In this notebook, You will do amazon review classification with BERT.[Download data from this (https://www.kaggle.com/snap/amazon-fine-food-reviews/data

It contains 5 parts as below. Detailed instrctions are given in the each cell. please 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 explain what to write.

3. please return outputs in the same format what we asked. Eg. Don't return List 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 wr

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

Every Grader function has to return True.

In [1]:
```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount("/gdrive", force_remount=True).
/gdrive
```

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
from tqdm import tqdm
```

In [3]:
```
tf.test.gpu_device_name()
```

Out[3]:
```
'/device:GPU:0'
```

Grader function 1

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

Out[4]:
```
True
```

Part-1: Preprocessing

In [ ]:
```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv('/gdrive/My Drive/BERT_Assignment/NLP_Transfer_Learning/Reviews.csv')
#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 [ ]:
```
#get only 2 columns - Text, Score
#drop the NAN values
df_text =  reviews[['Text','Score']]
df_text.head(2)
df_text.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   Text    568454 non-null  object
 1   Score   568454 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 8.7+ MB
```

In [ ]:
```
#Checking Nan value is present
df_text.isnull().values.any()
```

Out[14]:
```
False
```

In [ ]:
```
reviews = df_text[(df_text["Score"] > 3)| (df_text["Score"] <= 2) ]
reviews.shape
```

Out[15]:
```
(525814, 2)
```

In [ ]:
```
#function - replace score greater than 3 with 1 else 0
def f(row):
    if row['Score'] > 3:
        val = 1
    else:
        val = 0
    return val

reviews['New_score'] = reviews.apply(f, axis=1)
```

In [ ]:
```
reviews.drop('Score',axis=1,inplace=True)
reviews.rename(columns={'New_score':'Score'},inplace=True)
```

Grader function 2

In [ ]:
```
def grader_reviews():
    temp_shape = (reviews.shape == (525814, 2)) and (reviews.Score.value_counts()[1]==443777)
    assert(temp_shape == True)
    return True
grader_reviews()
```

Out[18]:
```
True
```

In [ ]:
```
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 [ ]:
```
import re
```

In [ ]:
```
#remove HTML from the Text column and save in the Text column only

# first see sample HTML tag in text file
for enu,i in (enumerate(range(100000))):
  #print(i)
  try:
    #print(a)
    a = reviews.Text[i]
    #print(a)
    val = re.findall(r'>(.+?)<', a)
    if len(val) >0:
      print('Sample HTML tag in index location',enu)
      print(a)
      break
  except KeyError as e:
    continue
```

In [ ]:
```
reviews.reset_index(drop=True,inplace=True)
reviews.head(3)
reviews.shape
```

In [ ]:
```
clean_text = []
for i in tqdm(range(reviews.shape[0])):
  a = reviews.Text[i]
  val = re.sub(r'<(.+?)>',"", a)
  clean_text.append(val)
```

sample after removed HTML tag refer below snap

```
[119] reviews.Text[14]
```

```
'royal canine is a great product Jake loves it he is now 3 months<br /><br /><a href="http://www.amazon.com/gp/product/B001VIY9KW">Royal C
um Puppy 32 Formula, 30-Pound Bag</a> old and 35  pounds boxer puppy'
```

```
[120]  a = reviews.Text[14]
       val = re.sub(r'<(.+?)>',"", a)
       val
```

```
'royal canine is a great product Jake loves it he is now 3 monthsRoyal Canin Dry Dog Food, Medium Puppy 32 Formula, 30-Pound Bag old and 3
```

In [ ]:
```
reviews['new_text'] = clean_text
reviews.head(3)
```

In [ ]:
```
#remove old text and rename new_text as Text
reviews.drop('Text',axis=1,inplace=True)
reviews.rename(columns={'new_text':'Text'},inplace=True)
```

In [ ]:
```
#print head 5
reviews.head(5)
```

Out[20]:

	Text	Score	len
64117	The tea was of great quality and it tasted lik...	1	30
418112	My cat loves this. The pellets are nice and s...	1	31
357829	Great product. Does not completely get rid of ...	1	41
175872	This gum is my favorite! I would advise every...	1	27
178716	I also found out about this product because of...	1	22

In [ ]:
```
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 64117 to 19261
Data columns (total 3 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   Text    100000 non-null  object
 1   Score   100000 non-null  int64 
 2   len     100000 non-null  int64 
dtypes: int64(2), object(1)
memory usage: 3.1+ MB
```

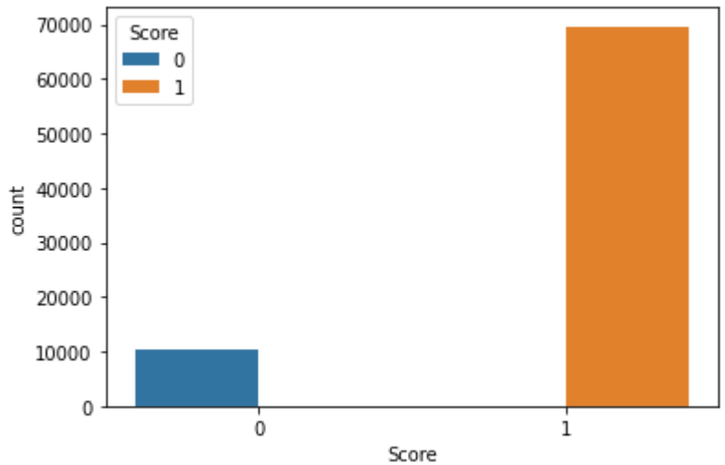
In [4]:
```
# Calling preprocessor
reviews = pd.read_csv('/gdrive/My Drive/BERT_Assignment/NLP_Transfer_Learning/preprocessed.csv')
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 3 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   Score   100000 non-null  int64 
 1   len     100000 non-null  int64 
 2   Text    100000 non-null  object
dtypes: int64(2), object(1)
memory usage: 2.3+ MB
```

In [5]:
```
#split the data into train and test data(20%) with Stratify sampling, random state 33,
from sklearn.model_selection import train_test_split
X = reviews.drop(['Score','len'],axis=1)
y = reviews['Score']
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify =y,random_state=33,test_size=0.2)
```

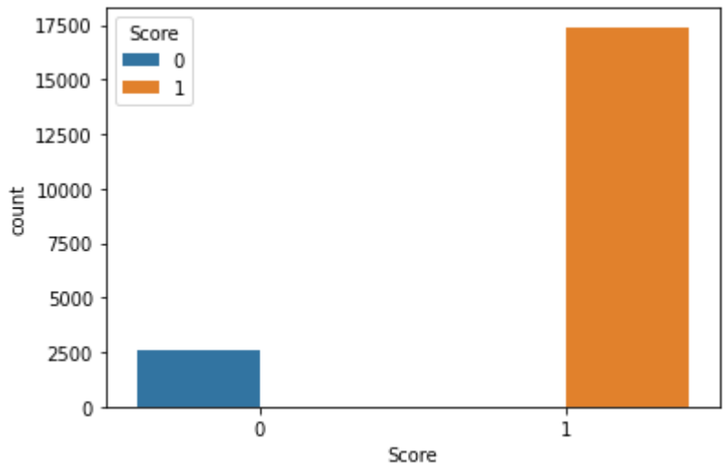
In [ ]:
```
#plot bar graphs of y_train and y_test
import seaborn as sns
sns.countplot( x=y_train,hue=y_train)
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6da3adb278>
```



In [ ]:
```
sns.countplot( x=y_test,hue=y_test)
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6df0435828>
```



In [ ]:
```
#saving to disk. if we need, we can load preprocessed data directly.
#reviews.to_csv('/gdrive/My Drive/BERT_Assignment/NLP_Transfer_Learning/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 (https://jalammar.github.io/illustrated-transformer/), BERT Paper (htt

s/1810.04805) and, This blog (https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/).

For this assignment, we are using BERT uncased Base model (https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1)
It uses L=12 hidden layers (i.e., Transformer blocks), a hidden size of H=768, and A=12 attention heads.

In [6]:
```
## 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
  # By refered above kaggle link, for text classification pooled_output is enough , representation of [batch_size, 768]
    # sequence_output of shape [batch_size, max_seq_length, 768]
bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_output)
```

In [7]:
```
bert_model.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
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 [8]:
```
bert_model.output
```

Out[8]:
```
<tf.Tensor 'keras_layer/StatefulPartitionedCall:0' shape=(None, 768) dtype=float32>
```

Part-3: Tokenization

In [97]:
```
#getting Vocab file
# refer - https://aihub.cloud.google.com/u/0/p/products%2F5f7e984e-f2e7-445f-9808-7ce751dcc1da
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
```

In [94]:
```
import os
!pip install sentencepiece
os.chdir('/gdrive/My Drive/BERT_Assignment/NLP_Transfer_Learning/')
from tokenization import FullTokenizer #- We have given tokenization.py file
```

```
Collecting sentencepiece
  Downloading https://files.pythonhosted.org/packages/d4/a4/d0a884c4300004a78cca907a6ff9a5e9fe4f090f5d95ab341c53d28cbc58/sente
p36m-manylinux1_x86_64.whl (https://files.pythonhosted.org/packages/d4/a4/d0a884c4300004a78cca907a6ff9a5e9fe4f090f5d95ab341c5
-0.1.91-cp36-cp36m-manylinux1_x86_64.whl) (1.1MB)
     |████████████████████████████████| 1.1MB 11.1MB/s eta 0:00:01
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.91
```

In [98]:
```
# 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

#install sentencepiece
from tokenization import FullTokenizer
tokenizer = FullTokenizer(vocab_file,do_lower_case)
```

Grader function 3

In [ ]:
```
#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[33]:
```
True
```

In [ ]:
```
X_train.values[5][0]
```

In [ ]:
```
# Create train and test tokens (X_train_tokens, X_test_tokens) from (X_train, X_test) using Tokenizer and


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

# 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 padding)

# 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_mask

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

# 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
```

In [ ]:
```
X_train.values[1][0]
```

In [62]:
```
def BERT_input(X_train):
  posistion_t = []
  mask_t = []
  segment_t = []
  for i in tqdm(range(len(X_train))):
    tokens = tokenizer.tokenize(X_train.values[i][0])
    tokens = tokens[0:(max_seq_length - 2)]
    tokens = ['[CLS]',*tokens,'[SEP]']
    tokenss = tokens.copy()
    if len(tokenss) < 55:
      #print('Count need to add', 55 - len(tokens))
      add_pad = 55 - len(tokenss)
      #print(add_pad)
      new_list = ['[PAD]']* add_pad
      tokenss.extend(new_list)

    posistion = np.array(tokenizer.convert_tokens_to_ids(tokenss))
    posistion_t.append(posistion)

    mask = np.array([1]*len(tokens)+ [0]*(max_seq_length - len(tokens)))

    mask_t.append(mask)

    segment =  np.array([0] *max_seq_length)
    segment_t.append(segment)
  return posistion_t,mask_t,segment_t


X_train_tokens,X_train_mask,X_train_segment = BERT_input(X_train)
X_test_tokens,X_test_mask,X_test_segment = BERT_input(X_test)
```

In [ ]:
```
X_train_tokens = np.array(X_train_tokens)
X_train_mask = np.array(X_train_mask)
X_train_segment = np.array(X_train_segment)
X_test_tokens = np.array(X_test_tokens)
X_test_mask = np.array(X_test_mask)
X_test_segment = np.array(X_test_segment)
```

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_le
# 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
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 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 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]
==================================================
```

In [9]:
```
import pickle
import os
os.chdir('/gdrive/My Drive/BERT_Assignment/NLP_Transfer_Learning/')
```

In [10]:
```
##save all your results to disk so that, no need to run all again.
#pickle.dump((X_train, X_train_tokens, X_train_mask, X_train_segment, y_train),open('/gdrive/My Drive/BERT_Assignment/NLP_Trar
#pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),open('/gdrive/My Drive/BERT_Assignment/NLP_Transfer_
```

In [11]:
```
#you can load from disk
X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.load(open("train_data.pkl", 'rb'))
X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(open("test_data.pkl", 'rb'))
```

Grader function 4

In [ ]:
```
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[37]:
```
True
```

Grader function 5

In [ ]:
```
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[38]:
```
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 [ ]:
```
bert_model.input
```

Out[39]:
```
[<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 [ ]:
```
bert_model.output
```

Out[40]:
```
<tf.Tensor 'keras_layer/StatefulPartitionedCall:0' shape=(None, 768) dtype=float32>
```

In [ ]:
```
# 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 [ ]:
```
# 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])
```

In [ ]:
```
##save all your results to disk so that, no need to run all again.
#pickle.dump((X_train_pooled_output, X_test_pooled_output),open('final_output.pkl','wb'))
```

In [21]:
```
X_train_pooled_output, X_test_pooled_output= pickle.load(open('final_output.pkl', 'rb'))
```

In [22]:
```
X_train_pooled_output.shape
```

Out[22]:
```
(80000, 768)
```

In [23]:
```
X_test_pooled_output.shape
```

Out[23]:
```
(20000, 768)
```

Grader function 6

In [ ]:
```
#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[43]:
```
True
```

In [ ]:
```
X_train_pooled_output.shape[1]
```

Out[81]:
```
768
```

In [ ]:

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 [12]:
```
##imports
from tensorflow.keras.models import Model
from tqdm import tqdm
from tensorflow.keras.layers import Conv1D,MaxPooling1D,Dense,Flatten,Input,Dropout,BatchNormalization
from tensorflow.keras.models import Model
import re
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, TensorBoard
from keras.models import Sequential
import datetime
import random as rn
from sklearn import metrics
import tensorflow as tf
```

KERAS OPTIMIZSATION

In [51]:
```
def auc_score(y_true, y_pred):

    auc_scoree =  tf.compat.v1.py_func(metrics.roc_auc_score, (y_true, y_pred), tf.double)
    return auc_scoree
```

In [52]:
```
filepath="/gdrive/My Drive/BERT_Assignment/NLP_Transfer_Learning/save_model/best_model-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_auc_score',  verbose=1, save_best_only=True, mode='max')
```

In [54]:
```

ACCURACY_THRESHOLD_test = 0.95
class myCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs={}):


      if(logs.get('val_auc_score') > ACCURACY_THRESHOLD_test) and (logs.get('auc_score') > ACCURACY_THRESHOLD_test) :
        print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD_test*100))
        self.model.stop_training = True



early_stop_auc_scores = myCallback()
```

In [55]:
```
# Early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=5, verbose=0, mode='auto',
    baseline=None, restore_best_weights=False
)
```

In [56]:
```
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/BERT_Assignment/NLP_Transfer_Learning/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=0, write_graph=True)
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [57]:
```
folder_name
```

Out[57]:
```
'20201009-073251'
```

In [58]:

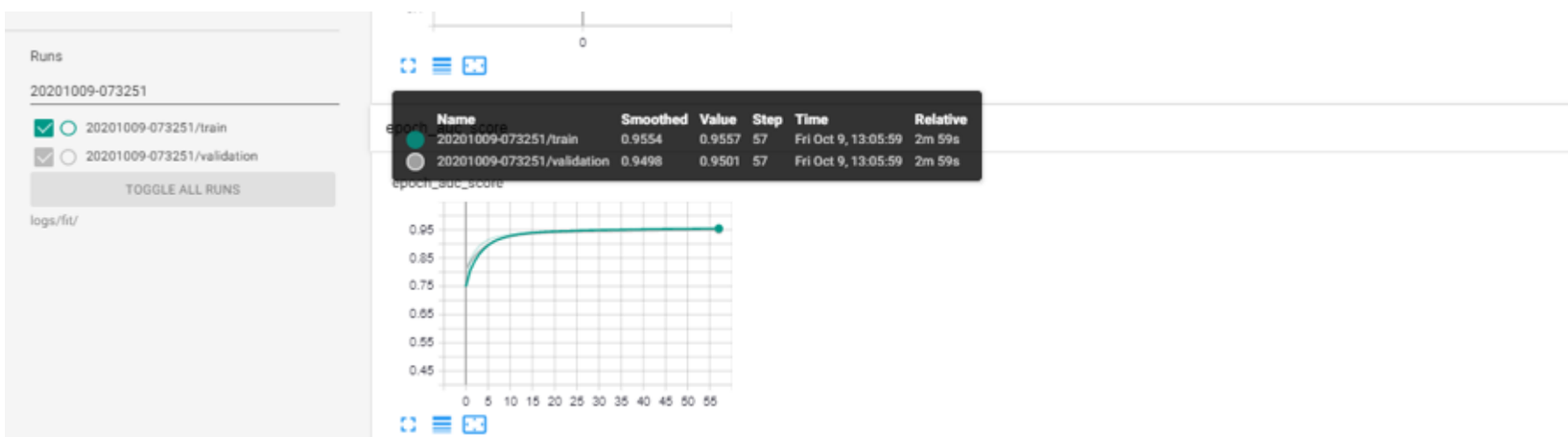
```
# Build model
def build_model(max_seq_length):
    os.environ['PYTHONHASHSEED'] = '0'
    tf.keras.backend.clear_session()
    rn.seed(0)

    bert_output = Input(shape=(max_seq_length,), name="input_ids")

    dense = Dense(256, activation='relu')(bert_output)
    pred = Dense(1, activation='sigmoid')(dense)

    model = tf.keras.models.Model(inputs=bert_output, outputs=pred)
    model.compile(loss='binary_crossentropy',
                  optimizer=tf.keras.optimizers.Adam(lr=2e-5),
                  metrics=[auc_score])
    return model



model = build_model(max_seq_length=X_train_pooled_output.shape[1])
```

In [59]:
```
model.fit(X_train_pooled_output, y_train, epochs=100,verbose=1,batch_size=128, validation_data=(X_test_pooled_output, y_test),
          callbacks =[checkpoint,tensorboard_callback,early_stop_auc_scores,early_stop])
```

```
t_model-45.h5
625/625 [==============================] - 3s 4ms/step - loss: 0.1751 - auc_score: 0.9530 - val_loss: 0.1813 - val_auc_score:
Epoch 46/100
612/625 [============================>.] - ETA: 0s - loss: 0.1743 - auc_score: 0.9540
Epoch 00046: val_auc_score improved from 0.94818 to 0.94846, saving model to /gdrive/My Drive/BERT_Assignment/NLP_Transfer_Lea
t_model-46.h5
625/625 [==============================] - 3s 4ms/step - loss: 0.1748 - auc_score: 0.9538 - val_loss: 0.1805 - val_auc_score:
Epoch 47/100
621/625 [============================>.] - ETA: 0s - loss: 0.1742 - auc_score: 0.9535
Epoch 00047: val_auc_score improved from 0.94846 to 0.94864, saving model to /gdrive/My Drive/BERT_Assignment/NLP_Transfer_Lea
t_model-47.h5
625/625 [==============================] - 3s 4ms/step - loss: 0.1742 - auc_score: 0.9534 - val_loss: 0.1816 - val_auc_score:
Epoch 48/100
622/625 [============================>.] - ETA: 0s - loss: 0.1743 - auc_score: 0.9540
Epoch 00048: val_auc_score improved from 0.94864 to 0.94869, saving model to /gdrive/My Drive/BERT_Assignment/NLP_Transfer_Lea
t_model-48.h5
625/625 [==============================] - 3s 4ms/step - loss: 0.1743 - auc_score: 0.9539 - val_loss: 0.1806 - val_auc_score:
Epoch 49/100
621/625 [============================>.] - ETA: 0s - loss: 0.1736 - auc_score: 0.9542
```

In [61]:
```
%tensorboard --logdir logs/fit/
```

Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.



Part-6: Creating a Data pipeline for BERT Model

1. Download data from here (https://drive.google.com/file/d/1QwjqTsqTX2vdy7fTmeXjxP3dq8IAVLpo/view?usp=sharing)
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

STEP - 2 Read_csv test file

In [128]:
```
# read test file
test_df = pd.read_csv('test.csv')
test_df.head(2)
```

Out[128]:

	Text
0	Just opened Greenies Joint Care (individually ...
1	This product rocks :) My mom was very happy w/...

STEP 3 - Remove all HTML tag

In [129]:
```
def clean_text(df):
  clean_text= []
  for i in tqdm(range(df.shape[0])):
    a = test_df.Text[i]
    val = re.sub(r'<(.+?)>',"", a)
    clean_text.append(val)
  return clean_text

test_df['new_text'] = clean_text(test_df)
```

```
100%|██████████| 352/352 [00:00<00:00, 65837.01it/s]
```

In [130]:
```
test_df.head(2)
```

Out[130]:

	Text	new_text
0	Just opened Greenies Joint Care (individually ...	Just opened Greenies Joint Care (individually ...
1	This product rocks :) My mom was very happy w/...	This product rocks :) My mom was very happy w/...

In [131]:
```
test_df.drop('Text',axis=1,inplace=True)
test_df.rename({'new_text':'Text'},inplace=True,axis=1)
```

In [132]:
```
test_df.head(2)
```

Out[132]:

	Text
0	Just opened Greenies Joint Care (individually ...
1	This product rocks :) My mom was very happy w/...

In [133]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 352 entries, 0 to 351
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Text    352 non-null    object
dtypes: object(1)
memory usage: 2.9+ KB
```

STEP 4 - TOKENIZATION

In [134]:
```
token,mask,segment=BERT_input(test_df)
```

```
100%|██████████| 352/352 [00:00<00:00, 1755.08it/s]
```

In [135]:
```
test_tokens = np.array(token)
test_mask = np.array(mask)
test_segment = np.array(segment)
```

STEP 5 - BERT MODEL

In [136]:
```
test_bert =bert_model.predict([test_tokens,test_mask,test_segment])
```

In [137]:
```
test_bert.shape
```

Out[137]:
```
(352, 768)
```

In [138]:
```
prediction = model.predict(test_bert)
```

In [139]:
```
prediction_df = pd.DataFrame(data=prediction,columns=['Prob_score'])
prediction_df.head(2)
```

Out[139]:

	Prob_score
0	0.140201
1	0.997037

In [140]:
```
def f(row):
    if row['Prob_score'] >= 0.5:
        val = 1
    else:
        val = 0
    return val

prediction_df['Result'] = prediction_df.apply(f, axis=1)
```

In [141]:
```
prediction_df.head(2)
```

Out[141]:

	Prob_score	Result
0	0.140201	0
1	0.997037	1

In [142]:
```
prediction_df['Result'].value_counts()
```

Out[142]:
```
1    304
0     48
Name: Result, dtype: int64
```

Total number of Postive review from 352 reviews - 304
Total number of Negative review from 352 reviews - 48

In [146]:
```
test_df['Predicted_label'] = prediction_df['Result']
```

In [148]:
```
# Sample for postive review
test_df[test_df['Predicted_label'] == 1].head(5)
```

Out[148]:

	Text	Predicted_label
1	This product rocks :) My mom was very happy w/...	1
2	The product was fine, but the cost of shipping...	1
3	I love this soup. It's great as part of a meal...	1
4	Getting ready to order again. These are great ...	1
7	If you need something to take with you to keep...	1

In [149]:
```
# Sample for Negative review
test_df[test_df['Predicted_label'] == 0].head(5)
```

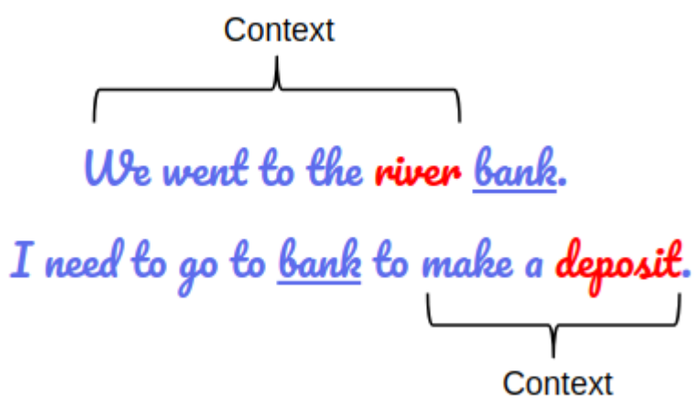
Out[149]:

	Text	Predicted_label
0	Just opened Greenies Joint Care (individually ...	0
5	These were delicious, but not wrapped as well ...	0
6	I will never again even CONSIDER a dog food wi...	0
12	I purchased this tea because I was told that i...	0
15	I searched for K-Cups and this product was lis...	0

Notes

Refer - https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/

1. What is BERT? - **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.
--> BERT is based on the Transformer architecture.
--> BERT is a “deeply bidirectional” model. Bidirectional means that BERT learns information from both the left and the ri
n’s context during the training phase.
Example



If we try to predict the nature of the word “bank” by only taking either the left or the right context, then we will be ma
t least one of the two given examples.

One way to deal with this is to consider both the left and the right context before making a prediction. That’s exactly wh
ill see later in the article how this is achieved.

From the Applied AI course content, BERT just kind of enhancement of Word2Vec
Word2Vec and GloVe - are embedding we using for most of text classifcation task, embedding for each word or character bu
er contextual relationship among word, but in BERT we going to consider those relation.

Example:
Another key limitation was that these models did not take the context of the word into account. Let’s take the above
“bank” example. The same word has different meanings in different contexts, right? However, an embedding like Word2Vec wil
ctor for “bank” in both the contexts.

That’s valuable information we are losing.

1. BERT’s Architecture for our task we used BERT base - 12 layers (transformer blocks), 12 attention heads, and 110 milli

1. Steps Follwed

Part 1 - Preprocessing
1. Clean the text from noise like HTML tag, change everything to lower to maintain consistency.
2. Split the data as train and test, startify as yes to maintain equal split among tha data using Target label

Part 2 - Creating BERT model
1. Need 3 inputs,
    i. Input of word as token
    ii. Masking - we added padding, Max word in our train data is 48 but we set our input as 55 so padding require here.
    iii. Segement - if single input text contains multiple sentence, then segment needed to represent, this contains only
tences per input

2. Call BERT model from hub.KerasLayer by passing three inputs to BERT model we use to get
    pooled_output - representation of [batch_size, 768] and **sequence_output [batch_size, max_seq_length, 768]** ,out task
so Pooled output is enough.

Part 3 - Tokenizsation

1. Initiated - vocab_file and do_lower_case - to get vocab file and change evrything to lower case
2. Create Train token and test token add '[CLS]' at start of the Tokens and '[SEP]' at the end of the tokens.
3. After step 2 completed , input is ready for BERT model.
    Note - to pass the input kindly change data type to array
4. To get Embedding BERT model , kindly do below steps and get the out of BERT model , then we can use this input for Neur
hine learning model like Logistic, Decision tree etc..

```
bert_model.predict([X_train_tokens,X_train_mask,X_train_segment])
```

5. As part of our task, we created simple Neural Network with 1 hidden unit.