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 [ ]:
```
#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 [ ]:
```
tf.test.gpu_device_name()
```

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

Grader function 1

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

Out[ ]:
```
True
```

Part-1: Preprocessing

In [ ]:
```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv("Reviews.csv")
#reviews.to_frame()
#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 [ ]:
```
reviews.head()
```

Out[ ]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

In [ ]:
```
type(reviews['Score'])
```
Out[ ]:
```
pandas.core.series.Series
```

In [ ]:
```
text_data = reviews[["Text" , "Score"]]
```

In [ ]:
```
reviews = text_data[text_data['Score'] != 3]

reviews.shape
```
Out[ ]:
```
(525814, 2)
```

In [ ]:
```
d= reviews['Score'].apply(lambda x: 1 if x > 3 else 0)
reviews['Score'] = d
reviews.head(5)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y
```

Out[ ]:

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

In [ ]:
```
reviews.Score.value_counts()
```
Out[ ]:
```
1    443777
0     82037
Name: Score, dtype: int64
```

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[ ]:
```
True
```

In [ ]:
```
reviews.shape
```
Out[ ]:
```
(525814, 2)
```

In [ ]:
```
from random import sample
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 [ ]:
```
#https://stackoverflow.com/questions/9662346/python-code-to-remove-html-tags-from-a-string
```

```
import re

def cleanhtml(raw_html):
  cleanr = re.compile('<.*?>')
  cleantext = re.sub(cleanr, '', raw_html)
  result = re.sub(r"http\S+", "", cleantext)
  return result

reviews['Text'] = [cleanhtml(i) for i in reviews['Text']]
reviews['Text'].head(5)
```

In [ ]:

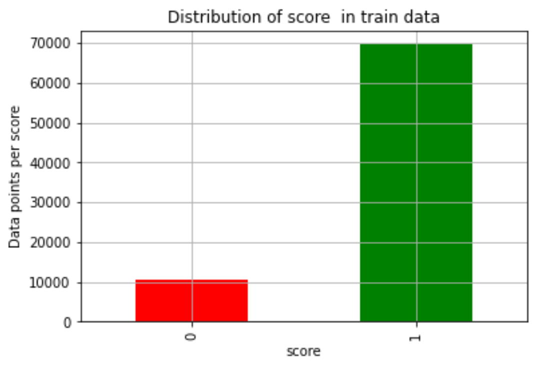
```
text_data = reviews['Text']
print(text_data)
```

```
64117     The tea was of great quality and it tasted lik...
418112    My cat loves this.  The pellets are nice and s...
357829    Great product. Does not completely get rid of ...
175872    This gum is my favorite!  I would advise every...
178716    I also found out about this product because of...
                                ...
336657    Using this coffee and a stove top espresso mak...
498034    THE TASTE OF THIS M&M IS THE BEST. I USED IT I...
357766    Excellent Tea. I enjoy a cup every now and the...
326811    These oatmeal cookies have a great spice taste...
19261     This is the best coffee ever! I will never dri...
Name: Text, Length: 100000, dtype: object
```

In [ ]:

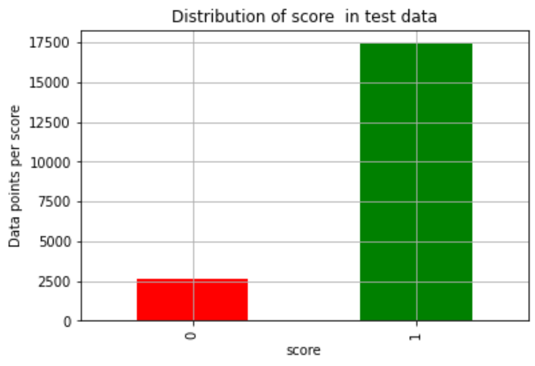
```
import matplotlib.pyplot as plt
train_class_distribution = y_train.value_counts().sort_index()
test_class_distribution = y_test.value_counts().sort_index()

my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar',color=my_colors)
plt.xlabel('score')
plt.ylabel('Data points per score')
plt.title('Distribution of score  in train data')
plt.grid()
plt.show()
```



In [ ]:

```
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar',color=my_colors)
plt.xlabel('score')
plt.ylabel('Data points per score')
plt.title('Distribution of score  in test data')
plt.grid()
plt.show()
```



In [ ]:

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

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

#BERT takes 3 inputs

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

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

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

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

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

In [ ]:
```
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 [ ]:
```
bert_model.output
```

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

Part-3: Tokenization

In [ ]:
```
#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 [ ]:
```
!pip install sentencepiece
```

```
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.7/dist-packages (0.1.95)
```

In [ ]:
```
%run tokenization.py
```

```
<Figure size 432x288 with 0 Axes>
```

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

```
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 [ ]: from keras.preprocessing import sequence
        print('original sentence :\n' , np.array(X_train.values[0].split()))
        print('number of words :',len(X_train.values[0].split()))
        print('='*50)
        max_seq_length = 55
        tokens = tokenizer.tokenize(X_train.values[0])
        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('token replace with 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 sequence length is ',np.array([0]*max_seq_length))
        print('='*50)
        #X_train_encoded_essay = t.texts_to_sequences(X_train['essay'])
        y = np.array(tokenizer.convert_tokens_to_ids(tokens))
        z = [int(i) for i in y]

        x = []
        for i in range(max_seq_length):
          if i < len(z):
            x.append(int(z[i]))
          else:
            x.append(int(0))

        print(x)
```

```
original sentence :
 ['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
token replace with 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]
==================================================
the sequence length 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]
==================================================
[101, 1045, 2018, 2196, 2699, 2023, 4435, 2077, 1010, 2061, 1045, 2001, 5191, 2055, 1996, 3737, 1012, 2009, 12595, 2307, 1012, 1037, 220
0, 3835, 5744, 4138, 2440, 14894, 1012, 2049, 2026, 2047, 5684, 3388, 1012, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

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

In [ ]:
```
max_seq_length = 55
def sen_to_token(data):
  Tokens , mask , sequence = list(), list(), list()
  for i in range(len(data)):
    tokens = tokenizer.tokenize(X_train.values[i])
    tokens = tokens[0:(max_seq_length-2)]
    tokens = ['[CLS]',*tokens,'[SEP]']
    tokenizer.convert_tokens_to_ids(tokens)
    y = np.array(tokenizer.convert_tokens_to_ids(tokens))
    z = [int(i) for i in y]
    x = []
    for i in range(max_seq_length):
      if i < len(z):
        x.append(int(z[i]))
      else:
        x.append(int(0))
    Tokens.append(np.array(x))

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

  return  np.array(Tokens) , np.array(mask) , np.array(sequence)
```

In [ ]:
```
X_train_tokens, X_train_mask, X_train_segment = sen_to_token(X_train)
X_test_tokens, X_test_mask, X_test_segment = sen_to_token(X_test)
```

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

In [ ]:
```
##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 [ ]:
```
#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[ ]:
```
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[ ]:
```
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[ ]:
```
[<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 [ ]:
```
bert_model.output
```

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

In [ ]:
```
X_train_tokens.shape,X_train_mask.shape,X_train_segment.shape
```

Out[ ]:
```
((80000, 55), (80000, 55), (80000, 55))
```

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])


#X_train_pooled_output=bert_model.predict([X_train_tokens[:100],X_train_mask[:100],X_train_segment[:100]])
```

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])
```

```
#X_test_pooled_output=bert_model.predict([X_test_tokens[:100],X_test_mask[:100],X_test_segment[:100]])
```

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 [ ]:
```
import pickle
```

In [ ]:
```
#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'))


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

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[ ]:
```
True
```

Part-5: Training a NN with 768 features

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

In [57]:
```
inputs_review = Input(shape=(768,),)

dense = Dense(768, activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(seed=0),
              kernel_regularizer='l2')(inputs_review)

drop_out1 = Dropout(0.2)(dense)

dense = Dense(100, activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(seed=0),
              kernel_regularizer='l2')(drop_out1)

dense = Dense(70, activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(seed=0),
              kernel_regularizer='l2')(dense)

drop_out2 = Dropout(0.2)(dense)

dense = Dense(40, activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(seed=0),
              kernel_regularizer='l2')(drop_out2)


Out = Dense(units=2,activation='softmax')(dense)

#Creating a model
model = Model(inputs_review,outputs=Out)
```

In [58]:
```
print(model.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 768)]             0
_________________________________________________________________
dense_5 (Dense)              (None, 768)               590592
_________________________________________________________________
dropout_2 (Dropout)          (None, 768)               0
_________________________________________________________________
dense_6 (Dense)              (None, 100)               76900
_________________________________________________________________
dense_7 (Dense)              (None, 70)                7070
_________________________________________________________________
dropout_3 (Dropout)          (None, 70)                0
_________________________________________________________________
dense_8 (Dense)              (None, 40)                2840
```

```
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 82
=================================================================
Total params: 677,484
Trainable params: 677,484
Non-trainable params: 0
_________________________________________________________________
None
```

In [49]:
```
y_train = tf.keras.utils.to_categorical(y_train, 2)
y_test = tf.keras.utils.to_categorical(y_test, 2)
```

In [50]:
```
from tensorflow import keras
%load_ext tensorboard

# Clear any logs from previous runs
!rm -rf ./logs/

import tensorflow as tf
import datetime
from tensorflow.keras.callbacks import ModelCheckpoint


log_dir="logs\\fit\\" + 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,write_grads=True)


callback_list = [tensorboard_callback]
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

In [72]:
```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=[tf.keras.metrics.AUC()])


model.fit(X_train_pooled_output,y_train,epochs=10, validation_data=(X_test_pooled_output,y_test), batch_size=1000,callbacks=callback_list)
```

```
Epoch 1/10
80/80 [==============================] - 3s 28ms/step - loss: 12.8219 - auc_1: 0.8872 - val_loss: 2.3216 - val_auc_1: 0.9478
Epoch 2/10
80/80 [==============================] - 1s 12ms/step - loss: 1.7354 - auc_1: 0.9355 - val_loss: 0.9058 - val_auc_1: 0.9517
Epoch 3/10
80/80 [==============================] - 1s 12ms/step - loss: 0.7929 - auc_1: 0.9445 - val_loss: 0.5578 - val_auc_1: 0.9600
Epoch 4/10
80/80 [==============================] - 1s 12ms/step - loss: 0.5130 - auc_1: 0.9576 - val_loss: 0.4491 - val_auc_1: 0.9643
Epoch 5/10
80/80 [==============================] - 1s 12ms/step - loss: 0.4205 - auc_1: 0.9574 - val_loss: 0.3524 - val_auc_1: 0.9690
Epoch 6/10
80/80 [==============================] - 1s 12ms/step - loss: 0.3586 - auc_1: 0.9651 - val_loss: 0.3264 - val_auc_1: 0.9700
Epoch 7/10
80/80 [==============================] - 1s 12ms/step - loss: 0.3361 - auc_1: 0.9661 - val_loss: 0.3135 - val_auc_1: 0.9701
Epoch 8/10
80/80 [==============================] - 1s 12ms/step - loss: 0.3168 - auc_1: 0.9689 - val_loss: 0.2984 - val_auc_1: 0.9725
Epoch 9/10
80/80 [==============================] - 1s 12ms/step - loss: 0.3053 - auc_1: 0.9704 - val_loss: 0.3869 - val_auc_1: 0.9583
Epoch 10/10
80/80 [==============================] - 1s 12ms/step - loss: 0.3110 - auc_1: 0.9673 - val_loss: 0.3077 - val_auc_1: 0.9710
```

Out[72]:
```
<tensorflow.python.keras.callbacks.History at 0x7ff1809a50d0>
```

In [75]:
```
%tensorboard --logdir 'logs\fit\20210312-143426'
```

Part-6: Creating a Data pipeline for BERT Model

1. Download data from here
2. Read the csv file
3. Remove all the html tags
4. Now do tokenization [Part 3 as mentioned above]
    - Create tokens,mask array and segment array

5. Get Embeddings from BERT Model [Part 4 as mentioned above] , let it be X_test
    - Print the shape of output(X_test.shape).You should get (352,768)

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

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


</pre>
```

Data pipeline for Bert Model

In [59]:
```
test = pd.read_csv("test.csv")
test.head()
```

Out[59]:

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

In [60]:
```
#removing all html tags
test['Text']= test['Text'].apply(lambda x : cleanhtml(x))
```

In [61]:
```
#tokenization
X_tokens,x_mask,x_segments = sen_to_token(test)
```

In [62]:
```
X_tokens.shape
```

Out[62]:
```
(352, 55)
```

In [63]:
```
#Getting Embeddings from BERT Model
X_test = bert_model.predict([X_tokens,x_mask,x_segments])
```

In [64]:
```
X_test.shape
```

Out[64]:
```
(352, 768)
```

In [67]:
```
y_pred=model.predict(X_test)
y_pred = y_pred.argmax(axis=-1)
y_pred
```

Out[67]:
```
array([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, 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, 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, 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, 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, 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, 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, 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, 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, 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])
```

In [68]:
```
#viewing first reviews
test['Text'][0]
```

Out[68]:
```
'Just opened Greenies Joint Care (individually sealed) in December 2011 and found small worm crawling all over it.  Next one looked fine,
but really supposed to trust these now?'
```

In [69]:
```
y_pred[0]
```

Out[69]:
```
1
```

In [70]:
```
test['Text'][1]
```

Out[70]:
```
'This product rocks :) My mom was very happy w/the product it was excatly as described we loved seeing all the candy and eating it all
:)'
```

In [71]:
```
y_pred[1]
```

Out[71]:
```
1
```