In [1]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
```

Text Classification

In [2]:

```
train = pd.read_csv("C:\\Users\\Acer\\OneDrive\\Desktop\\MKA 9 th Sept. Batch\\Deep
```

In [3]:

```
test = pd.read_csv("C:\\Users\\Acer\\OneDrive\\Desktop\\MKA 9 th Sept. Batch\\Deep L
```

In [4]:

```
print(train.shape)

print(test.shape)
```

```
(77946, 28)
(42157, 4)
```

In [5]:

```
def _load_and_shuffle_data(data_path,
                           file_name,
                           cols,
                           seed,
                           separator=',',
                           header=0):

    np.random.seed(seed)
    data_path = os.path.join(data_path, file_name)
    data = pd.read_csv(data_path, usecols=cols, sep=separator, header=header)
    return data.reindex(np.random.permutation(data.index))

def _split_training_and_validation_sets(texts, labels, validation_split):

    num_training_samples = int((1 - validation_split) * len(texts))
    return ((texts[:num_training_samples], labels[:num_training_samples]),
            (texts[num_training_samples:], labels[num_training_samples:]))

def load_tweet_weather_topic_classification_dataset(data_path,
                                                    validation_split=0.2,
                                                    seed=123):

    columns = [1] + [i for i in range(13, 28)]  # 1 - text, 13-28 - topics.
    data = _load_and_shuffle_data(data_path, 'train.csv', columns, seed)

    # Get tweet text and the max confidence score for the weather types.
    texts = list(data['tweet'])
    weather_data = data.iloc[:, 1:]

    labels = []
    for i in range(len(texts)):
        # Pick topic with the max confidence score.
        labels.append(np.argmax(list(weather_data.iloc[i, :].values)))

    return _split_training_and_validation_sets(
        texts, np.array(labels), validation_split)
```

In [6]:

```
from sklearn.model_selection import train_test_split
```

In [9]:

```
import os
for dirname, _, filenames in os.walk('/content/crowdflower-weather-twitter.zip'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

```

In [10]:

```
(train_data, train_labels),(test_data, test_labels)=load_tweet_weather_topic_classif
```

Tokenize and vectorize

In [11]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [12]:

```
#i am mouse - i, am, mouse, i am, am mouse

vectorizer = TfidfVectorizer(dtype='float32',
                             ngram_range=(1,2),
                             strip_accents='unicode',
                             decode_error='replace',
                             analyzer='word',
                             min_df=2)
```

In [13]:

```
x_train = vectorizer.fit_transform(train_data)
x_train.shape
```

```
C:\Users\Acer\anaconda37\lib\site-packages\sklearn\feature_extraction\tex
t.py:2060: UserWarning: Only (<class 'numpy.float64'>, <class 'numpy.float
32'>, <class 'numpy.float16'>) 'dtype' should be used. float32 'dtype' wil
l be converted to np.float64.
  warnings.warn(
```

Out[13]:

```
(62356, 85670)
```

In [14]:

```
x_test = vectorizer.transform(test_data)
```

In [15]:

```
#feature selection
selector = SelectKBest(f_classif, k=20000)
selector.fit(x_train, train_labels)
```

Out[15]:

```
SelectKBest(k=20000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]:

```
x_train = selector.transform(x_train)
x_test = selector.transform(x_test)
```

In [17]:

```
x_train = x_train.toarray()
x_test = x_test.toarray()
```

In [18]:

```
x_train.shape
```

Out[18]:

```
(62356, 20000)
```

In [19]:

```
x_test.shape
```

Out[19]:

```
(15590, 20000)
```

In [20]:

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

Neural network

ANN

In [21]:

```
from keras.models import Sequential
```

In [22]:

```
from keras.layers import Dense, Flatten, Dropout
```

Model creation - first attempt

Note: each text has 20000 features

In [23]:

```
weatherANN = Sequential()
```

In [24]:

```
weatherANN.add(Flatten())
weatherANN.add(Dense(units=512, activation='relu'))
#Add dropout regularization - a percentage of units should be set to 0. Eg: 0.25 - s
#be set to 0
weatherANN.add(Dropout(rate=0.25))
weatherANN.add(Dense(units=1, activation='sigmoid'))
```

In [25]:

```
weatherANN.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
```

In [26]:

```
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
filename = 'bestmodel_trial1.h5'
checkpoint = ModelCheckpoint(filename, monitor='val_accuracy', verbose=1, save_best_
#if monitored value doesnt improve for #patience epochs, then stop training
es = EarlyStopping(monitor='val_loss', patience=10)
#if monitored value doesnt improve for #patience epochs, LR(new) = LR(old)*factor
rd = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5)
```

In [27]:

```
history = weatherANN.fit(x_train, y_train, epochs=5, validation_split=0.2, callbacks
```

```
Epoch 1/5
1559/1559 [==============================] - ETA: 0s - loss: -7202.0425 -
accuracy: 0.0916
Epoch 1: val_accuracy improved from -inf to 0.09469, saving model to bestm
odel_trial1.h5
1559/1559 [==============================] - 181s 115ms/step - loss: -720
2.0425 - accuracy: 0.0916 - val_loss: -21491.5469 - val_accuracy: 0.0947 -
lr: 0.0010
Epoch 2/5
1559/1559 [==============================] - ETA: 0s - loss: -47311.9570 -
accuracy: 0.0916
Epoch 2: val_accuracy did not improve from 0.09469
1559/1559 [==============================] - 180s 115ms/step - loss: -4731
1.9570 - accuracy: 0.0916 - val_loss: -78290.1250 - val_accuracy: 0.0947 -
lr: 0.0010
Epoch 3/5
1559/1559 [==============================] - ETA: 0s - loss: -117731.4453
- accuracy: 0.0916
Epoch 3: val_accuracy did not improve from 0.09469
1559/1559 [==============================] - 173s 111ms/step - loss: -1177
31.4453 - accuracy: 0.0916 - val_loss: -161819.2031 - val_accuracy: 0.0947
- lr: 0.0010
Epoch 4/5
1559/1559 [==============================] - ETA: 0s - loss: -213001.5000
- accuracy: 0.0916
Epoch 4: val_accuracy did not improve from 0.09469
1559/1559 [==============================] - 185s 119ms/step - loss: -2130
01.5000 - accuracy: 0.0916 - val_loss: -269040.9688 - val_accuracy: 0.0947
- lr: 0.0010
Epoch 5/5
1559/1559 [==============================] - ETA: 0s - loss: -331133.1875
- accuracy: 0.0916
Epoch 5: val_accuracy did not improve from 0.09469
1559/1559 [==============================] - 185s 119ms/step - loss: -3311
33.1875 - accuracy: 0.0916 - val_loss: -398364.3750 - val_accuracy: 0.0947
- lr: 0.0010
```

Save the best model (W, b) by comparing accuracy/loss on validation set

In [29]:

```
from keras import models
#human accuracy - 1.0000
#training accuracy: 0.0916
#val_accuracy did not improve from 0.9469
#Avoidable bias = 1-0.091 = 0.1
#Variance = 1 - 0.94 = 0.16
finalmodel = models.load_model('bestmodel_trial1.h5')
```

loading best model based on val_accuracy

evaluate on test set

In [30]:

```
finalmodel.evaluate(x_test, y_test)
```

```
488/488 [==============================] - 5s 10ms/step - loss: -21934.714
8 - accuracy: 0.0921
```

Out[30]:

```
[-21934.71484375, 0.09211032837629318]
```

Plots

In [31]:

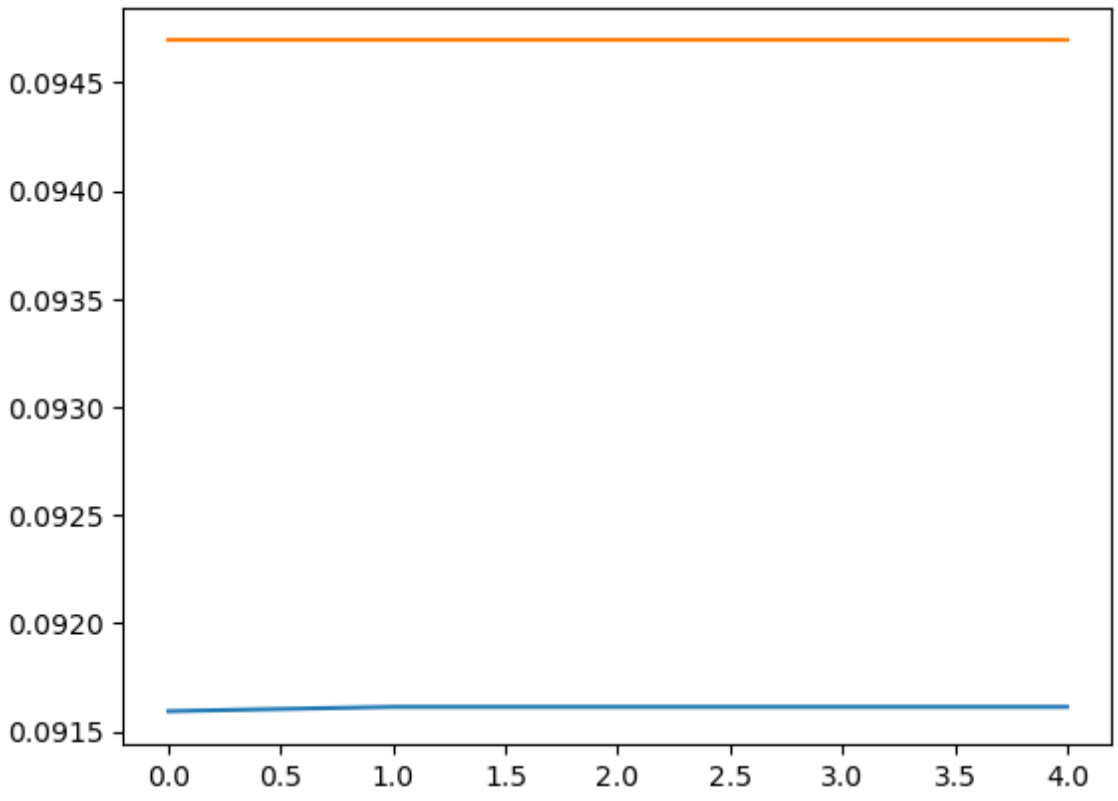
```
import matplotlib.pyplot as plt

```

In [38]:

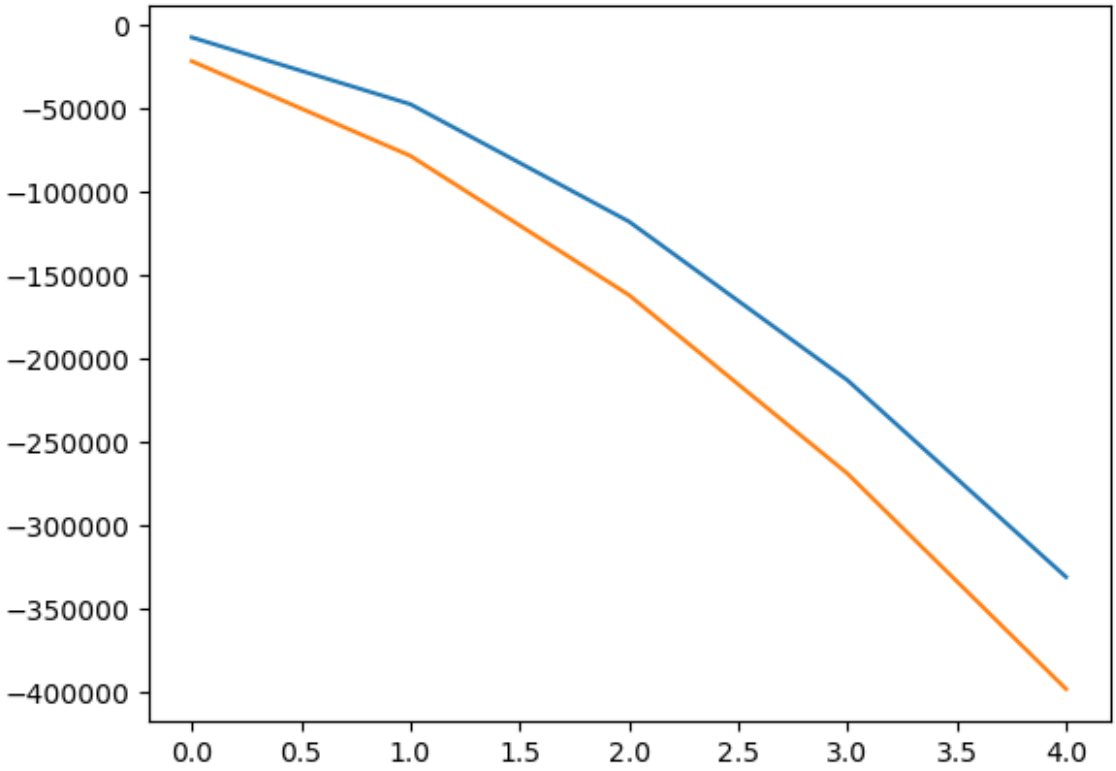
```
#plot accuracy with epochs - should increase with epochs

#training accuracy
plt.plot(history.history['accuracy'])
#validation accuracy
plt.plot(history.history['val_accuracy'])
plt.show()
```



In [33]:

```
#plot loss
import matplotlib.pyplot as plt
#training loss
plt.plot(history.history['loss'])
#validation loss
plt.plot(history.history['val_loss'])
plt.show()
```



Deployment

In [42]:

```
comment = ['At the park with erica, enjoying the weather. :)']

X_deployment = vectorizer.transform(comment)

X_deployment_best = selector.transform(X_deployment)

X_deployment_best_array = X_deployment_best.toarray()
```

In [44]:

```
finalmodel.predict(X_deployment_best_array)
```

```
1/1 [==============================] - 0s 128ms/step
```

Out[44]:

```
array([[1.]], dtype=float32)
```

In [ ]:

```
1
```