import pandas as pd *#Basic packages for creating dataframes and loading dataset*

import numpy as np

import matplotlib.pyplot as plt *#Package for visualization*

import re *#importing package for Regular expression operations*

from sklearn.model\_selection import train\_test\_split *#Package for splitting the data*

from sklearn.preprocessing import LabelEncoder *#Package for conversion of categorical to Numerical*

from keras.preprocessing.text import Tokenizer *#Tokenization*

from tensorflow.keras.preprocessing.sequence import pad\_sequences *#Add zeros or crop based on the length*

from keras.models import Sequential *#Sequential Neural Network*

from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D *#For layers in Neural Network*

from keras.utils.np\_utils import to\_categorical

from google.colab import drive

drive.mount('/content/gdrive')

import pandas as pd

*# Load the dataset as a Pandas DataFrame*

dataset = pd.read\_csv(path\_to\_csv, header=0)

*# Select only the necessary columns 'text' and 'sentiment'*

mask = dataset.columns.isin(['text', 'sentiment'])

data = dataset.loc[:, mask]

*# Keeping only the necessary columns*

data['text'] = data['text'].apply(lambda x: x.lower())

data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():

row[0] = row[0].replace('rt', ' ') *#Removing Retweets*

max\_fatures = 2000

tokenizer = Tokenizer(num\_words=max\_fatures, split=' ') *#Maximum words is 2000 to tokenize sentence*

tokenizer.fit\_on\_texts(data['text'].values)

X = tokenizer.texts\_to\_sequences(data['text'].values) *#taking values to feature matrix*

X = pad\_sequences(X) *#Padding the feature matrix*

embed\_dim = 128 *#Dimension of the Embedded layer*

lstm\_out = 196 *#Long short-term memory (LSTM) layer neurons*

def createmodel():

model = Sequential() *#Sequential Neural Network*

model.add(Embedding(max\_fatures, embed\_dim,input\_length = X.shape[1])) *#input dimension 2000 Neurons, output dimension 128 Neurons*

model.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2)) *#Drop out 20%, 196 output Neurons, recurrent dropout 20%*

model.add(Dense(3,activation='softmax')) *#3 output neurons[positive, Neutral, Negative], softmax as activation*

model.compile(loss = 'categorical\_crossentropy', optimizer='adam',metrics = ['accuracy']) *#Compiling the model*

return model

*# print(model.summary())*

labelencoder = LabelEncoder() *#Applying label Encoding on the label matrix*

integer\_encoded = labelencoder.fit\_transform(data['sentiment']) *#fitting the model*

y = to\_categorical(integer\_encoded)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,y, test\_size = 0.33, random\_state = 42) *#67% training data, 33%*

batch\_size = 32 *#Batch size 32*

model = createmodel() *#Function call to Sequential Neural Network*

model.fit(X\_train, Y\_train, epochs = 1, batch\_size=batch\_size, verbose = 2) *#verbose the higher, the more messages*

score,acc = model.evaluate(X\_test,Y\_test,verbose=2,batch\_size=batch\_size) *#evaluating the model*

print(score)

print(acc)

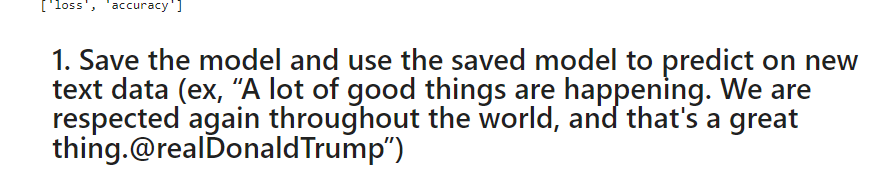
291/291 - 56s - loss: 0.8208 - accuracy: 0.6530 - 56s/epoch - 193ms/step

144/144 - 2s - loss: 0.7517 - accuracy: 0.6796 - 2s/epoch - 11ms/step

0.751739501953125

0.6795544028282166

print(model.metrics\_names) *#metrics of the model*



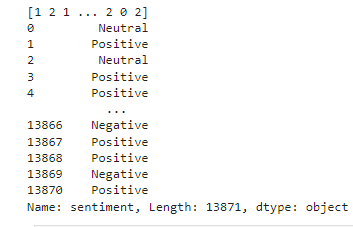
model.save('sentimentAnalysis.h5') *#Saving the model*

from keras.models import load\_model *#Importing the package for importing the saved model*

model= load\_model('sentimentAnalysis.h5') *#loading the saved model*

print(integer\_encoded)

print(data['sentiment'])



*# Predicting on the text data*

sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']

sentence = tokenizer.texts\_to\_sequences(sentence) *# Tokenizing the sentence*

sentence = pad\_sequences(sentence, maxlen=28, dtype='int32', value=0) *# Padding the sentence*

sentiment\_probs = model.predict(sentence, batch\_size=1, verbose=2)[0] *# Predicting the sentence text*

sentiment = np.argmax(sentiment\_probs)

print(sentiment\_probs)

if sentiment == 0:

print("Neutral")

elif sentiment < 0:

print("Negative")

elif sentiment > 0:

print("Positive")

else:

print("Cannot be determined")

1/1 - 0s - 22ms/epoch - 22ms/step

[0.3347626 0.16386913 0.5013683 ]

Positive

- 0s - 22ms/epoch - 22ms/step

[0.3347626 0.16386913 0.5013683 ]

Positive

**2. Apply GridSearchCV on the source code provided in the class**

In [45]:

from keras.wrappers.scikit\_learn import KerasClassifier *#importing Keras classifier*

from sklearn.model\_selection import GridSearchCV *#importing Grid search CV*

model = KerasClassifier(build\_fn=createmodel,verbose=2) *#initiating model to test performance by applying multiple hyper parameters*

batch\_size= [10, 20, 40] *#hyper parameter batch\_size*

epochs = [1, 2] *#hyper parameter no. of epochs*

param\_grid= {'batch\_size':batch\_size, 'epochs':epochs} *#creating dictionary for batch size, no. of epochs*

grid = GridSearchCV(estimator=model, param\_grid=param\_grid) *#Applying dictionary with hyper parameters*

grid\_result= grid.fit(X\_train,Y\_train) *#Fitting the model*

*# summarize results*

print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_)) *#best score, best hyper parameters*

