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 r

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 explai

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

Saved successfully!

ıks that we are given so that you will learn the conce

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

Every Grader function has to return True.

```
#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
```

```
!gdown --id 1GsD8JlAc_0yJ-1151LNr6rLw83RRUPgt
!gdown --id 13exfXiyiByluh1PfYK1EyZyizqxeCVG9
```

```
Downloading...
From: https://drive.google.com/uc?id=1GsD8JlAc_0yJ-1151LNr6rLw83RRUPgt
To: /content/Reviews.csv
100% 301M/301M [00:01<00:00, 185MB/s]
Downloading...
From: https://drive.google.com/uc?id=13exfXiyiByluh1PfYK1EyZyizqxeCVG9
To: /content/tokenization.py
100% 17.3k/17.3k [00:00<00:00, 26.2MB/s]
```

```
#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
```

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

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

Grader function 1

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

```
True
```

Part-1: Preprocessing

Saved successfully!

```
#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
```

```
#get only 2 columns - Text, Score
col_drop = ['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator', 'Helpfulne
reviews = reviews.drop(col_drop, axis=1)
```

```
#drop the NAN values
reviews = reviews.dropna()
```

```
reviews['Score'].isna().sum(), reviews['Text'].isna().sum()
```

```
(0, 0)
```

```
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 568454 entries, 0 to 568453
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   Score   568454 non-null  int64 
 1   Text    568454 non-null  object
dtypes: int64(1), object(1)
memory usage: 13.0+ MB
```

```
reviews.describe()
```

	Score
count	568454.000000
min	1.000000
25%	4.000000
50%	5.000000
75%	5.000000
max	5.000000

Saved successfully!

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

```
#if score == 3, remove the rows.
reviews.drop(reviews[reviews['Score'] == 3].index, inplace = True)
```

```
#if score> 3, set score = 1
#if score<=2, set score = 0
reviews['Score'] = (reviews['Score'] > 3).astype(int)
```

```
reviews.head()
```

	Score	Text
0	1	I have bought several of the Vitality canned d...
1	0	Product arrived labeled as Jumbo Salted Peanut...
2	1	This is a confection that has been around a fe...
3	0	If you are looking for the secret ingredient i...
4	1	Great taffy at a great price. There was a wid...

```
reviews.tail()
```

	Score	Text
568449	1	Great for sesame chicken..this is a good if no...
568450	0	I'm disappointed with the flavor. The chocolat...
568451	1	These stars are small, so you can give 10-15 o...
568452	1	These are the BEST treats for training and rew...
568453	1	I am very satisfied ,product is as advertised,...

```
reviews.info()
```

Saved successfully!

```
rame'>
o 568453
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Score   525814 non-null  int64
 1   Text    525814 non-null  object
dtypes: int64(1), object(1)
memory usage: 12.0+ MB
```

Grader function 2

```
def grader_reviews():
    temp_shape = (reviews.shape == (525814, 2)) and (reviews.Score.value_counts()[1]==4437
    assert(temp_shape == True)
    return True
grader_reviews()
```

```
True
```

```
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)
```

```
reviews.info()
```

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

```
#remove HTML from the Text column and save in the Text column only
import re
def remove_tags(string):
    result = re.sub('<.*?>','',string)
    return result
reviews['Text']=reviews['Text'].apply(lambda cw : remove_tags(cw))
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 64117 to 19261
Data columns (total 3 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   Score   100000 non-null  int64 
```

Saved successfully!

```
memory usage: 3.1+ MB
```

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

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

```
X = pd.DataFrame()
X['Text'] = reviews['Text']
X['len'] = reviews['len']
y = pd.DataFrame()
y['Score'] = reviews['Score']
```

```
X.shape, y.shape
```

```
((100000, 2), (100000, 1))
```

```
#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, test_size=0.20, random_state=33,
```

```
(X_train.shape, y_train.shape), (X_test.shape, y_test.shape)
```

```
(((80000, 2), (80000, 1)), ((20000, 2), (20000, 1)))
```

```
y_train = np.squeeze(y_train)
y_test = np.squeeze(y_test)
```

```
y_train.shape, y_test.shape
```

```
((80000,), (20000,))
```

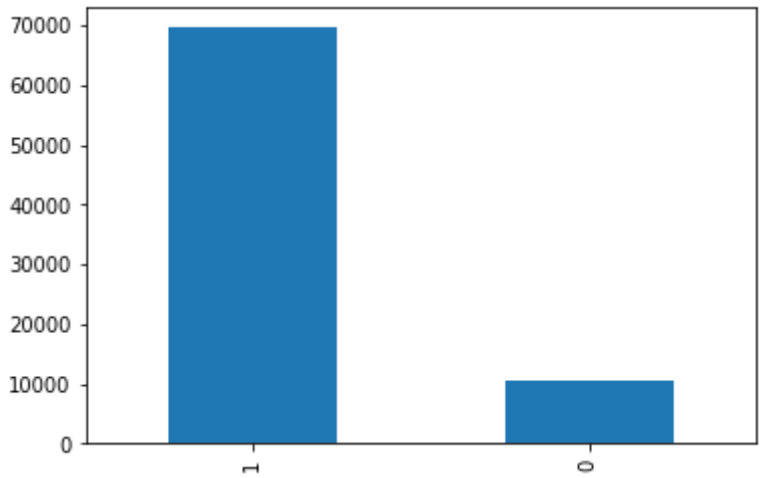
```
y_train.head(5)
```

```
33523     1
10855     1
390364    1
53902     1
```



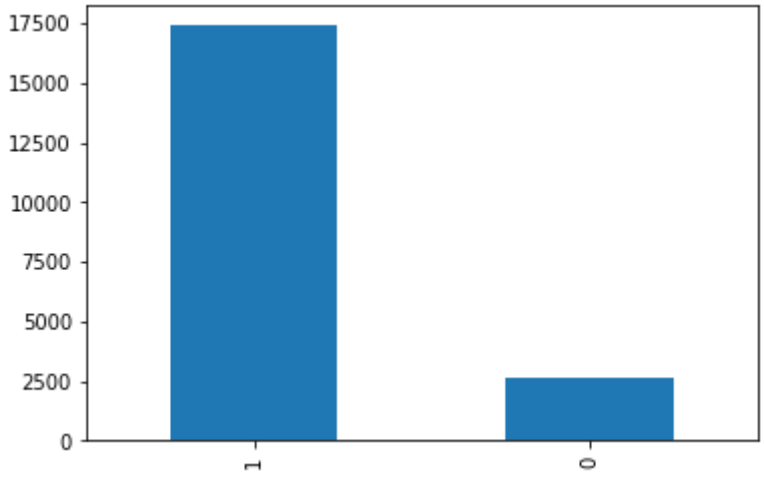
```
#plot bar graphs of y_train and y_test

#Bar Graph [y_train]

import matplotlib.pyplot as plt
y_train.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a0672a350>
```




```
#Bar Graph [y_test]
y_test.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09ff936650>
```




```
#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 BER
we will strongly recommend you to read Transformers, BERT Paper and, This blog.

Saved successfully!

BERT uncased Base model.
It uses L=12 hidden layers (i.e., Transformer blocks), a hidden size of H=768, and A=12

```
## 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 chang
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="inpu

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

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

#bert layer
```

```
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/
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
bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_output
```

```
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),        109482241   ['input_word_ids[0][
                                 (None, 55, 768)]                  'input_mask[0][0]',
                                                                   'segment_ids[0][0]
==========================================================================================
Total params: 109,482,241
                                                                  41
__________________________________________________________________________________________
```



```
bert_model.output
```

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

Part-3: Tokenization

```
#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()
```

```
!pip3 install tensorflow_text>=2.0.0rc0
!pip3 install sentencepiece
!pip3 install tf_sentencepiece
```

```
Collecting sentencepiece
  Downloading sentencepiece-0.1.96-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86
     |████████████████████████████████| 1.2 MB 5.2 MB/s
Installing collected packages: sentencepiece
```

```
Successfully installed sentencepiece-0.1.96
Collecting tf_sentencepiece
  Downloading tf_sentencepiece-0.1.90-py2.py3-none-manylinux1_x86_64.whl (2.1 MB)
     |████████████████████████████████| 2.1 MB 5.2 MB/s
Installing collected packages: tf-sentencepiece
Successfully installed tf-sentencepiece-0.1.90
```

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


# 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=vocab_file, do_lower_case=do_lower_case)
```

Grader function 3

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

```
 True
```



```
# Create train and test tokens (X_train_tokens, X_test_tokens) from (X_train, X_test) usin

# 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 (N

# if it is less than 55, add '[PAD]' token else truncate the tokens length.(similar to pad

# 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_ma

# Create a segment input for train and test. We are using only one sentence so all zeros.

# 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
```

```
#Checking the count of reviews of length greater than 50
see_train = [1 for i in X_train['len'] if i>50]
see_test = [1 for i in X_test['len'] if i>50]
len(see_train), len(see_test)
```

```
(0, 0)
```

```
def builder_tokenize(data_row):
    #print("original sentance : \n", np.array(X_train.values[1][0].split()))
    #print("number of words: ", len(X_train.values[1][0].split()))
    #print('='*50)
    tokens = tokenizer.tokenize(data_row)
    tokens = tokens[0:(max_seq_length-2)]
    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: ", np.array(tokenizer.convert_to
    #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)

    mask_1 = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
                                                    gth)
                                                    vert_tokens_to_ids(tokens)+[0]*(max_seq_length-len(tok
    #print(len(mask_1),len(seg_1), len(token_1))
    return token_1, mask_1, seg_1
```



```
X_train.shape
```

```
(80000, 2)
```

```
#Applying builder_tokenize function on X_train
%%time
from tqdm import tqdm
X_train_tokens = []
X_train_mask = []
X_train_segment = []
iter_num = len(X_train)
for row_num in tqdm(range(iter_num)):
  row = X_train.values[row_num][0]
  token_1, mask_1, seg_1 = builder_tokenize(row)
  X_train_tokens.append(token_1)
  X_train_mask.append(mask_1)
  X_train_segment.append(seg_1)

X_train_tokens = np.array(X_train_tokens)
X_train_mask = np.array(X_train_mask)
X_train_segment = np.array(X_train_segment)
```

```
100%|██████████| 80000/80000 [14:44<00:00, 90.42it/s]
CPU times: user 14min 26s, sys: 23.4 s, total: 14min 49s
Wall time: 14min 44s
```

```
print(type(X_train_tokens), type(X_train_mask), type(X_train_segment))
print(X_train_tokens.shape, X_train_mask.shape, X_train_segment.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
(80000, 55) (80000, 55) (80000, 55)
```

```
#Applying builder_tokenize function on X_test:
```

```
%%time
X_test_tokens = []
X_test_mask = []
X_test_segment = []
iter_num = len(X_test)
for row_num in tqdm(range(iter_num)):
  row = X_test.values[row_num][0]
  token_1, mask_1, seg_1 = builder_tokenize(row)
  X_test_tokens.append(token_1)
  X_test_mask.append(mask_1)
  X_test_segment.append(seg_1)

X_test_tokens = np.array(X_test_tokens)
X_test_mask = np.array(X_test_mask)
```

Saved successfully!

```
ment)
```

```
100%|██████████| 20000/20000 [00:35<00:00, 558.52it/s]CPU times: user 35.7 s, sys: 34
Wall time: 35.9 s
```

```
print(type(X_test_tokens), type(X_test_mask), type(X_test_segment))
print(X_test_tokens.shape, X_test_mask.shape, X_test_segment.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
(20000, 55) (20000, 55) (20000, 55)
```

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 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
```

Saved successfully!

```
import pickle


##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_
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),open('test_data.p


#you can load from disk
import pickle
X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.load(open("train_
X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(open("test_data.p
```

Grader function 4

```
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]
        (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()
```

```
True
```

Grader function 5


```
    if type(X_test_tokens) == np.ndarray:

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

```
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.

```
bert_model.input
```

```
[<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>]
```

```
bert_model.output
```

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

```
# 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])
```

```
# 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])
```

Saved successfully!

```
hat, no need to run all again.
pickle.dump((X_train_pooled_output, X_test_pooled_output),open('final_output.pkl','wb'))
```

```
X_train_pooled_output, X_test_pooled_output= pickle.load(open('final_output.pkl', 'rb'))
```

```
X_train_pooled_output.shape, X_test_pooled_output.shape
```

```
((80000, 768), (20000, 768))
```

Grader function 6

```
#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()
```

```
True
```

Part-5: Training a NN with 768 features

Create a NN and train the NN.

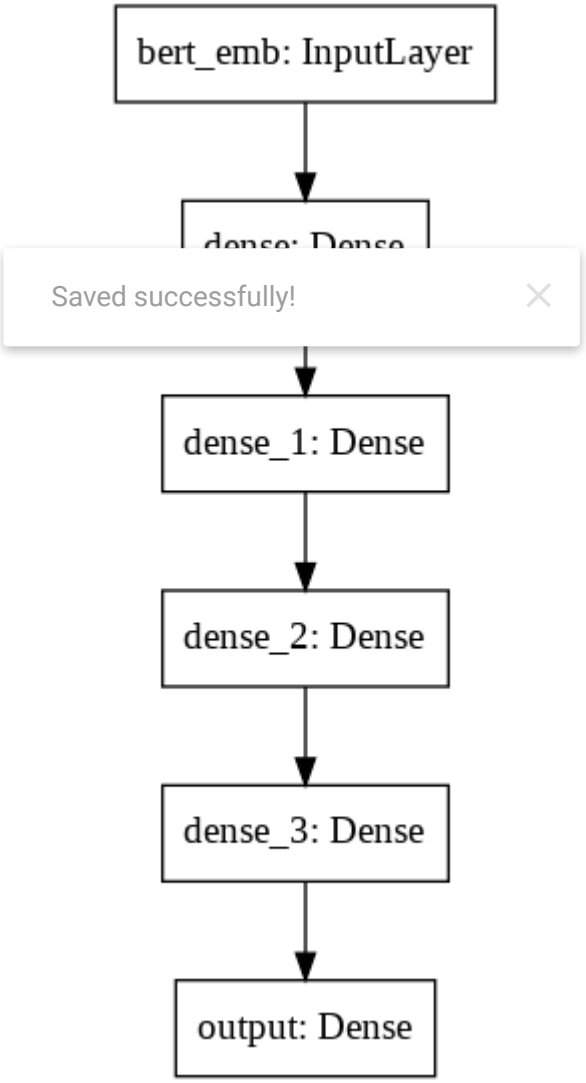
1. **You have to use AUC as metric. Do not use tf.keras.metrics.AUC**
   You have to write custom code for AUC and print it at the end of each epoch
2. You can use any architecture you want.
3. You have to use tensorboard to log all your metrics and Losses. You have to send thos
4. Print the loss and metric at every epoch.
5. You have to submit without overfitting and underfitting.

```
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout
from tensorflow.keras.models import Model
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
```

Saved successfully!

```
essions: https://pymotw.com/2/re/
import pickle
from tqdm import tqdm
import os
# !pip install chart_studio
# from chart_studio.plotly import plotly as py
# import plotly.express as px
# import plotly.offline as offline
# import plotly.graph_objs as go
# offline.init_notebook_mode()
from sklearn.metrics import roc_auc_score, f1_score
from keras.callbacks import Callback, EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
import logging
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import LearningRateScheduler
from keras.callbacks import TerminateOnNaN
from keras.callbacks import LambdaCallback
from keras.layers import Input, Embedding, LSTM, Dropout, BatchNormalization, Dense, conca
from keras.preprocessing.text import Tokenizer, one_hot
from keras import regularizers
from keras.optimizers import *
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model, load_model
import random as rn
```

```
import datetime
import numpy as np
import tensorflow as tf


##create an NN and
bert_emb = Input(shape=(768,),name="bert_emb")
x = Dense(256,activation="relu",kernel_initializer="he_normal",kernel_regularizer=regulari
#x=Dropout(0.5)(x)
x = Dense(128,activation="relu",kernel_initializer="he_normal",kernel_regularizer=regulari
#x=Dropout(0.4)(x)
x = Dense(64,activation="relu",kernel_initializer="he_normal",kernel_regularizer=regulariz
#x=Dropout(0.3)(x)
x = Dense(32,activation="relu",kernel_initializer="he_normal",kernel_regularizer=regulariz
output = Dense(2, activation='softmax', name='output')(x)
model = Model(inputs=[bert_emb],outputs=[output])


#Model Architechture:
tf.keras.utils.plot_model(model,show_shapes=False, show_layer_names=True, rankdir='TB',exp
```




```
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
bert_emb (InputLayer)        [(None, 768)]             0
_________________________________________________________________
dense (Dense)                (None, 256)               196864
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
output (Dense)               (None, 2)                 66
=================================================================
Total params: 240,162
Trainable params: 240,162
Non-trainable params: 0
_________________________________________________________________
```

```
train_data = [X_train_pooled_output]
test_data = [X_test_pooled_output]
```

```
from keras.utils import np_utils
Y_train = np_utils.to_categorical(y_train, 2)
Y_test = np_utils.to_categorical(y_test, 2)
```

Saved successfully!

```
= 1:
    else:
        return roc_auc_score(y_true, y_pred)
def auroc(y_true, y_pred):
    return tf.py_function(auc1, (y_true, y_pred), tf.double)
```

```
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=[auroc])
```

```
!pip install tensorboardcolab
%load_ext tensorboard
```

```
Requirement already satisfied: tensorboardcolab in /usr/local/lib/python3.6/dist-pack
```

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

```
from keras.callbacks import TensorBoard
```

```
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, LearningRat
def decay_fn(epoch, lr):
    if epoch < 10:
        return 0.001
    elif epoch >= 10 and epoch < 20:
```

```
        return 0.0001
    else:
        return 0.00001

lr_scheduler = LearningRateScheduler(decay_fn)



filepath = "{epoch:03d}-{val_auroc:.3f}.hdf5"
checkpoint_1 = ModelCheckpoint(filepath, monitor = "val_auroc", mode="max", save_best_only
#earlystop = EarlyStopping(monitor='val_auroc', patience=8, verbose=1, mode='max')
#reduce_lr = ReduceLROnPlateau(monitor='val_auroc', patience=5, factor=0.1, min_lr = 0.000
csv_log = CSVLogger('training_1.log')
NAME = 'model_1'
tensorboard_1 = TensorBoard(log_dir='logs\{}'.format(NAME),update_freq='epoch')
callbacks_1 = [tensorboard_1, checkpoint_1, csv_log, lr_scheduler]


history = model.fit(train_data,Y_train,batch_size=512,epochs=30,validation_data=(test_data
```

```
Epoch 00016: val_auroc did not improve from 0.95089
157/157 [==============================] - 2s 11ms/step - loss: 0.2472 - auroc: 0.9
Epoch 17/30
152/157 [============================>.] - ETA: 0s - loss: 0.2458 - auroc: 0.9554
Epoch 00017: val_auroc did not improve from 0.95089
157/157 [==============================] - 2s 11ms/step - loss: 0.2454 - auroc: 0.9
Epoch 18/30
156/157 [============================>.] - ETA: 0s - loss: 0.2446 - auroc: 0.9560
Epoch 00018: val_auroc did not improve from 0.95089
157/157 [==============================] - 2s 11ms/step - loss: 0.2446 - auroc: 0.9
Epoch 19/30
154/157 [============================>.] - ETA: 0s - loss: 0.2437 - auroc: 0.9562
Epoch 00019: val_auroc improved from 0.95089 to 0.95151, saving model to 019-0.952
157/157 [==============================] - 2s 11ms/step - loss: 0.2437 - auroc: 0.9
Epoch 20/30
152/157 [============================>.] - ETA: 0s - loss: 0.2417 - auroc: 0.9560
Epoch 00020: val_auroc did not improve from 0.95151
157/157 [==============================] - 2s 11ms/step - loss: 0.2412 - auroc: 0.9
Epoch 21/30
157/157 [==============================] - ETA: 0s - loss: 0.2397 - auroc: 0.9563
Epoch 00021: val_auroc improved from 0.95151 to 0.95152, saving model to 021-0.952
157/157 [==============================] - 2s 11ms/step - loss: 0.2397 - auroc: 0.9
Epoch 22/30
155/157 [============================>.] - ETA: 0s - loss: 0.2389 - auroc: 0.9567
Epoch 00022: val_auroc improved from 0.95152 to 0.95154, saving model to 022-0.952
157/157 [==============================] - 2s 11ms/step - loss: 0.2391 - auroc: 0.9
Epoch 23/30
157/157 [==============================] - ETA: 0s - loss: 0.2389 - auroc: 0.9567
Epoch 00023: val_auroc improved from 0.95154 to 0.95167, saving model to 023-0.952
157/157 [==============================] - 2s 11ms/step - loss: 0.2389 - auroc: 0.9
Epoch 24/30
154/157 [============================>.] - ETA: 0s - loss: 0.2390 - auroc: 0.9569
Epoch 00024: val_auroc did not improve from 0.95167
157/157 [==============================] - 2s 11ms/step - loss: 0.2387 - auroc: 0.9
Epoch 25/30
157/157 [==============================] - ETA: 0s - loss: 0.2386 - auroc: 0.9569
Epoch 00025: val_auroc did not improve from 0.95167
157/157 [==============================] - 2s 11ms/step - loss: 0.2386 - auroc: 0.9
Epoch 26/30
```

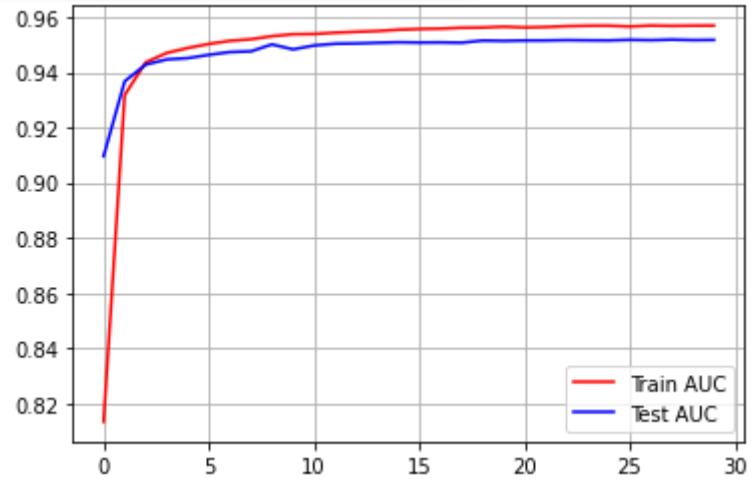
Saved successfully!

```
156/157 [============================>.] - ETA: 0s - loss: 0.2383 - auroc: 0.9566
Epoch 00026: val_auroc improved from 0.95167 to 0.95179, saving model to 026-0.952
157/157 [==============================] - 2s 11ms/step - loss: 0.2383 - auroc: 0.9
Epoch 27/30
155/157 [============================>.] - ETA: 0s - loss: 0.2384 - auroc: 0.9567
Epoch 00027: val_auroc did not improve from 0.95179
157/157 [==============================] - 2s 11ms/step - loss: 0.2382 - auroc: 0.9
Epoch 28/30
152/157 [============================>.] - ETA: 0s - loss: 0.2378 - auroc: 0.9570
Epoch 00028: val_auroc improved from 0.95179 to 0.95185, saving model to 028-0.952
157/157 [==============================] - 2s 11ms/step - loss: 0.2380 - auroc: 0.9
Epoch 29/30
152/157 [============================>.] - ETA: 0s - loss: 0.2377 - auroc: 0.9569
Epoch 00029: val_auroc did not improve from 0.95185
157/157 [==============================] - 2s 11ms/step - loss: 0.2380 - auroc: 0.9
Epoch 30/30
154/157 [============================>.] - ETA: 0s - loss: 0.2378 - auroc: 0.9568
Epoch 00030: val_auroc did not improve from 0.95185
```

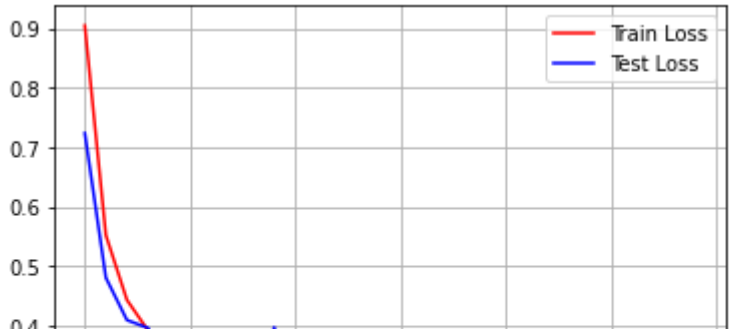
```
#Save Model to disk
model.save('028-0.952.hdf5')
```

```
from matplotlib import pyplot as plt
plt.plot(history.history['auroc'], 'r')
plt.plot(history.history['val_auroc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'b'})
```

Saved successfully!




```
plt.plot(history.history['loss'], 'r')
plt.plot(history.history['val_loss'], 'b')
plt.legend({'Train Loss': 'r', 'Test Loss':'b'})
plt.grid()
plt.show()
```




```
!ls
```

```
001-0.910.hdf5   008-0.948.hdf5   021-0.952.hdf5     model.png
002-0.937.hdf5   009-0.950.hdf5   022-0.952.hdf5     sample_data
003-0.943.hdf5   012-0.950.hdf5   023-0.952.hdf5     test_data.pkl
004-0.945.hdf5   013-0.950.hdf5   026-0.952.hdf5     train_data.pkl
005-0.945.hdf5   014-0.951.hdf5   028-0.952.hdf5     training_1.log
006-0.946.hdf5   015-0.951.hdf5   final_output.pkl
007-0.947.hdf5   019-0.952.hdf5  'logs\model_1'
```

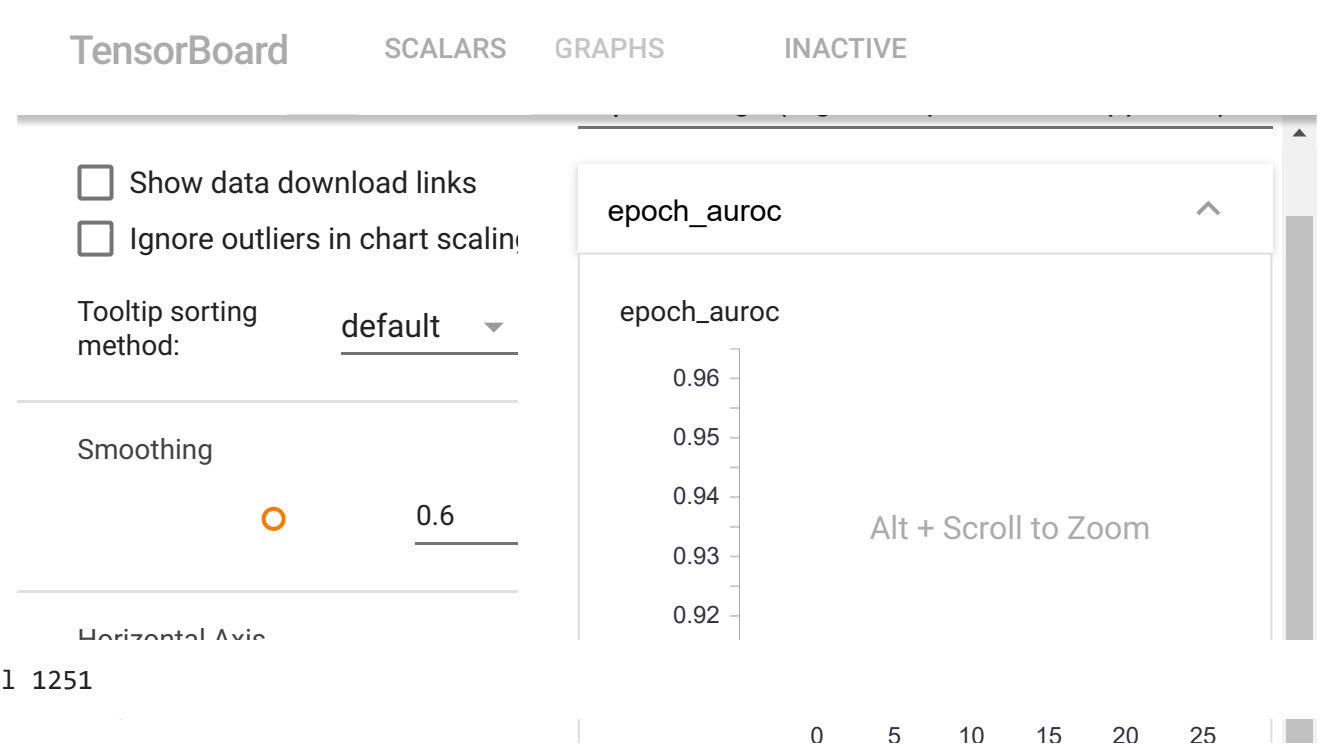
```
# Load the TensorBoard notebook extension
%load_ext tensorboard
```

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

```
NAME = 'model_1'
```

Saved successfully!

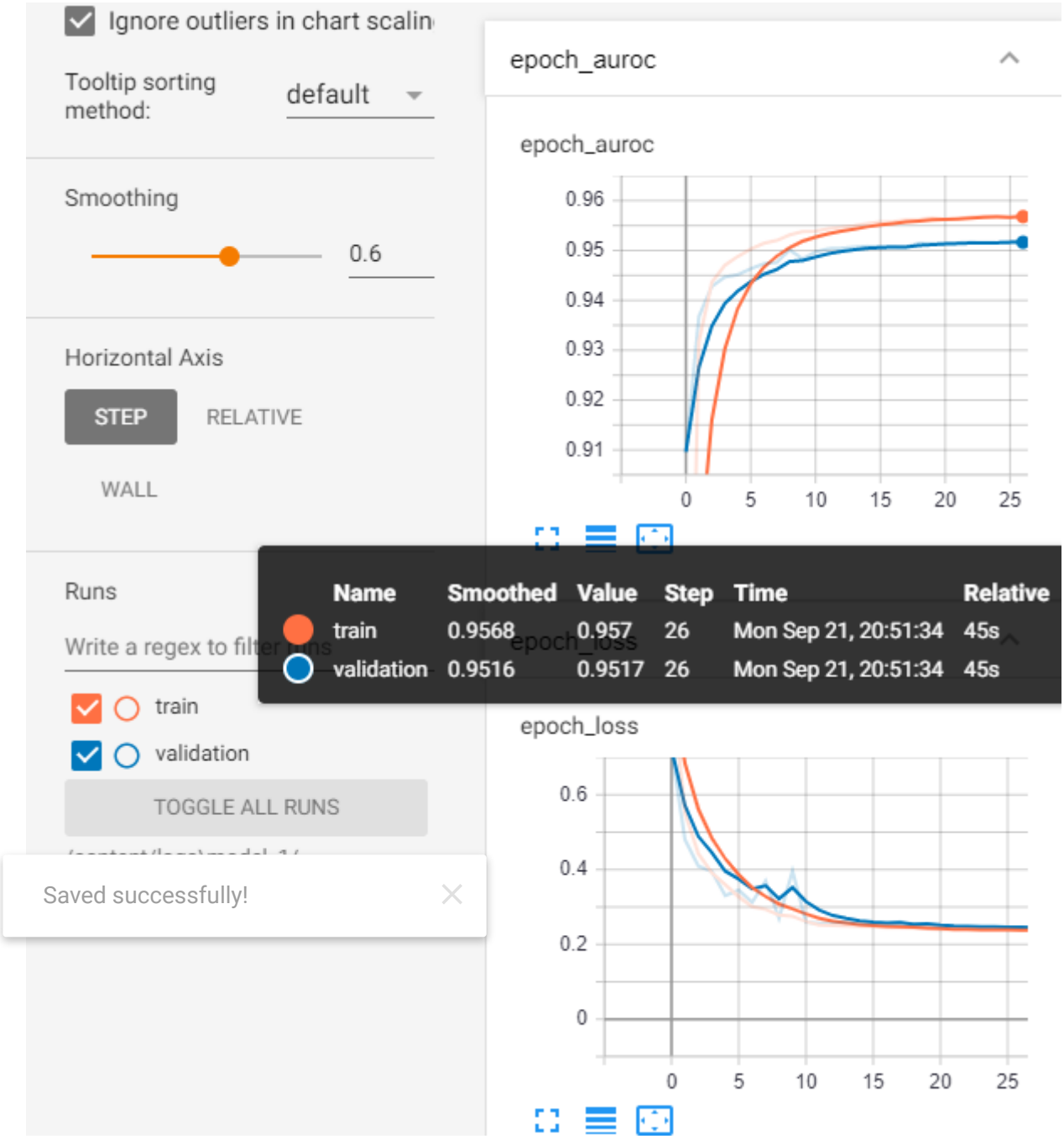
```
\model_1/'
```




```
!kill 1251
```

Model - Plots

Saved successfully!



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]

```
#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
import re
# !pip3 install tensorflow_text>=2.0.0rc0
# !pip3 install sentencepiece
# !pip3 install tf_sentencepiece
#Keep tokenization.py in the same folder as notebook
import tokenization
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout
from tensorflow.keras.models import Model
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import pickle
from tqdm import tqdm
import os
from sklearn.metrics import roc_auc_score, f1_score
from keras.callbacks import Callback, EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
import logging
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import LearningRateScheduler
from keras.callbacks import TerminateOnNaN
from keras.callbacks import LambdaCallback
dding, LSTM, Dropout, BatchNormalization, Dense, conca
Tokenizer, one_hot
from keras import regularizers
from keras.optimizers import *
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model, load_model
import random as rn
import datetime
import numpy as np
import tensorflow as tf
import keras


def preprocess_data(test_data):
    #remove HTML from the Text column and save in the Text column only
    #https://www.w3resource.com/python-exercises/pandas/string/python-pandas-string-exerci

    def remove_tags(string):
        result = re.sub('<.*?>','',string)
        return result
    test_data['Text']=test_data['Text'].apply(lambda cw : remove_tags(cw))
    #test_data.info()
    #Taking reviews with length of less than 50
    def get_wordlen(x):
        return len(x.split())
    test_data['len'] = test_data.Text.apply(get_wordlen)
    test_data = test_data[test_data.len<50]
```

Saved successfully!

```
    #reviews = reviews.sample(n=100000, random_state=30)   #for limiting to 100000 rows
    return test_data


def get_bert_model():
    ## 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 c
    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="

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

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

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



```
    not sequence out.
    #If you want to know about those, please read https://www.kaggle.com/questions-and-ans
    bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_ou

    return bert_model



def get_tokenizer():
    #getting Vocab file
    # Create tokenizer " Instantiate FullTokenizer"
    # name must be "tokenizer"
    vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
    do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
    tokenizer = tokenization.FullTokenizer(vocab_file=vocab_file, do_lower_case=do_lower_c
    return tokenizer


def builder_tokenize(data_row):
    #print("original sentance : \n", np.array(X_train.values[1][0].split()))
    #print("number of words: ", len(X_train.values[1][0].split()))
    #print('='*50)
    tokens = tokenizer.tokenize(data_row)
    tokens = tokens[0:(max_seq_length-2)]
    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: ", np.array(tokenizer.convert_to
    #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)

    mask_1 = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
    seg_1 = np.array([0]*max_seq_length)
    token_1 = np.array(tokenizer.convert_tokens_to_ids(tokens)+[0]*(max_seq_length-len(tok
    #print(len(mask_1),len(seg_1), len(token_1))
    return token_1, mask_1, seg_1


def get_tok_mask_seg(tokenizer, test_data):
    #Applying builder_tokenize function on X_train
    tokenizer = get_tokenizer()
    X_test_tokens = []
    X_test_mask = []
    X_test_segment = []
    iter_num = len(test_data)
    for row_num in tqdm(range(iter_num)):
      row = test_data.values[row_num][0]
      token_1, mask_1, seg_1 = builder_tokenize(row)
      X_test_tokens.append(token_1)
```

Saved successfully!

```
    X_test_tokens = np.array(X_test_tokens)
    X_test_mask = np.array(X_test_mask)
    X_test_segment = np.array(X_test_segment)
    return X_test_tokens, X_test_mask, X_test_segment

     100%|██████████| 352/352 [00:00<00:00, 1195.78it/s]CPU times: user 396 ms, sys: 20.3
     Wall time: 406 ms
```

```
def auc1(y_true, y_pred):
    if len(np.unique(y_true[:,1])) == 1:
        return 0.5
    else:
        return roc_auc_score(y_true, y_pred)
def auroc(y_true, y_pred):
    return tf.py_function(auc1, (y_true, y_pred), tf.double)
def get_saved_pretrained_nn_model():
    trained_model = keras.models.load_model('028-0.952.hdf5')
    return trained_model


#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 # fro
```

```
#https://towardsdatascience.com/3-ways-to-load-csv-files-into-colab-7c14fcbdcb92
from google.colab import files
uploaded = files.upload()


import pandas as pd
import io
test_data = pd.read_csv(io.BytesIO(uploaded['test.csv']))
# Dataset is now stored in a Pandas Dataframe


test_data.info() #should contain only 1 column i.e. 'Text'
```

```
<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
```

```
test_data = preprocess_data(test_data)


print(test_data.shape)
```

Saved successfully!

	Text	len
0	Just opened Greenies Joint Care (individually ...	29
1	This product rocks :) My mom was very happy w/...	27
2	The product was fine, but the cost of shipping...	21
3	I love this soup. It's great as part of a meal...	29
4	Getting ready to order again. These are great ...	44

```
bert_model = get_bert_model()
print(bert_model.summary())
print("\n\n\nbert_model.output:\n",bert_model.output)
```

```
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
__________________________________________________________________________________________________
None



bert_model.output:
 Tensor("keras_layer/StatefulPartitionedCall:0", shape=(None, 768), dtype=float32)
```

```
%%time
from tqdm import tqdm
X_test_tokens, X_test_mask, X_test_segment = get_tok_mask_seg(tokenizer, test_data)


print(type(X_test_tokens), type(X_test_mask), type(X_test_segment))
print(X_test_tokens.shape, X_test_mask.shape, X_test_segment.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
(352, 55) (352, 55) (352, 55)
```

```
print('bert_model.input:\n',bert_model.input)
                                    del.output) #should be (None,768)
```



```
 [<tf.Tensor 'input_word_ids:0' shape=(None, 55) dtype=int32>, <tf.Tensor 'input_mask
bert_model.output:
 Tensor("keras_layer/StatefulPartitionedCall:0", shape=(None, 768), dtype=float32)
```

```
# get the test output, BERT model will give one output so save in
# equivalent to X_test_pooled_output
X_test=bert_model.predict([X_test_tokens,X_test_mask,X_test_segment])
print(X_test.shape)  #should be (352,768)
```

```
(352, 768)
```

```
model = get_saved_pretrained_nn_model()
y_prob = model.predict(X_test)
y_classes = y_prob.argmax(axis=-1)


print("Predicted·Class·Labels·of·Test·Data(In Order): \n\n",y_classes)
```

```
Predicted Class Labels of Test Data(In Order):

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

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

1. In Part 1, I have successfully preprocess the data
2. In part 2, I have applied the BERT model in Keras, and found the important observations:

- Total params: 109,482,241
- Trainable params: 0
- Non-trainable params: 109,482,241

3. In part 3, I have applied Tokenization on X_train, X_test, and storing the data in pickel format. Also make sure that Type of all above token arrays should be numpy array not list
4. In part 4, I got the embeddings for each sentence in the Train and test data. BERT model ...est_pooled_output, now I saved all your results to disk so ...e 'final_output.pkl'.

Saved successfully!

5. In part 5, Cretaing and training a Neural Network with 768 features, used AUC as metric and print it at the end of each epoch, used tensorboard to show log all metrics and Losses. The model is balanced, neither overfitting nor underfitting.
6. In part 6, successfuly created a Data pipeline for BERT Model along with printing Predicted Class Labels of Test Data.

✓ 14m 44s completed at 12:10 AM

Saved successfully!