CS6005 DEEP LEARNING

MINI PROJECT 3 – 21/05/2021

Real and Fake News Detection with NLP

Name: Reuel Samuel Sam
Registration Number: 2018103053
Batch: P

NOTE: Link to Github Page: https://github.com/ReuelSam/Deep-Learning-Project---NLP

Abstract – Natural Language Processing (NLP) is a subfield of linguistics, computer science and artificial intelligence concerned with the interactions between computers and human language. This field focuses on processing and analyzing large amounts of natural language data. Processing of twitter data to infer sentiment or classification is one of the more popular aspects of NLP. This project focuses mainly on binary classification via detection of news as fake or real based on twitter data. The dataset used contains 7613 labeled tweets that have been split into training (6090) and testing data (1523). The model proposed was able to achieve a testing accuracy of 81.22%.

I. PROBLEM STATEMENT AND OVERVIEW

This project aims to tackle the NLP problem regarding the classification of tweets into Real News or Fake News. The dataset for this project is a collection of around 7000 tweets that are labeled as real or fake. The model proposed for this classification uses an Embedding and LSTM layer. These layers are newly applied concepts in relation to the previous projects. Apart from the model, due to the nature of text data, for analysis by a computer, a need for preprocessing the data was identified. This has been done by tokenizing the tweets after lemmatization and removal of stop words. Due to the difference in lengths in the tweets, the tweets have been padded to have equal length. An additional feature

P.T.O

II. DATASET DETAILS

This dataset was obtained after the cleaning and fine tuning of an existing dataset that has been used as the NLP Getting Started dataset competition on Kaggle [1]. This dataset has mainly 2 files – test and train – both comma-separated files. However, the test data does not have target labels and therefore cannot be used to create a confusion matrix.

As a result, this project has made use of only the train file. This file contains 7613 tweets that have been split into 6090 training examples and 1523 testing instances. The training set is further split to give a validation set during training with a factor of 0.2. The test file contains 5 columns – id, text, location, keyword and target. For the purposes of this project, the columns id, location and keyword are dropped since they do not affect training and classification.

This being a binary problem, there are only two target values:

- 1: Real News
- 2: Fake News

III. MODULES

Python comes with a rich library of modules that make coding rather convenient. This is no different when it comes to Deep Learning. Python has plenty of Deep Learning Libraries that can be made use of. Following are the modules that have been imported and applied in this project that are necessary for Natural Language Processing:

- *TensorFlow:* Open Source Machine Learning Platform
- *Keras:* Open Source Library that provides python interface for artificial neural networks
- *Matplotlib:* Graphical Library (Not necessarily required)
- *Seaborn:* Graphical Library
- *SKLearn:* Machine Library
- *Numpy:* Mathematical Library that supports array operations
- *Pandas:* Data Manipulation and analysis library
- *NLTK:* NLP Library for symbolic and statistical processing for English
- *Re:* Mathematical Library for the use of Regular Expressions
- *Textblob:* Library for processing textual data

Apart from Python modules, the code can be broken down into a set of modules:

- **Imports:** This section of the code is meant for processing all the imports of the necessary libraries for the execution of this project

- **Loading of Dataset:** The dataset has been loaded from Kaggle's rich collection of machine learning datasets. As explained in Section 2, only the Train data has been made use of and imported.

- **Preprocessing of Data:** Being textual data, plenty of preprocessing is required for this dataset. The preprocessing can be further broken down into more subdivisions as follows:
  - *Dropping of columns:* Columns that do not contribute to the results are removed (id, location, keywords)
  - *Text Processing:* Here, all tweets are first converted to lower case and special characters and unnecessary spaces are removed. Following this, English stop words are removed (taken from nltk). The tweets are then lemmatized to remove unnecessary suffixes and tense formed words.
  - *Tokenizing:* Since computers cannot process text data as it is, the tweets need to be tokenized, as in converted to numbers, for it to be processed. This is done by mapping the words in a dictionary and replacing the words with the mapped key. SKLearn's tokenizer was used to achieve this. Only the top 10000 words are considered in this dictionary.
  - *Padding:* For machine learning models, it is preferred to have equally sized data examples. However, since tweets differ in lengths, padding values are added to equal all the tweet lengths. The max length for a tweet has been chosen as 120.

- **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 300 features. This layer is then followed by an LSTM layer with 32 nodes. The data is then flattened before being passed to a Dense layer with 32 nodes and ReLU activation. Finally, the output is produced by another Dense layer with 2 nodes that uses Sigmoid Activation due to the binary nature of the problem statement. These layers have been elaborated and described in the next section.

- **Hyperparameters:**
  - Epochs: 25 (Callback and Early Stopping has been implemented)
  - Optimizer: Adam
  - Learning Rate: 0.0001
  - Training Batch Size: 128
  - Loss Function: Binary Cross Entropy
  - Max Tweet Length: 120
  - Vocabulary Size = 10000
  - Embedding Layer Dimension = 300
  - Early Stopping Patience = 7

IV. MODEL SUMMARY

The Model Architecture adopted to address the problem has been described in the prior section. The different layers added are further elaborated here:

- *Embedding Layer* [2]
  - Used to gain a dense representation of words and their relative meaning.
  - Requires three arguments
    - Input Dimensions: Size of Vocabulary – 10000
    - Output Dimensions: Size of feature vector – 300
    - Input Length: Size of each tweet – 120

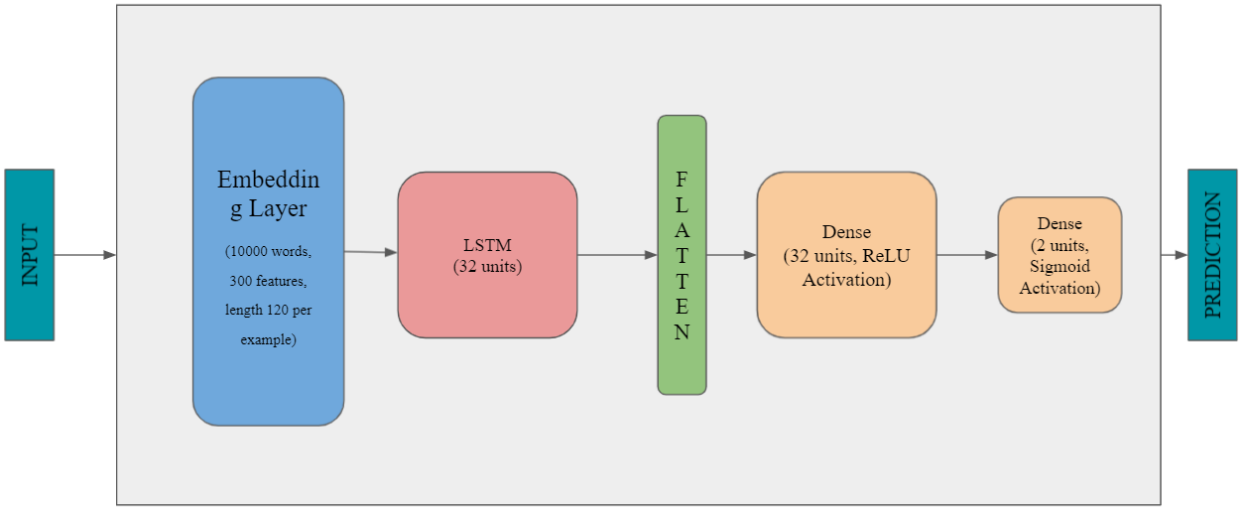
- *LSTM Layer* [3]
  - Long Short-Term Memory
  - Variation of RNN that is effective for predicting long sequences of data
  - Two main arguments:
    - Units: 32
    - return_sequences: set to **True** indicates returning last output as the output
- *Flatten Layer*
  - Converts the array of multiple dimensions into one single layer

- *Dense Layer*
  - Two Dense Layers are used
    - Layer 1: 32 units, ReLU Activation
    - Layer 2: 2 units, Sigmoid Activation

The Model Summary is given below:

```
In [118]: model.summary()

Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_10 (Embedding)     (None, 120, 300)          3000000
_________________________________________________________________
lstm_10 (LSTM)               (None, 120, 32)           42624
_________________________________________________________________
flatten_10 (Flatten)         (None, 3840)              0
_________________________________________________________________
dense_22 (Dense)             (None, 32)                122912
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 33
=================================================================
Total params: 3,165,569
Trainable params: 3,165,569
Non-trainable params: 0
_________________________________________________________________
```

The Model Architecture is shown below:



V. CODING SNAPSHOTS

1) *Imports*

Imports

In [310]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Flatten, GlobalAveragePooling1D
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

import nltk
import re
from nltk.corpus import stopwords
from textblob import Word, TextBlob

import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
```

2) *Loading of data set*

Load Dataset

In [263]:
```
train_df = pd.read_csv('train_data.csv')
```

3) *Data Pre-Processing*

Data Preprocessing

In [3]:
```
train_df.drop(columns=['id','keyword','location'], axis=1, inplace=True)
print(train_df.shape)
```

```
(7613, 2)
```

In [4]:
```
train_df.head()
```

	text	target
0	Our Deeds are the Reason of this # earthquake...	1
1	Forest fire near La Ronge Sask . Canada	1
2	All residents asked to ' shelter in place ' ...	1
3	13,000 people receive # wildfires evacuation ...	1
4	Just got sent this photo from Ruby # Alaska a...	1

In [5]:
```
train_df["text"]=train_df["text"].apply(lambda x: " ".join(x.lower() for x in x.split()))
train_df["text"]=train_df["text"].str.replace('[^\w\s]', '').replace('\d', '')

sw=stopwords.words('english')
train_df["text"]=train_df["text"].apply(lambda x: " ".join(x for x in x.split() if x not in sw))
train_df["text"]=train_df["text"].apply(lambda x:" ".join([Word(word).lemmatize() for word in x.split()]))

train_df.head()
```

	text	target
0	deed reason earthquake may allah forgive u	1
1	forest fire near la ronge sask canada	1
2	resident asked shelter place notified officer ...	1
3	13000 people receive wildfire evacuation order...	1
4	got sent photo ruby alaska smoke wildfire pour...	1

4) Data Splitting

Splitting Data

In [269]:
```
X_train, X_test, y_train, y_test = train_test_split(train_df['text'],train_df['target'], test_size=0.2, ran
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(6090,) (6090,) (1523,) (1523,)
```

5) Tokenizing and Padding

Tokenizing and Padding

In [271]:
```
vocab_size = 10000
oov_token = "<OOV>"
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)

tokenizer.fit_on_texts(X_train)
```

In [273]:
```
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

In [276]:
```
max_length = 120
trunc_type = 'post'
pad_type = 'post'

X_train_padded = pad_sequences(X_train, maxlen=max_length, truncating=trunc_type, padding=pad_type)
X_test_padded = pad_sequences(X_test, maxlen=max_length, truncating=trunc_type, padding=pad_type)
```

In [277]:
```
print(X_train_padded.shape, X_test_padded.shape)
```

```
(6090, 120) (1523, 120)
```

In [278]:
```
print(type(X_train_padded), type(X_test_padded))
print(type(y_train), type(y_test))
y_train = np.array(y_train)
y_test = np.array(y_test)
print(type(X_train_padded), type(X_test_padded))
print(type(y_train), type(y_test))
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
<class 'pandas.core.series.Series'> <class 'pandas.core.series.Series'>
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

6) *Model Creation*

Model Creation

In [107]:
```
embedding_dim = 300
```

In [117]:
```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
model.add(LSTM(32, return_sequences=True))
model.add(Flatten())
model.add(Dense(32, activation="relu"))
model.add(Dense(1, activation="sigmoid"))
```

In [118]:
```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_10 (Embedding)     (None, 120, 300)          3000000
_________________________________________________________________
lstm_10 (LSTM)               (None, 120, 32)           42624
_________________________________________________________________
flatten_10 (Flatten)         (None, 3840)              0
_________________________________________________________________
dense_22 (Dense)             (None, 32)                122912
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 33
=================================================================
Total params: 3,165,569
Trainable params: 3,165,569
Non-trainable params: 0
_________________________________________________________________
```

7) *Optimizer and model compilation*

Model Compilation

In [282]:
```
optimizer = tf.keras.optimizers.Adam(lr = 0.0001)
model.compile(
    loss = "binary_crossentropy",
    optimizer = optimizer,
    metrics = ["accuracy"]
)
```

8) Setting up of Checkpoint and Early Stopping to choose best model from epochs

```
In [283]: filepath = 'my_checkpoint.ckpt'
          cp = ModelCheckpoint(
              filepath=filepath,
              save_weights_only=True,
              save_best_only=True,
              monitor='val_accuracy',
              verbose=1
          )

In [284]: ep = EarlyStopping(
              monitor='val_accuracy',
              patience=7,
          )
```

9) Training

```
Model Training

In [122]: epochs=25
          history = model.fit(
              X_train_padded, y_train,
              validation_split = 0.2,
              callbacks=[cp,ep],
              batch_size=128,
              epochs=epochs
          )
```

10) Graphs

```
Graphs

In [123]: print(history.history.keys())
          plt.axes().set(facecolor ="white")
          plt.plot(history.history['accuracy'],color='c')
          plt.plot(history.history['val_accuracy'],color='m')
          plt.title('model accuracy').set_color('white')
          plt.ylabel('accuracy').set_color('white')
          plt.xlabel('epoch').set_color('white')
          plt.legend(['train', 'validation'], loc='upper left')

          plt.show()

          plt.axes().set(facecolor ="white")
          plt.plot(history.history['loss'])
          plt.plot(history.history['val_loss'])
          plt.legend(['train', 'validation'], loc='upper left')
          plt.title('model loss').set_color('white')
          plt.ylabel('loss').set_color('white')
          plt.xlabel('epoch').set_color('white')
          plt.show()
```

11) Model Evaluation

```
Model Evaluation

In [135]: model.load_weights(filepath)

          <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f68a816cb10>

In [125]: model.evaluate(X_test_padded, y_test)

          48/48 [==============================] - 0s 4ms/step - loss: 0.5242 - accuracy: 0.8050

          [0.5241948366165161, 0.8049901723861694]
```

12)Accuracy and Predictions

```
Predicting Test Data

In [294]: predictions = model.predict_classes(X_test_padded)

Accuracy

In [306]: accuracy = accuracy_score(y_test, predictions)
          print("Testing Accuracy: ", accuracy)

          Testing Accuracy:  0.8122127380170716
```

13) Confusion Matrix

```
Confusion Matrix

In [309]: print('Confusion Matrix')
          cm = confusion_matrix(y_test, predictions)
          print(sns.heatmap(confusion_matrix(y_test, predictions),annot=True,fmt="d"))
          print(classification_report(y_test, predictions, digits=5))
```

14) Predicting New Tweets

```
Predicting new tweets

In [148]: def preprocess_tweet(new_tweet):
              new_tweet = new_tweet.lower()
              new_tweet = re.sub('[^\w\s]', '', new_tweet)
              new_tweet = re.sub('\d', '', new_tweet)

              new_tweet = " ".join(x for x in new_tweet.split() if x not in sw)
              new_tweet = " ".join([Word(word).lemmatize() for word in new_tweet.split()])
              print("AFTER PRE-PROCESSING:", new_tweet)

              tokenizer.fit_on_texts(new_tweet)

              new_tweet = tokenizer.texts_to_sequences([new_tweet])

              new_tweet =  pad_sequences(new_tweet, maxlen=max_length, truncating=trunc_type, padding=pad_type)

              return new_tweet

          def predict_newTweet(tweet):
              print("Input Tweet: ", tweet)
              emb_tweet = preprocess_tweet(tweet)
              result = model.predict_classes(emb_tweet)
          #     print(result)
              print("RESULT: " + "Real News" if result[0][0] == 1 else "Fake News")
```

VI. RESULTS

The different outputs corresponding to the driver code has been shown below:

1) Training

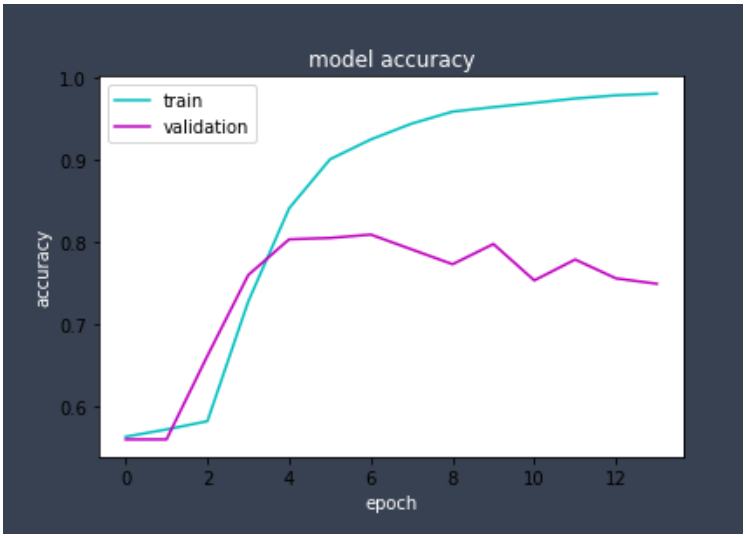
```
Epoch 1/25
39/39 [==============================] - 4s 53ms/step - loss: 0.6867 - accuracy: 0.5482 - val_loss: 0.6826 - val_accuracy: 0.5599

Epoch 00001: val_accuracy improved from -inf to 0.55993, saving model to my_checkpoint.ckpt
Epoch 2/25
39/39 [==============================] - 2s 44ms/step - loss: 0.6752 - accuracy: 0.5811 - val_loss: 0.6718 - val_accuracy: 0.5599

Epoch 00002: val_accuracy did not improve from 0.55993
Epoch 3/25
39/39 [==============================] - 2s 43ms/step - loss: 0.6578 - accuracy: 0.5812 - val_loss: 0.6380 - val_accuracy: 0.6617

Epoch 00003: val_accuracy improved from 0.55993 to 0.66174, saving model to my_checkpoint.ckpt
Epoch 4/25
39/39 [==============================] - 2s 43ms/step - loss: 0.6077 - accuracy: 0.7127 - val_loss: 0.5558 - val_accuracy: 0.7603

Epoch 00004: val_accuracy improved from 0.66174 to 0.76026, saving model to my_checkpoint.ckpt
Epoch 5/25
39/39 [==============================] - 2s 43ms/step - loss: 0.4554 - accuracy: 0.8302 - val_loss: 0.4794 - val_accuracy: 0.8038

Epoch 00005: val_accuracy improved from 0.76026 to 0.80378, saving model to my_checkpoint.ckpt
Epoch 6/25
39/39 [==============================] - 2s 49ms/step - loss: 0.2825 - accuracy: 0.8942 - val_loss: 0.5026 - val_accuracy: 0.8054

Epoch 00006: val_accuracy improved from 0.80378 to 0.80542, saving model to my_checkpoint.ckpt
Epoch 7/25
39/39 [==============================] - 2s 43ms/step - loss: 0.1925 - accuracy: 0.9257 - val_loss: 0.5420 - val_accuracy: 0.8095

Epoch 00007: val_accuracy improved from 0.80542 to 0.80952, saving model to my_checkpoint.ckpt
Epoch 8/25
39/39 [==============================] - 2s 43ms/step - loss: 0.1484 - accuracy: 0.9475 - val_loss: 0.5913 - val_accuracy: 0.7915

Epoch 00008: val_accuracy did not improve from 0.80952
Epoch 9/25
39/39 [==============================] - 2s 43ms/step - loss: 0.1165 - accuracy: 0.9621 - val_loss: 0.6631 - val_accuracy: 0.7734

Epoch 00009: val_accuracy did not improve from 0.80952
Epoch 10/25
39/39 [==============================] - 2s 43ms/step - loss: 0.0852 - accuracy: 0.9679 - val_loss: 0.7745 - val_accuracy: 0.7980

Epoch 00010: val_accuracy did not improve from 0.80952
Epoch 11/25
39/39 [==============================] - 2s 42ms/step - loss: 0.0771 - accuracy: 0.9700 - val_loss: 0.8313 - val_accuracy: 0.7537

Epoch 00011: val_accuracy did not improve from 0.80952
Epoch 12/25
39/39 [==============================] - 2s 44ms/step - loss: 0.0562 - accuracy: 0.9781 - val_loss: 0.9130 - val_accuracy: 0.7791

Epoch 00012: val_accuracy did not improve from 0.80952
Epoch 13/25
39/39 [==============================] - 2s 47ms/step - loss: 0.0555 - accuracy: 0.9783 - val_loss: 1.0001 - val_accuracy: 0.7562

Epoch 00013: val_accuracy did not improve from 0.80952
Epoch 14/25
39/39 [==============================] - 2s 42ms/step - loss: 0.0430 - accuracy: 0.9821 - val_loss: 1.0871 - val_accuracy: 0.7496

Epoch 00014: val_accuracy did not improve from 0.80952
```

2) Accuracy and evaluation

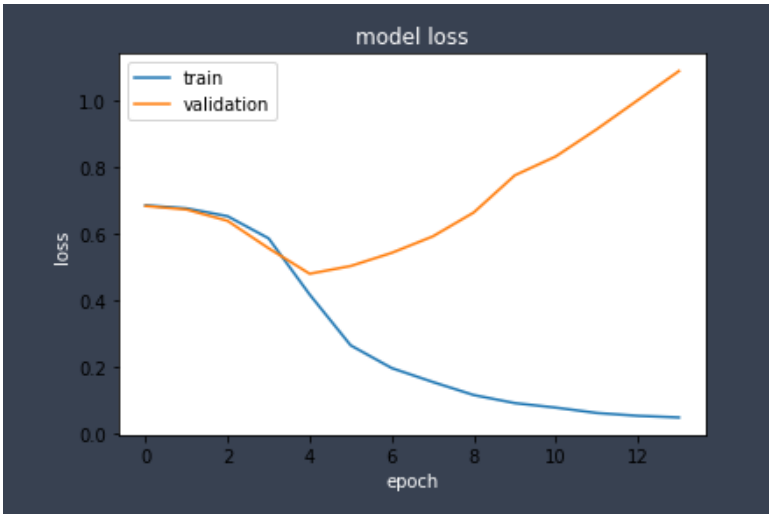
```
In [125]: model.evaluate(X_test_padded, y_test)

          48/48 [==============================] - 0s 4ms/step - loss: 0.5242 - accuracy: 0.8050

          [0.5241948366165161, 0.8049901723861694]
```

```
In [127]: accuracy = accuracy_score(y_test, predictions)
          print("Testing Accuracy: ", accuracy)

           Testing Accuracy:  0.8049901510177282
```

3) Model Accuracy Graph

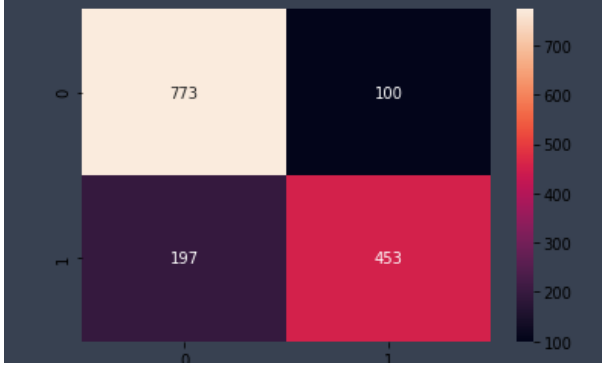


4) Model Loss Graph



5) *Confusion Matrix*

```
Confusion Matrix
AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0    0.79691   0.88545   0.83885       873
           1    0.81917   0.69692   0.75312       650

    accuracy                        0.80499      1523
   macro avg    0.80804   0.79119   0.79598      1523
weighted avg    0.80641   0.80499   0.80226      1523
```



6) *New Tweet Predictions*

```
In [151]: new_tweet = "Fires start in apartment buildings in Chicago; injuring 20 civilians #CNN"
          predict_newTweet(new_tweet)

          Input Tweet:  Fires start in apartment buildings in Chicago; injuring 20 civilians #CNN
          AFTER PRE-PROCESSING: fire start apartment building chicago injuring civilian cnn
          RESULT: Real News

In [153]: new_tweet = "hello this is a random tweet :)"
          predict_newTweet(new_tweet)

          Input Tweet:  hello this is a random tweet :)
          AFTER PRE-PROCESSING: hello random tweet
          Fake News
```

VII. CONCLUSIONS

As seen from the output screenshots, the model performs well with the dataset. It managed to score a Training Accuracy of 98.21% in the 14th epoch before the early stopping was activated. The validation accuracy appeared to drop from Epoch 7. Therefore, early stopping came into play and the weights with Validation Accuracy of 80.95% were taken. The Validation Loss was observed to be 0.5420. This however was not the lowest value since that was observed at epoch 5 with a value 0.4794. The callback function monitored the validation accuracy and the model at the 7th epoch was saved as it has the highest validation accuracy (of 80.95%). On testing, a Testing Accuracy of 80.499% was noted.

As a result, the basics of Natural Language Processing for classification of tweets as Fake or Real News and its implementation have been understood via this experimental project.

VIII. REFERENCES

[1] https://www.kaggle.com/vbmokin/nlp-with-disaster-tweets-cleaning-data

[2] https://machinelearningmastery.com/

[3] https://machinelearningknowledge.ai/