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. 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 writing.

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

Every Grader function has to return True.

In [ ]:
```
#in this assignment you need two files reviews.csv and tokenization file
#you can use gdown module to import both the files in colab from Google drive
#the syntax is for gdown is !gdown --id file_id
#please run the below cell to import the required files
```

In [ ]:
```
# !gdown --id 1GsD8JlAc_0yJ-1151LNr6rLw83RRUPgt
# !gdown --id 13exfXiyiByluh1PfYK1EyZyizqxeCVG9
```

In [ ]:
```
#pip install seaborn
```

In [42]:
```
#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 [43]:
```
tf.test.gpu_device_name()
```

Out[43]:
```
'/device:GPU:0'
```

Grader function 1

In [44]:
```
def grader_tf_version():
    assert((tf.__version__)>'2')
    return True
grader_tf_version()
```

Out[44]:
```
True
```

Part-1: Preprocessing

In [45]:
```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv(r"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 [46]:

```
#get only 2 columns - Text, Score
#drop the NAN values

reviews = reviews[['Text','Score']]
reviews[pd.isnull(reviews).any(axis=1)]
reviews.dropna(inplace=True)
```

In [47]:

```
#if score> 3, set score = 1
#if score<=2, set score = 0
#if score == 3, remove the rows.
```

In [48]:

```
ind = reviews[reviews['Score']==3].index
reviews.drop(ind,axis=0,inplace=True)

reviews.loc[reviews['Score']<=2, "Score"] = 0

reviews.loc[reviews["Score"] >3, "Score"] = 1
```

In [49]:

```
reviews.Score.value_counts()
```

Out[49]:

```
1    443777
0     82037
Name: Score, dtype: int64
```

Grader function 2

In [50]:

```
reviews.shape
```

Out[50]:

```
(525814, 2)
```

In [51]:

```
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[51]:

```
True
```

In [52]:

```
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 [53]:

```
#remove HTML from the Text column and save in the Text column only

#fil = '<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});'
#reviews['Text'] = reviews['Text'].str.replace(r'<[^<>]*>', '', regex=True)
reviews['Text'] = reviews['Text'].str.replace(r'<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});', '', regex=Tru
```

In [54]:

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

Out[54]:

	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 [55]:

```
X = reviews[['Text','len']]
y = reviews['Score']
```

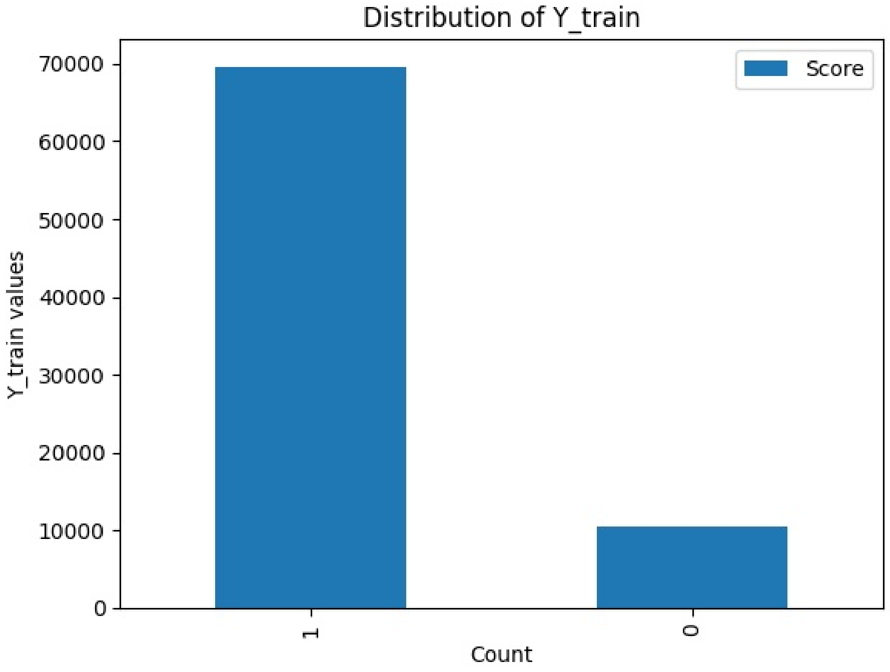
In [56]:

```
#split the data into train and test data(20%) with Stratify sampling, random state 33,

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     stratify=y,
                                                     test_size=0.20,
                                                    random_state=33,
                                                    shuffle=True)
```

In [ ]:

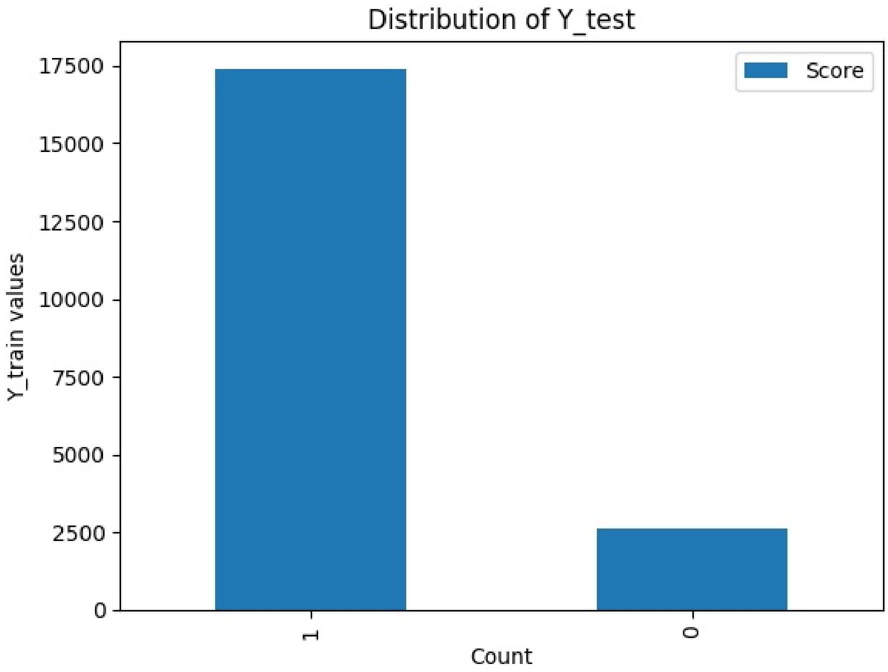
In [16]:

```
#plot bar graphs of y_train and y_test

import seaborn as sns
import matplotlib.pyplot as plt
y_train_ = pd.DataFrame(y_train.value_counts())
y_train_.plot.bar()
plt.xlabel("Count")
plt.ylabel("Y_train values")
plt.title("Distribution of Y_train")
plt.show()
```



In [17]:

```
y_test_ = pd.DataFrame(y_test.value_counts())
y_test_.plot.bar()
plt.xlabel("Count")
plt.ylabel("Y_train values")
plt.title("Distribution of Y_test")
plt.show()
```



In [18]:
```
reviews
```

Out[18]:

	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
...	...	...	...
336657	Using this coffee and a stove top espresso mak...	1	39
498034	THE TASTE OF THIS M&M IS THE BEST. I USED IT I...	1	28
357766	Excellent Tea. I enjoy a cup every now and the...	1	21
326811	These oatmeal cookies have a great spice taste...	1	23
19261	This is the best coffee ever! I will never dri...	1	28

100000 rows × 3 columns

In [19]:
```
#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 [19]:
```
import os
## 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
#"https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4"
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/2", trainable=False)
#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 [20]:

```
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 [21]:

```
bert_model.output
```

Out[21]:

```
<KerasTensor: shape=(None, 768) dtype=float32 (created by layer 'keras_layer')>
```

Part-3: Tokenization

In [22]:

```
#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 [23]:

```
import sys
sys.path.insert(0, "C:/Users/darsh/Downloads/Srujan/NLP_Transfer_learning_assignment/tokenization.py")
#pip install sentencepiece
```

In [24]:

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

In [ ]:

In [26]:

```
# 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
```

In [25]:

```
# if you are getting error for sentencepiece module you can install it using below command while running this c
#!pip install sentencepiece
tokenizer=tokenization.FullTokenizer(vocab_file,do_lower_case )
```

Grader function 3

In [32]:

```
#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[32]:

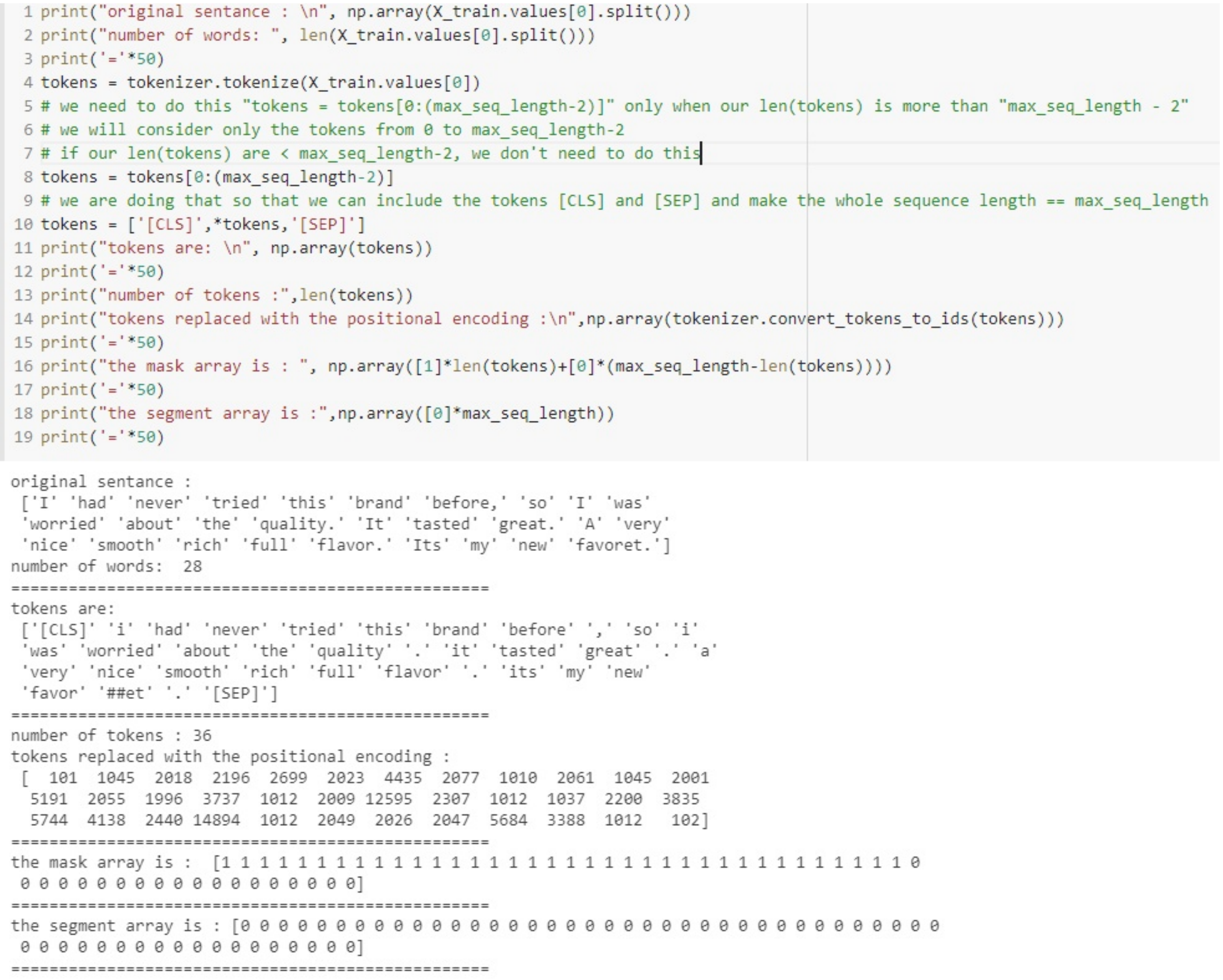
```
True
```

In [33]:

```
# 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

# 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 [ ]:

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]
==================================================
```

train tokenization

In [34]:

```python
from tqdm import tqdm
X_train_tokens = list()
X_train_mask = list()
X_train_segment = list()
max_seq_length = 55
for i in tqdm(range(X_train.shape[0])):
    tokens = tokenizer.tokenize(X_train['Text'].values[i])
    tokens=tokens[0:(max_seq_length-2)]
    tokens=['[CLS]',*tokens,'[SEP]']
    if len(tokens)<max_seq_length:
        aa = ['[PAD]'] * (max_seq_length-len(tokens))
        tokens_=[*tokens] + aa
        X_train_mask.append(([1]*len(tokens)+[0]*(tokens_.count('[PAD]'))))
        tokens = tokenizer.convert_tokens_to_ids(tokens_)
```

```
            #print("<55",len(tokens_),tokens_.count('[PAD]'),tokens_,'\n')
        else:
            #print('>=55',len(tokens))
            X_train_mask.append(([1]*len(tokens)))
            tokens = tokenizer.convert_tokens_to_ids(tokens)

        X_train_tokens.append(tokens)
        seg = [0 for i in range(len(tokens))]
        X_train_segment.append(seg)

    print(np.array(X_train_tokens).shape,np.array(X_train_mask).shape,np.array(X_train_mask).shape)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 80000/80000 [00:25<00:00, 3170
.77it/s]
(80000, 55) (80000, 55) (80000, 55)
```

In [35]:
```
X_train_tokens = np.array(X_train_tokens)
X_train_mask = np.array(X_train_mask)
X_train_segment = np.array(X_train_segment)
```

In [ ]:

test tokenization

In [36]:
```
from tqdm import tqdm
X_test_tokens = list()
X_test_mask = list()
X_test_segment = list()
#max_seq_length = 55
for i in tqdm(range(X_test.shape[0])):
    tokens = tokenizer.tokenize(X_test['Text'].values[i])
    tokens=tokens[0:(max_seq_length-2)]
    tokens=['[CLS]',*tokens,'[SEP]']
    if len(tokens)<max_seq_length:
        aa = ['[PAD]'] * (max_seq_length-len(tokens))
        tokens_=[*tokens] + aa
        X_test_mask.append(([1]*len(tokens)+[0]*(tokens_.count('[PAD]'))))
        tokens = tokenizer.convert_tokens_to_ids(tokens_)
        #print("<55",len(tokens_),tokens_.count('[PAD]'),tokens_,'\n')
    else:
        #print('>=55',len(tokens))
        X_test_mask.append(([1]*len(tokens)))
        tokens = tokenizer.convert_tokens_to_ids(tokens)

    X_test_tokens.append(tokens)
    seg = [0 for i in range(len(tokens))]
    X_test_segment.append(seg)

print(np.array(X_test_tokens).shape,np.array(X_test_mask).shape,np.array(X_test_segment).shape)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 20000/20000 [00:06<00:00, 2909
.37it/s]
(20000, 55) (20000, 55) (20000, 55)
```

In [ ]:

In [37]:
```
X_test_tokens = np.array(X_test_tokens)
X_test_mask = np.array(X_test_mask)
X_test_segment = np.array(X_test_segment)
```

In [ ]:

In [ ]:

In [38]:
```
import pickle
```

In [39]:
```
##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('train_data.pkl','wb'))
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),open('test_data.pkl','wb'))
```

In [36]:
```
#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 [40]:
```
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)
        #print('temp_shapes',temp_shapes)


        segment_temp = not np.any(X_train_segment)
        #print('segment_temp',segment_temp)


        mask_temp = np.sum(X_train_mask==0) == np.sum(X_train_tokens==0)
        #print('mask_temp',mask_temp)


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

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

        out = temp_shapes and segment_temp and mask_temp and no_cls and no_sep
        #print('out',out)

    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[40]: True

Grader function 5

In [41]:

```
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[41]: 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 [42]:

```
bert_model.input
```

Out[42]:

```
[<KerasTensor: shape=(None, 55) dtype=int32 (created by layer 'input_word_ids')>,
 <KerasTensor: shape=(None, 55) dtype=int32 (created by layer 'input_mask')>,
 <KerasTensor: shape=(None, 55) dtype=int32 (created by layer 'segment_ids')>]
```

In [43]:

```
bert_model.output
```

Out[43]:

```
<KerasTensor: shape=(None, 768) dtype=float32 (created by layer 'keras_layer')>
```

In [44]:
```
# get the train output, BERT model will give one output so save in
# X_train_pooled_output
#this cell will take some time to execute, make sure thay you have stable internet connection
X_train_pooled_output=bert_model.predict([X_train_tokens,X_train_mask,X_train_segment])
```

In [45]:
```
# 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 [46]:
```
##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 [26]:
```
import pickle
```

In [27]:
```
X_train_pooled_output, X_test_pooled_output= pickle.load(open('final_output.pkl', 'rb'))
```

In [ ]:
```

```

Grader function 6

In [49]:
```
#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[49]:
```
True
```

Part-5: Training a NN with 768 features

Create a NN and train the NN.
1. **You have to use AUC as metric. Do not use tf.keras.metrics.AUC**
 You have to write custom code for AUC and print it at the end of each epoch
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 [35]:
```
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout, LSTM, Conv1D, Flatten, Conv2D, Embedding
from tensorflow.keras.models import Model,Sequential
```

In [36]:
```
##create an Neural Network and train your model on X_train_pooled_output and y_train
# you can start as follows
input_layer=Input(shape=(X_train_pooled_output.shape[1],))
```

In [37]:
```
import os
import datetime
import tensorflow as tf
log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)
```

In [38]:
```
%load_ext tensorboard
```

In [39]:
```
checkpoint_path = "C:/Users/darsh/Downloads/Srujan/NLP_Transfer_learning_assignment/cpp1-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
```

In [57]:
```
xtrain=X_train_pooled_output.reshape(-1,768,1)
xtest=X_test_pooled_output.reshape(-1,768,1)

ytrain = tf.keras.utils.to_categorical(y_train.values, num_classes=2)
ytest= tf.keras.utils.to_categorical(y_test.values, num_classes=2)
```

In [ ]:
```

```

In [58]:

```
model1 = Sequential()
input_layer = Input(shape=(xtrain.shape[1],xtest.shape[2]))
model1.add(input_layer)
model1.add(Conv1D(128, 3, activation='relu',padding='same')) # input_shape = (768,1)
model1.add(Conv1D(256, 3,  activation='relu',padding='same'))
model1.add(Conv1D(256, 3,  activation='relu',padding='same'))
# flat
model1.add(Flatten())

model1.add(Dense(2, activation='softmax'))
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d (Conv1D)              (None, 768, 128)          512       
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 768, 256)          98560     
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 768, 256)          196864    
_________________________________________________________________
flatten (Flatten)            (None, 196608)            0         
_________________________________________________________________
dense (Dense)                (None, 2)                 393218    
=================================================================
Total params: 689,154
Trainable params: 689,154
Non-trainable params: 0
_________________________________________________________________
```

In [59]:

```
import tensorflow as tf
from sklearn.metrics import roc_auc_score

def auc(y_true, y_pred):
    return tf.numpy_function(roc_auc_score, (y_true, y_pred), tf.double)
```

In [ ]:

In [ ]:

In [60]:

```
from tensorflow.keras.optimizers import SGD,Adam
opt = Adam()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint('./NLP_TL_Best_model.h5', save_weights_only=False, save_best_only=True, \
                                       mode='min', monitor='loss',verbose=1,save_freq='epoch'),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', patience=1,mode='min',verbose=1),
    tensorboard_callback
]
model1.compile(
    loss='categorical_crossentropy',
    optimizer="adam",
    metrics=["accuracy",auc]
)
```

In [ ]:

In [61]:

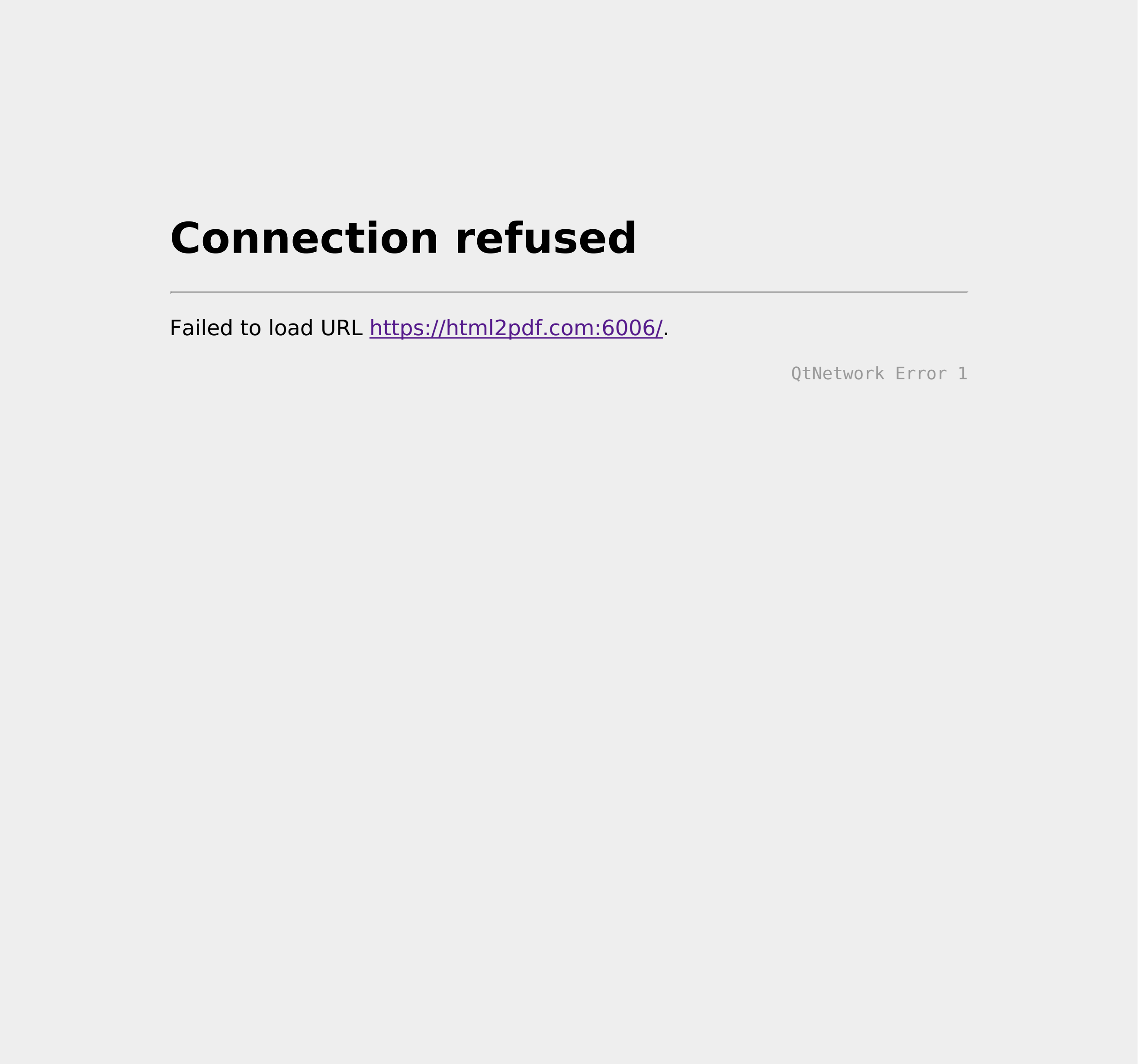
```
model1.fit(xtrain,ytrain,batch_size=128,
           validation_data=(xtest,ytest),callbacks=callbacks,epochs=5,verbose=1)
```

```
Epoch 1/5
625/625 [==============================] - 66s 88ms/step - loss: 0.2486 - accuracy: 0.9007 - auc: 0.8968 - val_
loss: 0.1938 - val_accuracy: 0.9201 - val_auc: 0.9412

Epoch 00001: loss improved from inf to 0.24860, saving model to .\NLP_TL_Best_model.h5
Epoch 2/5
625/625 [==============================] - 70s 113ms/step - loss: 0.1925 - accuracy: 0.9222 - auc: 0.9490 - val
_loss: 0.1861 - val_accuracy: 0.9283 - val_auc: 0.9459

Epoch 00002: loss improved from 0.24860 to 0.19253, saving model to .\NLP_TL_Best_model.h5
Epoch 3/5
625/625 [==============================] - 67s 108ms/step - loss: 0.1866 - accuracy: 0.9242 - auc: 0.9529 - val
_loss: 0.1868 - val_accuracy: 0.9262 - val_auc: 0.9487

Epoch 00003: loss improved from 0.19253 to 0.18660, saving model to .\NLP_TL_Best_model.h5
Epoch 4/5
625/625 [==============================] - 67s 107ms/step - loss: 0.1790 - accuracy: 0.9274 - auc: 0.9553 - val
_loss: 0.1884 - val_accuracy: 0.9255 - val_auc: 0.9467

Epoch 00004: loss improved from 0.18660 to 0.17902, saving model to .\NLP_TL_Best_model.h5
Epoch 5/5
625/625 [==============================] - 67s 107ms/step - loss: 0.1781 - accuracy: 0.9286 - auc: 0.9570 - val
_loss: 0.1760 - val_accuracy: 0.9318 - val_auc: 0.9506

Epoch 00005: loss improved from 0.17902 to 0.17806, saving model to .\NLP_TL_Best_model.h5
```

Out[61]:
```
<keras.callbacks.History at 0x225bedafee0>
```

In [ ]:

In [62]:
```
%tensorboard --logdir "C:/Users/darsh/Downloads/Srujan/NLP_Transfer_learning_assignment/logs/fits/20221201-0422
```



Connection refused

Failed to load URL https://html2pdf.com:6006/.

QtNetwork Error 1

In [ ]:

In [ ]:

In [ ]:

```

```

Part-6: Creating a Data pipeline for BERT Model

1. Pipeline is a way to codify and automate the workflow.
2. Download the test.csv file from here here

In [57]:

```
#there is an alterante way to load files from Google drive directly to your Colab session
# you can use gdown module to import the files as follows
#for example for test.csv you can write your code as !gdown --id file_id (remove the # from next line and run i
```

In [ ]:

```

```

In [63]:

```
#read the csv file
test_df= pd.read_csv('test.csv')
```

1. You have to write a function that takes the test_df,trained model and the required parameters as input.
2. Perform all the preproceesing steps inside the function.
- Remove all the html tags
- Now do tokenization [Part 3 as mentioned above]
- Create tokens,mask array and segment array
- 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)
3. Predit the output of X_test with the neural network model which we trained earlier.

4. Return the occurences of class labels from the function.
The output should be the count of datapoints classified as 1 or 0.

In [64]:

```
test_df['Text'] = test_df['Text'].str.replace(r'<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});', '', regex=Tru
```

In [71]:

```
from tqdm import tqdm
Xtest_tokens = list()
Xtest_mask = list()
Xtest_segment = list()
#max_seq_length = 55
for i in tqdm(range(test_df.shape[0])):
    tokens = tokenizer.tokenize(test_df['Text'].values[i])
    tokens=tokens[0:(max_seq_length-2)]
    tokens=['[CLS]',*tokens,'[SEP]']
    if len(tokens)<max_seq_length:
        aa = ['[PAD]'] * (max_seq_length-len(tokens))
        tokens_=[*tokens] + aa
        Xtest_mask.append(([1]*len(tokens)+[0]*(tokens_.count('[PAD]'))))
        tokens = tokenizer.convert_tokens_to_ids(tokens_)
        #print("<55",len(tokens_),tokens_.count('[PAD]'),tokens_,'\n')
    else:
        #print('>=55',len(tokens))
        Xtest_mask.append(([1]*len(tokens)))
        tokens = tokenizer.convert_tokens_to_ids(tokens)

    Xtest_tokens.append(tokens)
    seg = [0 for i in range(len(tokens))]
    Xtest_segment.append(seg)

#print(np.array(Xtest_tokens).shape,np.array(Xtest_mask).shape,np.array(Xtest_segment).shape)
Xtest_tokens = np.array(Xtest_tokens)
Xtest_mask = np.array(Xtest_mask)
Xtest_segment = np.array(Xtest_segment)
X_test_=bert_model.predict([Xtest_tokens,Xtest_mask,Xtest_segment])
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 352/352 [00:00<00:00, 2810
.09it/s]
```

In [72]:

```
X_test_.shape
```

Out[72]:

```
(352, 768)
```

In [73]:

```
pred = (model1.predict(X_test_.reshape(-1,768,1)))
```

In [76]:

```
out = list()
for i in range(pred.shape[0]):
    out.append(np.argmax(pred[i]))
```

In [77]:
```
out.count(0),out.count(1)
```

Out[77]:
```
(29, 323)
```

In [65]:
```
def data_pipeline(test_df,tokenizer,bert_model,model1):
    test_df['Text'] = test_df['Text'].str.replace(r'<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});', '', regex
    from tqdm import tqdm
    Xtest_tokens = list()
    Xtest_mask = list()
    Xtest_segment = list()
    #max_seq_length = 55
    for i in tqdm(range(test_df.shape[0])):
        tokens = tokenizer.tokenize(test_df['Text'].values[i])
        tokens=tokens[0:(max_seq_length-2)]
        tokens=['[CLS]',*tokens,'[SEP]']
        if len(tokens)<max_seq_length:
            aa = ['[PAD]'] * (max_seq_length-len(tokens))
            tokens_=[*tokens] + aa
            Xtest_mask.append(([1]*len(tokens)+[0]*(tokens_.count('[PAD]'))))
            tokens = tokenizer.convert_tokens_to_ids(tokens_)
            #print("<55",len(tokens_),tokens_.count('[PAD]'),tokens_,'\n')
        else:
            #print('>=55',len(tokens))
            Xtest_mask.append(([1]*len(tokens)))
            tokens = tokenizer.convert_tokens_to_ids(tokens)

        Xtest_tokens.append(tokens)
        seg = [0 for i in range(len(tokens))]
        Xtest_segment.append(seg)

    #print(np.array(Xtest_tokens).shape,np.array(Xtest_mask).shape,np.array(Xtest_segment).shape)
    Xtest_tokens = np.array(Xtest_tokens)
    Xtest_mask = np.array(Xtest_mask)
    Xtest_segment = np.array(Xtest_segment)
    X_test_=bert_model.predict([Xtest_tokens,Xtest_mask,Xtest_segment])

    pred = (model1.predict(X_test_.reshape(-1,768,1)))
    out = list()
    for i in range(pred.shape[0]):
        out.append(np.argmax(pred[i]))
    return out.count(0),out.count(1)
```

In [ ]:
```

```

In [67]:
```
negative_class,positive_class = data_pipeline(test_df,tokenizer,bert_model,model1)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 352/352 [00:00<00:00, 1691
.66it/s]
```

In [68]:
```
negative_class,positive_class
```

Out[68]:
```
(39, 313)
```

Please write your observations at the end of notebook and explain each and every step you followed in solving this assignment.

```
1. Preprocessing:

    a. Downloaded the amazon reviews data and only Text, Score features were considered.
    b. Reviews which have a score greater than 3 is considered as a positive class or
class_label=1
    c. Reviews that have a score less than or equal to 2 are considered as a negative class or
class_label=0
    d. Reviews that have a score equal to 3 are dropped
    e. From this new dataset with features = [Text, Score, Class_label], a random sample of
100000 reviews are selected
       whose word_length is less than 50.

2. Creating a BERT model from the Tensorflow HUB:

    a. We have created this model to get word embeddings that are used in the model.
    b. BERT model takes 3-inputs i.e [input_word_ids, Segment_ids,input_mask]

3. Tokenization:

    a. To generate these inputs, we use the tokenization.py file and generate these 3-inputs
    b. Preprocessing of Text like creating input_ids,input_mask,segment_ids and padding also
```

```
applied to obtain
      Text of size equals max_sequence_length. This is done because we can send a batch of
rows at a time.


4. Getting the pre-trained embedding Vector for a given review from the BERT.
    a. Now all these 3 inputs are sent to the BERT model to get the word embeddings of size
768 dimensions for all rows
       in data.


5. Using the embedding data apply NN and classify the reviews:
    a. Using these word embeddings from the BERT model, a neural network with 3-Conv1D layers
and a Dense layer is
       created.
    b. The input layer for this neural network is modified
    c. Train and test datasets were created and class_label is converted to categorical type.
    d. Adam optimizer, custom AUC metric, categorical_crossentropy as a loss were considered
for the neural network
       model
    e. Model has been trained for 5 epochs with batch_size=128.


6. Creating a Data Pipeline for BERT Model:
    a. created a data pipeline that does all the above steps and returns the counts of each
label.
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js