In [ ]:
```
pip install transformers
```

In [1]:
```
import subprocess
from ast import literal_eval

def run(command):
    process = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE)
    out, err = process.communicate()
    print(out.decode('utf-8').strip())

print('# CPU')
run('cat /proc/cpuinfo | egrep -m 1 "^model name"')
run('cat /proc/cpuinfo | egrep -m 1 "^cpu MHz"')
run('cat /proc/cpuinfo | egrep -m 1 "^cpu cores"')

print('# RAM')
run('cat /proc/meminfo | egrep "^MemTotal"')

print('# GPU')
run('lspci | grep VGA')

print('# OS')
run('uname -a')
```

```
# CPU



# RAM

# GPU

# OS
```

In [2]:
```
import pandas as pd
import numpy as np
import sklearn
from sklearn.model_selection import GroupKFold
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
# import tensorflow_hub as hub
import tensorflow as tf
# import bert_tokenization as tokenization
import tensorflow.keras.backend as K
import os
from scipy.stats import spearmanr
from math import floor, ceil
from transformers import *
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import seaborn as sns
import string
import re    #for regex

np.set_printoptions(suppress=True)
print(tf.__version__)
```

```
2.3.1
```

Choose model

In [3]:
```
from transformers import BertTokenizer, BertModel


MODEL_TYPE = 'bert-base-uncased'
MAX_SIZE = 200
BATCH_SIZE = 500

tokenizer = BertTokenizer.from_pretrained(MODEL_TYPE)
```

1. Read data and tokenizer

Read tokenizer and data, as well as defining the maximum sequence length that will be used for the input to Bert (maximum is usually 512 tokens)

In [4]:
```
HAS_ANS = False
training_sample_count = 1000 # 4000
training_epochs = 2 # 3
test_count = 1000
running_folds = 1 # 2
MAX_SEQUENCE_LENGTH = 300
```

original dataset

In [ ]:
```
ls
```

In [5]:
```
df = pd.read_csv('train.csv')
df = df[['text','is_humor']]
print(df)
X = df.iloc[:, 0].values
Y = df.iloc[:, 1].values

print(X)
print(Y)
```

```
                                                   text  is_humor
0     TENNESSEE: We're the best state. Nobody even c...         1
1     A man inserted an advertisement in the classif...         1
2     How many men does it take to open a can of bee...         1
3     Told my mom I hit 1200 Twitter followers. She ...         1
4     Roses are dead. Love is fake. Weddings are bas...         1
...                                                 ...       ...
7995  Lack of awareness of the pervasiveness of raci...         0
7996    Why are aspirins white? Because they work sorry         1
7997  Today, we Americans celebrate our independence...         1
7998  How to keep the flies off the bride at an Ital...         1
7999  "Each ounce of sunflower seeds gives you 37% o...         0

[8000 rows x 2 columns]
["TENNESSEE: We're the best state. Nobody even comes close. *Elevennessee w
alks into the room* TENNESSEE: Oh shit..."
 'A man inserted an advertisement in the classifieds "Wife Wanted". The nex
t day, he received 1000 of replies, all reading: "You can have mine." Free
delivery also available at your door step'
 'How many men does it take to open a can of beer? None. It should be open
by the time she brings it to the couch.'
 ...
 'Today, we Americans celebrate our independence from Britain while plannin
g our escape to Canada.'
 'How to keep the flies off the bride at an Italian wedding Keep a bucket o
f shit next to her'
 '"Each ounce of sunflower seeds gives you 37% of your daily need for vitam
in E" vitamin health']
[1 1 1 ... 1 1 0]
```

In [6]:
```
label_encoder_Y = LabelEncoder()
Y = label_encoder_Y.fit_transform(Y)
df_train = df
df_test = pd.read_csv("public_dev.csv")
df_test = df_test[['text']]

print("TRAIN Dataset -> ")
print(df_train.head())
print(" Test Dataset ->")
print(df_test.head())

df_test = df_test[:test_count]
print("Training entries: {}, test entries: {}".format(len(df_train), len(df_t
```

```
TRAIN Dataset ->
                                                text  is_humor
0  TENNESSEE: We're the best state. Nobody even c...         1
1  A man inserted an advertisement in the classif...         1
2  How many men does it take to open a can of bee...         1
3  Told my mom I hit 1200 Twitter followers. She ...         1
4  Roses are dead. Love is fake. Weddings are bas...         1
 Test Dataset ->
                                                text
0  What's the difference between a Bernie Sanders...
1     Vodka, whisky, tequila. I'm calling the shots.
2     French people don't masturbate They Jacque off
3  A lot of Suicide bombers are Muslims - I don't...
4  What happens when you fingerbang a gypsy on he...
Training entries: 8000, test entries: 1000
```

In [7]:
```
print(len(df),len(df_train),len(df_test))
display(df_train.head())
display(df_test.head())
```

```
8000 8000 1000
```

	text	is_humor
0	TENNESSEE: We're the best state. Nobody even c...	1
1	A man inserted an advertisement in the classif...	1
2	How many men does it take to open a can of bee...	1
3	Told my mom I hit 1200 Twitter followers. She ...	1
4	Roses are dead. Love is fake. Weddings are bas...	1

	text
0	What's the difference between a Bernie Sanders...
1	Vodka, whisky, tequila. I'm calling the shots.
2	French people don't masturbate They Jacque off
3	A lot of Suicide bombers are Muslims - I don't...
4	What happens when you fingerbang a gypsy on he...

In [8]:
```
print(list(df_train.columns))
```

```
['text', 'is_humor']
```

In [9]:
```
output_categories = list(df_train.columns[[1]])
input_categories = list(df_train.columns[[0]])

if HAS_ANS:
    output_categories = list(df_train.columns[11:])
    input_categories = list(df_train.columns[[1,2,5]])


TARGET_COUNT = len(output_categories)

print('\ninput categories:\n\t', input_categories)
print('\noutput TARGET_COUNT:\n\t', TARGET_COUNT)
print('\noutput categories:\n\t', output_categories)
```

```
input categories:
         ['text']

output TARGET_COUNT:
         1

output categories:
         ['is_humor']
```

```
#### 2. Preprocessing functions

These are some functions that will be used to preprocess the raw text data into
useable Bert inputs.<br>

*update 4:* credits to [Minh](https://www.kaggle.com/dathudeptrai) for this
implementation. If I'm not mistaken, it could be used directly with other
Huggingface transformers too! Note that due to the 2 x 512 input, it will
require significantly more memory when finetuning BERT.
```

In [10]:

```
def _convert_to_transformer_inputs(title, question, answer, tokenizer, max_se
    """Converts tokenized input to ids, masks and segments for transformer (i

    def return_id(str1, str2, truncation_strategy, length):

        inputs = tokenizer.encode_plus(str1, str2,
            add_special_tokens=True,
            max_length=length,
            truncation_strategy=truncation_strategy)

        input_ids =  inputs["input_ids"]
        input_masks = [1] * len(input_ids)
        input_segments = inputs["token_type_ids"]
        padding_length = length - len(input_ids)
        padding_id = tokenizer.pad_token_id
        input_ids = input_ids + ([padding_id] * padding_length)
        input_masks = input_masks + ([0] * padding_length)
        input_segments = input_segments + ([0] * padding_length)

        return [input_ids, input_masks, input_segments]

    input_ids_q, input_masks_q, input_segments_q = return_id(
        title, None, 'longest_first', max_sequence_length)

    input_ids_a, input_masks_a, input_segments_a = return_id(
        '', None, 'longest_first', max_sequence_length)

    return [input_ids_q, input_masks_q, input_segments_q,
            input_ids_a, input_masks_a, input_segments_a]

def compute_input_arrays(df, columns, tokenizer, max_sequence_length):
    input_ids_q, input_masks_q, input_segments_q = [], [], []
    input_ids_a, input_masks_a, input_segments_a = [], [], []
    for _, instance in tqdm(df[columns].iterrows()):
        t, q, a = instance.text, instance.text, instance.text

        ids_q, masks_q, segments_q, ids_a, masks_a, segments_a = \
        _convert_to_transformer_inputs(t, q, a, tokenizer, max_sequence_lengt

        input_ids_q.append(ids_q)
        input_masks_q.append(masks_q)
        input_segments_q.append(segments_q)
        input_ids_a.append(ids_a)
        input_masks_a.append(masks_a)
        input_segments_a.append(segments_a)

    return [np.asarray(input_ids_q, dtype=np.int32),
            np.asarray(input_masks_q, dtype=np.int32),
            np.asarray(input_segments_q, dtype=np.int32),
            np.asarray(input_ids_a, dtype=np.int32),
            np.asarray(input_masks_a, dtype=np.int32),
            np.asarray(input_segments_a, dtype=np.int32)]

def compute_output_arrays(df, columns):
    return np.asarray(df[columns])
```

In [11]:
```
outputs = compute_output_arrays(df_train, output_categories)
inputs = compute_input_arrays(df_train, input_categories, tokenizer, MAX_SEQU
test_inputs = compute_input_arrays(df_test, input_categories, tokenizer, MAX_
```

```
HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info', l
ayout=Layout(width='20px'), max=1.0…

Truncation was not explicitly activated but `max_length` is provided a spec
ific value, please use `truncation=True` to explicitly truncate examples to
max length. Defaulting to 'longest_first' truncation strategy. If you encod
e pairs of sequences (GLUE-style) with the tokenizer you can select this st
rategy more precisely by providing a specific strategy to `truncation`.


HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info', l
ayout=Layout(width='20px'), max=1.0…
```

3. Create model

~~`compute_spearmanr()`~~ `mean_squared_error` is used to compute the competition metric for the validation set

`create_model()` contains the actual architecture that will be used to finetune BERT to our dataset.

In [12]:
```
def compute_spearmanr_ignore_nan(trues, preds):
    rhos = []
    for tcol, pcol in zip(np.transpose(trues), np.transpose(preds)):
        rhos.append(spearmanr(tcol, pcol).correlation)
    return np.nanmean(rhos)

def create_model():
    q_id = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)
    a_id = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)

    q_mask = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)
    a_mask = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)

    q_atn = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)
    a_atn = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)

    config = BertConfig() # print(config) to see settings
    config.output_hidden_states = False # Set to True to obtain hidden states
    # caution: when using e.g. XLNet, XLNetConfig() will automatically use xl

    bert_model = TFBertModel.from_pretrained('bert-base-uncased', config=conf

    # if config.output_hidden_states = True, obtain hidden states via bert_mo
    q_embedding = bert_model(q_id, attention_mask=q_mask, token_type_ids=q_at
    a_embedding = bert_model(a_id, attention_mask=a_mask, token_type_ids=a_at

    q = tf.keras.layers.GlobalAveragePooling1D()(q_embedding)
    a = tf.keras.layers.GlobalAveragePooling1D()(a_embedding)

#     x = tf.keras.layers.Concatenate()([q, q])

    x = tf.keras.layers.Dropout(0.2)(q)

    x = tf.keras.layers.Dense(TARGET_COUNT, activation='sigmoid')(x)

    model = tf.keras.models.Model(inputs=[q_id, q_mask, q_atn, ], outputs=x)

    return model
```

4. Obtain inputs and targets, as well as the indices of the train/validation splits

In [13]:
```
output_categories
```

Out[13]: ['is_humor']

5. Training, validation and testing

Loops over the folds in gkf and trains each fold for 3 epochs --- with a learning rate of 3e-5 and batch_size of 6. A simple binary crossentropy is used as the objective-/loss-function.

In [14]:
```
# Evaluation Metrics
import sklearn
def print_evaluation_metrics(y_true, y_pred, label='', is_regression=True, la
    print('==================', label2)
    ### For regression
    if is_regression:
        print('mean_absolute_error',label,':', sklearn.metrics.mean_absolute_
        print('mean_squared_error',label,':', sklearn.metrics.mean_squared_er
        print('r2 score',label,':', sklearn.metrics.r2_score(y_true, y_pred))
        #     print('max_error',label,':', sklearn.metrics.max_error(y_true,
        return sklearn.metrics.mean_squared_error(y_true, y_pred)
    else:
        ### FOR Classification
        print('balanced_accuracy_score',label,':', sklearn.metrics.balanced_a
        print('average_precision_score',label,':', sklearn.metrics.average_pr
        print('balanced_accuracy_score',label,':', sklearn.metrics.balanced_a
        print('accuracy_score',label,':', sklearn.metrics.accuracy_score(y_tr
        print('f1_score',label,':', sklearn.metrics.f1_score(y_true, y_pred))

        matrix = sklearn.metrics.confusion_matrix(y_true, y_pred)
        print(matrix)
        TP,TN,FP,FN = matrix[1][1],matrix[0][0],matrix[0][1],matrix[1][0]
        Accuracy = (TP+TN)/(TP+FP+FN+TN)
        Precision = TP/(TP+FP)
        Recall = TP/(TP+FN)
        F1 = 2*(Recall * Precision) / (Recall + Precision)
        print('Acc', Accuracy, 'Prec', Precision, 'Rec', Recall, 'F1',F1)
        return sklearn.metrics.accuracy_score(y_true, y_pred)

print_evaluation_metrics([1,0], [0.9,0.1], '', True)
print_evaluation_metrics([1,0], [1,1], '', False)
```

```
==================
mean_absolute_error  : 0.09999999999999999
mean_squared_error  : 0.009999999999999998
r2 score  : 0.96
==================
balanced_accuracy_score  : 0.5
average_precision_score  : 0.5
balanced_accuracy_score  : 0.5
accuracy_score  : 0.5
f1_score  : 0.6666666666666666
[[0 1]
 [0 1]]
Acc 0.5 Prec 0.5 Rec 1.0 F1 0.6666666666666666
```

Out[14]: 0.5

Loss function selection

Regression problem between 0 and 1, so binary_crossentropy and mean_absolute_error seem good.

Here are the explanations: https://www.dlology.com/blog/how-to-choose-last-layer-activation-and-loss-function/ (https://www.dlology.com/blog/how-to-choose-last-layer-activation-and-loss-function/)

In [15]:
```
min_acc = 1000000
min_test = []
valid_preds = []
test_preds = []
best_model = False
for LR in np.arange(1e-5, 2e-5, 3e-5).tolist():
    print('>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>')
    print('LR=', LR)
    gkf = GroupKFold(n_splits=5).split(X=df_train.text, groups=df_train.text)

    for fold, (train_idx, valid_idx) in enumerate(gkf):
        if fold not in range(running_folds):
            continue
        train_inputs = [(inputs[i][train_idx])[:training_sample_count] for i
        train_outputs = (outputs[train_idx])[:training_sample_count]

        valid_inputs = [inputs[i][valid_idx] for i in range(len(inputs))]
        valid_outputs = outputs[valid_idx]

        print(np.array(train_inputs).shape, np.array(train_outputs).shape)
#         print(train_idx[:10], valid_idx[:10])

        K.clear_session()
        model = create_model()
        optimizer = tf.keras.optimizers.Adam(learning_rate=LR)
        model.compile(loss='binary_crossentropy', optimizer=optimizer)
        for xx in range(1):
            model.fit(train_inputs, train_outputs, epochs=training_epochs, ba
            # model.save_weights(f'bert-{fold}.h5')
            valid_preds.append(model.predict(valid_inputs))
            #rho_val = compute_spearmanr_ignore_nan(valid_outputs, valid_prea
            #print('validation score = ', rho_val)
            acc = print_evaluation_metrics(np.array(valid_outputs), np.array(
            if acc < min_acc:
                print('new acc >> ', acc)
                min_acc = acc
                best_model = model
#                 min_test = model.predict(test_inputs)
#                 test_preds.append(min_test)
            print(' ')
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
LR= 1e-05
(6, 1000, 300) (1000, 1)
```

```
Some layers from the model checkpoint at bert-base-uncased were not used
when initializing TFBertModel: ['nsp___cls', 'mlm___cls']
- This IS expected if you are initializing TFBertModel from the checkpoin
t of a model trained on another task or with another architecture (e.g. i
nitializing a BertForSequenceClassification model from a BertForPreTraini
ng model).
- This IS NOT expected if you are initializing TFBertModel from the check
point of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification
model).
All the layers of TFBertModel were initialized from the model checkpoint
at bert-base-uncased.
```

```
If your task is similar to the task the model of the checkpoint was train
ed on, you can already use TFBertModel for predictions without further tr
aining.

Epoch 1/5
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/ber
t/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mi
nimizing the loss.
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/ber
t/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mi
nimizing the loss.
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/ber
t/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mi
nimizing the loss.
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/ber
t/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when mi
nimizing the loss.
32/32 [==============================] - 1809s 57s/step - loss: 0.5794
Epoch 2/5
32/32 [==============================] - 1764s 55s/step - loss: 0.2468
Epoch 3/5
32/32 [==============================] - 1799s 56s/step - loss: 0.1068
Epoch 4/5
32/32 [==============================] - 1843s 58s/step - loss: 0.0420
Epoch 5/5
32/32 [==============================] - 1467s 46s/step - loss: 0.0116
==================
mean_absolute_error on #1 : 0.09396692630552934
mean_squared_error on #1 : 0.0735357907604336
r2 score on #1 : 0.6849682639069455
new acc >>  0.0735357907604336
```

In [16]:
```
best_model.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected
to
==========================================================================================
input_1 (InputLayer)            [(None, 300)]        0
__________________________________________________________________________________________
input_3 (InputLayer)            [(None, 300)]        0
__________________________________________________________________________________________
input_5 (InputLayer)            [(None, 300)]        0
__________________________________________________________________________________________
tf_bert_model (TFBertModel)     TFBaseModelOutputWit 109482240   input_1[0]
[0]
                                                                 input_3[0]
[0]
                                                                 input_5[0]
[0]
__________________________________________________________________________________________
global_average_pooling1d (Globa (None, 768)          0           tf_bert_mo
del[0][0]
__________________________________________________________________________________________
dropout_37 (Dropout)            (None, 768)          0           global_ave
rage_pooling1d[0][0]
__________________________________________________________________________________________
dense (Dense)                   (None, 1)            769         dropout_37
[0][0]
==========================================================================================
Total params: 109,483,009
Trainable params: 109,483,009
Non-trainable params: 0
__________________________________________________________________________________________
```

In [17]:
```
print('best acc >> ', acc)
```

```
best acc >>  0.0735357907604336
```

In [18]:
```
len(valid_inputs[0])
```

Out[18]: 1600

In [19]:
```
print(valid_outputs.shape, valid_preds[-1].shape)
print_evaluation_metrics(np.array(valid_outputs), np.array(valid_preds[-1]),
```

```
(1600, 1) (1600, 1)
==================
mean_absolute_error  : 0.09396692630552934
mean_squared_error  : 0.0735357907604336
r2 score  : 0.6849682639069455
```

Out[19]:
```
0.0735357907604336
```

In [20]:
```
# %%time
min_test = best_model.predict(test_inputs)
```

Regression submission

In [22]:
```
df_sub = df_test.copy()
# df_sub['pred'] = np.average(test_preds, axis=0) # for weighted average set
df_sub['pred'] = min_test

print(df_sub)
# df_sub.to_csv('sub10.csv', index=False)
```

```
                                                text      pred
0    What's the difference between a Bernie Sanders...  0.999945
1       Vodka, whisky, tequila. I'm calling the shots.  0.889932
2       French people don't masturbate They Jacque off  0.999590
3    A lot of Suicide bombers are Muslims - I don't...  0.999841
4    What happens when you fingerbang a gypsy on he...  0.999941
..                                                 ...       ...
995  boss: what are you doing inventor of the bagpi...  0.999972
996  I told him his views were pretty extreme and i...  0.014071
997  "Mum, all the black kids call each other Nigga...  0.999093
998  In honor of Fathers Day, I'm gonna bring you "...  0.001903
999  I don't know why Coca-Cola and Pepsi are fight...  0.999948

[1000 rows x 2 columns]
```

In [23]:
```
for i in range(len(min_test)):
    min_test[i] = min_test[i] * 4
df_sub['humor_rating'] = min_test
```

Binary submission

In [25]:
```
for split in np.arange(0.1, 0.80, 0.1).tolist():
    df_sub['is_humor'] = (df_sub['pred'] > split)


df_sub.to_csv('colbert_test.csv', index=False)
df_sub.head()
```

Out[25]:

	text	pred	humor_rating	is_humor
0	What's the difference between a Bernie Sanders...	0.999945	3.999778	True
1	Vodka, whisky, tequila. I'm calling the shots.	0.889932	3.559730	True
2	French people don't masturbate They Jacque off	0.999590	3.998360	True
3	A lot of Suicide bombers are Muslims - I don't...	0.999841	3.999364	True
4	What happens when you fingerbang a gypsy on he...	0.999941	3.999764	True

In [ ]:
```

```