeros(X_test_tokens.shape)

# type of all the above arrays should be numpy arrays

# after execution of this cell, you have to get
# X_train_tokens, X_train_mask, X_train_segment
# X_test_tokens, X_test_mask, X_test_segment
```

```
CPU times: user 46.3 s, sys: 444 ms, total: 46.8 s
Wall time: 46.8 s
```

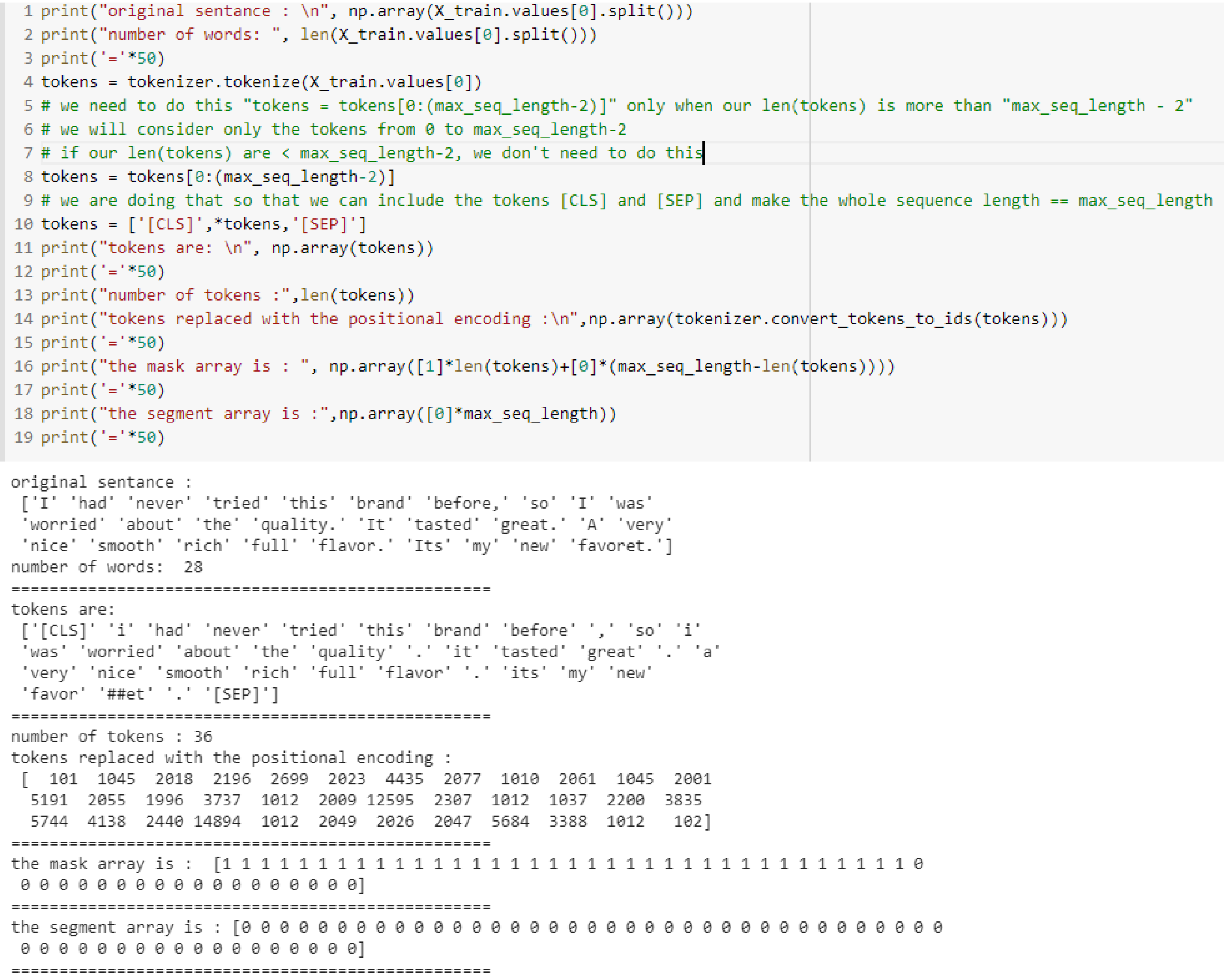
In [24]:

```
%%time

def convert_tokens_to_ids(data, tokenizer):
  ids = []
  for row in data:
    ids.append(tokenizer.convert_tokens_to_ids(row))

  return np.array(ids)
```

```
X_train_tokens = convert_tokens_to_ids(X_train_tokens, tokenizer)
X_test_tokens = convert_tokens_to_ids(X_test_tokens, tokenizer)
```

```
CPU times: user 2.82 s, sys: 29.7 ms, total: 2.84 s
Wall time: 2.85 s
```

Example



```
print("original sentance : \n", np.array(X_train.values[0].split()))
print("number of words: ", len(X_train.values[0].split()))
print('='*50)
tokens = tokenizer.tokenize(X_train.values[0])
# we need to do this "tokens = tokens[0:(max_seq_length-2)]" only when our len(tokens) is more than "max_seq_length - 2"
# we will consider only the tokens from 0 to max_seq_length-2
# if our len(tokens) are < max_seq_length-2, we don't need to do this
tokens = tokens[0:(max_seq_length-2)]
# we are doing that so that we can include the tokens [CLS] and [SEP] and make the whole sequence length == max_seq_length
tokens = ['[CLS]',*tokens,'[SEP]']
print("tokens are: \n", np.array(tokens))
print('='*50)
print("number of tokens :",len(tokens))
print("tokens replaced with the positional encoding :\n",np.array(tokenizer.convert_tokens_to_ids(tokens)))
print('='*50)
print("the mask array is : ", np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens))))
print('='*50)
print("the segment array is :",np.array([0]*max_seq_length))
print('='*50)
```

```
original sentance : 
 ['I' 'had' 'never' 'tried' 'this' 'brand' 'before,' 'so' 'I' 'was'
 'worried' 'about' 'the' 'quality.' 'It' 'tasted' 'great.' 'A' 'very'
 'nice' 'smooth' 'rich' 'full' 'flavor.' 'Its' 'my' 'new' 'favoret.']
number of words:  28
==================================================
tokens are: 
 ['[CLS]' 'i' 'had' 'never' 'tried' 'this' 'brand' 'before' ',' 'so' 'i'
 'was' 'worried' 'about' 'the' 'quality' '.' 'it' 'tasted' 'great' '.' 'a'
 'very' 'nice' 'smooth' 'rich' 'full' 'flavor' '.' 'its' 'my' 'new'
 'favor' '##et' '.' '[SEP]']
==================================================
number of tokens : 36
tokens replaced with the positional encoding :
 [  101  1045  2018  2196  2699  2023  4435  2077  1010  2061  1045  2001
  5191  2055  1996  3737  1012  2009 12595  2307  1012  1037  2200  3835
  5744  4138  2440 14894  1012  2049  2026  2047  5684  3388  1012   102]
==================================================
the mask array is :  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
==================================================
the segment array is : [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
==================================================
```

Grader function 4

In [27]:

```
def grader_alltokens_train():
    out = False

    if type(X_train_tokens) == np.ndarray:

        temp_shapes = (X_train_tokens.shape[1]==max_seq_length) and (X_train_mask.shape[
1]==max_seq_length) and \
        (X_train_segment.shape[1]==max_seq_length)

        segment_temp = not np.any(X_train_segment)

        mask_temp = np.sum(X_train_mask==0) == np.sum(X_train_tokens==0)

        no_cls = np.sum(X_train_tokens==tokenizer.vocab['[CLS]'])==X_train_tokens.shape[
0]

        no_sep = np.sum(X_train_tokens==tokenizer.vocab['[SEP]'])==X_train_tokens.shape[
0]

        out = temp_shapes and segment_temp and mask_temp and no_cls and no_sep
    else:
        print('Type of all above token arrays should be numpy array not list')
        out = False
    assert(out==True)
    return out

grader_alltokens_train()
```

Out[27]:

```
True
```

Grader function 5

In [28]:

```
def grader_alltokens_test():
    out = False
    if type(X_test_tokens) == np.ndarray:

        temp_shapes = (X_test_tokens.shape[1]==max_seq_length) and (X_test_mask.shape[1]
==max_seq_length) and \
        (X_test_segment.shape[1]==max_seq_length)

        segment_temp = not np.any(X_test_segment)

        mask_temp = np.sum(X_test_mask==0) == np.sum(X_test_tokens==0)

        no_cls = np.sum(X_test_tokens==tokenizer.vocab['[CLS]'])==X_test_tokens.shape[0]

        no_sep = np.sum(X_test_tokens==tokenizer.vocab['[SEP]'])==X_test_tokens.shape[0]

        out = temp_shapes and segment_temp and mask_temp and no_cls and no_sep

    else:
        print('Type of all above token arrays should be numpy array not list')
        out = False
    assert(out==True)
    return out
grader_alltokens_test()
```

Out[28]:

```
True
```

Part-4: Getting Embeddings from BERT Model

We already created the BERT model in the part-2 and input data in the part-3.
We will utlize those two and will get the embeddings for each sentence in the
Train and test data.

In [29]:

```
bert_model.input
```

Out[29]:

```
[<tf.Tensor 'input_word_ids:0' shape=(None, 55) dtype=int32>,
 <tf.Tensor 'input_mask:0' shape=(None, 55) dtype=int32>,
 <tf.Tensor 'segment_ids:0' shape=(None, 55) dtype=int32>]
```

In [30]:

```
bert_model.output
```

Out[30]:

```
<tf.Tensor 'keras_layer/Identity:0' shape=(None, 768) dtype=float32>
```

In [31]:

```
# get the train output, BERT model will give one output so save in
# X_train_pooled_output
X_train_pooled_output=bert_model.predict([X_train_tokens,X_train_mask,X_train_segment])
```

In [34]:

```
# get the test output, BERT model will give one output so save in
# X_test_pooled_output
X_test_pooled_output=bert_model.predict([X_test_tokens,X_test_mask,X_test_segment])
```

Grader function 6

In [36]:

```
#now we have X_train_pooled_output, y_train
#X_test_pooled_ouput, y_test

#please use this grader to evaluate
def greader_output():
    assert(X_train_pooled_output.shape[1]==768)
    assert(len(y_train)==len(X_train_pooled_output))
    assert(X_test_pooled_output.shape[1]==768)
    assert(len(y_test)==len(X_test_pooled_output))
    assert(len(y_train.shape)==1)
    assert(len(X_train_pooled_output.shape)==2)
    assert(len(y_test.shape)==1)
    assert(len(X_test_pooled_output.shape)==2)
    return True
greader_output()
```

Out[36]:

```
True
```

Part-5: Training a NN with 768 features

Create a NN and train the NN.

1. **You have to use AUC as metric.**
2. You can use any architecture you want.
3. You have to use tensorboard to log all your metrics and Losses. You have to send those logs.
4. Print the loss and metric at every epoch.
5. You have to submit without overfitting and underfitting.

In [37]:

```
##imports
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Model
```

In [53]:

```
class ReviewClassifier(Model):
  def __init__(self):
    super(ReviewClassifier, self).__init__()

    self.dense_1 = Dense(64, activation='relu')
    self.dense_2 = Dense(32, activation='relu')
    self.dense_3 = Dense(16, activation='relu')
    self.classify = Dense(1, activation='sigmoid')

    self.dropout_1 = Dropout(0.2)
    self.dropout_2 = Dropout(0.2)
    self.dropout_3 = Dropout(0.2)
  def call(self, inputs):
    x = self.dense_1(inputs)
    x = self.dropout_1(x)

    x = self.dense_2(x)
    x = self.dropout_2(x)

    x = self.dense_3(x)
    x = self.dropout_3(x)

    x = self.classify(x)

    return x
```

In [54]:

```
review_classifier = ReviewClassifier()
review_classifier.compile(loss='binary_crossentropy', optimizer='adam')
review_classifier.build((None, 768))
review_classifier.summary()
```

```
Model: "review_classifier_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              multiple                  49216
_________________________________________________________________
dense_7 (Dense)              multiple                  2080
_________________________________________________________________
dense_8 (Dense)              multiple                  528
_________________________________________________________________
dense_9 (Dense)              multiple                  17
_________________________________________________________________
dropout_4 (Dropout)          multiple                  0
_________________________________________________________________
dropout_5 (Dropout)          multiple                  0
_________________________________________________________________
dropout_6 (Dropout)          multiple                  0
=================================================================
Total params: 51,841
Trainable params: 51,841
Non-trainable params: 0
_________________________________________________________________
```

In [49]:

```
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import (Callback, EarlyStopping,
                                        TensorBoard, ReduceLROnPlateau)

class ROCCallback(Callback):

  def __init__(self, validation_data):
    super(ROCCallback, self).__init__()
    self.validation_data = validation_data

  def on_epoch_end(self, epoch, logs={}):
    probs = self.model.predict(self.validation_data[0])
    y_true = self.validation_data[1]
    score = roc_auc_score(y_true, probs)
    logs['auc'] = score
```

In [50]:

```
class AchieveTarget(Callback):

  def __init__(self, target):
    super(AchieveTarget, self).__init__()
    self.target = target

  def on_epoch_end(self, epoch, logs={}):
    acc = logs['auc']
    if acc >= self.target:
      self.model.stop_training = True
```

In [51]:

```
roc_callback = ROCCallback((X_test_pooled_output, y_test))
early_stopping = EarlyStopping(patience=5)
tensorboard = TensorBoard()
reduce_lr = ReduceLROnPlateau(patience=3)
target = AchieveTarget(0.95)

callbacks = [
             roc_callback,
             early_stopping,
             tensorboard,
             reduce_lr,
             target,
]
```

In [55]:

```
!rm -rf ./logs/*

history = review_classifier.fit(X_train_pooled_output, y_train, batch_size=32, epochs=10
0,
                    callbacks=callbacks, validation_data=(X_test_pooled_output, y_test))
```

```
Epoch 1/100
2344/2344 [==============================] - 11s 5ms/step - loss: 0.3151 - val_loss: 0.27
04 - auc: 0.9293 - lr: 0.0010
Epoch 2/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2972 - val_loss: 0.25
36 - auc: 0.9167 - lr: 0.0010
Epoch 3/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2924 - val_loss: 0.26
96 - auc: 0.9056 - lr: 0.0010
Epoch 4/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2916 - val_loss: 0.23
28 - auc: 0.9353 - lr: 0.0010
Epoch 5/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2557 - val_loss: 0.23
51 - auc: 0.9279 - lr: 0.0010
Epoch 6/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2422 - val_loss: 0.20
84 - auc: 0.9400 - lr: 0.0010
Epoch 7/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2389 - val_loss: 0.20
98 - auc: 0.9476 - lr: 0.0010
Epoch 8/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2340 - val_loss: 0.20
35 - auc: 0.9419 - lr: 0.0010
Epoch 9/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2337 - val_loss: 0.20
35 - auc: 0.9472 - lr: 0.0010
Epoch 10/100
2344/2344 [==============================] - 9s 4ms/step - loss: 0.2297 - val_loss: 0.205
2 - auc: 0.9449 - lr: 0.0010
Epoch 11/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2287 - val_loss: 0.19
74 - auc: 0.9454 - lr: 0.0010
Epoch 12/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2290 - val_loss: 0.19
94 - auc: 0.9452 - lr: 0.0010
Epoch 13/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2274 - val_loss: 0.20
40 - auc: 0.9493 - lr: 0.0010
Epoch 14/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2257 - val_loss: 0.19
45 - auc: 0.9477 - lr: 0.0010
Epoch 15/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2245 - val_loss: 0.21
```

```
20 - auc: 0.9426 - lr: 0.0010
Epoch 16/100
2344/2344 [==============================] - 10s 4ms/step - loss: 0.2258 - val_loss: 0.19
78 - auc: 0.9509 - lr: 0.0010
```

Part-6: Creating a Data pipeline for BERT Model

1. Download data from here

2. Read the csv file

3. Remove all the html tags

4. Now do tokenization [Part 3 as mentioned above]

    - Create tokens,mask array and segment array

5. Get Embeddings from BERT Model [Part 4 as mentioned above] , let it be X_test

    - Print the shape of output(X_test.shape).You should get (352,768)

6. Predit the output of X_test with the Neural network model which we trained earlier.

7. Print the occurences of class labels in the predicted output

</pre>

In [76]:

```
class BertVectorizer():

  def __init__(self, max_sequence_len, tokenizer, bert_model):
    super(BertVectorizer, self).__init__()
    self.max_sequence_len = max_seq_length
    self.tokenizer = tokenizer
    self.bert_model = bert_model


  def get_bert_vectors(self, X):

    X.map(lambda x: re.sub(r'<.*?>', '', x))

    tokens = X.map(self.tokenizer.tokenize)

    tokens = pad_sequences(tokens, self.max_sequence_len - 2,
                           value='[PAD]', dtype=object)

    tokens = self._complete_sequence(tokens)
    self.mask = (tokens != '[PAD]')
    self.segment = np.zeros(tokens.shape)
    self.tokens = self._convert_tokens_to_ids(tokens)

    return self.bert_model.predict([self.tokens, self.mask, self.tokens])


  def _complete_sequence(self, X):
    completed_sequences = []
    for row in X:
      completed_sequences.append(['[CLS]', *row, '[SEP]'])
    return np.array(completed_sequences)


  def _convert_tokens_to_ids(self, X):
    ids = []
    for row in X:
      ids.append(self.tokenizer.convert_tokens_to_ids(row))

    return np.array(ids)
```

In [77]:

```
def pipeline(X, max_sequence_len, tokenizer, bert_model, classifier):
  vectorizer = BertVectorizer(max_sequence_len, tokenizer, bert_model)
  vectors = vectorizer.get_bert_vectors(X)
  print(vectors.shape)
  return classifier.predict(vectors)
```

In [78]:

```
def classify_reviews(reviews):
  global max_seq_length
  global tokenizer
  global bert_model
  global review_classifier
  return pipeline(reviews, max_seq_length, tokenizer, bert_model, review_classifier)
```

In [79]:

```
X_test = pd.read_csv('./test.csv')
preds = classify_reviews(X_test['Text'])
```

```
(352, 768)
```

In [84]:

```
labels = [1 if x > 0.5 else 0 for x in preds]
sum(labels)
```

Out[84]:

```
322
```

In [85]:

```
print(labels)
```

```
[0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```