CS6005
Deep Learning
Mini Project – NLP
Sentiment analysis on Tweets from GOP debate

Date: 21/05/2021

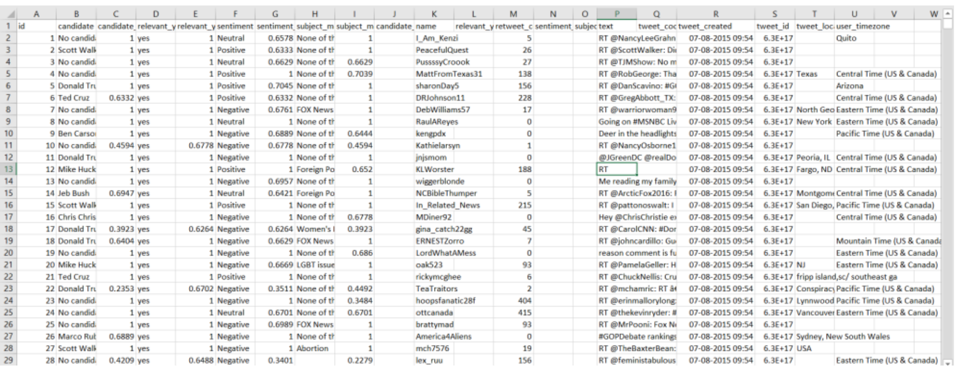
Sayf Hussain Z
2018103059
P - Batch

Problem statement

The aim of the project is to perform sentiment analysis on tweets on the First GOP debate and to classify the tweets as either positive or negative tweets. For this purpose, a suitable dataset was identified, preprocessed to take only the essential features from the tweets and a model was trained.

Dataset Details

The dataset consists of around 17,728 tweets of sentiments positive, neutral and negative [1]. The dataset contains information like origin of tweet, userid of tweet, time of tweet, user identification number and the unprocessed tweet along with the sentiment behind the tweet. From this dataset only the tweet and its sentiment were considered for the model and the other attributes were dropped.



id	candidate	candidate_	relevant_y	relevant_y	sentiment	sentiment_	subject_m	subject_m	candidate_	name	relevant_y	retweet_c	sentiment_	subjec	text	tweet_coo	tweet_created	tweet_id	tweet_loc	user_timezone
1	No candid	1	yes	1	Neutral	0.6578	None of th	1		I_Am_Kenzi		5			RT @NancyLeeGrahn		07-08-2015 09:54	6.3E+17		Quito
2	Scott Walk	1	yes	1	Positive	0.6333	None of th	1		PeacefulQuest		26			RT @ScottWalker: Di		07-08-2015 09:54	6.3E+17		
3	No candid	1	yes	1	Neutral	0.6629	None of th	0.6629		PussssyCroook		27			RT @TJMShow: No m		07-08-2015 09:54	6.3E+17		
4	No candid	1	yes	1	Positive	1	None of th	0.7039		MattFromTexas31		138			RT @RobGeorge: Tha		07-08-2015 09:54	6.3E+17	Texas	Central Time (US & Canada)
5	Donald Tru	1	yes	1	Positive	0.7045	None of th	1		sharonDay5		156			RT @DanScavino: #G		07-08-2015 09:54	6.3E+17		Arizona
6	Ted Cruz	0.6332	yes	1	Positive	0.6332	None of th	1		DRJohnson11		228			RT @GregAbbott_TX:		07-08-2015 09:54	6.3E+17		Central Time (US & Canada)
7	No candid	1	yes	1	Negative	0.6761	FOX News	1		DebWilliams57		17			RT @warriorwoman9		07-08-2015 09:54	6.3E+17	North Geo	Eastern Time (US & Canada)
8	No candid	1	yes	1	Neutral	1	None of th	1		RaulAReyes		0			Going on #MSNBC Liv		07-08-2015 09:54	6.3E+17	New York	Eastern Time (US & Canada)
9	Ben Carso	1	yes	1	Negative	0.6889	None of th	0.6444		kengpdx		0			Deer in the headlights		07-08-2015 09:54	6.3E+17		Pacific Time (US & Canada)
10	No candid	0.4594	yes	0.6778	Negative	0.6778	None of th	0.4594		Kathielarsyn		1			RT @NancyOsborne1		07-08-2015 09:54	6.3E+17		
11	Donald Tru	1	yes	1	Negative	1	None of th	1		jnjsmom		0			@JGreenDC @realDo		07-08-2015 09:54	6.3E+17	Peoria, IL	Central Time (US & Canada)
12	Mike Huck	1	yes	1	Positive	1	Foreign Po	0.652		KLWorster		188			RT		07-08-2015 09:54	6.3E+17	Fargo, ND	Central Time (US & Canada)
13	No candid	1	yes	1	Negative	0.6957	None of th	1		wiggerblonde		0			Me reading my family		07-08-2015 09:54	6.3E+17		
14	Jeb Bush	0.6947	yes	1	Neutral	0.6421	Foreign Po	1		NCBibleThumper		5			RT @ArcticFox2016: I		07-08-2015 09:54	6.3E+17	Montgom	Central Time (US & Canada)
15	Scott Walk	1	yes	1	Positive	1	None of th	1		In_Related_News		215			RT @pattonoswalt: I		07-08-2015 09:54	6.3E+17	San Diego,	Pacific Time (US & Canada)
16	Chris Chris	1	yes	1	Negative	1	None of th	0.6778		MDiner92		0			Hey @ChrisChristie ex		07-08-2015 09:54	6.3E+17		Central Time (US & Canada)
17	Donald Tru	0.3923	yes	0.6264	Negative	0.6264	Women's I	0.3923		gina_catch22gg		45			RT @CarolCNN: #Dor		07-08-2015 09:54	6.3E+17		
18	Donald Tru	0.6404	yes	1	Negative	0.6629	FOX News	1		ERNESTZorro		7			RT @johncardillo: Gu		07-08-2015 09:54	6.3E+17		Mountain Time (US & Canada
19	No candid	1	yes	1	Negative	1	None of th	0.686		LordWhatAMess		0			reason comment is fu		07-08-2015 09:54	6.3E+17		Eastern Time (US & Canada)
20	Mike Huck	1	yes	1	Negative	0.6669	LGBT issue	1		oak523		93			RT @PamelaGeller: H		07-08-2015 09:54	6.3E+17	NJ	Eastern Time (US & Canada)
21	Ted Cruz	1	yes	1	Positive	1	None of th	1		rickymcghee		6			RT @ChuckNellis: Cru		07-08-2015 09:54	6.3E+17	fripp island,sc/ southeast ga	
22	Donald Tru	0.2353	yes	0.6702	Negative	0.3511	None of th	0.4492		TeaTraitors		2			RT @mchamric: RT â€		07-08-2015 09:54	6.3E+17	Conspiracy	Pacific Time (US & Canada)
23	No candid	1	yes	1	Negative	1	None of th	0.3484		hoopsfanatic28f		404			RT @erinmallorylong:		07-08-2015 09:54	6.3E+17	Lynnwood	Pacific Time (US & Canada)
24	No candid	1	yes	1	Neutral	0.6701	None of th	0.6701		ottcanada		415			RT @thekevinryder: #		07-08-2015 09:54	6.3E+17	Vancouver	Eastern Time (US & Canada)
25	No candid	1	yes	1	Negative	0.6989	FOX News	1		brattymad		93			RT @MrPooni: Fox Ne		07-08-2015 09:54	6.3E+17		
26	Marco Rul	0.6889	yes	1	Negative	1	None of th	1		America4Aliens		0			#GOPDebate rankings		07-08-2015 09:54	6.3E+17	Sydney, New South Wales	
27	Scott Walk	1	yes	1	Negative	1	Abortion	1		mch7576		19			RT @TheBaxterBean:		07-08-2015 09:54	6.3E+17	USA	
28	No candid	0.4209	yes	0.6488	Negative	0.3401		0.2279		lex_ruu		156			RT @feministabulous		07-08-2015 09:54	6.3E+17		Eastern Time (US & Canada)

Dataset with all attributes

Modules

Reading the data:
The dataset used is 'First GOP Twitter sentiment' of which only the Tweet and its sentiment attribute were considered. The tweets had the sentiments positive, neutral and negative of which all tweets with sentiment neutral were discarded to avoid dataset imbalance. The tweets were loaded into a variable and its corresponding sentiment were loaded into another variable.

Preprocessing the data:
All tweets were first processed to replace any special symbols or numbers with blank spaces after which the strings were all converted to lowercase. Each tweet was further split into words and Porter Stemmer was used to remove suffix or prefix (like -ing, -ed) from the words. Then the words were checked to see if they were stop-words (downloaded from nltk), all stop-words were discarded from the tweet. Unique words from the dataset are then used to create a dictionary of size 10000 words. The textual tweet is then one-hot encoded where each word of the tweet is assigned the index number from the dictionary. The tweets were all padded with zero (mode=pre) to maintain consistent size of input for the model.

Model Creation:
The model used for this project is a Sequential Model since the output from each layer is used as the input for the next layer. The first layer in this model is an Embedding Layer that converts every word into a vector with 80 features. This layer is then followed by an LSTM layer with 16 nodes. The data is then passed to a Dense layer of 32 nodes with ReLU activation and a L1 regularizer with 0.01 as the regularizing parameter to prevent overfitting. Additionally, a dropout layer was also used. Finally, the output is produced by another Dense layer with 1 node that uses Sigmoid Activation.

Embedding layer is used to give a dense representation of words and their relative meaning.

Parameters used:

Parameter	Value
Loss	Binary Cross entropy
Optimizer	Adam
Learning rate	0.001
Batch size	128
Epochs	10
Input Length	20
Embedding layer features	80

Test-train split:

Sentiment	Training	Validation	Testing
Negative	6102	678	1712
Positive	1623	180	433
Total	7725	858	2145

Coding screenshots

Importing libraries

In [1]:
```
import pandas as pd
import tensorflow as tf
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
from keras.layers import Dropout
import nltk
import re
from nltk.corpus import stopwords
```

Reading the dataset, dropping irrelevant attributes and entries with neutral sentiment, one-hot encoding the sentiment attribute

In [2]:
```
df=pd.read_csv('data/Sentiment.csv')
df.head()
```

Out[2]:

	id	candidate	candidate_confidence	relevant_yn	relevant_yn_confidence	sentiment	sentiment_confidence	subject_matter	subject_matter_confidence	candi
0	1	No candidate mentioned	1.0	yes	1.0	Neutral	0.6578	None of the above	1.0000	
1	2	Scott Walker	1.0	yes	1.0	Positive	0.6333	None of the above	1.0000	
2	3	No candidate mentioned	1.0	yes	1.0	Neutral	0.6629	None of the above	0.6629	
3	4	No candidate mentioned	1.0	yes	1.0	Positive	1.0000	None of the above	0.7039	
4	5	Donald Trump	1.0	yes	1.0	Positive	0.7045	None of the above	1.0000	

5 rows × 21 columns

In [3]:
```
df = df[df.sentiment != "Neutral"]
```

In [4]:
```
X=df.drop(['candidate','candidate_confidence','relevant_yn','relevant_yn_confidence','sentiment_confidence','subject_matter','sul
X
```

Out[4]:

	id	text
1	2	RT @ScottWalker: Didn't catch the full #GOPdeb...
3	4	RT @RobGeorge: That Carly Fiorina is trending ...
4	5	RT @DanScavino: #GOPDebate w/ @realDonaldTrump...
5	6	RT @GregAbbott_TX: @TedCruz: "On my first day ...
6	7	RT @warriorwoman91: I liked her and was happy ...
...	...	...
13866	13867	RT @cappy_yarbrough: Love to see men who will ...
13867	13868	RT @georgehenryw: Who thought Huckabee exceede...
13868	13869	RT @Lrihendry: #TedCruz As President, I will a...
13869	13870	RT @JRehling: #GOPDebate Donald Trump says tha...
13870	13871	RT @Lrihendry: #TedCruz headed into the Presid...

10729 rows × 2 columns

In [5]:
```
Y=df['sentiment']
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
Y=le.fit_transform(Y)
Y
```

Out[5]:
```
array([1, 1, 1, ..., 1, 0, 1])
```

In [6]:
```
le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print(le_name_mapping)
```

```
{'Negative': 0, 'Positive': 1}
```

Setting the dictionary size to 10000 and resetting the index for the Tweets

In [7]:
```
voc_size=10000
```

In [8]:
```
X.reset_index(inplace=True)
X
```

Out[8]:

	index	id	text
0	1	2	RT @ScottWalker: Didn't catch the full #GOPdeb...
1	3	4	RT @RobGeorge: That Carly Fiorina is trending ...
2	4	5	RT @DanScavino: #GOPDebate w/ @realDonaldTrump...
3	5	6	RT @GregAbbott_TX: @TedCruz: "On my first day ...
4	6	7	RT @warriorwoman91: I liked her and was happy ...
...	...	...	...
10724	13866	13867	RT @cappy_yarbrough: Love to see men who will ...
10725	13867	13868	RT @georgehenryw: Who thought Huckabee exceede...
10726	13868	13869	RT @Lrihendry: #TedCruz As President, I will a...
10727	13869	13870	RT @JRehling: #GOPDebate Donald Trump says tha...
10728	13870	13871	RT @Lrihendry: #TedCruz headed into the Presid...

10729 rows × 3 columns

Preprocessing the tweets by getting rid of special characters, numbers, converting the tweet to lowercase, Stemming the words, removing stop-words and one-hot encoding the Tweets

In [9]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[9]: True

In [10]:
```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
corpus = []
for i in range(0, len(X)):
    review = re.sub('[^a-zA-Z]', ' ', X['text'][i])
    review = review.lower()
    review = review.split()

    review1 = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review1 = ' '.join(review1)
    corpus.append(review1)
```

In [11]:
```
onehot_repr=[one_hot(words,voc_size)for words in corpus]
```

In [12]:
```
sent_length=20
embedded_docs=pad_sequences(onehot_repr,padding='pre',maxlen=sent_length)
print(embedded_docs)
```

```
[[   0    0    0 ... 5630 1170 9066]
 [   0    0    0 ... 4551 7015 8436]
 [   0    0    0 ... 1903 5630 1170]
 ...
 [   0    0    0 ... 4861 9333 7015]
 [   0    0    0 ... 9546  581 4171]
 [   0    0    0 ... 5630 1170 2142]]
```

Building the model

In [13]:
```
embedding_vector_features=80
model=Sequential()
model.add(Embedding(voc_size,embedding_vector_features,input_length=sent_length))
model.add(LSTM(16))
model.add(Dense(32,activation='relu',kernel_regularizer=tf.keras.regularizers.l1(0.01)))
model.add(Dropout(0.2))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),metrics=['accuracy'])
print(model.summary())
model_save = ModelCheckpoint('weights.h5',
                             save_best_only = True,
                             save_weights_only = True,
                             monitor = 'val_accuracy',
                             mode = 'max', verbose = 1)
```

Converting the dataset into an numpy array and printing number of instances category wise

In [14]:
```
import numpy as np
X_final=np.array(embedded_docs)
y_final=np.array(Y)
X_final.shape,y_final.shape
```

Out[14]: ((10729, 20), (10729,))

In [15]:
```
count0=0
count1=0
for i in range(0,len(y_final)):
    if y_final[i] == 0:
        count0=count0+1
    else:
        count1=count1+1

print(count0)
print(count1)
```

```
8493
2236
```

In [16]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.20, random_state=42)
```

In [17]:
```
count0=0
count1=0
for i in range(0,len(y_train)):
    if y_train[i] == 0:
        count0=count0+1
    else:
        count1=count1+1

print(count0)
print(count1)
```

```
6780
1803
```

In [18]:
```
count0=0
count1=0
for i in range(0,len(y_test)):
    if y_test[i] == 0:
        count0=count0+1
    else:
        count1=count1+1

print(count0)
print(count1)
```

```
1713
433
```

Fitting the model on the training data

In [19]:
```
history=model.fit(X_train,y_train,validation_split=0.1,epochs=10,batch_size=128,callbacks = [model_save])
```

Visualizing the training and validation accuracy and the training and validation loss

In [20]:
```
import matplotlib.pyplot as plt
print(history.history.keys())
plt.plot(history.history['accuracy'],color='c')
plt.plot(history.history['val_accuracy'],color='m')
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'],color='c')
plt.plot(history.history['val_loss'],color='m')
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Evaluating the model on the test set

In [21]:
```
model.load_weights('./weights.h5')
count = 0
predictions = model.predict(X_test)
for i in range(len(predictions)):
    if(predictions[i]>0.5):
        predictions[i]=1
    else:
        predictions[i]=0
    if (y_test[i] == predictions[i]):
        count+=1

print("Accuracy = ", count/len(predictions) * 100.0)
```

```
Accuracy =  85.13513513513513
```

Visualizing the confusion matrix and classification report

In [22]:
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
print('Confusion Matrix')
cm = confusion_matrix(y_test, predictions)
print(cm)
print(sns.heatmap(confusion_matrix(y_test,predictions),annot=True,fmt="d"))
print(classification_report(y_test,predictions, digits=5))
```

Predicting new tweets

In [23]:
```
def predict_emotion(stri):
    review = re.sub('[^a-zA-Z]', ' ', stri)
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    onehot_repr = [one_hot(review,voc_size)]
    embed = pad_sequences(onehot_repr,padding='pre',maxlen=sent_length)
    predicti = model.predict(embed)
    if(predicti>0.5):
        predicti=1
    else:
        predicti=0
    return le.classes_[(predicti)]
```

In [24]:
```
predict_emotion("@realDonaldTrump delivered the highest ratings in the history of presidential debates.")
```

Results

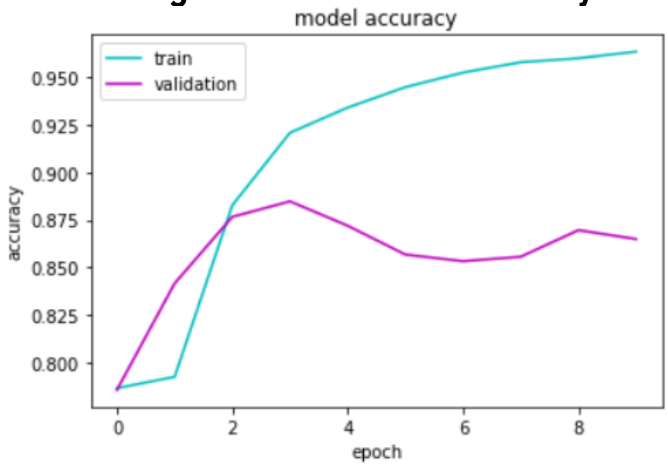
Model summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 20, 80)            800000    
_________________________________________________________________
lstm (LSTM)                  (None, 16)                6208      
_________________________________________________________________
dense (Dense)                (None, 32)                544       
_________________________________________________________________
dropout (Dropout)            (None, 32)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33        
=================================================================
Total params: 806,785
Trainable params: 806,785
Non-trainable params: 0
_________________________________________________________________
None
```

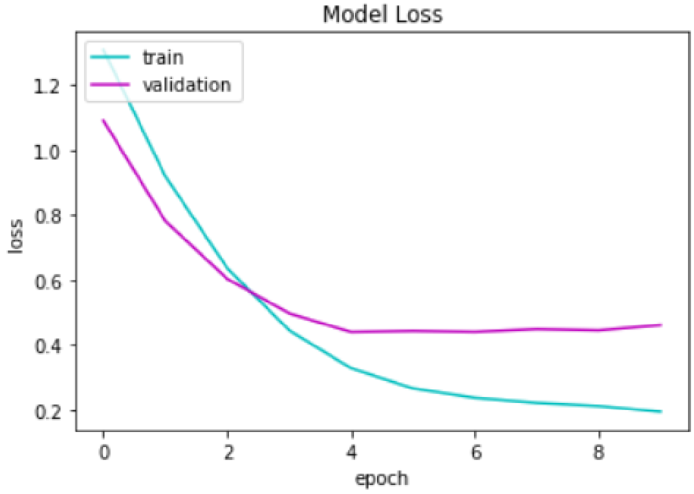
Training the model

```
Epoch 1/10
61/61 [==============================] - 4s 34ms/step - loss: 1.4323 - accuracy: 0.7707 - val_loss: 1.0900 - val_accuracy: 0.78
58

Epoch 00001: val_accuracy improved from -inf to 0.78580, saving model to weights.h5
Epoch 2/10
61/61 [==============================] - 1s 19ms/step - loss: 1.0008 - accuracy: 0.7915 - val_loss: 0.7815 - val_accuracy: 0.84
17

Epoch 00002: val_accuracy improved from 0.78580 to 0.84168, saving model to weights.h5
Epoch 3/10
61/61 [==============================] - 1s 19ms/step - loss: 0.6763 - accuracy: 0.8716 - val_loss: 0.6036 - val_accuracy: 0.87
66

Epoch 00003: val_accuracy improved from 0.84168 to 0.87660, saving model to weights.h5
Epoch 4/10
61/61 [==============================] - 1s 19ms/step - loss: 0.4731 - accuracy: 0.9209 - val_loss: 0.4975 - val_accuracy: 0.88
47

Epoch 00004: val_accuracy improved from 0.87660 to 0.88475, saving model to weights.h5
Epoch 5/10
61/61 [==============================] - 1s 20ms/step - loss: 0.3476 - accuracy: 0.9322 - val_loss: 0.4399 - val_accuracy: 0.87
19

Epoch 00005: val_accuracy did not improve from 0.88475
Epoch 6/10
61/61 [==============================] - 1s 19ms/step - loss: 0.2712 - accuracy: 0.9471 - val_loss: 0.4429 - val_accuracy: 0.85
68

Epoch 00006: val_accuracy did not improve from 0.88475
Epoch 7/10
61/61 [==============================] - 1s 19ms/step - loss: 0.2330 - accuracy: 0.9531 - val_loss: 0.4401 - val_accuracy: 0.85
33

Epoch 00007: val_accuracy did not improve from 0.88475
Epoch 8/10
61/61 [==============================] - 1s 20ms/step - loss: 0.2178 - accuracy: 0.9614 - val_loss: 0.4489 - val_accuracy: 0.85
56
Epoch 00008: val_accuracy did not improve from 0.88475
Epoch 9/10
61/61 [==============================] - 1s 19ms/step - loss: 0.2122 - accuracy: 0.9608 - val_loss: 0.4451 - val_accuracy: 0.86
96

Epoch 00009: val_accuracy did not improve from 0.88475
Epoch 10/10
61/61 [==============================] - 1s 19ms/step - loss: 0.1976 - accuracy: 0.9634 - val_loss: 0.4615 - val_accuracy: 0.86
50

Epoch 00010: val_accuracy did not improve from 0.88475
```

Model Training and Validation accuracy vs Epochs



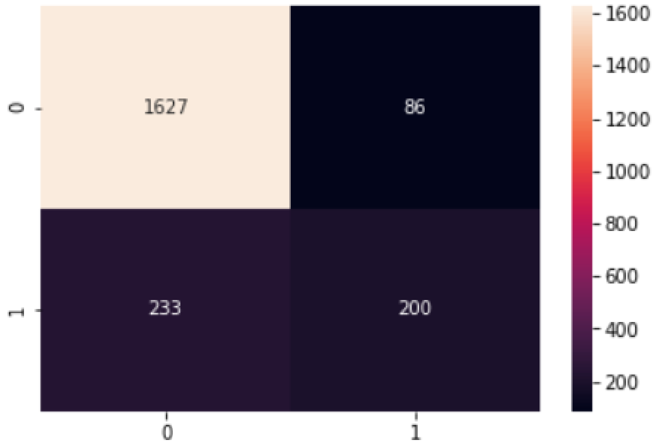
Model Training and validation loss vs epochs



Evaluating on test set

```
Accuracy =  85.13513513513513
```

Confusion Matrix for test set



Classification report for test set

	precision	recall	f1-score	support
0	0.87473	0.94980	0.91072	1713
1	0.69930	0.46189	0.55633	433
accuracy			0.85135	2146
macro avg	0.78702	0.70584	0.73352	2146
weighted avg	0.83933	0.85135	0.83921	2146

New prediction output

```
In [23]: def predict_emotion(stri):
             review = re.sub('[^a-zA-Z]', ' ', stri)
             review = review.lower()
             review = review.split()
             review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
             review = ' '.join(review)
             onehot_repr = [one_hot(review,voc_size)]
             embed = pad_sequences(onehot_repr,padding='pre',maxlen=sent_length)
             predicti = model.predict(embed)
             if(predicti>0.5):
                 predicti=1
             else:
                 predicti=0
             return le.classes_[(predicti)]

In [24]: predict_emotion("@realDonaldTrump delivered the highest ratings in the history of presidential debates.")

Out[24]: 'Positive'
```

Conclusion

As seen from the output screenshots, the model performs well with the dataset. It managed to score a Training Accuracy of 96.34% in 10 epochs of training. The model achieved the best validation accuracy of 88.47% in the 4th epoch of training and this set of weights were saved. The callback function monitored the validation accuracy and the model at the 4th epoch was saved as it has the highest validation accuracy (of 88.47%). On testing, a Testing Accuracy of 85.13% was achieved.

References

[1] https://www.kaggle.com/crowdflower/first-gop-debate-twitter-sentiment