**Abstract**

As social media and Internet have become one of the main sources for news broadcasting, for the ease news reader, there are numerous amount of fake posts produced. These mass-media play a vital role influencing the society. As the news goes viral, it's always challenging to determine whether they or their sources are genuine or fake. Most of the time, there’s someone who can take advantage of these fake news and manipulates the public opinion over a certain matter.

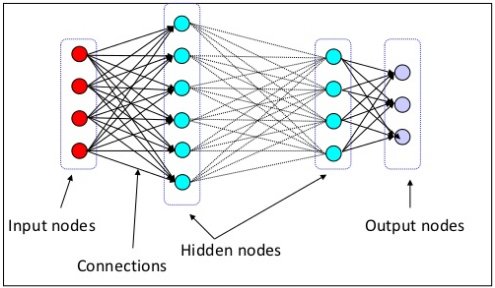
The proposed software system classifies those posts on the basis of their credibility. With a large amount of data available, the deep learning approach is preferred for training the model for the detection of credibility of the post. This project performs a simple posts detection method based on one of the artificial intelligence algorithms – Deep Neural Network.

While there exist different other mechanisms for the similar work, the goal of this project is to examine how this particular method works for this given large labeled news dataset and comparing the outcome with other similar techniques.. Using an artificial intelligence algorithm along with a large dataset helps in evaluating recent data.

**CHAPTER - 1**

**INTRODUCTION**

* 1. **Introduction to Deep Learning**

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smart phones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users’ interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

*Figure 1: Deep learning (Multi-layered Neural Network)*

Deep Learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. It is a subset of machine learning in Artificial Intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. It has evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world. This data, known simply as Big Data, is drawn from sources like social media, internet search engines, e-commerce platforms, online cinemas and more. This enormous amount of data is readily accessible and can be shared through FIN-TECH applications like cloud computing. However, the data, which normally is unstructured, is so vast that it could take decades for humans to comprehend it and extract relevant information. Companies realize the incredible potential that can result from unraveling this wealth of information, and are increasingly adapting to Artificial Intelligence (AI) systems for automated support.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

Deep learning, a subset of machine learning, utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

* 1. **Applications of Deep learning in Natural Language Processing**

The use of machine learning in NLP has been mostly limited to numerical optimization of weights for human designed representations and features from the text data. The goal of deep or representation learning is to automatically develop features or representations from the raw text material appropriate for a wide range of NLP tasks.

Neural network based deep learning methods have been shown to perform well on various NLP tasks such as language modeling, machine translation, part-of-speech tagging, named entity recognition, sentiment analysis, and paraphrase detection. The most attractive aspect of deep learning methods is their ability to perform these tasks without external hand-designed resources or time-intensive feature engineering. It is found that the NNLM can be successfully trained in two steps:

1. Continuous word vectors are learned using a simple model which eliminates the non linearity in the upper neural network layer and share the projection layer for all words.
2. The N -gram NNLM is trained on top of the word vectors. After removing the second step in the NNLM, the simple model is used to learn Word Embedding, where the simplicity allows the use of very large amount of data.

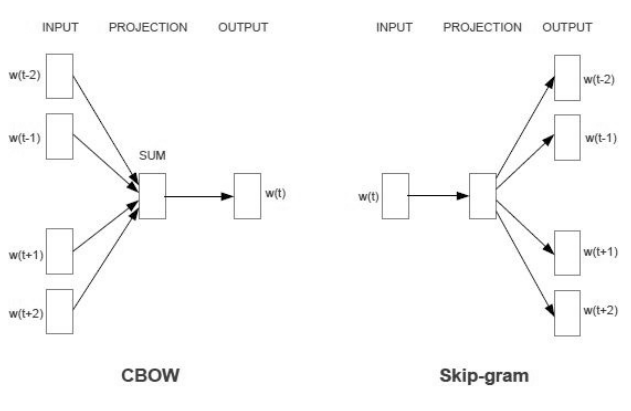


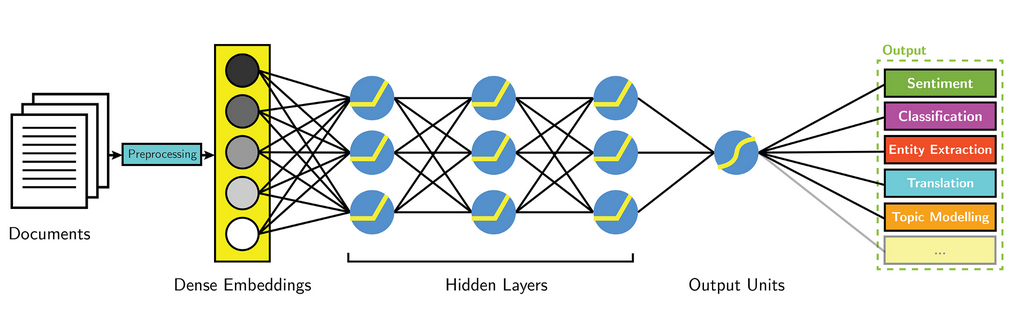
Figure 7: The CBOW architecture and the Skip-gram architecture

This gives rise to a ***word embedding*** model called Continuous Bag-of-Words Model (CBOW).

* 1. **Word Embedding**

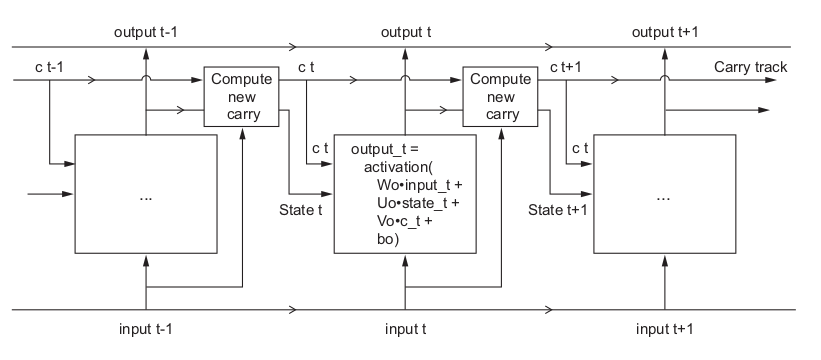
Since the goal is no longer computing probabilities of word sequences as in LMs, the word embedding system here is made more effective by not only to predict the current word based on the context but also to perform inverse prediction known as “Skip-gram” model.

A more recent study on applying deep learning methods to machine translation where the phrase-translation component, rather than the LM component in the machine translation system is replaced by the neural network models with semantic word embedding. The estimation of the translation model probabilities of a phrase-based machine translation system is carried out using neural networks. The translation probability of phrase pairs is learned using continuous-space representations induced by neural networks.



* 1. **Long Short-Term Memory**

The underlying Long Short-Term Memory ( LSTM ) algorithm was developed by Hochreiter and Schmidhuber in 1997; it was the culmination of their research on the vanishing gradient problem through training, by isolating the memory unit and the output unit where the current memory unit linearly depends upon the previous memory unit in a chained process. The architecture having three gates: input gate providing set of input to neuron, the forget gate to eliminate the unwanted information from input set based on previous input and the output gate to represent the expected output, it enhances the performance of the model. LSTM saves information for later, thus preventing older signals from gradually vanishing during processing. As proposed in [13], we found that word embedding suited LSTM than one-hot encoding.



* 1. **Introduction to Project**

The News media is one of the most important form to share news to the audiences and has widen its reach through different mediums: either printed, broadcast or through Internet. Due to the vast influence of Internet and ease of access of the reader, the online media posts and blogs have become one of the main sources for news broadcasting. The news through these posts and blogs spread real fast across the web world. Since most of the blogs and posts are produced by social or self journalist, most of the time, they are biased. These mass-media play a vital role in influencing the society. As the news goes viral, it's always challenging to determine whether they or their sources are genuine or fake. Most of the time, there’s someone who can take advantage of these fake news and manipulates the public opinion over a certain matter. These posts appear most frequent before or during election year. There are several articles proposed describing detection of such posts. The difference between these article and articles on the similar topics is that in this paper deep neural network along with the concept of word embedding and LSTM was to obtain the credibility of those posts. It was later, tested on a relatively new data set, which gave an opportunity to evaluate its performance on a recent data.

* 1. **Scope of present study**

With the increasing the mass media everyday, the proposed work will help to reduce the speed of the influence of fake news on society. As like ticket confirmation chances in railways system, this work will help the reader or viewer to identify how credible the news are. Before getting manipulated by the content, the reader could easily find whether the post is genuine or manipulated.

* 1. **Organization of project report**

The project report is organized into seven chapters.

**Chapter 1: INTRODUCTION**

This chapter discusses brief description of Deep Learning, Word Embedding, Long Short-term Memory concept and their use in text processing. This chapter will focuses on the reason behind the proposed architecture of the Deep Neural Network model.

**Chapter 2: LITERATURE SURVEY**

This section describes the study consideration taken during the project. The keeping the previous related work in account, and all their respective findings. This chapter briefly explain the related published work that were referred during the proposed work.

**Chapter 3: PROJECT ANALYSIS**

This chapter deals with the motivation behind the problem and the problem statement. The current scenario is described in the existing system. Proposed system is discussed which is the main idea of the project. Objectives of the project are thoroughly analyzed.

**Chapter 4: SYSTEM DESIGN & METHODOLOGY**

In this chapter the proposed DNN model has been discussed. The Model architecture is constructed with different layers and is compiled before it’s trained. All the aspects involved in preprocessing, compiling and training has been over viewed.

**Chapter 5: ENVIRONMENT SETUP**

This chapter gives a detail description of all the requirements either software or hardware. It presents an evaluation of the input and output requirements, computer and storage requirements and communication requirements.

**Chapter 6: IMPLEMENTATION**

This chapter contains the code of various modules from retrieving training data and processing it to display the output of the proposed trained model on given input data.

**Chapter 7: RESULTS AND DISCUSSIONS**

This chapter evaluates the results obtained from the various training data and the outputs are discussed at each stage.

**Chapter 8: CONCLUSIONS AND FUTURE ENHANCEMENTS**

This chapter describes the final conclusions and evaluates if the project meets the desired objectives successfully. The chapter also explores the future enhancements that can be accomplished in order to modify the project and make it more accurate.

**CHAPTER - 2**

**LITERATURE SURVEY**

1. **Post Detection Using Naive Bayes Classifier.**

In this paper, the authors have proposed a detection technique that uses the concept of naive Bayes classifier where the training dataset trains the model and later uses test dataset for the evaluation of the model checking its accuracy. Taking an interval between [0.2; 0.75] for the unconditional probability of fact and [0.5; 0.9] for the true probability, the author had obtained threshold of 0.59 and 0.8 respectively. With that threshold, the author have classified the post falling in between both threshold as genuine post and rest as fake posts. Through the experiment, the authors have shown that the accuracy of the model approaches reaches the desired accuracy and stated could be higher if the training dataset is large.

1. **Feature Analysis for Posts Detection through Random Forests.**

Through this paper, the authors have provided an analysis of the main review and reviewer-concentric feature to detect fake posts implementing a supervised classifier, Random Forests. Taking single feature, burst feature and overall feature in consideration, the authors have evaluated different parameters of classification and have obtained the desired accuracy.

1. **Investigation on Social Media post detection.**

In this paper, the authors have explored the various existing approaches which address the prevention of unwanted blogs or posts on social media websites, addressing the various existing techniques that focus on the fake post detection. The authors have investigated the comparative analysis of the existing techniques focusing on Support Vector Machine (SVM), Decision Tree model and Neural Network model, that performs classification on the posts and found out how the data on the social media are prune to tempered with while other site discards the fake posts.

1. **Comparative study of neural network model:**

With the existence of different neural network model around to work on, this paper made a comparison between the LSTM, GRU, RNN and Feed forward network model on the small data set as well as the larger data sets. Through this comparative study, the authors found that gated networks like GRU and LSTM’s performance was better as compared to RNN and feed-forward network models.

**CHAPTER – 3**

1. **Project Analysis**
   1. **Problem Statement**

The trends of sharing viral fake post, on the social media platform are increasing day by day, which is affecting the decision making capability of the user. There requires a system that provides the opportunity to verify the credibility of the post that are available on the social media platform.

This project aims to fulfill all those needs. As the existing systems are not completely based on artificial intelligence, the performance accuracy will remain unimproved although the dataset grows, in contrast to that, the proposed model performance will gradually increases with the increase in available dataset.

* 1. **Objectives**

This project objective to detect the posts over social media by predicting the credibility of the post and the source. The proposed work performs a simple posts detection method based on one of the artificial intelligence algorithms – Deep Neural Network. The deep model is designed in such a way that with the increase in the data set fed to the model, it will perform better every time.

* 1. **Existing system**
     1. In [1] the author have described their method for fake news detection using Naive Bayes Classifier.
     2. In [2] the author have developed a method using Random Forest algorithm and classifying on the basis of different features.
     3. In [3] the authors provide an novel approach to define available techniques for the matter. In this paper, the classification of those posts on the basis of their credibility is performed.
  2. **Proposed system**

With a large amount of data available, an artificial intelligence algorithm - Deep Learning approach is preferred in training the model for the detection of credibility of the post. The aim of the study is to examine how this particular method works for this particular problem given a manually labeled news dataset.

* 1. **System Specifications**

The proposed software system is compatible with all three operating systems, whether it’s WINDOWS, LINUX or MAC. All it needs is the environment to execute. With some basic requirements, here are some of the system specification on which the proposed work has been performed:

* + 1. **Hardware Components**

|  |  |  |
| --- | --- | --- |
| **Components** | **Specifications** | **Tested on** |
| RAM | 8GB or greater | DDR3 8 GB |
| Processor | Core i5 or greater | Core i5 |
| HDD | 500 GB or greater | 1 TB |
| OS Type | 64 bit | 64 bit |
| GRAPHIC | NVIDIA, AMD | NVIDIA |
| Computing Capability | 4 or greater | - |
| KEY BOARD | - | - |
| MOUSE | - | - |
| MONITOR | - | - |

* + 1. **Software Components**

|  |  |  |
| --- | --- | --- |
| **Components** | **Specifications** | **Tested on** |
| OPERATING SYSTEM | Windows 7 or <,  MAC OS,  Ubuntu 14.04 or < | Ubuntu 18.04 |
| TEXT EDITOR | Any | Sublime text 3 |
| PYTHON | 3.x | 3.6.6 |
| TENSORFLOW | - | - |
| KERAS | - | - |
| PANDAS | - | - |
| SCIKIT LEARN | - | - |
| NUMPY | - | - |
| MATPLOTLIB | - | - |

**CHAPTER – 4**

1. **Model Design & Methodology**
   1. **Model Architecture**

The core building block of neural networks is the layer, a data-processing module that you can think of as a filter for data. Some data goes in, and it comes out in a more useful form. Specifically, layers extract representations out of the data fed into them hopefully, representations that are more meaningful for the problem at hand. Most of deep learning consists of chaining together simple layers that will implement a form of progressive data distillation. A deep-learning model is like a sieve for data processing, made of a succession of increasingly refined data filters the layers. As in fig[1], our model is composed of four different layers:

* + 1. **Embedding Layer**

Keras offers an [Embedding](https://keras.io/layers/embeddings/" \l "embedding) layer that can be used for neural networks on text data. It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the [Tokenizer API](https://keras.io/preprocessing/text/" \l "tokenizer) also provided with Keras. The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.

It is a flexible layer that can be used in a variety of ways, such as:

* It can be used alone to learn a word embedding that can be saved and used in another model later.
* It can be used as part of a deep learning model where the embedding is learned along with the model itself.
* It can be used to load a pre-trained word embedding model, a type of transfer learning.
  + 1. **LSTM Layer**

Long short-term memory (LSTM) units are units of a [recurrent neural network](https://en.wikipedia.org/wiki/Recurrent_neural_network" \o "Recurrent neural network) (RNN). An RNN composed of LSTM units is often called an *LSTM network*. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

* + 1. **Dense Layers**

A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected. The layer has a weight matrix **W,** a bias vector **b,** and the activation of previous layer **a.** The following is the doc string of class Dense from the Keras documentation:

*output = activation(dot(input, kernel) + bias)*

where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer.

* + 1. **Dropout Layer**

Dropout is a [regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics)" \o "Regularization (mathematics)) technique for reducing [overfitting](https://en.wikipedia.org/wiki/Overfitting" \o "Overfitting) in [neural networks](https://en.wikipedia.org/wiki/Neural_networks" \o "Neural networks) by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks.[[1]](https://en.wikipedia.org/wiki/Dropout_(neural_networks)" \l "cite_note-MyUser_Arxiv.org_July_26_2015c-1) The term "dropout" refers to dropping out units (both hidden and visible) in a [neural network](https://en.wikipedia.org/wiki/Neural_network" \o "Neural network).

* 1. **Methodology**

The proposed work is completed in five steps, from cleaning the data to classification of the posts. Each steps is initiated as the program approaches forward. These are the followed five steps:

1. Data Preprocessing
2. Embedding Preprocessing
3. Building the network
4. Compiling the Network Model
5. Classification of posts
   * 1. **Data Preprocessing**

The major challenge is to find out dataset of the posts or blogs on which detail analysis is to be performed and here, we have taken the labelled dataset from the Kaggle, collected from different stream pages. From the available 20 fields, we have selected some relevant fields to retrieve news articles as our input dataset. With over 14000 records to work on, we have split the data into train and test in the ratio of 4:1 (train data: 80%, test data: 20%). Further, we have performed the steps of data preprocessing techniques over both, training dataset and test dataset. Since the input to our model is only numeric tensor values, the data preprocessing on these dataset are carried on these set of steps:

* + - 1. **Data Cleaning:**

In this step, we have parsed the data collected from the different stream and from the obtained CSV file, we have abstracted the relevant fields to numpy array. for our model, we have selected the post's text and its label to differentiate them in according. Further, we have ignored the data having null values to reduce training accuracy. Later, we have used the parsed dataset with no missing values for the vectorization step.

* + - 1. **Vectorization:**

In the step, we have converted the parsed input text data from the previous step into numeric tensors which will behave as inputs to the Deep Learning Model. After we vectorize the input text, we have further, passed those vectors to the model for training purpose which achieved in two steps:

First, we performed Tokenization, where we split the test string are into n-grams chunks and, then we create their associated numeric vectors using the word embedding concept, where we have embedded a predefined word embedding Global Vectors for Word Representation (GloVe)[4].

* + 1. **Embedding Preprocessing**

Deep neural network model requires a large dataset to learn powerful features, in failing in feeding that, we have here used a predefined word embedding stacked to the model through Embedding Layer. As the predefined embedding is an archive, to add it to model, we have followed two steps:

1. Parse the file into map index:

Here, we have created a key-value pair for each of the indices present in the word embedding to create the mapping sample for our text input tensors which further behaves as an associate to the numeric tensors while training

1. Construct an embedding matrix to pass it to the layer:

After we created a mapping file for input data, now we have constructed a matrix with the same dimensions as of our input dataset. ie.

*embedding\_matrix=[max,dim]*

This matrix has a shape of max\_words (m) from the input dataset that we are using and embedding\_dimension (d), that the key-value pair can acquire.

* + 1. **Building the Network Model**

In this section, we have described our proposed network model for classification.

1. **The network architecture**

It is the core building block of the network model. And our proposed model follows the Sequential Network Model Architecture, where each node in a layer is connected to each node of the adjacent layer.

1. **The activation function**

The activation function, also known as tensor operations, defines the behaviour of output from the node for the given input or set of inputs. When activated, it helps the neural network model to reduce the learning transformation of tensors by deciding which information from the input is relevant by transforming them non linearly.

output = activation(dot(input, kernel) + bias)

where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer.

With different types of activation function available, our network model has used :

* 1. **ReLU**

ReLU or Rectified Linear Unit function, defined as:

***F(x) = x+ = max(0,x)***

where x is the input, is a non-linear function that back propagates the errors. Depending on the input behaviour, relevant or irrelevant, it activates the neurons making the network parse and easy to compute



* 1. **Sigmoid function:**

This function is preferred when the model is expected to generate binary output or in the range[0,1].

Mathematically,



where x is a large set of output from the previous layer. We have used the Sigmoid function in our output layer of the network for the binary classification of the posts.

* + 1. **Compiling the Network Model:**

In this section, the compilation of the proposed network model is performed based on two primary factors:

1. **Loss function :**

This function is employed to minimizes the loss in quality of the selected set of parameters based on their induced score with ground truth labels, and for our data set, we have chosen binary cross-entropy to obtain the probabilistic output.

Binary cross-entropy is considered to be an ideal loss function while using the Sigmoid function in the output layer and expected output is either in the range of 0 and 1.

1. **Optimizer:**

The optimizer reduces the different cost function that determines the update of the network based on the loss function. For the proposed model, we have chosen RMSprop algorithm for deep learning model.

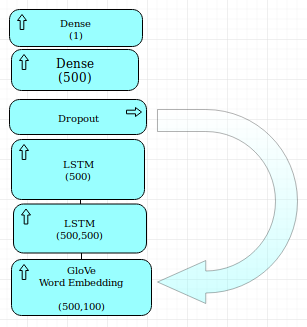
RMSprop [15] has proven its effectiveness as a modification of Adagrad algorithm by discarding the extreme past history. The exponentially decaying average and the learning rate of 0.001 helps to optimize the model. It minimizes the loss during training due to loss function and mini-batch samples.

* + 1. **Classification of posts**

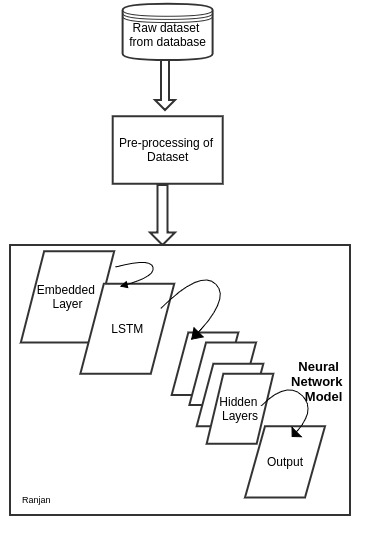
For the better distribution, we made random shuffle to the dataset, and after that, we divided them into three subsets: training dataset, validation dataset, test dataset. The neural network will then use the training dataset for training classifier model whereas it will use validation dataset for tuning some global parameters of the classifier. The model then uses the test dataset to estimate how well the model performs on new data.

At this final module, we are going to feed the test dataset to the trained model to analyze the prediction accuracy of the model. Based on obtained accuracy, we are going to evaluate different classification parameters and if possible, even enhancing the accuracy.

* 1. **Project Model Layered Design**



* 1. **Project Flow Design**



**CHAPTER – 5**

1. **Environment Step-up**

This setup is proposed for Linux system (UBUNTU 18.04). This installation process may vary from system to system, but the step remains the same.

* 1. **Prepare computer:**

*sudo apt-get update  
 sudo apt-get upgrade   
 sudo apt-get install build-essential cmake g++ gfortran   
 sudo apt-get install git pkg-config python-dev   
 sudo apt-get install software-properties-common wget  
 sudo apt-get autoremove   
 sudo rm -rf /var/lib/apt/lists/\**

* 1. **Install NVIDIA drivers:**

Next step is to install correct nvidia driver.

Check whether nvidia is present or not:

*lspci | grep -i nvidia*

Next step is to add proprietary repository of nvidia drivers.

*sudo add-apt-repository ppa:graphics-drivers/ppa  
sudo apt-get update  
sudo apt-get install nvidia-375*

Once nvidia driver is installed, restart the computer.

* 1. **Cuda-8 installation**

Next step is to install cuda-8. Download the appropriate package file.

Installation steps are the same as that on their website,

Next step is to add libraries to .bashrc file,

*echo 'export PATH = /usr/local/cuda/bin:$PATH' >> ~/.bashrc  
echo 'export LD\_LIBRARY\_PATH = /usr/local/cuda/lib64 : $PATH' >> ~/.bashrc  
source ~/.bashrc*

After sourcing the bashrc file, the CUDA version can be verified using

*nvcc -V*

* 1. **cuDNN Installation**

Next step is to install cuDNN. cuDNN can be downloaded from NVIDIA’s webpage https://developer.nvidia.com/cudnn. Extract cuDNN and install.

*cd ~/Downloads/  
 tar xvf cudnn\*.tgz  
 cd cuda  
 sudo cp \*/\*.h /usr/local/cuda/include/  
 sudo cp \*/libcudnn\* /usr/local/cuda/lib64/  
 sudo chmod a+r /usr/local/cuda/lib64/libcudnn\**

* 1. **Install python stuff**

Next step is to install standard python libraries,

*sudo apt-get update && apt-get install -y python-numpy python-scipy*

*python-nose python-h5py python-skimage python-matplotlib python-pandas python-sklearn python-sympy*

*sudo apt-get clean && sudo apt-get autoremove*

*sudo rm -rf /var/lib/apt/lists/\**

Next install more python goodies like virtual environment, pip, pip3, etc.

*sudo apt-get update*

*sudo apt-get install git python-dev python3-dev python-numpy*

*python3-numpy build-essential python-pip python3-pip*

*python-virtualenv swig python-heel libcurl3-dev*

*sudo apt-get install -y libfreetype6-dev libpng12-dev*

*pip3 install -U matplotlib ipython[all] jupyter pandas scikit-image*

* 1. **Install openBLAS from its repository**

OpenBLAS is linear algebra library that is faster than standard Atlas.

*mkdir ~/git  
cd ~/git  
git clone https://github.com/xianyi/OpenBLAS.git  
cd OpenBLAS  
make FC=gfortran -j16  
sudo make PREFIX=/usr/local install*

* 1. **Install tensorflow**

Next install tensorflow, easiest method is to use pip3

*pip3 install --upgrade tensorflow-gpu*

*or*

*pip3 install --upgrade tensorflow*

* 1. **Install keras**

Install keras using pip3

*sudo pip3 install keras*

* 1. **Virtual environments**

Last step is to use virtual environments to organize projects. For now, I will use tensorflow3. Create a virtual environment named tensorflow3. A different more appropriate name can be chosen.

*virtualenv --system-site-packages -p python3 tensorflow3*

Next, source the new environment and install tensorflow

*source ~/tensorflow3/bin/activate*

* 1. **Code/text editor**

Next step is to add your favorite code/text editor.

Sublime text: Installation

*sudo add-apt-repository ppa:webupd8team/sublime-text-3  
sudo apt-get update  
sudo apt-get install sublime-text-installer*

**CHAPTER – 6**

1. **Implementation**
   1. **Dataset Preprocessing**

*import csv*

*import sys*

*import pandas as pd*

*csv.field\_size\_limit(sys.maxsize)*

*def load\_csv():*

*with open('fake1.csv') as file:*

*data = pd.read\_csv(file)*

*print('\n'+'\*'\*80 )*

*print('Printing Text Data')*

*print('\*'\*80 + '\n')*

*working\_data = data['text'].head()*

*print(working\_data)*

*print('\*'\*80)*

*text = []*

*text\_text = []*

*text\_labels = []*

*set\_labels=set()*

*count = 0*

*text\_text = data.iloc[:,5]*

*print(text\_text.astype)*

*text\_labels = data.iloc[:,19]*

*for row in text\_labels:*

*set\_labels.add(row)*

*print('\n'+'\*'\*80 )*

*print('Printing Distinct Labels')*

*print('\*'\*80 + '\n')*

*print(set\_labels)*

*labels=[]*

*for i,row in enumerate(text\_labels):*

*if(row =='conspiracy'):*

*text\_labels=text\_labels.replace(row,'1')*

*elif(row =='fake'):*

*text\_labels =text\_labels.replace(row,'1')*

*elif(row =='bs'):*

*if(float(data.loc[i][12])>0):*

*text\_labels =text\_labels.replace(row,'1')*

*else:*

*text\_labels =text\_labels.replace(row,'0')*

*elif(row =='bias'):*

*if(float(data.loc[i][12])>0):*

*text\_labels =text\_labels.replace(row,'1')*

*else:*

*text\_labels =text\_labels.replace(row,'0')*

*elif(row =='hate'):*

*text\_labels =text\_labels.replace(row,'1')*

*else:*

*text\_labels =text\_labels.replace(row,'0')*

*return(text\_text.astype(str),text\_labels)*

*load\_csv()*

* 1. **Input text Tokenizing**

*from keras.preprocessing.text import Tokenizer*

*from keras.preprocessing.sequence import pad\_sequences*

*import numpy as np*

*def tokenize(texts,labels,maxlen,max\_words):*

*tokenizer = Tokenizer(num\_words=max\_words)*

*tokenizer.fit\_on\_texts(texts)*

*sequences = tokenizer.texts\_to\_sequences(texts)*

*word\_index = tokenizer.word\_index*

*print('Found %s unique tokens in the training dataset.' % len(word\_index))*

*data = pad\_sequences(sequences, maxlen=maxlen)*

*print('-'\*80 )*

*print('Printing Length of dataset')*

*print(len(data))*

*print('-'\*80)*

*labels = np.asarray(labels)*

*print('-'\*80 )*

*print('Shape of data tensor:', data.shape)*

*print('Shape of label tensor:', labels.shape)*

*print("Tokenizing performed successfully on dataset.")*

*print('\*'\*80)*

*indices = np.arange(data.shape[0])*

*np.random.shuffle(indices)*

*data = data[indices]*

*labels = labels[indices]*

*x\_train = data*

*y\_train = labels*

*return (x\_train,y\_train,word\_index)*

* 1. **Word Embedding Tokenizing**

*import os*

*import numpy as np*

*glove\_dir = 'glove.6B/'*

*embeddings\_index = {}*

*print('\n'+'\*'\*80 )*

*print('Loading Words Embedding...')*

*print('-'\*80)*

*print(os.path.join(glove\_dir,'glove.6B.100d.txt'))*

*f = open(os.path.join(glove\_dir,'glove.6B.100d.txt'))*

*for line in f:*

*values = line.split()*

*word = values[0]*

*coefs = np.asarray(values[1:], dtype='float32')*

*embeddings\_index[word] = coefs*

*f.close()*

*print('Found %s word vectors.' % len(embeddings\_index))*

*embedding\_dim = 100*

*def embedd(max\_words,word\_index,embedding\_matrix):*

*for word, i in word\_index.items():*

*embedding\_vector = embeddings\_index.get(word)*

*if i < max\_words:*

*if embedding\_vector is not None:*

*# Words not found in embedding index will be all-zeros.*

*embedding\_matrix[i] = embedding\_vector*

*def generate\_matrix(max\_words,word\_index):*

*embedding\_matrix = np.zeros((max\_words, embedding\_dim))*

*embedd(max\_words,word\_index,embedding\_matrix)*

*return(embedding\_matrix,embedding\_dim)*

* 1. **Building and Compiling**

*from keras.layers import LSTM, Dense, Embedding, Flatten, Dropout*

*from keras.models import Sequential*

*import csv\_check as cs*

*import embedding\_layer as emb\_layer*

*import load\_data as ld*

*import text\_tokenizing as tt*

*from sklearn.model\_selection import train\_test\_split*

*#Some constant values*

*maxlen = 500 # We will cut reviews after 100 words*

*max\_words = 50000 # We will only consider the top 50,000 words in the dataset*

*#labeling the data set*

*texts,labels = cs.load\_csv()*

*print(labels)*

*#spliting into train and test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(texts, labels,test\_size=0.2)*

*#sampling the text data*

*print('\n'+'\*'\*80 )*

*print('Tokenizer Fittig on Training dataset....')*

*print('-'\*80)*

*x\_train,y\_train,word\_index = tt.tokenize(X\_train,y\_train,maxlen,max\_words)*

*print('\n'+'\*'\*80 )*

*print('Tokenizer Fittig on Test dataset........')*

*print('-'\*80)*

*X\_test, y\_test, words= tt.tokenize(X\_test,y\_test,maxlen,max\_words)*

*embedding\_matrix, embedding\_dim = emb\_layer.generate\_matrix(max\_words,word\_index)*

*model = Sequential()*

*model.add(Embedding(max\_words,*

*embedding\_dim,*

*input\_length=maxlen))*

*model.add(LSTM(maxlen,return\_sequences=True))*

*model.add(LSTM(maxlen))*

*model.add(Dropout(0.5))*

*model.add(Dense(2500, activation='relu'))*

*model.add(Dense(1, activation='sigmoid'))*

*print('\n'+'\*'\*80 )*

*print('Printing the proposed model summary........')*

*print('-'\*80)*

*model.summary()*

*model.layers[0].set\_weights([embedding\_matrix])*

*model.layers[0].trainable = False*

*model.compile(optimizer='rmsprop',*

*loss='binary\_crossentropy',*

*metrics=['acc'])*

*print('\n'+'\*'\*80 )*

*print('Compiling the proposed model summary........')*

*print('-'\*80 )*

*print('Printing the proposed model JSON FILE........')*

*print('-'\*80)*

*model\_json = model.to\_json()*

*print(model\_json)*

*print('-'\*80)*

*history = model.fit(x\_train, y\_train,*

*epochs=6,*

*batch\_size=100,*

*validation\_data=(X\_test, y\_test))*

*print('\n'+'\*'\*80 )*

*print('Training the proposed model ........')*

*print('-'\*80 )*

*print('Saving Trained Model to file : Pre\_Trained\_Global\_Model..')*

*print('-'\*80)*

*with open("model21.json", "w") as json\_file:*

*json\_file.write(model\_json)*

*# serialize weights to HDF5*

*model.save\_weights('pre\_trained\_glove\_model.h5')*

*print("Saved model to disk")*

* 1. **Testing on Test Dataset**

*from keras.models import model\_from\_json*

*from keras.preprocessing.text import Tokenizer*

*from keras.preprocessing.sequence import pad\_sequences*

*from sklearn.metrics import accuracy\_score*

*# load json and create model*

***def getTest(x\_itest,y\_itest):***

*json\_file = open('model.json', 'r')*

*loaded\_model\_json = json\_file.read()*

*json\_file.close()*

*model = model\_from\_json(loaded\_model\_json)*

*# load weights into new model*

*model.load\_weights("pre\_trained\_glove\_model.h5")*

*print("Loaded model from disk")*

*model.compile(optimizer='rmsprop',*

*loss='binary\_crossentropy',*

*metrics=['acc'])*

*x\_test,y\_test = x\_itest,y\_itest*

*clf\_probs = model.evaluate(x\_test,y\_test)*

*return(clf\_probs)*

* 1. **Plotting the Result**

*import matplotlib.pyplot as plt*

*acc = history.history['acc']*

*val\_acc = history.history['val\_acc']*

*loss = history.history['loss']*

*val\_loss = history.history['val\_loss']*

*epochs = range(1, len(acc) + 1)*

*plt.plot(epochs, acc, 'bo', label='Training acc')*

*plt.plot(epochs, val\_acc, 'b', label='Validation acc')*

*plt.title('Training and validation accuracy')*

*plt.legend()*

*plt.figure()*

*plt.plot(epochs, val\_loss, 'b', label='Validation loss')*

*plt.plot(epochs, loss, 'bo', label='Training loss')*

*plt.title('Training and validation loss')*

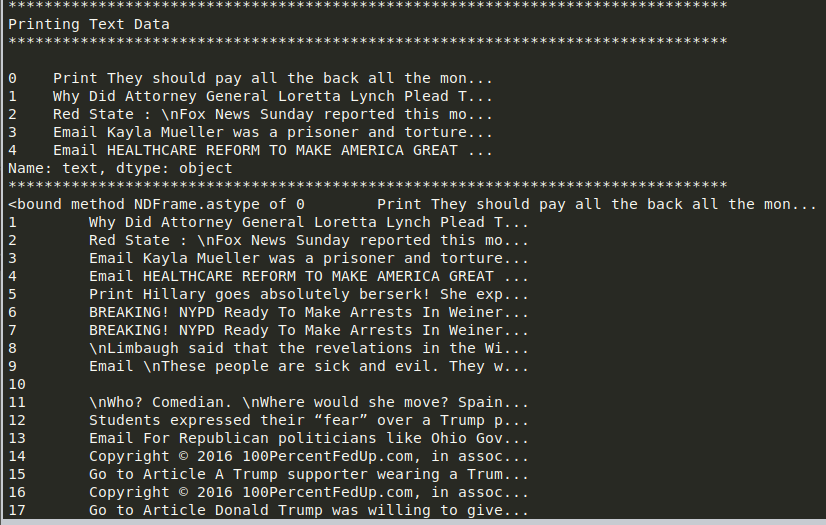
*plt.legend()*

*plt.show()*

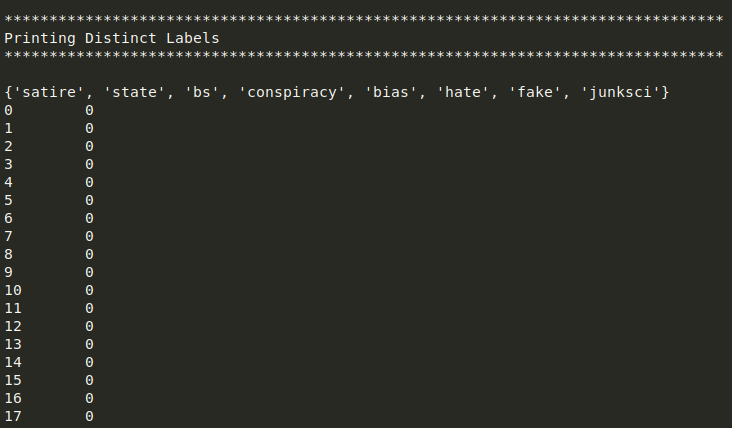
**CHAPTER – 7**

**RESULTS AND DISCUSSIONS**

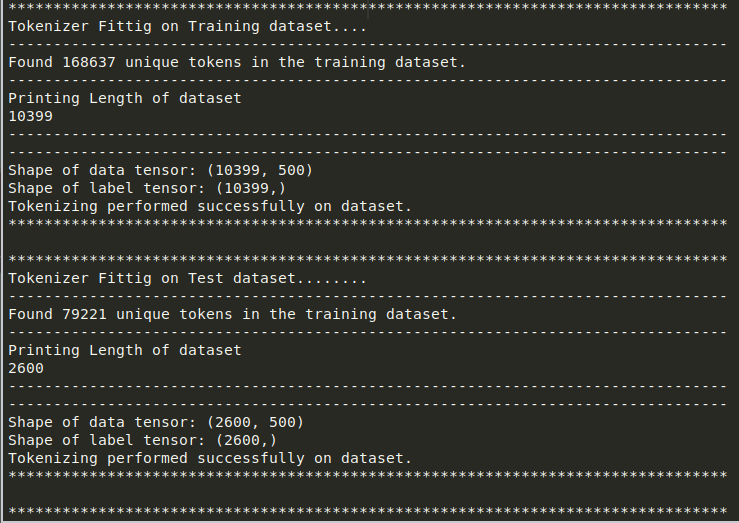
Here the figure XX, shows the input text posts (data), extracted from CSV file which was collected from different stream sources.

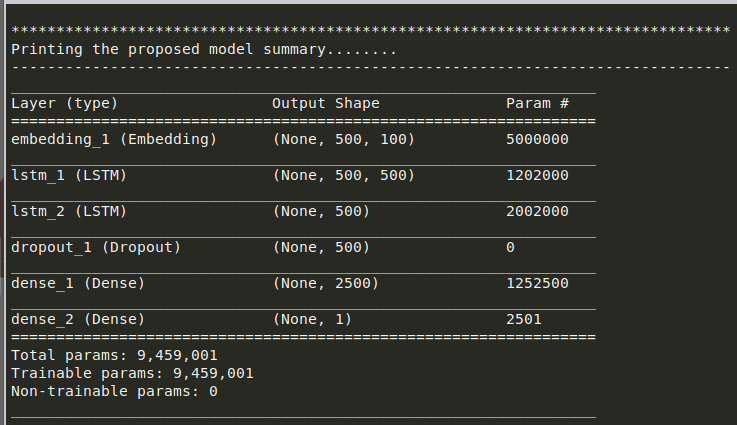
****

This second snap displays the labels to the input text. The categorized labels are examined and are converted to binary labels, where 0 or zero represents genuine post and 1 or one represents fake ones.



The third screenshot describes the process of Tokenization, displaying all the intermediate results and printing them:



The fourth picture describes the network architecture of the proposed mode, where different layers with different parameters are stacked over each other..

After the model is built, here the the json format of that proposed model saved to the disk:

{"class\_name": "Sequential", "config": {"name": "sequential\_1", "layers": [{"class\_name": "Embedding", "config": {"name": "embedding\_1", "trainable": false, "batch\_input\_shape": [null, 500], "dtype": "float32", "input\_dim": 50000, "output\_dim": 100, "embeddings\_initializer": {"class\_name": "RandomUniform", "config": {"minval": -0.05, "maxval": 0.05, "seed": null}}, "embeddings\_regularizer": null, "activity\_regularizer": null, "embeddings\_constraint": null, "mask\_zero": false, "input\_length": 500}}, {"class\_name": "LSTM", "config": {"name": "lstm\_1", "trainable": true, "return\_sequences": true, "return\_state": false, "go\_backwards": false, "stateful": false, "unroll": false, "units": 500, "activation": "tanh", "recurrent\_activation": "hard\_sigmoid", "use\_bias": true, "kernel\_initializer": {"class\_name": "VarianceScaling", "config": {"scale": 1.0, "mode": "fan\_avg", "distribution": "uniform", "seed": null}}, "recurrent\_initializer": {"class\_name": "Orthogonal", "config": {"gain": 1.0, "seed": null}}, "bias\_initializer": {"class\_name": "Zeros", "config": {}}, "unit\_forget\_bias": true, "kernel\_regularizer": null, "recurrent\_regularizer": null, "bias\_regularizer": null, "activity\_regularizer": null, "kernel\_constraint": null, "recurrent\_constraint": null, "bias\_constraint": null, "dropout": 0.0, "recurrent\_dropout": 0.0, "implementation": 1}}, {"class\_name": "LSTM", "config": {"name": "lstm\_2", "trainable": true, "return\_sequences": false, "return\_state": false, "go\_backwards": false, "stateful": false, "unroll": false, "units": 500, "activation": "tanh", "recurrent\_activation": "hard\_sigmoid", "use\_bias": true, "kernel\_initializer": {"class\_name": "VarianceScaling", "config": {"scale": 1.0, "mode": "fan\_avg", "distribution": "uniform", "seed": null}}, "recurrent\_initializer": {"class\_name": "Orthogonal", "config": {"gain": 1.0, "seed": null}}, "bias\_initializer": {"class\_name": "Zeros", "config": {}}, "unit\_forget\_bias": true, "kernel\_regularizer": null, "recurrent\_regularizer": null, "bias\_regularizer": null, "activity\_regularizer": null, "kernel\_constraint": null, "recurrent\_constraint": null, "bias\_constraint": null, "dropout": 0.0, "recurrent\_dropout": 0.0, "implementation": 1}}, {"class\_name": "Dropout", "config": {"name": "dropout\_1", "trainable": true, "rate": 0.5, "noise\_shape": null, "seed": null}}, {"class\_name": "Dense", "config": {"name": "dense\_1", "trainable": true, "units": 2500, "activation": "relu", "use\_bias": true, "kernel\_initializer": {"class\_name": "VarianceScaling", "config": {"scale": 1.0, "mode": "fan\_avg", "distribution": "uniform", "seed": null}}, "bias\_initializer": {"class\_name": "Zeros", "config": {}}, "kernel\_regularizer": null, "bias\_regularizer": null, "activity\_regularizer": null, "kernel\_constraint": null, "bias\_constraint": null}}, {"class\_name": "Dense", "config": {"name": "dense\_2", "trainable": true, "units": 1, "activation": "sigmoid", "use\_bias": true, "kernel\_initializer": {"class\_name": "VarianceScaling", "config": {"scale": 1.0, "mode": "fan\_avg", "distribution": "uniform", "seed": null}}, "bias\_initializer": {"class\_name": "Zeros", "config": {}}, "kernel\_regularizer": null, "bias\_regularizer": null, "activity\_regularizer": null, "kernel\_constraint": null, "bias\_constraint": null}}]}, "keras\_version": "2.2.4", "backend": "tensorflow"}

The below execution displays the first epoch from the training of the model with six epoch:

100/10399 [..............................] - ETA: 57:22 - loss: 0.6912 - acc: 0.5600

200/10399 [..............................] - ETA: 1:02:39 - loss: 0.4838 - acc: 0.7550

300/10399 [..............................] - ETA: 1:04:28 - loss: 0.5996 - acc: 0.6200

400/10399 [>.............................] - ETA: 1:03:48 - loss: 0.6171 - acc: 0.6975

500/10399 [>.............................] - ETA: 1:06:24 - loss: 0.6441 - acc: 0.6380

600/10399 [>.............................] - ETA: 1:08:30 - loss: 0.5712 - acc: 0.6917

700/10399 [=>............................] - ETA: 1:10:28 - loss: 0.5291 - acc: 0.7243

800/10399 [=>............................] - ETA: 1:11:42 - loss: 0.4742 - acc: 0.7587

900/10399 [=>............................] - ETA: 1:11:25 - loss: 0.5398 - acc: 0.7744

1000/10399 [=>............................] - ETA: 1:10:37 - loss: 0.5158 - acc: 0.7920

1100/10399 [==>...........................] - ETA: 1:10:48 - loss: 0.4858 - acc: 0.8073

1200/10399 [==>...........................] - ETA: 1:11:18 - loss: 0.4750 - acc: 0.8158

1300/10399 [==>...........................] - ETA: 1:11:43 - loss: 0.4558 - acc: 0.8269

1400/10399 [===>..........................] - ETA: 1:11:47 - loss: 0.4407 - acc: 0.8350

1500/10399 [===>..........................] - ETA: 1:12:05 - loss: 0.4237 - acc: 0.8433

1600/10399 [===>..........................] - ETA: 1:12:27 - loss: 0.4071 - acc: 0.8512

1700/10399 [===>..........................] - ETA: 1:12:34 - loss: 0.3873 - acc: 0.8594

1800/10399 [====>.........................] - ETA: 1:12:27 - loss: 0.3722 - acc: 0.8661

1900/10399 [====>.........................] - ETA: 1:12:27 - loss: 0.3665 - acc: 0.8700

2000/10399 [====>.........................] - ETA: 1:12:21 - loss: 0.3584 - acc: 0.8740

2100/10399 [=====>........................] - ETA: 1:12:06 - loss: 0.3640 - acc: 0.8738

2200/10399 [=====>........................] - ETA: 1:11:41 - loss: 0.3634 - acc: 0.8768

2300/10399 [=====>........................] - ETA: 1:11:26 - loss: 0.3582 - acc: 0.8796

2400/10399 [=====>........................] - ETA: 1:11:08 - loss: 0.3542 - acc: 0.8817

2500/10399 [======>.......................] - ETA: 1:10:37 - loss: 0.3493 - acc: 0.8840

2600/10399 [======>.......................] - ETA: 1:10:10 - loss: 0.3405 - acc: 0.8877

2700/10399 [======>.......................] - ETA: 1:09:29 - loss: 0.3412 - acc: 0.8889

2800/10399 [=======>......................] - ETA: 1:09:10 - loss: 0.3379 - acc: 0.8911

2900/10399 [=======>......................] - ETA: 1:08:45 - loss: 0.3323 - acc: 0.8931

3000/10399 [=======>......................] - ETA: 1:08:03 - loss: 0.3290 - acc: 0.8947

3100/10399 [=======>......................] - ETA: 1:07:08 - loss: 0.3250 - acc: 0.8961

3200/10399 [========>.....................] - ETA: 1:06:23 - loss: 0.3234 - acc: 0.8972

3300/10399 [========>.....................] - ETA: 1:05:46 - loss: 0.3190 - acc: 0.8991

3400/10399 [========>.....................] - ETA: 1:05:11 - loss: 0.3134 - acc: 0.9012

3500/10399 [=========>....................] - ETA: 1:04:22 - loss: 0.3095 - acc: 0.9029

3600/10399 [=========>....................] - ETA: 1:03:11 - loss: 0.3063 - acc: 0.9042

3700/10399 [=========>....................] - ETA: 1:02:06 - loss: 0.3024 - acc: 0.9057

3800/10399 [=========>....................] - ETA: 1:00:49 - loss: 0.3021 - acc: 0.9061

3900/10399 [==========>...................] - ETA: 59:31 - loss: 0.2990 - acc: 0.9082

4000/10399 [==========>...................] - ETA: 58:15 - loss: 0.3039 - acc: 0.9082

4100/10399 [==========>...................] - ETA: 57:00 - loss: 0.3012 - acc: 0.9095

4200/10399 [===========>..................] - ETA: 55:49 - loss: 0.3007 - acc: 0.9102

4300/10399 [===========>..................] - ETA: 54:38 - loss: 0.2977 - acc: 0.9116

4400/10399 [===========>..................] - ETA: 53:35 - loss: 0.2960 - acc: 0.9125

4500/10399 [===========>..................] - ETA: 52:33 - loss: 0.2946 - acc: 0.9131

4600/10399 [============>.................] - ETA: 51:22 - loss: 0.2921 - acc: 0.9141

4700/10399 [============>.................] - ETA: 50:07 - loss: 0.2947 - acc: 0.9134

4800/10399 [============>.................] - ETA: 48:51 - loss: 0.2940 - acc: 0.9142

4900/10399 [=============>................] - ETA: 47:41 - loss: 0.2913 - acc: 0.9151

5000/10399 [=============>................] - ETA: 46:32 - loss: 0.2894 - acc: 0.9158

5100/10399 [=============>................] - ETA: 45:22 - loss: 0.2891 - acc: 0.9159

5200/10399 [==============>...............] - ETA: 44:16 - loss: 0.2891 - acc: 0.9162

5300/10399 [==============>...............] - ETA: 43:10 - loss: 0.2876 - acc: 0.9170

5400/10399 [==============>...............] - ETA: 42:06 - loss: 0.2869 - acc: 0.9174

5500/10399 [==============>...............] - ETA: 40:58 - loss: 0.2860 - acc: 0.9176

5600/10399 [===============>..............] - ETA: 39:57 - loss: 0.2843 - acc: 0.9182

5700/10399 [===============>..............] - ETA: 38:55 - loss: 0.2812 - acc: 0.9193

5800/10399 [===============>..............] - ETA: 37:55 - loss: 0.2779 - acc: 0.9203

5900/10399 [================>.............] - ETA: 36:51 - loss: 0.2754 - acc: 0.9212

6000/10399 [================>.............] - ETA: 35:48 - loss: 0.2734 - acc: 0.9218

6100/10399 [================>.............] - ETA: 34:47 - loss: 0.2714 - acc: 0.9225

6200/10399 [================>.............] - ETA: 33:46 - loss: 0.2690 - acc: 0.9232

6300/10399 [=================>............] - ETA: 32:47 - loss: 0.2686 - acc: 0.9237

6400/10399 [=================>............] - ETA: 31:48 - loss: 0.2686 - acc: 0.9237

6500/10399 [=================>............] - ETA: 30:50 - loss: 0.2670 - acc: 0.9242

6600/10399 [==================>...........] - ETA: 29:53 - loss: 0.2655 - acc: 0.9247

6700/10399 [==================>...........] - ETA: 28:57 - loss: 0.2672 - acc: 0.9243

6800/10399 [==================>...........] - ETA: 28:02 - loss: 0.2679 - acc: 0.9247

6900/10399 [==================>...........] - ETA: 27:06 - loss: 0.2672 - acc: 0.9251

7000/10399 [===================>..........] - ETA: 26:13 - loss: 0.2666 - acc: 0.9253

7100/10399 [===================>..........] - ETA: 25:19 - loss: 0.2667 - acc: 0.9252

7200/10399 [===================>..........] - ETA: 24:27 - loss: 0.2664 - acc: 0.9254

7300/10399 [====================>.........] - ETA: 23:34 - loss: 0.2647 - acc: 0.9260

7400/10399 [====================>.........] - ETA: 22:42 - loss: 0.2638 - acc: 0.9264

7500/10399 [====================>.........] - ETA: 21:51 - loss: 0.2626 - acc: 0.9268

7600/10399 [====================>.........] - ETA: 21:00 - loss: 0.2624 - acc: 0.9268

7700/10399 [=====================>........] - ETA: 20:10 - loss: 0.2612 - acc: 0.9273

7800/10399 [=====================>........] - ETA: 19:20 - loss: 0.2608 - acc: 0.9273

7900/10399 [=====================>........] - ETA: 18:31 - loss: 0.2592 - acc: 0.9278

8000/10399 [======================>.......] - ETA: 17:42 - loss: 0.2589 - acc: 0.9280

8100/10399 [======================>.......] - ETA: 16:54 - loss: 0.2575 - acc: 0.9285

8200/10399 [======================>.......] - ETA: 16:06 - loss: 0.2584 - acc: 0.9284

8300/10399 [======================>.......] - ETA: 15:19 - loss: 0.2577 - acc: 0.9287

8400/10399 [=======================>......] - ETA: 14:32 - loss: 0.2565 - acc: 0.9290

8500/10399 [=======================>......] - ETA: 13:45 - loss: 0.2574 - acc: 0.9288

8600/10399 [=======================>......] - ETA: 12:59 - loss: 0.2584 - acc: 0.9285

8700/10399 [========================>.....] - ETA: 12:13 - loss: 0.2575 - acc: 0.9289

8800/10399 [========================>.....] - ETA: 11:27 - loss: 0.2582 - acc: 0.9287

8900/10399 [========================>.....] - ETA: 10:43 - loss: 0.2581 - acc: 0.9287

9000/10399 [========================>.....] - ETA: 9:58 - loss: 0.2582 - acc: 0.9286

9100/10399 [=========================>....] - ETA: 9:13 - loss: 0.2569 - acc: 0.9290

9200/10399 [=========================>....] - ETA: 8:29 - loss: 0.2565 - acc: 0.9291

9300/10399 [=========================>....] - ETA: 7:45 - loss: 0.2564 - acc: 0.9291

9400/10399 [==========================>...] - ETA: 7:01 - loss: 0.2553 - acc: 0.9296

9500/10399 [==========================>...] - ETA: 6:18 - loss: 0.2533 - acc: 0.9302

9600/10399 [==========================>...] - ETA: 5:35 - loss: 0.2542 - acc: 0.9301

9700/10399 [==========================>...] - ETA: 4:52 - loss: 0.2538 - acc: 0.9303

9800/10399 [===========================>..] - ETA: 4:10 - loss: 0.2535 - acc: 0.9303

9900/10399 [===========================>..] - ETA: 3:27 - loss: 0.2531 - acc: 0.9303

10000/10399 [===========================>..] - ETA: 2:45 - loss: 0.2524 - acc: 0.9305

10100/10399 [============================>.] - ETA: 2:03 - loss: 0.2515 - acc: 0.9308

10200/10399 [============================>.] - ETA: 1:22 - loss: 0.2519 - acc: 0.9307

10300/10399 [============================>.] - ETA: 40s - loss: 0.2515 - acc: 0.9310 .

10399/10399 [==============================] - 4535s 436ms/step -

Epoch 1/6

loss: 0.2497 - acc: 0.9315 - val\_loss: 0.2163 - val\_acc: 0.9535

Epoch 2/6

loss: 0.2397 - acc: 0.9394 - val\_loss: 0.2033 - val\_acc: 0.9542

Epoch 3/6

loss: 0.2255 - acc: 0.9446 - val\_loss: 0.1860 - val\_acc: 0.9542

Epoch 4/6

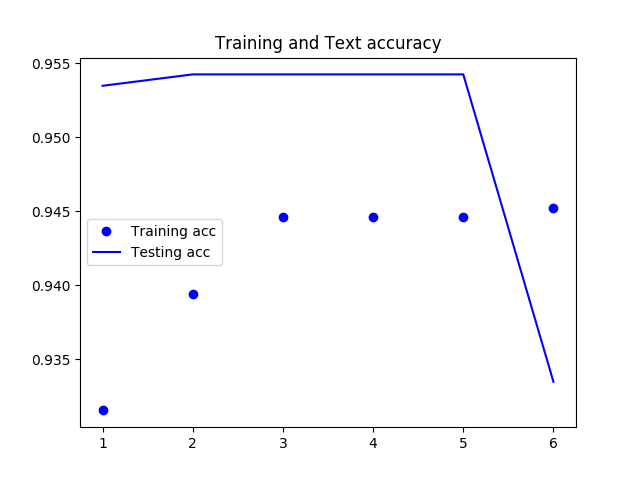
loss: 0.2100 - acc: 0.9446 - val\_loss: 0.2004 - val\_acc: 0.9542

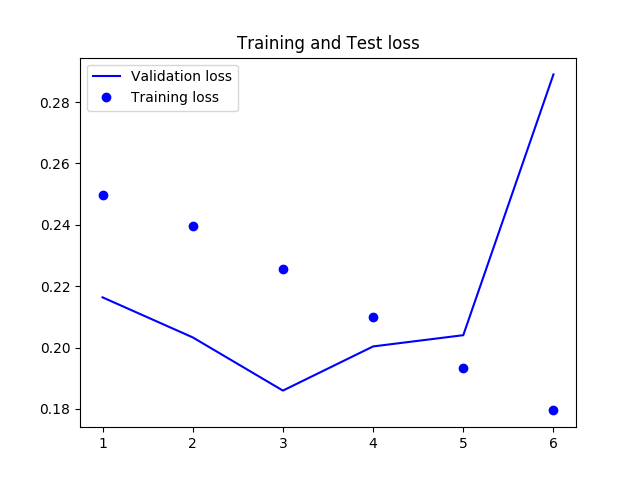
Epoch 5/6

loss: 0.1933 - acc: 0.9446 - val\_loss: 0.2040 - val\_acc: 0.9542

Epoch 6/6

loss: 0.1795 - acc: 0.9452 - val\_loss: 0.2890 - val\_acc: 0.9335





**CHAPTER – 8**

**CONCLUSION AND FUTURE ENHANCEMENT**

**7.1. Conclusion**

In this project, the online news posts from the different stream are downloaded which is in CSV format. It is then converted into python data frame which is further used in training to the model. The word embedding from GloVe is downloaded which is further used to map the numeric tensor token during the process of vectorization. These files serve as training data. A model based upon Deep Neural Network with LSTM is used to fit the training data. After multiple epochs, the model became able to take comments from the user and provide relevant responses. The accuracy of the model is tested using various input comments. The responses from the model is also analyzed. The model became fairly decent on providing relevant output to the input comments. However, it also suffered from the drawbacks of having less data set for training. However, the accuracy can be easily improved by using more training data.

**7.2. Future Enhancement**

The DNN model can be made more responsive and more accurate by training the above model with more amount of training data. For this, high speed GPU is required. Alternatively, premium services from AWS(Amazon Web Services) can be used to fit the data in short amount of time. In future, stronger GPUs will be available in cheaper prices. Hence, more data can be trained in short amount of time. This will make the DNN model more accurate.

**REFERENCES:**

1. Jason Brownlee (Feb 2018) *How to Use Word Embedding Layers for Deep Learning with Keras,* Available at: *https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/* (Accessed: 18th November 2018).
2. Jeffrey Pennington, Richard Socher, Christopher D. Manning (2014) *GloVe: Global Vectors for Word Representation,* Available at: *https://nlp.stanford.edu/projects/glove/* (Accessed: Nov 2018).
3. Felix A. Gers, Jiirgen Schmidhuber,Fred Cummins (1999) 'Learning t o Forget: Continual Prediction with LSTM', *Artificial Neural Networks, 7 10 September 1999, Conference Publication No. 470 IEE 1999,* (), pp. 850-855.
4. FRANÇOIS CHOLLET (2018) *Deep Learning with Python*, ISBN 9781617294433 edn., 20 Baldwin Road PO Box 761 Shelter Island, NY 11964: Manning Publications Co..
5. G. Shanmugasundaram (2017) 'Investigation on Social Media Spam Detection', *International Conference on Innovations in information Embedded and Communication Systems (ICIIECS),* 17(978-1-5090-3294-5), pp. .
6. Julien Fontanarava, Gabriella Pasi,Marco Viviani (2017) 'Feature Analysis for Fake Review Detection through Supervised Classification', *International Conference on Data Science and Advanced Analytics,* 17(978-1-5090-5004-8), pp. 658-666.
7. Li Deng and Dong Yu (2014) *Deep Learning: Methods and Applications*, Volume 7 Issues 3-4, ISSN: 1932-8346 edn., Tokyo: Foundations and Trends in Signal Processing.  
   Megan Risdal (2016) *Getting Real about Fake News,* Available at: *https://www.kaggle.com/mrisdal/fake-news/home* (Accessed: Nov 2018).
8. Mykhailo Granik, Volodymyr Mesyura (2017) 'Fake News Detection Using Naive Bayes Classifier', *First Ukraine Conference on Electrical and Computer Engineering,* 17(978-1-5090-3006-4), pp. 900-903.
9. Suman Ravuri,Andreas Stolcke (2015) 'A COMPARATIVE STUDY OF NEURAL NETWORK MODELS FOR LEXICAL INTENT CLASSIFICATION', *ASRU,* 15(978-1-4799-7291-3), pp. 368-374.
10. Sydney A. Barnard, Soon M. Chung, Vincent A. Schmidt (2017) 'Content-based Clustering and Visualization of Social Media Text Messages', *International Conference on Data and Software Engineering (ICoDSE),* 17(978-1-5386-1449-5), pp. .
11. Vivek Yadav (Jun 23, 2017) *Deep learning setup for Ubuntu 16.04: Tensorflow 1.2, keras, opencv3, python3, cuda8 and cudnn5.1,* Available at: *https://medium.com/@vivek.yadav/deep-learning-setup-for-ubuntu-16-04-tensorflow-1-2-keras-opencv3-python3-cuda8-and-cudnn5-1-324438dd46f0* (Accessed: 18th November 2018).
12. Yan Huang, Wei Wang, Liang Wang, Tieniu Tan (2014) 'A General Nonlinear Embedding Framework Based on Deep Neural Network', *22nd International Conference on Pattern Recognition,* 14(1051-4651), pp. 732-737.