BERT_

April 22, 2021

[ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

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

[ ]:
```
project_path = "/content/gdrive/My Drive/Colab Notebooks/NLP Transfer Learning␣
 ↪BERT/NLP Transfer Learning/"
review_csv_path = '/content/gdrive/My Drive/Colab Notebooks/NLP Transfer␣
 ↪Learning BERT/NLP Transfer Learning/Reviews.csv'
```

[ ]:
```
#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
from sklearn.model_selection import train_test_split
import copy
import pickle
import matplotlib.pyplot as plt
import tensorboard

%load_ext tensorboard
!rm -rf ./logs/
```

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

[ ]:
```
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

[ ]:
```
#get only 2 columns - Text, Score
reviews = pd.read_csv(review_csv_path)
reviews = pd.DataFrame({'Score':reviews.Score , 'Text':reviews.Text})



# If any NAN values
print("If any nan values for Text column {}".format(reviews['Text'].isnull().
 ↪values.any()))
print("If any nan values for Score column {}".format(reviews['Score'].isnull().
 ↪values.any()))



#if score> 3, set score = 1
#if score<=2, set score = 0
#if score == 3, remove the rows.
reviews.loc[reviews['Score'] <= 2, 'Score'] = 0
reviews.loc[reviews['Score'] > 3, 'Score'] = 1


# Drop rows, with Score = 3
reviews.drop(reviews[reviews['Score'] == 3].index, inplace = True)


# INFO
reviews.info()
```

```
If any nan values for Text column False
If any nan values for Score column False
<class 'pandas.core.frame.DataFrame'>
Int64Index: 525814 entries, 0 to 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
```

[ ]:
```
reviews.Score.value_counts()
```

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

Grader function 2

[ ]:
```
def grader_reviews():
    temp_shape = (reviews.shape == (525814, 2)) and (reviews.Score.
 ↪value_counts()[1]==443777)
    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.head(2)
```

[ ]:
```
        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
```

[ ]:
```
np.max(reviews['len'])
```

[ ]:
```
49
```

[ ]:
```
#remove HTML from the Text column and save in the Text column only
# https://stackoverflow.com/questions/9662346/
 ↪python-code-to-remove-html-tags-from-a-string

def cleanhtml(raw_html):
  cleanr = re.compile('<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});')
  cleantext = re.sub(cleanr, '', raw_html)
  return cleantext

reviews['Text'] = reviews.Text.apply(cleanhtml)
```

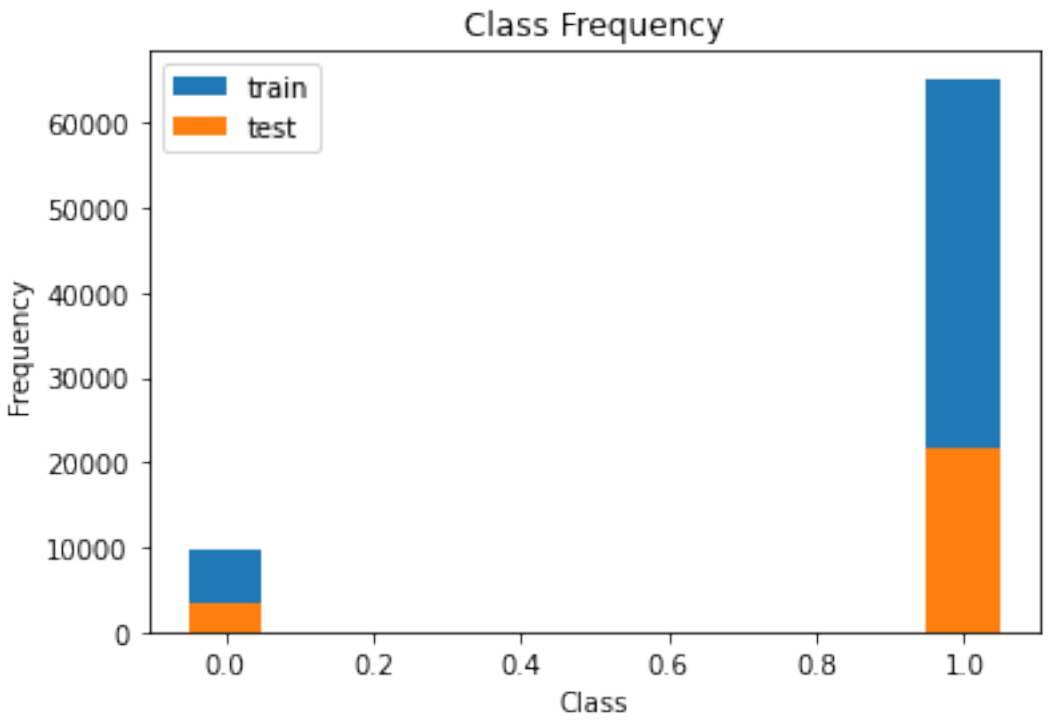
[ ]:
```
#print head 5
reviews.head(5)
```

[ ]:
```
        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
```

[ ]:
```
#split the data into train and test data(20%) with Stratify sampling, random␣
 ↪state 33,
y = reviews['Score']
X = reviews.drop(['Score'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪random_state=33)
```

[ ]:
```
# plot bar graphs of y_train and y_test
# https://www.cognitivecoder.com/2018/03/29/
 ↪3-quick-ways-to-create-graphs-of-your-class-distributions-in-python/



unique, counts = np.unique(y_train, return_counts=True)
plt.bar(unique, counts, width=0.1, label='train')

unique, counts = np.unique(y_test, return_counts=True)
plt.bar(unique, counts, width=0.1, label="test")

plt.title('Class Frequency')
plt.xlabel('Class')
plt.ylabel('Frequency')

plt.legend()
plt.show()
```



[ ]:
```
#saving to disk. if we need, we can load preprocessed data directly.
preprocessed_path = project_path + "preprocessed.csv"
reviews.to_csv(preprocessed_path, index=False)
```

[ ]:
```
## 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 = 49

#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)
```

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

```
Model: "functional_1"
__________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_word_ids (InputLayer)     [(None, 49)]         0
__________________________________________________________________________________
__________________
input_mask (InputLayer)         [(None, 49)]         0
__________________________________________________________________________________
__________________
segment_ids (InputLayer)        [(None, 49)]         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
__________________________________________________________________________________
__________________
```

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

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

[ ]:
```
#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()
```

[ ]:
```
vocab_file, do_lower_case
```

[ ]:
```
(b'gs://tfhub-modules/tensorflow/bert_en_uncased_L-12_H-768_A-12/1/uncompressed/
assets/vocab.txt',
 True)
```

[ ]:
```
#import tokenization - We have given tokenization.py file
# https://mg.readthedocs.io/
 ↪importing-local-python-modules-from-jupyter-notebooks/sys-path-in-notebook/
 ↪path-notebook.html

! pip install sentencepiece

import os
import sys
sys.path.insert(0, os.path.abspath('/content/gdrive/My Drive/Colab Notebooks/
 ↪NLP Transfer Learning BERT/NLP Transfer Learning'))

import tokenization
```

```
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.6/dist-
packages (0.1.94)
```

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

tokenizer = tokenization.FullTokenizer(vocab_file, do_lower_case)
```

Grader function 3

[ ]:
```
#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)
```

[ ]:
```
True
```

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

# np.ndarray(shape=(X_train.shape[0], max_seq_length))
X_train_tokens = []
X_train_mask = []
X_train_segment = []

X_test_tokens = []
X_test_mask = []
X_test_segment = []


for i, sentence in enumerate(X_train.Text):
  tokens = tokenizer.tokenize(sentence)
  tokens = tokens[0:(max_seq_length-2)]
  tokens = ['[CLS]', *tokens , '[SEP]']
  if len(tokens) < max_seq_length:
    for i, token in enumerate(tokens):
      while len(tokens) != max_seq_length:
        tokens.insert(-1, '[PAD]')

  if len(tokens) > max_seq_length:
    for i, token in enumerate(tokens):
      while len(tokens) != max_seq_length:
        del tokens[-2]

  X_train_tokens.append(np.array(tokenizer.convert_tokens_to_ids(copy.
 ↪deepcopy(tokens))))

  for i, token in enumerate(tokens):
    if token == '[PAD]':
      tokens[i] = 0
```

```
    else:
      tokens[i] = 1

  X_train_mask.append(np.array(copy.deepcopy(tokens)))

  for i, token in enumerate(tokens):
    tokens[i] = 0

  X_train_segment.append(np.array(copy.deepcopy(tokens)))

# Based on padding, create the mask for Train and Test ( 1 for real token, 0␣
 ↪for '[PAD]'),
# it will also same shape as input tokens (None, 55) save those in␣
 ↪X_train_mask, X_test_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


for i, sentence in enumerate(X_test.Text):
  tokens = tokenizer.tokenize(sentence)
  tokens = tokens[0:(max_seq_length-2)]
  tokens = ['[CLS]', *tokens , '[SEP]']
  if len(tokens) < max_seq_length:
    for i, token in enumerate(tokens):
      while len(tokens) != max_seq_length:
        tokens.insert(-1, '[PAD]')

  if len(tokens) > max_seq_length:
    for i, token in enumerate(tokens):
      while len(tokens) != max_seq_length:
        del tokens[-2]

  X_test_tokens.append(np.array(tokenizer.convert_tokens_to_ids(copy.
 ↪deepcopy(tokens))))

  for i, token in enumerate(tokens):
    if token == '[PAD]':
      tokens[i] = 0
```

```
    else:
      tokens[i] = 1

  X_test_mask.append(np.array(copy.deepcopy(tokens)))

  for i, token in enumerate(tokens):
    tokens[i] = 0

  X_test_segment.append(np.array(copy.deepcopy(tokens)))
```

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

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

Example

[ ]:
```
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_data.pkl','wb'))
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment,␣
 ↪y_test),open('test_data.pkl','wb'))
```

[ ]:
```
#you can load from disk
# train_data_path = "/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer␣
 ↪Learning BERT/NLP Transfer Learning/train_data.pkl"
# test_data_path = "/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer␣
 ↪Learning BERT/NLP Transfer Learning/test_data.pkl"

# X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.
 ↪load(open(train_data_path, 'rb'))
# X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.
 ↪load(open(test_data_path, 'rb'))
```

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]==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()
```

[ ]: True

Grader function 5

[ ]:
```
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()
```

[ ]:
```
True
```

[ ]:
```
bert_model.input
```

[ ]:
```
[<tf.Tensor 'input_word_ids:0' shape=(None, 49) dtype=int32>,
 <tf.Tensor 'input_mask:0' shape=(None, 49) dtype=int32>,
 <tf.Tensor 'segment_ids:0' shape=(None, 49) 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])
```

[ ]:
```
X_train_pooled_output.shape
```

[ ]:
```
(75000, 768)
```

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

[ ]:
```
##save all your results to disk so that, no need to run all again.
# bert_embeddings_path = "/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer␣
 ↪Learning BERT/NLP Transfer Learning/final_output.pkl"
# pickle.dump((X_train_pooled_output,␣
 ↪X_test_pooled_output),open(bert_embeddings_path,'wb'))
```

[ ]:
```
bert_embeddings_path = "/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer␣
 ↪Learning BERT/NLP Transfer Learning/final_output.pkl"

X_train_pooled_output, X_test_pooled_output= pickle.
 ↪load(open(bert_embeddings_path, 'rb'))
```

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
```

[ ]:
```
X_train_pooled_output.shape
```

[ ]:
```
(75000, 768)
```

[ ]:
```
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout, LSTM,␣
↪BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l1_l2, L1, L2, L1L2, l1, l2
from tensorflow.keras.initializers import he_normal
```

[ ]:
```
##create an NN and
tf.keras.backend.clear_session()

input = Input(shape=(X_train_pooled_output.shape[1], ))

# lstm = LSTM(units=128,input_shape=(X_train_pooled_output.shape[1], 1),␣
↪activation='relu', kernel_initializer=he_normal(), name="lstm_1") (input)

dense_1 = Dense(units=512, activation='relu', kernel_initializer=he_normal(),␣
↪kernel_regularizer=l1(0.0001), name="dense_1") (input)

drop_out1 = Dropout(rate=0.1, name='dropout_1') (dense_1)

dense_2 = Dense(units=256, activation='relu', kernel_initializer=he_normal(),␣
↪kernel_regularizer=l1(0.0001), name="dense_2") (drop_out1)
```

```
drop_out2 = Dropout(rate=0.1, name='dropout_2') (dense_2)

dense_3 = Dense(units=128, activation='relu', kernel_initializer=he_normal(),␣
↪kernel_regularizer=l1(0.0001), name="dense_3") (drop_out2)


batch_normal = BatchNormalization() (dense_3)

output = Dense(units=2, activation='softmax', name="output") (batch_normal)

model = Model(input, output)

model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 768)]             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               393728
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization (BatchNo (None, 128)               512
_________________________________________________________________
output (Dense)               (None, 2)                 258
=================================================================
Total params: 558,722
Trainable params: 558,466
Non-trainable params: 256
_________________________________________________________________
```

[ ]:
```
def auroc(y_true, y_pred):
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)
```

[ ]:
```
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,␣
↪EarlyStopping
import datetime
```

```
y_train_ohe = tf.keras.utils.to_categorical(y_train)
y_test_ohe = tf.keras.utils.to_categorical(y_test)

y_train_ohe.shape, y_test_ohe.shape




def auroc(y_true, y_pred):
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)

opt = tf.keras.optimizers.Adam(learning_rate=0.0001)


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

# auc_callback = AUC_Callback()
modelpath = "/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/
 ↪NLP Transfer Learning/bert_model.h5"

checkpoint_callback = ModelCheckpoint(modelpath, monitor='val_auroc',␣
 ↪verbose=1, save_best_only=True, mode='max')

reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',mode='min', factor=0.
 ↪01, patience=2, min_lr=0.00000001)

es_callback = EarlyStopping(monitor='val_loss', mode='min', verbose=0,␣
 ↪patience=2)

# nan_callback = tf.keras.callbacks.TerminateOnNaN()

log_dir="logs/fit/model_1_" + 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)




history = model.fit(X_train_pooled_output,
                    y_train_ohe,
                    validation_data = (X_test_pooled_output, y_test_ohe),
                    verbose=1,
                    epochs=20,
```

```
                    batch_size=500,
                    callbacks=[reduce_lr_callback, tensorboard_callback,␣
↪checkpoint_callback, es_callback]) # starts training
```

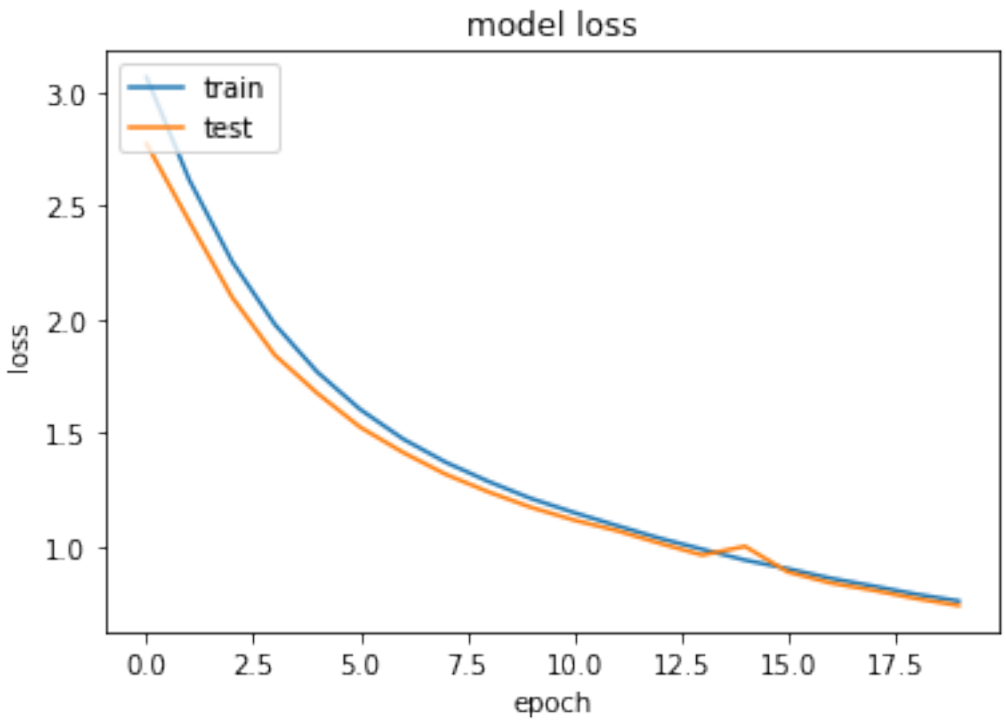
```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the
`TensorBoard` Callback.

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

Epoch 1/20
  1/150 […] - ETA: 0s - loss: 3.5256 - auroc:
0.4911WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after
2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after
2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.

  2/150 […] - ETA: 4s - loss: 3.5258 - auroc:
0.4690WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0122s vs `on_train_batch_end` time: 0.0422s).
Check your callbacks.

WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.0122s vs `on_train_batch_end` time: 0.0422s). Check
your callbacks.

145/150 [============================>.] - ETA: 0s - loss: 3.0740 - auroc:
0.7255
Epoch 00001: val_auroc improved from -inf to 0.90285, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 2s 12ms/step - loss: 3.0657 - auroc:
0.7289 - val_loss: 2.7720 - val_auroc: 0.9029
Epoch 2/20
144/150 [============================>..] - ETA: 0s - loss: 2.6211 - auroc:
0.8743
Epoch 00002: val_auroc improved from 0.90285 to 0.92588, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 2.6134 - auroc:
0.8752 - val_loss: 2.4302 - val_auroc: 0.9259
```

```
Epoch 3/20
145/150 [============================>.] - ETA: 0s - loss: 2.2597 - auroc:
0.9104
Epoch 00003: val_auroc improved from 0.92588 to 0.93023, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 2.2549 - auroc:
0.9102 - val_loss: 2.0973 - val_auroc: 0.9302
Epoch 4/20
148/150 [============================>.] - ETA: 0s - loss: 1.9793 - auroc:
0.9208
Epoch 00004: val_auroc improved from 0.93023 to 0.93657, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 8ms/step - loss: 1.9778 - auroc:
0.9208 - val_loss: 1.8424 - val_auroc: 0.9366
Epoch 5/20
149/150 [============================>.] - ETA: 0s - loss: 1.7674 - auroc:
0.9261
Epoch 00005: val_auroc improved from 0.93657 to 0.94135, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 1.7667 - auroc:
0.9262 - val_loss: 1.6734 - val_auroc: 0.9413
Epoch 6/20
149/150 [============================>.] - ETA: 0s - loss: 1.6032 - auroc:
0.9319
Epoch 00006: val_auroc improved from 0.94135 to 0.94374, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 1.6028 - auroc:
0.9318 - val_loss: 1.5243 - val_auroc: 0.9437
Epoch 7/20
150/150 [==============================] - ETA: 0s - loss: 1.4731 - auroc:
0.9369
Epoch 00007: val_auroc improved from 0.94374 to 0.94625, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 1.4731 - auroc:
0.9369 - val_loss: 1.4136 - val_auroc: 0.9463
Epoch 8/20
147/150 [============================>.] - ETA: 0s - loss: 1.3710 - auroc:
0.9379
Epoch 00008: val_auroc improved from 0.94625 to 0.94656, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 1.3702 - auroc:
0.9379 - val_loss: 1.3171 - val_auroc: 0.9466
```

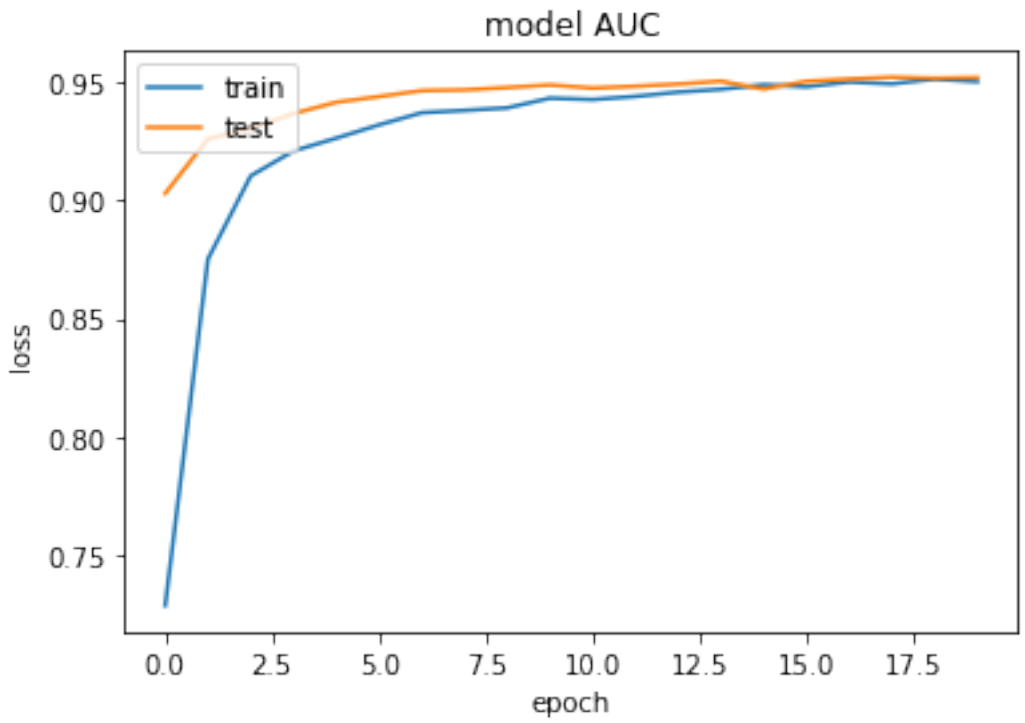
```
Epoch 9/20
146/150 [============================>.] - ETA: 0s - loss: 1.2872 - auroc:
0.9385
Epoch 00009: val_auroc improved from 0.94656 to 0.94756, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 1.2855 - auroc:
0.9390 - val_loss: 1.2395 - val_auroc: 0.9476
Epoch 10/20
147/150 [============================>.] - ETA: 0s - loss: 1.2111 - auroc:
0.9429
Epoch 00010: val_auroc improved from 0.94756 to 0.94865, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 1.2102 - auroc:
0.9431 - val_loss: 1.1714 - val_auroc: 0.9487
Epoch 11/20
148/150 [============================>.] - ETA: 0s - loss: 1.1493 - auroc:
0.9424
Epoch 00011: val_auroc did not improve from 0.94865
150/150 [==============================] - 1s 8ms/step - loss: 1.1489 - auroc:
0.9425 - val_loss: 1.1157 - val_auroc: 0.9473
Epoch 12/20
147/150 [============================>.] - ETA: 0s - loss: 1.0920 - auroc:
0.9436
Epoch 00012: val_auroc did not improve from 0.94865
150/150 [==============================] - 1s 8ms/step - loss: 1.0913 - auroc:
0.9438 - val_loss: 1.0694 - val_auroc: 0.9481
Epoch 13/20
145/150 [============================>.] - ETA: 0s - loss: 1.0370 - auroc:
0.9455
Epoch 00013: val_auroc improved from 0.94865 to 0.94902, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 1.0358 - auroc:
0.9456 - val_loss: 1.0139 - val_auroc: 0.9490
Epoch 14/20
148/150 [============================>.] - ETA: 0s - loss: 0.9874 - auroc:
0.9469
Epoch 00014: val_auroc improved from 0.94902 to 0.95023, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 0.9873 - auroc:
0.9468 - val_loss: 0.9616 - val_auroc: 0.9502
Epoch 15/20
149/150 [============================>.] - ETA: 0s - loss: 0.9399 - auroc:
0.9488
Epoch 00015: val_auroc did not improve from 0.95023
```

```
150/150 [==============================] - 1s 8ms/step - loss: 0.9398 - auroc:
0.9487 - val_loss: 1.0000 - val_auroc: 0.9469
Epoch 16/20
146/150 [============================>.] - ETA: 0s - loss: 0.9020 - auroc:
0.9481
Epoch 00016: val_auroc did not improve from 0.95023
150/150 [==============================] - 1s 8ms/step - loss: 0.9016 - auroc:
0.9479 - val_loss: 0.8874 - val_auroc: 0.9502
Epoch 17/20
148/150 [============================>.] - ETA: 0s - loss: 0.8598 - auroc:
0.9499
Epoch 00017: val_auroc improved from 0.95023 to 0.95116, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 0.8595 - auroc:
0.9499 - val_loss: 0.8389 - val_auroc: 0.9512
Epoch 18/20
149/150 [============================>.] - ETA: 0s - loss: 0.8251 - auroc:
0.9491
Epoch 00018: val_auroc improved from 0.95116 to 0.95193, saving model to
/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning BERT/NLP Transfer
Learning/bert_model.h5
150/150 [==============================] - 1s 9ms/step - loss: 0.8251 - auroc:
0.9491 - val_loss: 0.8068 - val_auroc: 0.9519
Epoch 19/20
148/150 [============================>.] - ETA: 0s - loss: 0.7891 - auroc:
0.9512
Epoch 00019: val_auroc did not improve from 0.95193
150/150 [==============================] - 1s 8ms/step - loss: 0.7892 - auroc:
0.9511 - val_loss: 0.7702 - val_auroc: 0.9515
Epoch 20/20
145/150 [============================>.] - ETA: 0s - loss: 0.7597 - auroc:
0.9502
Epoch 00020: val_auroc did not improve from 0.95193
150/150 [==============================] - 1s 8ms/step - loss: 0.7594 - auroc:
0.9500 - val_loss: 0.7397 - val_auroc: 0.9518
```

[ ]:
```python
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



[ ]:
```
plt.plot(history.history['auroc'])
plt.plot(history.history['val_auroc'])
plt.title('model AUC')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



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

```
<IPython.core.display.Javascript object>
```

[ ]:
```
test_data_path = "/content/gdrive/MyDrive/Colab Notebooks/NLP Transfer Learning␣
↪BERT/NLP Transfer Learning/test.csv"
df = pd.read_csv(test_data_path)
```

[ ]:
```
max(df.astype('str').applymap(lambda x: len(x)).max())
```

[ ]:
```
318
```

[ ]:
```
df.head(1)
```

[ ]:
```
                                                Text
0  Just opened Greenies Joint Care (individually …
```

[ ]:
```
df['Text'][0]
```

[ ]:
```
'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?'
```

0.1 Data cleaning

[ ]:
```
def cleanhtml(raw_html):
  cleanr = re.compile('<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});')
  cleantext = re.sub(cleanr, '', raw_html)
  return cleantext
```

[ ]:
```
df['Text'] = df.Text.apply(cleanhtml)
```

[ ]:
```
df['Text'][0]
```

[ ]:
```
'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?'
```

[ ]:
```
def get_wordlen(x):
    return len(x.split())
df['len'] = df.Text.apply(get_wordlen)
```

[ ]:
```
test_seq_max_len = np.max(df['len'])
```

[ ]:
```
test_seq_max_len
```

[ ]:
```
49
```

0.2 Tokenization

[ ]:
```
tokens_list = []
masks_list = []
segments_list = []

for i, sentence in enumerate(df.Text):
  tokens = tokenizer.tokenize(sentence)
  tokens = tokens[0:(max_seq_length-2)]
  tokens = ['[CLS]', *tokens , '[SEP]']
  if len(tokens) < max_seq_length:
    for i, token in enumerate(tokens):
      while len(tokens) != max_seq_length:
        tokens.insert(-1, '[PAD]')

  if len(tokens) > max_seq_length:
    for i, token in enumerate(tokens):
      while len(tokens) != max_seq_length:
        del tokens[-2]

  tokens_list.append(np.array(tokenizer.convert_tokens_to_ids(copy.
↪deepcopy(tokens))))
```

```
  for i, token in enumerate(tokens):
    if token == '[PAD]':
      tokens[i] = 0
    else:
      tokens[i] = 1

  masks_list.append(np.array(copy.deepcopy(tokens)))

  for i, token in enumerate(tokens):
    tokens[i] = 0

  segments_list.append(np.array(copy.deepcopy(tokens)))
```

[ ]:
```
tokens_list = np.array(tokens_list)
masks_list = np.array(masks_list)
segments_list = np.array(segments_list)
```

0.3 Embeddings from BERT

[ ]:
```
# dependencies = {
#     'auroc': auroc
# }
```

[ ]:
```
# bert_model = tf.keras.models.load_model('/content/gdrive/MyDrive/Colab␣
 ↪Notebooks/NLP Transfer Learning BERT/NLP Transfer Learning/bert_model.h5',
#                                         dependencies)
```

[ ]:
```
X_test=bert_model.predict([tokens_list,masks_list,segments_list])
```

[ ]:
```
X_test.shape
```

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

[ ]:
```
test_predicted_prob = model.predict(X_test)
test_pred = []
```

[ ]:
```
for item in test_predicted_prob:
  if item[1] > item[0]:
    test_pred.append(1)
  else:
    test_pred.append(0)
```

[ ]:
```
data = {"Text":df['Text'], "label":test_pred}
```

[ ]:
```
df_pred = pd.DataFrame(data)
```

[ ]:
```
count_1 = 0
count_0 = 0

for index, row in df_pred.iterrows():
    if row['label'] == 0:
      count_0 += 1
    else:
      count_1 += 1
```

0.4 Class label count

[ ]:
```
print("0 --> {}".format(count_0))
print("1 --> {}".format(count_1))
```

```
0 --> 40
1 --> 312
```

1 Procedure

taking sentences which has word length < 50

maximum input sequence length $= 512$ for bert

inputs to BERT to get vector representaion of each sentence. 1. tokens

1. only separing words
2. SEP is used for separating out multiple sentences in input to BERT.
3. CLS for start of a input.
4. segment
5. for each sentence we will have a number to it's respective words.
6. single Input (0,0,0,0)
7. 2 sentence input, we will have: e.g (0,0,0,0,0,0, 1, 1, 1)
8. masking
9. way to tell which token to consider and which token to ignore.
10. 1=consider, 0=do not consider

from bert model, we need to get embeddings 1. token_embedding 1. vector representation of each word in a sentence
2. shape (max_len, 768)

2. segment embedding
    1. to separate out multiple text input
    2. e.g if 2 inputs at a time then 0,1
    3. shape (max_len, 768)

3. Masking
    1. (shape=(max_len, 768))
4. then add all these embeddings element wise (max_len, 768)(positional) + (max_len, 768)(segment) + (max_len, 768)(token)
5. we will have (1, max_len, 768) ebedding for each input.

feeding my model with this input(embeding)

1. more dense and dropout layer i will add, model becomes more overfit
2. more units of neurons i will add model becomes accuracy
3. more dropout i will add, model looses accuracy

[ ]: