SENTIMENT ANALYSIS USING DEEP NEURAL NETWORK ALGORITHM LONG SHORT-TERM MEMORY (LSTM) WITH TENSORFLOW AND KERAS

NAME: krishna pavan kalyan Emani , STUDENT ID:B00819618  
COURSE: M.Sc Artificial intelligence **ULSTER UNIVERSITY**  
EMAIL: [krishsaikalyan13@gmail.com](mailto:krishsaikalyan13@gmail.com)

*Abstract*—sentiment analysis is a domain of natural language processing (NLP) plays a vital role in categorization of polarity into three outcomes positive, negative and neutral based on text reviews extracted from feedback given by customers through online platforms like Trip Advisor and yelp datasets. The aim of sentiment classification in this project is to develop deep neural algorithm with Long short-term memory (LSTM) to calculate effective and performance accuracy of business firms through online datasets. The process of achieving evaluation involves importing datasets, data preprocessing with bigrams and word embeddings and convert recycled text data input into numerical values with TensorFlow, keras and sckitlearn label encoders then model building with deep neural network by adding three layers initially, normalized vectors input is fed into embedding layer

Keywords—Bigrams, Word embeddings, one hot encoder, Label encoder, embedding layer, LSTM layer, dense layers

# **Introduction**

Sentiment analysis reveals that the autonomic computing tools are helpful in extraction of meaningful context in large data sequences instantaneously. Sentiment mining is utilized for various types of classification such as subjectivity of text data, emotion detection and justification of ratings etc. sentiment classification is currently becoming a key method for all kinds of business firms to know the effectiveness of services offered to end users and supports to resurge in adverse conditions by changing their strategies according to present scenario by the integration of sentiment analysis. [1]

Sentiment analysis is being a field of natural language processing relentlessly and updating its ability to process novelty in emerging divergence of languages. The aim of the sentiment analysis is to classify the textual sequences into subjectivity and polarity, in this project ‘0’ being objective and ‘1’ represents subjective. Subjective format of a text requires polarity sorted into three categories positive, neutral and negative, if the sentence is in unclear and undeclared it is referred as objective and if it is direct opinion classified as subjective.[1]

Semantics plays a crucial part in uprooting actual sense of end user reactions and hidden emotions of a text and complexity of a natural language is being simplified by semantic approach of text data. However, natural language processing (NLP) when integrate with deep neural networks carry out amazed outcomes of given network language data.

Sentiment analysis is flexible, reliable and capable of maintaining large data sets. It supports wide range of text analytic tools and paving a way for building as per prerequisites needed for neural network algorithms like conventional RNN and LSTM networks.

RNN:

Recurrent neural network (RNN) is multi-level architecture with interconnected layers as hidden layers. RNN neural network process current input state at a time step ‘t’ from the output of previous input state ‘t-1’. RNN learns from past sequences to predict current output. it is largely helpful to determine the outcomes of short-term prerequisites. Hidden layers preserve the previous inputs which can be reverted at any point of prediction of outputs. RNN repeated the task more often with hidden layer and back propagation method is a medium to detect errors in difference between squares of actual output and predicted output.[1]

The back propagation tradition in RNN is helpful in adjusting the weights of inputs according to required output. The computational sequential methods present in RNN helps to normalize input labels into one to many and many to one method. However, backpropagation method has its own drawbacks when performing in RNN.one of the desperate deficit is to identify errors through backpropagation process while adjusting the wights of initial layers, the loss becomes negligible to recognize the inaccuracy in initial time states due to poor identification of memory preserved in hidden layers. This is known as vanishing gradient problem.[1]

INTERNAL LAYER INSIDE RNN

# Ease of Use

Fig 1: Internal layer inside rnn

## Where **‘**t’ is current input state

‘t-1’ is previous hidden input state

Output h(t) = t+t-1, tanh function sums both input hidden state,

However, conventional RNN’s lack in formatting long texts and sequences and doesn’t able to recycle unessential sequences.

Long-short term memory (LSTM):

LSTM network is learned from RNN. LSTM is one of the deep neural network algorithms mainly used to process long term sequences in short length in a short time span. LSTM network is efficient to filter unmerited texts in sentences with help of forget gate and became master in vanishing gradient problem previously identified in RNN. [1]

**Multiplication addition tanh sigmoid**

**Concatenation**

FIG 2: INTERNAL ARCHITECTURE OF LSTM NETWORK

In the above diagram, cell state represents current state of memory, and hidden state stores memory of both unnecessary and highlighted information, however the forget gate makes unlearned all unmerited sequences. this process requires three gate a) forget gate b) input gate c) output gate. [1]

a) Forget gate: the initial stage of LSTM network is forgot gate, the concatenated vector inputs from previous hidden state (Ht-1) and input (St) can be passed through sigmoid function (s) process the vector inputs into two binary outputs ‘1’ represents to allow the data to flow from forget gate and ‘0’ represents to terminate the data from forget gate. The mathematical computation for forget is given by

F = σ (Ht-1 & St)

b) input gate: the sequence of both previous hidden state (Ht-1) and input (St). the two replicas of vector input can be proceeded through sigmoid and activation function and then final product of both the functions are given by:

I = σ (Ht-1 & St) \* tanh (Ht-1 & St)

The equation for current cell output is derived from the input gate is given by:

Ct = (F \* Ct) + I

**INTERNAL ARCHITECTURE FOR PROPSED METHODOLOGY**:

[II] METHODOLOGY:

The process of achieving sentiment analysis with LSTM network requires following methods.

## RAW DATA:

Import text data by downloading trip advisor dataset from one of the online sources and read the csv file by deploying pandas package in PyCharm environment and delete if any null values are present in text data used as main source for training and testing of dataset.

## DATA CLEANING

Data cleaning is necessary for making input csv easy to understand and processed by neural network.

#### Recycling of text data includes eliminating URL’s

#### converting upper case letter to lower case letters.

c)terminating all the special symbols like !, @, #, $,

%, ^, &, \*, (, ).

#### d)Removing all tags from text sequences

e) Eliminating all the punctuation and character.

## TEXT PREPROCESSING METHODS

## BI-GRAMS: Intially,bi-grams are derived from N-gram.N-gram is adjacent sequences of n-items like words, symbols and text from a given corpus text or document. Where ‘N’ is the range to be selected for feature extraction of given data. In this project, consider N = 2 which is union of two words in a text or sentence known as bigrams. For example in the given array,

[“the sun rises in the East”] in this array, the bigrams are given as “the sun”, “sun rises”, “rises in”, “in the”, “the East”

N-grams are mainly used to distribute long sequences of given text data, sentences into small pair of given words and to calculate the occurrences of frequencies of input ‘n’ value from text corpus.[3]

## D.ONE HOT ENCODING:one hot encoding is one of the text preprocesing method to encode categorical variables into binary values represented in 1’s and 0’s. in this process each input categorical strings and integers are transformed into numerical vectors by creating new attributes for each category in a binary column as sparse amtrix. One hot encoder denoted null value if the token is not presented in the related category.one hot encoder is fed as input to neural network.[2]

## E.LABEL ENCODER: label encoder converts the categorical data into numerical data of target data column in a dataset.it allocates each unique value to each discrete label in a column starting from 0 to n-1 classes, where n is the number of different categorical variables. Label encoder transforms categorical colums into numerical columns by selecting individual columns manually. It reiterate same digit to the same label.[1]

F. TOKENIZER: word tokenizer is used to divide the text sequences into word segments. Tokenization is a traditional method used in NLP, which make effortless for processing into neural network. It paves the way for performing word vectors. there are mainly 3 ways to perform word tokenization.1) white space tokenization 2) dictionary-based tokenization 3) sub word tokenization

### 1)WHITE SPACE TOKENIZATION:Uses text data input and break every time it has spaces between the words. It is easy when performing meaningful text sequences like English language.

2)*DICTIONARY TOKENIZATION:* It maps tokens from dictionary and gives count to the tokens from dictionary

EMBEDDING LAYER:

The word embeddings are originated from Natural Language processing methods (NLP). it is a traditional process of converting words in a text document into vectors matrix form, there are so many ways of approaching feature extraction of word embeddings. the aim of word embeddings is to compute and represent numerical values to transformed vector matrix form. There are different methods to start with word embeddings one of the most common path is to encode the words using one hot encoder to change strings into numerical array in sparse matrix type. in order to obtain word embeddings with one hot encoder. Primary goal is to tokenize the targeted words which are going to fed as input for one hot encoding. Tokenizer separates the words in a dataset and gives count to words mapped from dictionary and place the most common words together with semantic analysis of text and label each word with numerical value.[1]

The output generated from word tokenizer is further fed as input to one hot encoder inter changes words into sparse dense matrix array shape and allocate binary values to each token and passed as input to embedding layer and acts as initial layer in neural networks. [1]

LSTM LAYER ARCHITECTURE

The output of vector matrix is insert into LSTM layer to categorize values into three outcomes positive, negative and netural. LSTM layer is extension of RNN layer with number of hidden layers interconnected with each other.

[[

FIG 3 internal layer architecture

[III]. DEVELOPE THE PYTHON CODE [2]

#import libraries required for reading csv file

import pandas as pd

import numpy as np

#import packages for data cleaning and preprocessing with scikit learn and tensorflow backend

from nltk.corpus import stopwords

from keras .preprocessing.text import Tokenizer

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

from sklearn.preprocessing import LabelEncoder

#import scikit learn to split train and test models

From sklearn.model\_selection import train\_test\_split

#import packages for model building

From keras. import models

From keras.import layers

From keras.import Dense, Embedding, Dropout

bf=pd.read\_csv(‘C:\\Users\\KALYAN\\Downloads\\tripadvisor\_reviews. csv’)

read the csv file using pandas and select the path folder to variable ‘bf’.

#check for any null values present in csv file

bf.isnull().values.any()

#see the output of initial columns and rows

Print(bf.head( ))

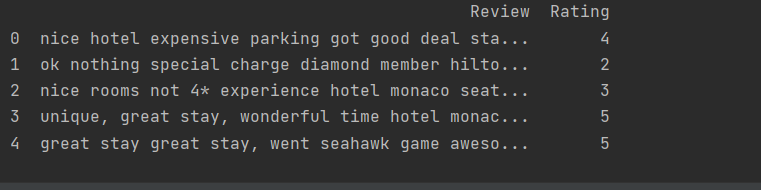


FIG 4 : csv dataset

#import preprocessing functions () to perform preprocessing of text

def preprocess\_text(se)

# eliminate html tags

sentenc = remove\_tags(se)

#remove punctuating and numbers

sentenc = re.sub(‘[^a-zA-Z]’ , ‘ ’ , sentenc)

#to remove single charcter

Sentenc = re.sub(r” \s+[a-zA-Z]\s+”, ‘ ’ , sentenc)

#to remove multiple spaces

sentence = re.sub(r’ \s+’ , ‘ ’ ,sentenc)

return sentenc

#importing re function defined to restore any open and closing space < > with a space.

import re

TAG\_RE = re.compile(r’<[^>]+>’ )

def remove\_tags(text):

return TAG\_RE.sub(‘ ’, text)

#import countvectorizer from sklearn

From sklearn.feature\_extraction.text import countvectorizer

#import textblob dictionary to find inbuilt words

import textblob from TextBlob

#import stopwords from corpus to refine unmerited words in text document

from nltk.corpus import stopwords

countvectorize transforms words into matrix token numbers to text data column

we can add stopwords manually and slected English and but from the stopwords

stoplis = stopwords.words(‘English’) + [‘but’]

apply countvectorizer to stoplis and specify ngram count to ngram\_range parameter.

[IV] FEATURE REPRESENTATION OF BIGRAMS:

c\_ve = countVectorizer(stop\_words=stoplis, ngram\_range=(2,3))

#matrix form of ngrams and choose column name to transform into matrix column.

ngrams = c\_ve.fit\_transform(df[‘reviews’])

#caluclate frequency of ngrams by converting into array form

count\_valu = ngrams.toarray( ).sum(axis=0)

#generate list of bigrams with frequency and reverse upper case to lower case order and count no of occurrences of word in a Data Frame.

voca = c\_ve.vocabulary\_

bf\_ngram = pd.DataFrame(sorted( [ (count\_valu[i],k) for k, i in voca.items ( ) ], reverse=True]

#apply polarity and subjectivity to representation of bigram column through TextBlob

bf\_ngram [‘polarity’] = bf\_ngram[‘bigram’].apply(lambda x:TextBlob.polarity)

bf\_ngram[‘subjective’] = bf\_ngram[‘bigram’].apply(lambda x.TextBlob.subjectivity)

print(bf\_ngram.head (10))

[V]. OUTPUT ON BIGRAM REPRESENTATION WITH POLARITY AND SUBJECTIVITY CLASSIFICATION:

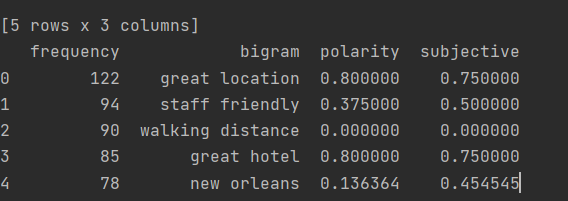


FIG 5: sentiment classification and bigrams

CODE FOR SENTIMENT CLASSIFICATION FROM BIGRAMS:

def get Analysis(score):

if score < 0:

return ‘negative -ve’

elif score == 0:

return ‘Neutral’

else:

return ‘positive +ve’

#create analysis column in text data and apply analysis to polarity column in the dataset.

bf\_ngram[‘Analysis’]= bf\_ngram[‘polarity’].apply(getAnalysis)

print(bf\_ngram.head (10))

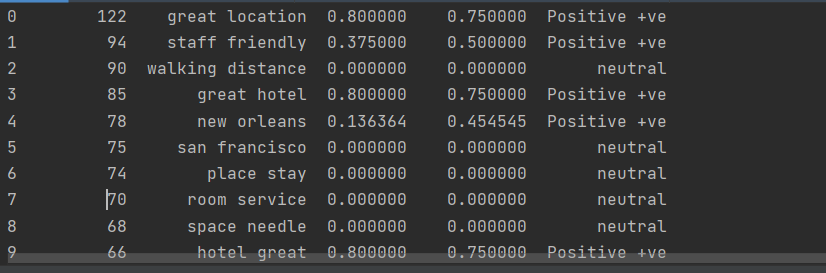


FIG 6 : Polarity classification

NOW SPLIT THE DATASET INTO TRAINING AND TESTING DATA

#split test date for 33 percent and remaining part to 67% for training data set imported from train\_test\_split with keras and tensorflow backend

train, test = train\_test\_split(bf\_ngram, test\_size = 0.33, randomstate = 42)

#divide the dataset into x and y parameters to train and test the data

x = bf\_ngram[‘bigram’]

y = bf\_ngram[‘Analysis’]

#now split the test and tarin dataset using test\_train\_split into x and y parameters, test\_size value gives percentage of test size to calculate in a given dataset

RANDOM \_STATE

random\_state is the replication of train and test and the value given to random\_state parameter represents number of iterations for train and test datasets to maintain same values in outputs.

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33, random\_state=42)

REPRESENTATION OF WORD EMBEDDINGS USING WORD TOKENIZER NAD ONE HOT ENCODER

#tokenizer gives number count to words mapped from dictionary. number words parameter in tokenize define maximum number of words given as input to tokenizer and fit tokenizer on x\_train parameter .

tok = Tokenizer(num\_words=1000,

filters=’!”#$%( )\*+,-,/: ;<=>?@[\\]^\_’~\t\n’ ,

lower=True,

split=“ ”)

tok.fit\_on\_texts(x\_train)

#print fitted tokenizer on documents using and format tokens with number count

Print(‘Fitted tokenizer on {} documents .format(tok.document\_count))

Print(‘{} words in dictionary’ .format(tok\_num\_words))



FIG 7: Tokenizer output

#print the most similar token counts located in close distance

Print(‘Top 10 most common words are:’, collections.counter(tok.word\_counts).most\_common(1000))



FIG 8: Tokenization of words

#import sequences from keras model which performs sequential operations on x\_train and x\_test values

x\_train\_se = tok.texts\_to\_sequences(x\_train)

x\_test\_se = tok\_texts\_to\_sequences(x\_test)

#import one hot encoder to encode sequences into numerical variables

#define one hot encoder and given a range of 1000 words extracted from sequence for computation

def one\_hot\_se(seq, nb\_featur = 1000):

#set the null values if token is not present in sequence using np. zeros and 1 if a value are present in sequence

o = np.zeros((len(seq), nb\_featur))

for i, s in enumerate(seq):

o[i, s] = 1.

return o

x\_train\_o = one\_hot\_se(x\_train\_se)

x\_test\_o = one\_hot\_se(x\_test\_se)

print(‘ “{} is converted into {}’ .format(x\_train\_se[10], x\_train\_o[10]))

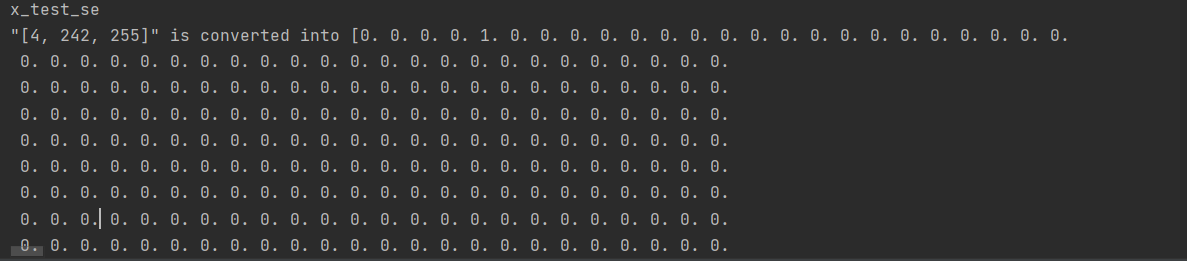


FIG 9: One hot encoder output

#import LabelEncoder function to transform categorical variables and strings in y train and test to numerical labels

#fit LabelEncoder function to y\_train and y\_test data sets and convert into binary variables.

l = LabelEncoder( )

y\_train\_l = l.fit\_transform(y\_train)

Y\_test\_l = l.transform(y\_test)

Y\_train\_o = to\_categorical(y\_train\_l)

Y\_test\_o = to\_categorical(y\_test\_l)

Print(‘ ‘”{}” is converted into {}’ .format(y\_train[11], y\_train\_l[11]))



FIG 10: Label encoder output

VALIDATION DATA:extarct validation data from training set and validate data to evaluate accuracy of the neural networks and performance of hyperparameters

EVALUATION AND PERFORMANNCE:

1)EMBEDDING LAYER:

Input\_dim: input\_dim parameter is the vocabulary length of distinctive words given as input to neural network.

2)output\_dim: output\_ dim represents the output of word embedding layer that gives computation of numerical values as output.

SPATIAL DROPOUT 1D: spatial dropout 1D layer improves the validation accuracy by removing each row of 1D feature maps but not each and every element in an array.

For example: [1,1,1] -> [1,1,0]

DROPOUT: Dropout layer act as regularization model in evolution of the learning rate and efficiency in training accuracy and to avoid overfitting by selecting arbitrary inputs set to ‘0’and up-rate the inputs which are not set to ‘’.

Activation function:

Activation function classifies the output into the representation shape in LSTM network SoftMax function, and also used to categorize the output nodes into sum of probabilities of ‘1’. For example, in this project SoftMax function systemize input into three binary outcomes positive, negative and neutral. It is utilized to process non-linear functions and tanh is standard activation function for LSTM networks.

OPTIMIZER:

To uprade performance of training and validation accuracy and to make learning datasets effort less. Rmsprop optimizer suitable for this project. The main charcterstics of rmsprop optimizer is to maintain reduced mean of the squares of the gradient by root of this average.

LOSS FUNCTION:

It measures and generates the output of predicted values and actual values to reduce the fault rate. Binary cross entropy fits for the softmax activation function which process multiple categories classification models and computes binary values into multiple outcomes.

Epochs: Number of reiterations in our training data undergoes during the train data and test data set to improve validation and training accuracy.

BATCH SIZE: It is number of repetitions made by batch size parameter. The main goal is to divide training data set into mini batches of training and validation test and fed into network for every hyperparameter updates.

[V]. PYTHONCODE FOR EVALUATION AND PERFORMANCE:

bas\_model = sequential()

bas\_model.add(Embedding(input\_dim=1000, output\_dim=128)

bas\_model.add(spatialDropout1D(0.8))

bas\_model.add(LSTM(128, dropout=0.8, recurrent\_dropout=0.8))

bas\_model.add(Dense(3,activation=‘softmax’))

print(bas\_model.summary())

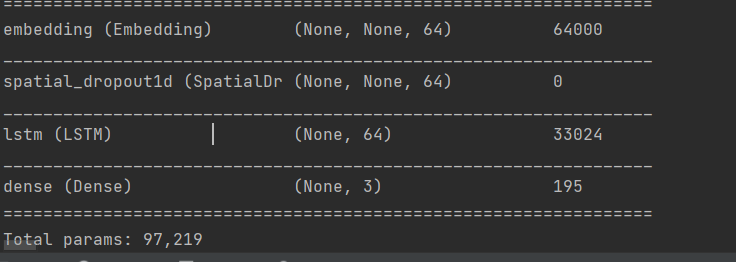


FIG 11: Model summary

[VI] EVALUATION AND PERFORMANCE OF TRAIN DATA ACCURACY:

def dp\_mode(mode)

mode.compile(optimizer = ‘rmsprop’

loss= ‘binary\_crossentropy’

metrics= [‘accuracy’])

histo = mode.fit(x\_train\_res

y\_train\_res

epochs=10

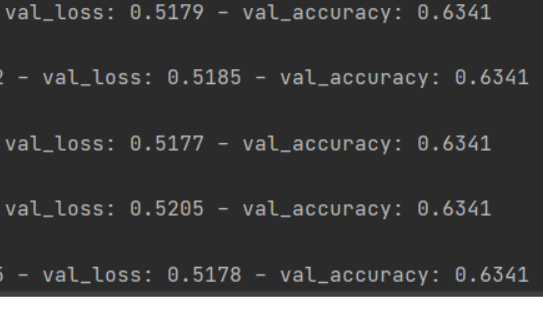
batch\_size=32

validation\_split=0.3

return histo

bas\_histo = dp\_mode(bas\_model)

validation accuracy of 63.41% and training accuracy of 63.45% and loss accuracy of 53%



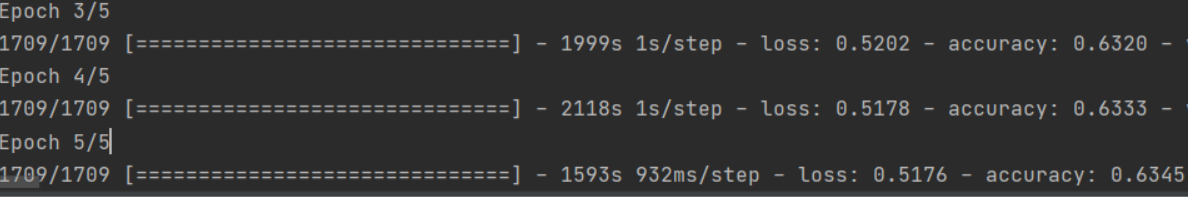


FIG 12 & 13: Validation and train loss and accuracy

EVALUATION PERFORMANCE OF TESTING DATASET:

def test\_mode(mode):

mode.fit(x\_train\_o

y\_train\_o

epoch=3

batch\_siz=128

)

result = mode.evaluate(x\_test\_o, y\_test\_o)

return result

base\_result = test\_mode(bas\_mode)

print(‘/n’)

print(‘Test accuracy of baseline model: {0: .3f}%’.format(base\_result[1]\*100))

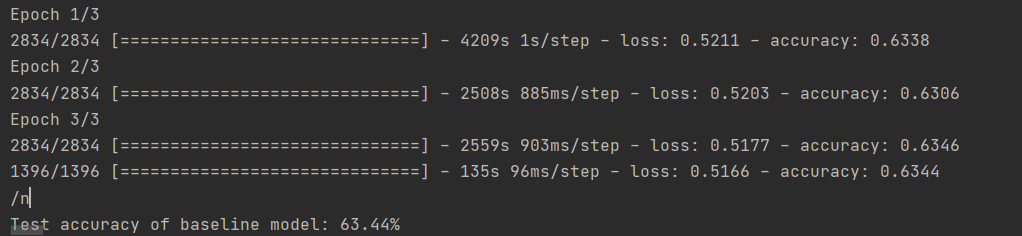


FIG13: Output of test accuracy

MODEL FOR REDUCED LAYERS:

Removing SpatialDropot1D layer, Dropout layer and decrease number of hidden layers in LSTM units to 32 and Embedding output\_dim in embedded layer to 32

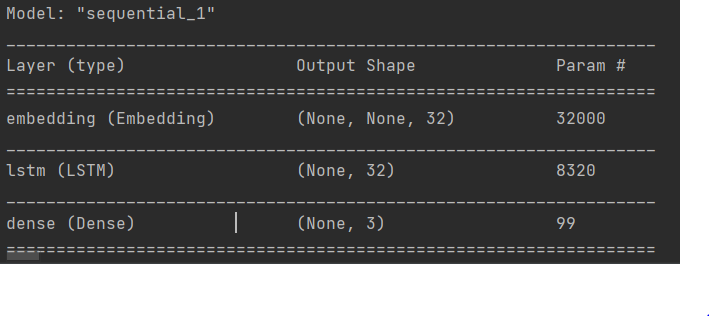


FIG 14: Reduced model summary

When compare with adding SpatiaDropout layer and Dropout layer there is a slight variation in without dropout layers with train accuracy of 63.36% and validitation accuracy of 63.31% and test accuracy of 63.48%

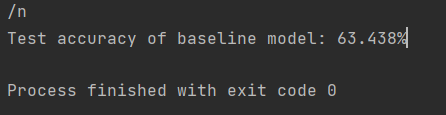


FIG 15: Output of reduced validation, train and test loss and accuracy

REFERENCES

### <https://learning.oreilly.com/library/view/neural-network-projects/9781789138900/5c3532cc-0b87-4c73-ab59-633172e048e1.xhtml>

### [https://learning.oreilly.com/library/view/making-sense-of/9781430265931/A978-1-4302-6593- 1\_2\_Chapter.html](https://learning.oreilly.com/library/view/making-sense-of/9781430265931/A978-1-4302-6593-%20%201_2_Chapter.html)

3) https://towardsdatascience.com/text-analysis-basics-in-python-443282942ec5