



Savitribai Phule Pune University Home Fourth Year of Computer Engineering (2019 Course) 410255: Laboratory Practice V		
<b>Teaching Scheme</b> <b>Practical: 2 Hours/Week</b>	<b>Credit</b> <b>01</b>	<b>Examination Scheme</b> <b>Term Work: 50 arks</b> <b>Practical: 50 Marks</b>
<b>Companion Course:</b> High Performance Computing(410250), Deep Learning(410251)		
<b>Course Objectives:</b> <ul style="list-style-type: none"> <li>To understand and implement searching and sorting algorithms.</li> <li>To learn the fundamentals of GPU Computing in the CUDA environment.</li> <li>To illustrate the concepts of Artificial Intelligence/Machine Learning(AI/ML).</li> <li>To understand Hardware acceleration.</li> <li>To implement different deep learning models.</li> </ul>		
<b>Course Outcomes:</b> <p> <b>CO1: Analyze and measure</b> performance of sequential and parallel algorithms.  <b>CO2: Design and Implement</b> solutions for multicore/Distributed/parallel environment.  <b>CO3: Identify and apply</b> the suitable algorithms to solve AI/ML problems.  <b>CO4: Apply</b> the technique of Deep Neural network for implementing Linear regression and classification.  <b>CO5: Apply</b> the technique of Convolution (CNN) for implementing Deep Learning models.  <b>CO6: Design and develop</b> Recurrent Neural Network (RNN) for prediction.         </p>		
<b>Guidelines for Instructor's Manual</b> <p>Laboratory Practice V is for practical hands on for core courses High Performance Computing and Data Learning. The instructor's manual is to be developed as a hands-on resource and as ready reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface etc), University syllabus, conduction and Assessment guidelines, topics under consideration-concept, objectives, outcomes, set of typical applications/assignments/ guidelines, references among others.</p>		
<b>Guidelines for Student's Laboratory Journal</b> <p>The laboratory assignments are to be submitted by student in the form of journal. Journal may</p>		

consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software and Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, Algorithm/Database design, test cases, conclusion/analysis). Program codes with sample output of all performed assignments are to be submitted as softcopy.

### **Guidelines for Laboratory /Term Work Assessment**

Continuous assessment of laboratory work is to be done based on overall performance and lab assignments performance of student. Each lab assignment assessment will assign grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each lab assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness reserving weightage for successful mini-project completion and related documentation.

### **Guidelines for Practical Examination**

- Both internal and external examiners should jointly frame suitable problem statements for practical examination based on the term work completed.
- During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement.
- The supplementary and relevant questions may be asked at the time of evaluation to test the student's for advanced learning, understanding of the fundamentals, effective and efficient implementation.
- Encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. So adhering to these principles will consummate our team efforts to the promising boost to the student's academics.

### **Guidelines for Laboratory Conduction**

- List of recommended programming assignments and sample mini-projects is provided for reference.
- Referring these, Course Teacher or Lab Instructor may frame the assignments/mini-project by understanding the prerequisites, technological aspects, utility and recent trends related to the respective courses.
- Preferably there should be multiple sets of assignments/mini-project and distribute among batches of students.
- Real world problems/application based assignments/mini-projects create interest among learners serving as foundation for future research or startup of business projects.
- Mini-project can be completed in group of 2 to 3 students.

- Software Engineering approach with proper documentation is to be strictly followed.
- Use of open source software is to be encouraged.
- Instructor may also set one assignment or mini-project that is suitable to respective course beyond the scope of syllabus.

Operating System recommended :- 64-bit Open source Linux or its derivative

Programming Languages: Object Oriented Languages

C++/JAVA/PYTHON/R

Programming tools recommended: Front End: Java/Perl/PHP/Python/Ruby/.net, Backend :

MongoDB/MYSQL/Oracle, Database Connectivity : ODBC/JDBC

### Suggested List of Laboratory Experiments/Assignments

#### 410250: High Performance Computing

**Any 4 Assignments and 1 Mini Project is Compulsory**

##### Group 1

1. Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS .
2. Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.
3. Implement Min, Max, Sum and Average operations using Parallel Reduction.
4. Write a CUDA Program for :
  1. Addition of two large vectors
  2. Matrix Multiplication using CUDA C
5. Implement HPC application for AI/ML domain.

##### Group 2

6. Mini Project: Evaluate performance enhancement of parallel Quicksort Algorithm using MPI
7. Mini Project: Implement Huffman Encoding on GPU
8. Mini Project: Implement Parallelization of Database Query optimization
9. Mini Project: Implement Non-Serial Polyadic Dynamic Programming with GPU Parallelization

#### 410251: Course Code : Deep Learning

**Any 3 Assignments and 1 Mini Project is Compulsory**

##### Group 1

1.	Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.
2.	<b>Classification using Deep neural network</b> (Any One from the following) <ol style="list-style-type: none"> <li>1. Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset <a href="https://archive.ics.uci.edu/ml/datasets/letter+recognition">https://archive.ics.uci.edu/ml/datasets/letter+recognition</a></li> <li>2. Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset</li> </ol>
3.	<b>Convolutional neural network (CNN)</b> (Any One from the following) <ul style="list-style-type: none"> <li>• Use any dataset of plant disease and design a plant disease detection system using CNN.</li> <li>• Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.</li> </ul>
4.	<b>Recurrent neural network (RNN)</b> Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.
<b>Group 2</b>	
5.	Mini Project: Human Face Recognition
6.	Mini Project: Gender and Age Detection: predict if a person is a male or female and also their age
7.	Mini Project: Colorizing Old B&W Images: color old black and white images to colorful images

@The CO-PO Mapping Matrix

CO/PO	P O 1	P O 2	P O 3	PO4	P O 5	P O 6	PO7	P O 8	P O 9	PO1 0	PO1 1	P O 12
CO1	1	-	1	1	-	2	1	-	-	-	-	-
CO2	1	2	1	-	-	1	-	-	-	-	-	1
CO3	-	1	1	1	1	1	-	-	-	-	-	-
CO4	3	3	3	-	3	-	-	-	-	-	-	-
CO5	3	3	3	3	3	-	-	-	-	-	-	-
CO6	3	3	3	3	3	-	-	-	-	-	-	-
CO7	3	3	3	3	3		-	-	-	-	-	-



**INDEX**

Sr. No.	Title of Assignment	CO	PO
	<b>Group 1</b>		
1	Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.	CO3, CO4	1,2,3,4, 5
2	Classification using Deep neural network (Any One from the following) 1. Multiclass classification using Deep Neural Network Example: Use the OCR letter recognition dataset <a href="https://archive.ics.uci.edu/ml/datasets/letter+recognition">https://archive.ics.uci.edu/ml/datasets/letter+recognition</a> 2. Binary classification using Deep Neural Networks Example: Classify movie reviews into "positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset.	CO3, CO4	1,2,3,4, 5
3	Convolutional neural network (CNN) (Any One from the following) Use any dataset of plant disease and design a plant disease detection system using CNN.  Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.	CO5	1,2,3,4, 5
4	Recurrent Neural Network(RNN) Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.	CO5	1,2,3,4, 5
5	Vlab: To demonstrate the perceptron learning law.		
	<b>Group 2</b>		
	Mini Project: 1. Human Face Recognition 2. Gender and Age Detection: predict if a person is a male or female and also their age 3. Colorizing Old B&W Images: color old black and white images to colorful images	CO3	2,3,4,5, 6

Use Open Source Software.

**Software Required:**

1. 64 bit open source operating system
2. Object oriented languages C++ /Java/PYTHON/R.
3. Programming tools recommended:  
Front End: Java/Perl/PHP/Python/Ruby/.net,  
Backend : MongoDB/MYSQL/Oracle, Database Connectivity : ODBC/JDBC

**Write-ups must include:**

- **Group:**
- **Assignment No.**
- **Title**
- **Problem Statement**
- **Prerequisites**
- **Course Objectives**
- **Course Outcomes**
- **Theory(in brief)**
- **Algorithm**
- **Test Cases**
- **Conclusion**
- **FAQs:**
- **Output: Printout of program with output.**

### **Prerequisite**

**Title : Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch.  
Document the distinct**

features and functionality of the packages. Aim: Study and installation of following Deep learning Packages :

- i.TensorFlow
- ii.Keras
- iii.Theano
- iV . PyTorch

### **Theory :**

**1)What is Deep learning ?**

**2)What are various packages in python for supporting Machine Learning libraries and which are mainly used for Deep Learning ?**

**3)Compare Tensorflow / Keras/Theano and PyTorch on following points(make a table) :**

- i. Available Functionality
- ii. GUI status
- iii. Versions.
- iv. Features
- v. Compatibility with other environments.
- vi. Specific Application domains.

**4) Enlist the Models Datasets and pretrained models, Libraries and Extensions , Tools related to Tensorflow also**

**discuss any two case studies like (PayPal, Intel, Etc. ) related to TensorFlow.**

**[Ref:<https://www.tensorflow.org/about>]**

**5) Explain the Keras Ecosystem.(keras tuner,kerasNLP,keras,Autokaras and Model Optimization.) Also explain**

following concepts related to keras :

1. Developing sequential Model
2. Training and validation using the inbuilt functions

3. Parameter Optimization. [Ref: <https://keras.io/>]

**6) Explain a simple Theano program. 7) Explain PyTorch Tensors . And also explain Uber's Pyro, Tesla Autopilot.[<https://pytorch.org/>] Steps/ Algorithm**

## **Installation of Tensorflow On Ubuntu:**

### **1. Install the Python Development Environment:**

You need to download Python, the PIP package, and a virtual environment. If these packages are already installed, you

can skip this step. You can download and install what is needed by visiting the following links:

<https://www.python.org/>

<https://pip.pypa.io/en/stable/installing/><https://docs.python.org/3/library/venv.html>

To install these packages, run the following commands in the terminal:

```
sudo apt update
```

```
sudo apt install python3-dev python3-pip python3-venv
```

### **2. Create a Virtual Environment**

Navigate to the directory where you want to store your Python 3.0 virtual environment. It can be in your home

```
directory, or any other directory where your user can read and write permissions. mkdir tensorflow_files
```

```
cd tensorflow_files
```

Now, you are inside the directory. Run the following command to create a virtual environment:

```
python3 -m virtual environment
```

```
virtual environment
```

The command above creates a directory named virtual environment. It contains a copy of the Python binary, the PIP package

manager, the standard Python library, and other supporting files.



### 3. Activate the Virtual Environment

`source virtualenv/bin/activate` Once the environment is activated, the virtual environment's `bin` directory will be added

to the beginning of the `$PATH` variable. Your shell's prompt will alter, and it will show the name of the virtual

environment you are currently using, i.e. `virtualenv`.

### 4. Update PIP

`pip install --upgrade pip`

### 5. Install TensorFlow

The virtual environment is activated, and it's up and running. Now, it's time to install the TensorFlow package. `pip install -- upgrade TensorFlow`

### Installation of Keras on Ubuntu :

#### Prerequisite :

Python version 3.5 or above.

#### STEP 1: Install and Update Python3 and Pip

Skip this step if you already have Python3 and Pip on your machine. `sudo apt install python3 python3.pip sudo pip3 install -- upgrade pip`

#### STEP 2: Upgrade Setuptools

`pip3 install -- upgrade setuptools`

#### STEP 3: Install TensorFlow

`pip3 install tensorflow`

Verify the installation was successful by checking the software package information:

`pip3 show tensorflow`

#### STEP 4: Install Keras

`pip3 install keras`

Verify the installation by displaying the package information:

`pip3 show keras`

[<https://phoenixnap.com/kb/how-to-install-keras-on-linux>]

**Installation of Theano on Ubuntu:**

**Step 1:** First of all, we will install Python3 on our Linux Machine. Use the following command in the terminal to install Python3. `sudo apt-get install python3`

**Step 2:** Now, install the pip module

`sudo apt install python3-pip`

**Step 3:** Now, install the Theano

Verifying Theano package Installation on Linux using PIP

`python3 -m pip show theano`

**Installation of PyTorch**

First, check if you are using python's latest version or not. Because PyGame requires python 3.7 or a higher version

`python3 --version`

`pip3 --version`

`pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f`

[https://download.pytorch.org/whl/torch\\_stable.html](https://download.pytorch.org/whl/torch_stable.html)

[Ref : <https://www.geeksforgeeks.org/install-pytorch-on-linux/>]

**Python Libraries and functions required****1.Tensorflow,keras numpy :**

NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import

numpy use

`import numpy as np`

**pandas:**

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on

top of the Python programming language. To import pandas use

`import pandas as pd`

**sklearn :**

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a

selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering

and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is

built upon NumPy, SciPy and Matplotlib. For importing train\_test\_split use from sklearn.model\_selection import

train\_test\_split

**2. For Theano Requirements:**

Python3

Python3-pip

NumPy

SciPy

BLAS

Sample Code with comments

**1. Tensorflow Test program:2. Keras Test Program:**

```
1 from tensorflow import keras
```

```
from keras import datasets
```

```
#
```

```
# Load MNIST data
```

```
#
```

```
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
```

```
#
```

```
# Check the dataset loaded
```

```
#
```

```
train_images.shape, test_images.shape
```

**3. Theano test program**

```
# Python program showing
```

```
# addition of two scalars
# Addition of two scalars
import numpy
import theano.tensor as T
from theano import function
# Declaring two variables
x = T.dscalar('x')
y = T.dscalar('y')
# Summing up the two numbers
z=x+y
# Converting it to a callable object
# so that it takes matrix as parameters
f = function([x, y], z)
f(5, 7)
4. Test program for PyTorch## The usual imports
import torch
import torch.nn as nn
## print out the pytorch version used
print(torch. version )
```

**Output of Code:**

**Note: Run the code and attach your output of the code here. Conclusion :**

Tensorflow , PyTorch,Keras and Theano all these packages are installed and ready for Deep learning applications . As

per application domain and dataset we can choose the appropriate package and build the required type of Neural Network.

**GROUP: 1****ASSIGNMENT NO: 1**

**AIM: Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.**

**PROBLEM STATEMENT:** Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

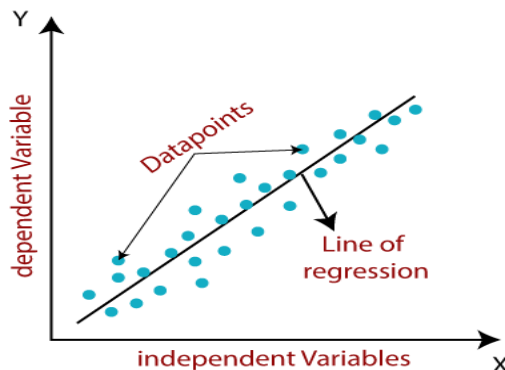
**PREREQUISITES:** Machine Learning

**COURSE OBJECTIVE:** Apply the technique of Deep Neural network for implementing Linear regression and classification.

**THEORY:**

**Deep Learning :** Deep learning is a subfield of machine learning focusing on learning data representations as successive layers of increasingly meaningful representations.

**Linear Regression:** It is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering and the number of independent variables being used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :  $y = \theta_1 + \theta_2 \cdot x$

While training the model we are given :

x: input training data (univariate – one input variable(parameter))

y: labels to data (supervised learning)

**Training the model** – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best  $\theta_1$  and  $\theta_2$  values.

$\theta_1$ : intercept

$\theta_2$ : coefficient of x

Once we find the best  $\theta_1$  and  $\theta_2$  values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

### How to update $\theta_1$ and $\theta_2$ values to get the best fit line ?

#### Cost Function (J):

By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the  $\theta_1$  and  $\theta_2$  values, to reach the best value that minimize the error between predicted y value (pred) and true y value (y).

Cost function(J) of Linear Regression is the Root Mean Squared Error (RMSE) between predicted y value (pred) and true y value (y).

#### Gradient Descent:

To update  $\theta_1$  and  $\theta_2$  values in order to reduce Cost function (minimizing RMSE value) and achieve the best fit line the model uses Gradient Descent. The idea is to start with random  $\theta_1$  and  $\theta_2$  values and then iteratively updating the values, reaching minimum cost.

#### Algorithm:-

Step 1: Download the data set of Boston Housing Prices.

(<https://www.kaggle.com/c/boston-housing>).

Step 2: Importing Libraries

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

Step 3: Importing Data

```
from sklearn.datasets import load_boston
```

```
boston = load_boston()
```

Step 4: Converting data from nd-array to data frame and adding feature names to the data

```
data = pd.DataFrame(boston.data)
```

```
data.columns = boston.feature_names
```

Step 5: Adding 'Price' (target) column to the data

```
data['Price'] = boston.target
```

Step 6: Getting input and output data and further splitting data to training and testing dataset.



```
# Input Data
x = boston.data
# Output Data
y = boston.target
```

Step 7: splitting data to training and testing dataset.

```
#from sklearn.cross_validation import train_test_split
#the submodule cross_validation is renamed and deprecated to model_selection
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_state=0)
```

Step 8: #Applying Linear Regression Model to the dataset and predicting the prices.

```
# Fitting Multi Linear regression model to training model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(xtrain, ytrain)
# predicting the test set results
y_pred = regressor.predict(xtest)
```

Step 9: Plotting Scatter graph to show the prediction

```
# results - 'ytrue' value vs 'y_pred' value
plt.scatter(ytest, y_pred, c='green')
plt.xlabel("Price: in $1000's")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Linear Regression")
plt.show()
```

Step 10: Results of Linear Regression.

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(ytest, y_pred)
print("Mean Square Error : ", mse)
```

**Input:** Dataset of Boston Housing Prices. This dataset concerns the housing prices in the housing city of Boston. The dataset provided has 506 instances with 14 features.

**Output:** Prediction of Boston Housing Prices by plotting graph to show prediction

**Conclusion:-**Housing Prices of Boston city predicted using Linear Regression

**FAQ'S**

**Questions:**

- 1) What is Linear regression?
- 2) What are different types of linear regressions?
- 3) Applications where linear regression is used.
- 4) What are the limitations of linear regression?

**GROUP: A****ASSIGNMENT NO: 2**

**AIM: Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use the IMDB dataset.**

**PROBLEM STATEMENT:** Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use the IMDB dataset.

**PREREQUISITES: Machine learning**

**COURSE OBJECTIVE:** To implement different deep learning model

**COURSE OUTCOME:** Identify and apply the suitable algorithm to solve AI/ML problems.

Apply the techniques of Deep Neural Network for implementing Linear regression and Classification.

**THEORY:**

In this case study, our objective is to classify movie reviews as positive or negative. This is a classic *binary classification*, which aims to predict one of two classes (*positive* vs. *negative*). To predict whether a review is positive or negative, we will use the text of the movie review.

Throughout this case study you will learn a few new concepts:

- Vectorizing text with one-hot encoding
- Regularization with:
  - Learning rate
  - Model capacity
  - Weight decay
  - Dropout

**Package requirements:**

```
library(keras)      # for deep learning
library(tidyverse)  # for dplyr, ggplot2, etc.
library(testthat)   # unit testing
library(glue)       # easy print statements
```

**The IMDB dataset:**

Our data consists of 50,000 movie reviews from [IMDB](#). This data has been curated and supplied to us via keras; however, tomorrow we will go through the process of preprocessing the original data on our own. First, let's grab our data and unpack them into training vs test and features vs labels.

```
imdb <- dataset_imdb(num_words = 10000)
c(c(reviews_train, y_train), c(reviews_test, y_test)) %<-% imdb

length(reviews_train)  # 25K reviews in our training data
```

```
[1] 25000
```

```
length(reviews_test)   # 25K reviews in our test data
```

```
[1] 25000
```

**4. Understanding our data:**

The reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset. For example, the integer “14” encodes the 14th most frequent word in the data. Actually, since the numbers 1, 2, and 3 are reserved to identify:

1. start of a sequence
2. unknown words

## 3. padding

```
reviews_train[[1]]
```

```
[1] 1 14 22 16 43 530 973 1622 1385 65 458 4468 66 3941 4 173
[17] 36 256 5 25 100 43 838 112 50 670 2 9 35 480 284 5
[33] 150 4 172 112 167 2 336 385 39 4 172 4536 1111 17 546 38
[49] 13 447 4 192 50 16 6 147 2025 19 14 22 4 1920 4613 469
[65] 4 22 71 87 12 16 43 530 38 76 15 13 1247 4 22 17
[81] 515 17 12 16 626 18 2 5 62 386 12 8 316 8 106 5
[97] 4 2223 5244 16 480 66 3785 33 4 130 12 16 38 619 5 25
[113] 124 51 36 135 48 25 1415 33 6 22 12 215 28 77 52 5
[129] 14 407 16 82 2 8 4 107 117 5952 15 256 4 2 7 3766
[145] 5 723 36 71 43 530 476 26 400 317 46 7 4 2 1029 13
[161] 104 88 4 381 15 297 98 32 2071 56 26 141 6 194 7486 18
[177] 4 226 22 21 134 476 26 480 5 144 30 5535 18 51 36 28
[193] 224 92 25 104 4 226 65 16 38 1334 88 12 16 283 5 16
[209] 4472 113 103 32 15 16 5345 19 178 32
```

We can map the integer values back to the original word index (`dataset_imdb_word_index()`).

The integer number corresponds to the position in the word count list and the name of the vector is the actual word.

```
word_index <- dataset_imdb_word_index() %>%
  unlist() %>%
  sort() %>%
  names()

# The indices are offset by 3 since 0, 1, and 2 are reserved for "padding",
# "start of sequence", and "unknown"
reviews_train[[1]] %>%
  map_chr(~ ifelse(.x >= 3, word_index[.x - 3], "<UNK>")) %>%
  cat()
```

Our response variable is just a vector of 1s (positive reviews) and 0s (negative reviews).

```
str(y_train)
```

```
int [1:25000] 1 0 0 1 0 0 1 0 1 0 ...
```

```
# our labels are equally balanced between positive (1s) and negative (0s)
# reviews
table(y_train)
```

```
y_train
  0    1
12500 12500
```

## 5. Preparing the features:

All inputs and response values in a neural network must be tensors of either floating-point or integer data. Moreover, our feature values should not be relatively large compared to the randomized initial weights *and* all our features should take values in roughly the same range.

Consequently, we need to *vectorize* our data into a format conducive to neural networks. For this data set, we'll transform our list of article reviews to a 2D tensor of 0s and 1s representing if the word was used (aka one-hot encode)

```
# number of unique words will be the number of features
n_features <- c(reviews_train, reviews_test) %>%
  unlist() %>%
  max()

# function to create 2D tensor (aka matrix)
vectorize_sequences <- function(sequences, dimension = n_features) {
  # Create a matrix of 0s
  results <- matrix(0, nrow = length(sequences), ncol = dimension)

  # Populate the matrix with 1s
  for (i in seq_along(sequences))
    results[i, sequences[[i]]] <- 1
  results
}

# apply to training and test data
x_train <- vectorize_sequences(reviews_train)
x_test <- vectorize_sequences(reviews_test)

# unit testing to make sure certain attributes hold
expect_equal(ncol(x_train), n_features)
expect_equal(nrow(x_train), length(reviews_train))
expect_equal(nrow(x_test), length(reviews_test))
```

Our transformed feature set is now just a matrix (2D tensor) with 25K rows and 10K columns (features).

```
dim(x_train)
```

```
[1] 25000 9999
```

Let's check out the first 10 rows and columns:

```
x_train[1:10, 1:10]
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
[1,]	1	1	0	1	1	1	1	1	1	0
[2,]	1	1	0	1	1	1	1	1	1	0
[3,]	1	1	0	1	0	1	1	1	1	0
[4,]	1	1	0	1	1	1	1	1	1	1
[5,]	1	1	0	1	1	1	1	1	0	1
[6,]	1	1	0	1	0	0	0	1	0	1
[7,]	1	1	0	1	1	1	1	1	1	0
[8,]	1	1	0	1	0	1	1	1	1	1
[9,]	1	1	0	1	1	1	1	1	1	0
[10,]	1	1	0	1	1	1	1	1	1	1




## 6. Preparing the labels:

In contrast to MNIST, the labels of a binary classification will just be one of two values, 0 (negative) or 1 (positive). We do not need to do any further preprocessing.

```
str(y_train)
```

```
int [1:25000] 1 0 0 1 0 0 1 0 1 0 ...
```

## 7. Initial model:

Since we are performing binary classification, our output activation function will be the *sigmoid activation function* . Recall that the sigmoid activation is used to predict the probability of the output being positive. This will constrain our output to be values ranging from 0-100%.

```
network <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = n_features) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

[Hide](#)

```
summary(network)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	160000
dense_1 (Dense)	(None, 16)	272
dense_2 (Dense)	(None, 1)	17
Total params: 160,289		
Trainable params: 160,289		
Non-trainable params: 0		

We're going to use *binary\_crossentropy* since we only have two possible classes.

```
network %>% compile(  
  optimizer = "rmsprop",  
  loss = "binary_crossentropy",  
  metrics = "accuracy"  
)
```

Now let's train our network for 20 epochs and we'll use a batch size of 512 because, as you'll find out, this model overfits very quickly (remember, large batch sizes compute more accurate gradient descents that traverse the loss more slowly).

```
history <- network %>% fit(  
  x_train,  
  y_train,  
  epochs = 20,  
  batch_size = 512,  
  validation_split = 0.2  
)
```

Check out our initial results:

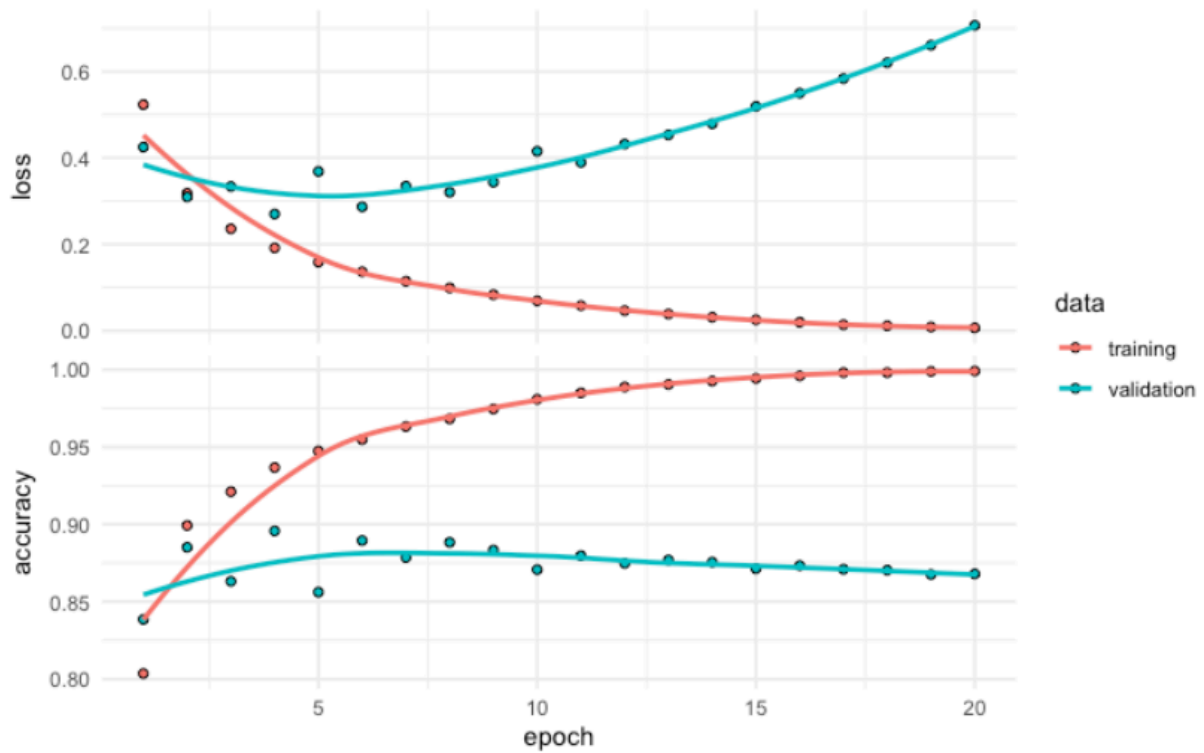
```
best_epoch <- which.min(history$metrics$val_loss)  
best_loss <- history$metrics$val_loss[best_epoch] %>% round(3)  
best_acc <- history$metrics$val_accuracy[best_epoch] %>% round(3)  
  
glue("Our optimal loss is {best_loss} with an accuracy of {best_acc*100}%")
```

```
Our optimal loss is 0.27 with an accuracy of 89.6%
```

In the previous module, we had the problem of underfitting; however looking at our learning curve for this model it's obvious that we have an overfitting problem.

Hide

```
plot(history)
```



**CONCLUSION:** In this story, we applied the concepts of Deep Neural Network on the IMDB dataset. I would recommend trying out other datasets as well.

## FAQ's

### 1. Explain types of loss function with example

In mathematical optimization and decision theory, a loss function or cost function (sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.

### 2. What are L1 and L2 loss functions?

L1 and L2 are two common loss functions in machine learning/deep learning which are mainly used to minimize the error. L1 loss function is also known as Least Absolute Deviations in short LAD. L2 loss function is also known as Least square errors in short LS.

### 3. What is deep networking?

What is a deep neural network? At its simplest, a neural network with some level of complexity, usually at least two layers, qualifies as a deep neural network (DNN), or deep net for short. Deep nets process data in complex ways by employing sophisticated math modeling

**4. What is a deep neural network with an example?**

DL deals with training large neural networks with complex input output transformations. One example of DL is the mapping of a photo to the name of the person(s) in photo as they do on social networks and describing a picture with a phrase is another recent application of DL.

**5. Why are deep neural networks used?**

Deep networks require a large amount of annotated data for training. With efficient training algorithms, deep neural networks are capable of separating millions of labeled images. Moreover, the trained network can also be used for learning efficient image representations for other similar benthic data sets.

**GROUP: A****ASSIGNMENT NO: 3**

**AIM:** Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.(Convolutional neural network (CNN))

**PROBLEM STATEMENT:** Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.(Convolutional neural network (CNN))

**PREREQUISITES:** Machine Learning

**COURSE OBJECTIVE:** To implement different deep learning model

**COURSE OUTCOME:** Apply the techniques of Convolution (CNN) for implementing Deep Learning models

**THEORY:**

**Steps :**

- 1. Installing Jupyter notebook with python3**
- 2. import the required libraries-numpy,matplotlib.pyplot,pandas,seaborn**
- 3. Importing Data (kaggle and scikit-learn library) and Checking out**
- 4. Exploratory Data Analysis**
- 5. Get Data Ready For Training CNN model**
- 6. Split Data into Train, Test**

The Fashion-MNIST dataset is proposed as a more challenging replacement dataset for the MNIST dataset.

It is a dataset composed of 60,000 small square 28×28 pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The mapping of all 0-9 integers to class labels is listed below.

- 0: T-shirt/top
- 1: Trouser
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal

- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle boot

It is a more challenging classification problem than MNIST and top results are achieved by deep learning convolutional neural networks with a classification accuracy of about 90% to 95% on the hold out test dataset.

The example below loads the Fashion-MNIST dataset using the Keras API and creates a plot of the first nine images in the training dataset.

```
1 # example of loading the fashion mnist dataset
2 from matplotlib import pyplot
3 from keras.datasets import fashion_mnist
4 # load dataset
5 (trainX, trainy), (testX, testy) = fashion_mnist.load_data()
6 # summarize loaded dataset
7 print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
8 print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
9 # plot first few images
10 for i in range(9):
11     # define subplot
12     pyplot.subplot(330 + 1 + i)
13     # plot raw pixel data
14     pyplot.imshow(trainX[i], cmap=pyplot.get_cmap('gray'))
15 # show the figure
16 pyplot.show()
```

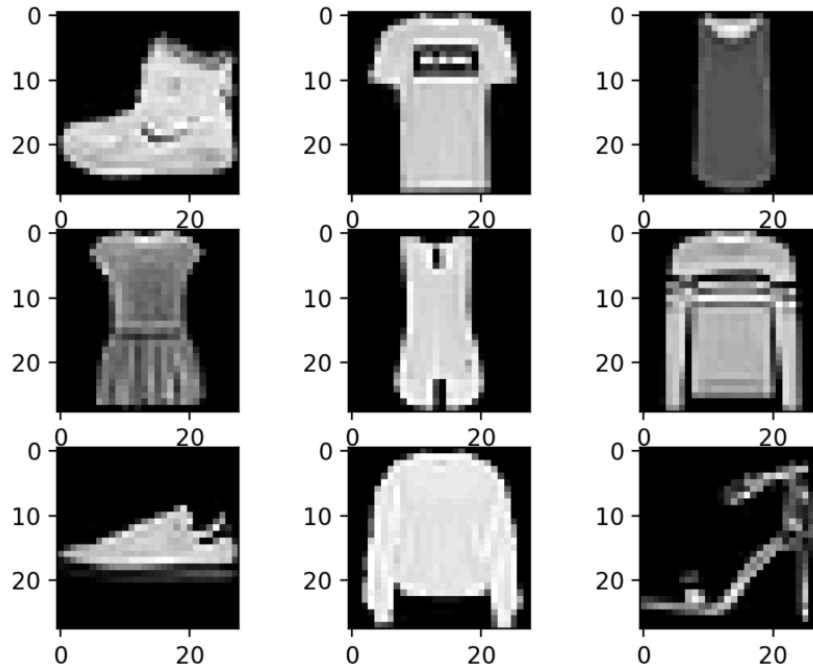
Running the example loads the Fashion-MNIST train and test dataset and prints their shape.

We can see that there are 60,000 examples in the training dataset and 10,000 in the test dataset and that images are indeed square with 28×28 pixels.

```
1 Train: X=(60000, 28, 28), y=(60000,)
2 Test: X=(10000, 28, 28), y=(10000,)
```

A plot of the first nine images in the dataset is also created showing that indeed the images are grayscale photographs of items of clothing.





## a. Model Evaluation Methodology

The Keras API supports this by specifying the “*validation\_data*” argument to the *model.fit()* function when training the model, that will, in turn, return an object that describes model performance for the chosen loss and metrics on each training epoch.

```
1 # record model performance on a validation dataset during training
2 history = model.fit(..., validation_data=(valX, valY))
```

In order to estimate the performance of a model on the problem in general, we can use k-fold cross-validation, perhaps 5-fold cross-validation. This will give some account of the model’s variance with both respect to differences in the training and test datasets and the stochastic nature of the learning algorithm. The performance of a model can be taken as the mean performance across k-folds, given with the standard deviation, that could be used to estimate a confidence interval if desired.

We can use the KFold class from the scikit-learn API to implement the k-fold cross-validation evaluation of a given neural network model.

```
1 # example of k-fold cv for a neural net
2 data = ...
3 # prepare cross validation
4 kfold = KFold(5, shuffle=True, random_state=1)
5 # enumerate splits
6 for train_ix, test_ix in kfold.split(data):
7     model = ...
8     ...
```

## b. How to Develop a Baseline Model

### i. Load Dataset

The images are all pre-segmented (e.g. each image contains a single item of clothing), that the images all have the same square size of 28×28 pixels, and that the images are grayscale. Therefore, we can load the images and reshape the data arrays to have a single color channel.

```
1 # load dataset
2 (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
3 # reshape dataset to have a single channel
4 trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
5 testX = testX.reshape((testX.shape[0], 28, 28, 1))
```

We also know that there are 10 classes and that classes are represented as unique integers.

We can, therefore, use a one hot encoding for the class element of each sample, transforming the integer into a 10 element binary vector with a 1 for the index of the class value. We can achieve this with the *to\_categorical()* utility function.

### ii. Prepare Pixel Data

```
1 # convert from integers to floats
2 train_norm = train.astype('float32')
3 test_norm = test.astype('float32')
4 # normalize to range 0-1
5 train_norm = train_norm / 255.0
6 test_norm = test_norm / 255.0
```

The *prep\_pixels()* function below implement these behaviors and is provided with the pixel values for both the train and test datasets that will need to be scaled.

```
1 # scale pixels
2 def prep_pixels(train, test):
3     # convert from integers to floats
4     train_norm = train.astype('float32')
5     test_norm = test.astype('float32')
6     # normalize to range 0-1
7     train_norm = train_norm / 255.0
8     test_norm = test_norm / 255.0
9     # return normalized images
10    return train_norm, test_norm
```

### iii. Define Model

```
1 # define cnn model
2 def define_model():
3     model = Sequential()
4     model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
5     model.add(MaxPooling2D((2, 2)))
6     model.add(Flatten())
7     model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
8     model.add(Dense(10, activation='softmax'))
9     # compile model
10    opt = SGD(lr=0.01, momentum=0.9)
11    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
12    return model
```

#### iv. Evaluate Model

```
1 # evaluate a model using k-fold cross-validation
2 def evaluate_model(dataX, dataY, n_folds=5):
3     scores, histories = list(), list()
4     # prepare cross validation
5     kf = KFold(n_folds, shuffle=True, random_state=1)
6     # enumerate splits
7     for train_ix, test_ix in kf.split(dataX):
8         # define model
9         model = define_model()
10        # select rows for train and test
11        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
12        # fit model
13        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
14        # evaluate model
15        _, acc = model.evaluate(testX, testY, verbose=0)
16        print('> %.3f' % (acc * 100.0))
17        # append scores
18        scores.append(acc)
19        histories.append(history)
20    return scores, histories
```

**CONCLUSION:** In this story, we applied the concepts of Convolution Neural Network on the MNIST Fashion dataset. I would recommend trying out other datasets as well.

#### FAQ's

1. What is the use of the convolution layer in CNN?
2. What are the advantages of using CNN over DNN?
3. Why is CNN preferred over ANN for image data?
4. How would you visualise features of CNN in an image classification task?
5. What do you understand about shared weights in CNN?
6. Explain the role of a fully connected (FC) layer in CNN.
7. What is the importance of parameter sharing
8. Explain the different types of Pooling.

**GROUP: A****ASSIGNMENT NO: 4**

**AIM: Recurrent neural network ( RNN) Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.**

**PROBLEM STATEMENT:** Recurrent neural network ( RNN) Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.

**PREREQUISITES:** Machine Learning

**COURSE OBJECTIVE:** To implement different deep learning model

**COURSE OUTCOME:** Design and develop recurrent neural network for prediction.

**THEORY:**

Recurrent Neural Network Model (RNN) for a dataset having continuous data such as Google Stock Prices and how to effectively measure the accuracy of the model using the R2 value.

We will be developing the model, step-by-step from preprocessing onwards, and at the end of the article, I will give you the code of the model that I have created.

**Domain Knowledge on Stock Pricing**

Generally speaking, the prices in the stock market are driven by supply and demand. This makes the stock market similar to other economic markets. When a stock is sold, a buyer and seller exchange money for share ownership. The price for which the stock is purchased becomes the new market price.

Likewise, in Google (the mother company is Alphabet Inc.) the stock prices are made available to the world so that the investors can get an idea about the performance of the company when deciding about their available stocks and for new investors to make wise decisions. You can check for the real-time Google stock prices from here.

## Knowledge on Recurrent Neural Networks (RNNs)

Simply saying, we use RNN for modeling sequence/series data, where we have to analyze a series of data when predicting something. In other words, in series data, a sequence of historical data points must be considered when making predictions, and cannot just derive conclusions just by overlooking the data.

Therefore, unlike other models such as MultiLayer Perceptron, and other simple machine learning models used for prediction, when using an RNN, first we should define the sequence of data to be considered when training one data point into the model. And when we take one data point also, there can be various features that can be derived from that data point, which should also be considered when making predictions, which we call dimensions of a particular data point.

Therefore, when passing the dataset into the RNN, the ultimate size of the dataset would be;

$(\# \text{ of data samples}) * (\# \text{ of dimensions per data point}) * (\text{length of sequence})$

## 2. Overview of the Dataset

The dataset I'll be using in this tutorial is the Google Stock Prices dataset as mentioned earlier, which is available from Kaggle, accessible from [here](#).

When inspecting the dataset, we can see that the data contains stock prices from 2012/1/3 to 2016/12/30. The dataset contains 1258 data points (samples) that are given under 6 fields.

The attributes in this dataset are, 'Date', 'Open', 'High', 'Low', 'Close' and 'Volume'.

Open refers to the started stock price for a particular day, while Close means the ended stock price value. High and Low refers to the maximum and minimum stock price values that were achieved during the same day.

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7380500
1	1/4/2012	331.27	333.87	329.08	666.45	5749400
2	1/5/2012	329.83	330.75	326.89	657.21	6590300
3	1/6/2012	328.34	328.77	323.68	648.24	5405900
4	1/9/2012	322.04	322.29	309.46	620.76	11688800
5	1/10/2012	313.70	315.72	307.30	621.43	8824000
6	1/11/2012	310.59	313.52	309.40	624.25	4817800
7	1/12/2012	314.43	315.26	312.08	627.92	3764400
8	1/13/2012	311.96	312.30	309.37	623.28	4631800
9	1/17/2012	314.81	314.81	311.67	626.86	3832800

(1258, 6)

The above figure gives an overview of the dataset. We could see that all are numerical values. The original dataset contains the values in the Volume field as thousand separated values. Therefore, upon loading the dataset, I have specifically mentioned a thousand parameters as comma (,).

When plotting the attributes corresponding to the Date, we could see the graphs as follows.

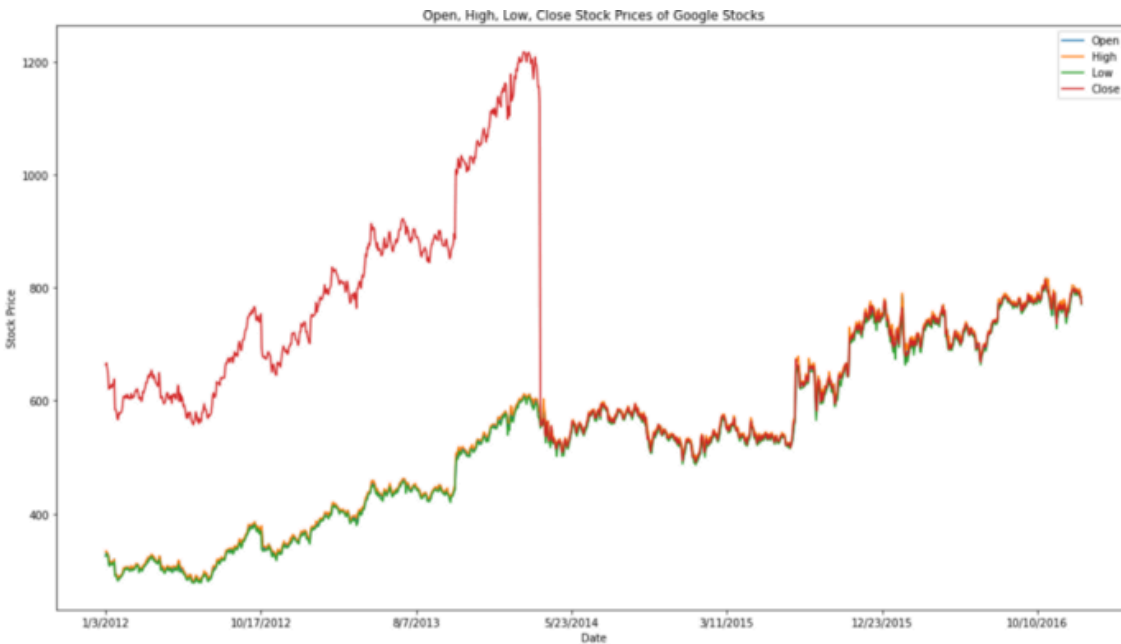


Figure 2: Open, High, Low, Close Stock Prices

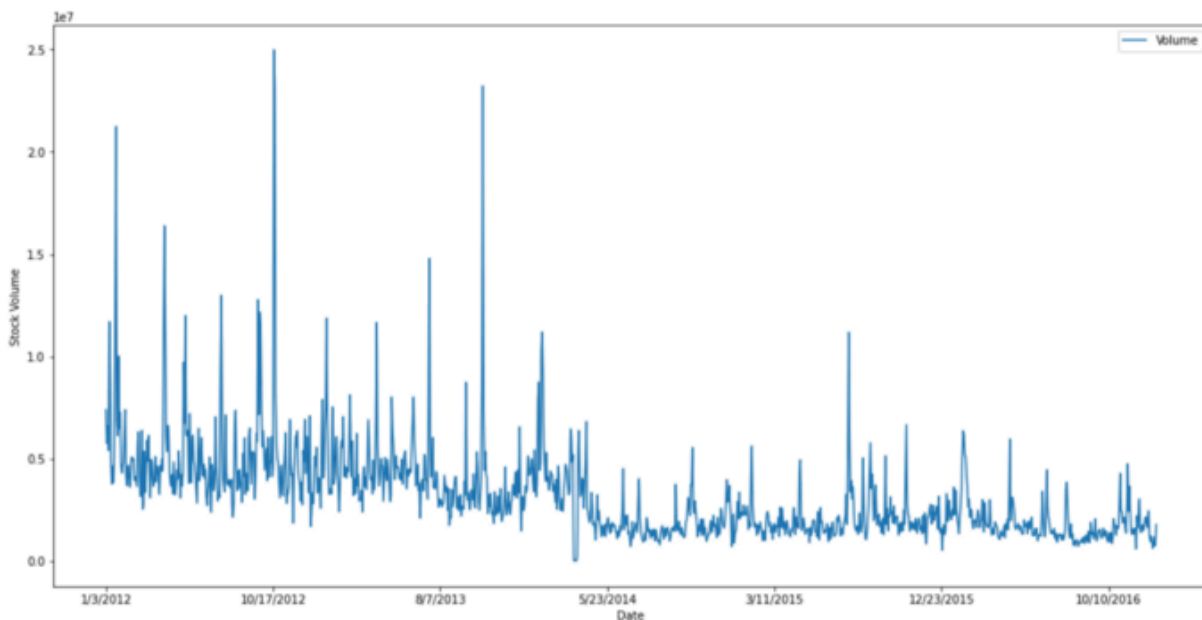


Figure 3: Volume corresponding to Date



When observing the above two plots given in Figures 2 and 3, we could see that there is an abnormality in the data prior to 2014 April. The closing stock price cannot be higher than the High stock price for a given day, and there are significant fluctuations in the Volume values in this period as well

### 3. Developing the Recurrent Neural Network (RNN) Model using TensorFlow Keras LSTM

The development of the model is done in 3 steps as follows.

- Preprocessing
- Feature Engineering
- Developing the Model and Predicting Stock Volume

#### a. Preprocessing

Preprocessing data includes handling missing values and outliers, applying feature coding techniques if needed, scale & standardize features. When we checked for any missing values, we could see that there are no missing values in the dataset.

According to Figure 1, all the values in the dataset are numerical values. Therefore, we do not have to encode any of the fields.

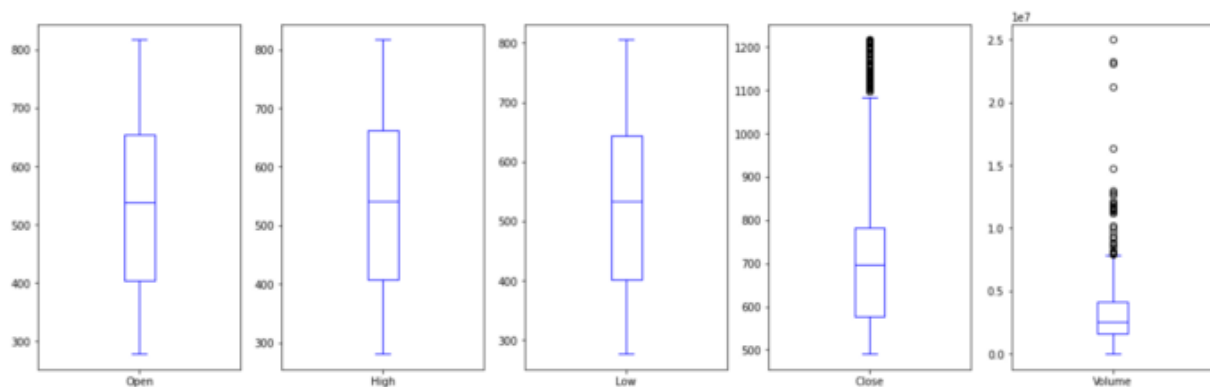


Figure 4: Box plots of the features Open, High, Low, Close

When checking for outliers using box plots, we could see that there are outliers in the Close and Volume attributes as shown in Figure 4. With this matter, when we consider the issue of the abnormal nature of data points before 2014/04, there is a significant problem in data before this date.

However when considering the number of data points in the dataset (1258), if I remove the data points prior to 2014/03/26, the data points in the dataset will be reduced from 1258 to 698 after the removal of the abnormal data.

Since this will affect our dataset and also training of the RNN model, I have not removed the outlier data and have taken the data to generate a Correlation Matrix in the next step.

I have done feature scaling using the MinMaxScaler, which is said to be better to be used on regression-related / continuous data.

### b. Feature Engineering

Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of the model. Here, I have used the Correlation matrix to conduct feature engineering.

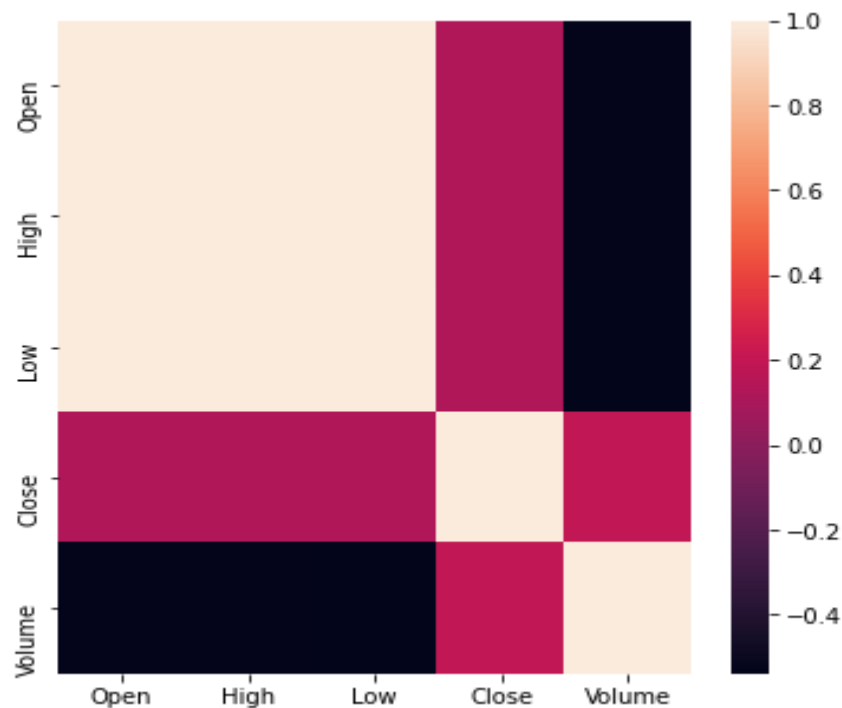


Figure 5: Correlation Matrix for the dataset

When applying the Correlation matrix to the dataset, we can see that the attributes High, Low, Open have a very high correlation with each other, which makes it logical for us to just take one of them for our model development. We can also see that Volume has very less correlation with these three attributes.

On the other hand, we can see that Close has very less correlation with the other attributes. However, as we discussed earlier, we can think that this is affected by the abnormality of values in the Close attribute.

Therefore, I have done the following changes to the dataset after analyzing the features.

- I decided to remove the Close attribute (column) from the dataset considering the abnormality of the data values in this feature.
- Remove Open and Low attributes and keep only the High attribute since they have a very high correlation with High and keeping one from those 3 is enough.

	1	4
0	0.096401	0.295258
1	0.098344	0.229936
2	0.092517	0.263612
3	0.088819	0.216179
4	0.076718	0.467797
...	...	...
1253	0.955292	0.024650
1254	0.964853	0.031286
1255	0.958074	0.045891
1256	0.942574	0.029491
1257	0.936691	0.070569

1258 rows × 2 columns

Figure 6: Dataset after cleaning

### **c. Developing the Model and Predicting Stock Volume**

When developing an RNN Model, we have to reshape the data that we are feeding into the model. To do that, first, we have to find a pattern in the data available and define the number of time steps according to the pattern.

When considering the plots we have created for this we could not see a clear cut pattern/trend in the data by just visual inspection. Since there are two features in our dataset after cleaning, I have used a split sequence method for a multivariate dataset as mentioned here.

In RNN, since we are dealing with time-series data, we have to specify how many past data points we will be considered when generating the sequence. In this tutorial, I have taken 60 past data points (time steps) when generating the data sequences.

Next, I have split the data sequences into training and testing data, and the training data further into train and validation data. I have not used the test data to do the validation because validation data are used to fine-tune the model, and if I used the testing data for validation purposes, then those data will be already seen by the model when trying to predict them later. Next, I am defining the model using Tensorflow Keras LSTM and Dense layers as follows.

In the LSTM layer, I have not specified the activation function explicitly because in the Keras Documentation it is specified that the default activation function for LSTM is 'tanh', and the default recurrent activation function is sigmoid. Therefore those default activation functions will be used here.

When fitting the model with training data and fine-tuning with validation data, I have used 250 epochs for this purpose. When visualizing the training and validation rates with respective epochs, the following graph is obtained.

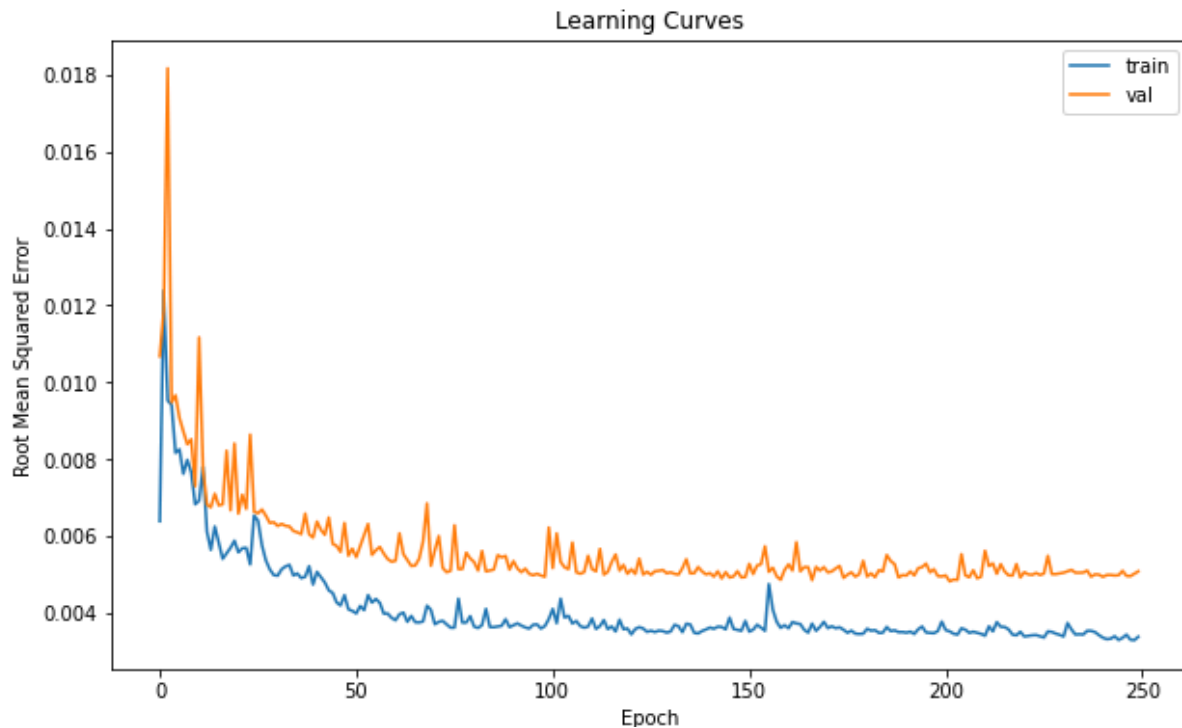


Figure 7: Training and Validation RMSE according to epochs

Even though I have used only 250 epochs here with a batch size of 32 for mini-batch learning, these parameters can be changed.

You can read more about deciding on these parameters of neural network models from .

## Model Evaluation and Predictions

Next, I used the `model.evaluate()` and `model.predict()` methods.

The `model.evaluate()` is used to only check the MSE, RMSE, MAE values of our model when the testing data is used. This does not provide us with the results the model would predict. Therefore to see the results that the model would predict, I have used the `model.predict()` function. Later I have plotted the actual and predicted values of the test data to see how well our model has performed.

Then prediction was done on the test data that I have separated at the beginning, and the following graph shows how the actual test data and test predictions are.

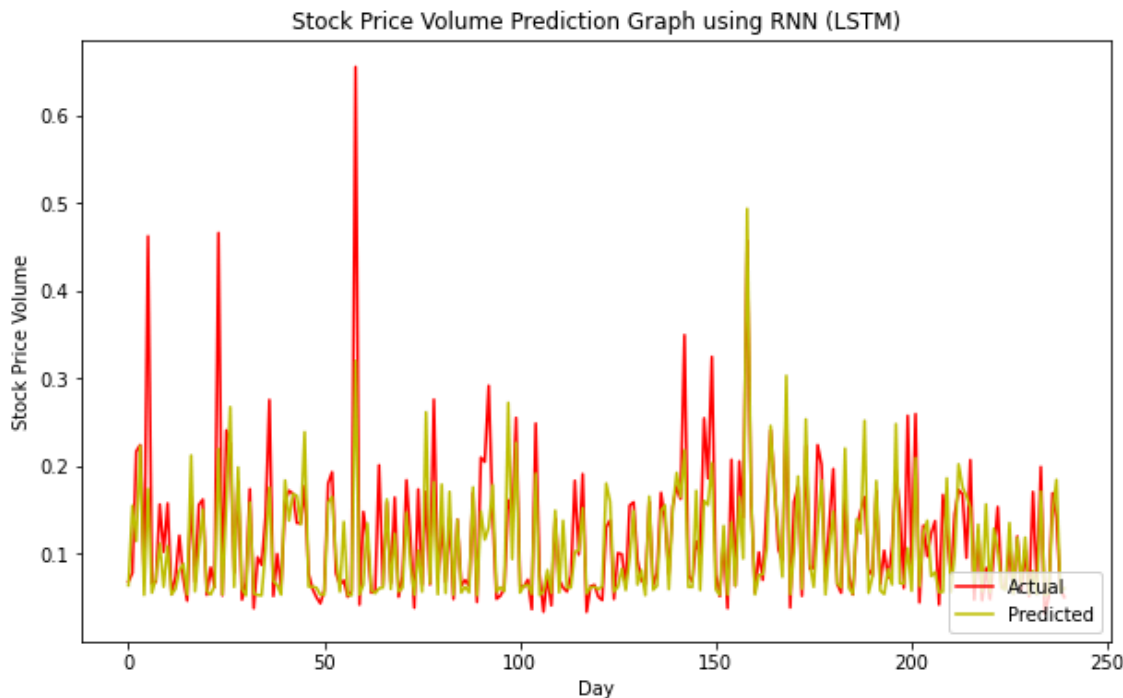


Figure 8: Actual vs. Predicted values for the test data

### Using R Squared to Test the Accuracy of the Model

The R squared is a metric we can use to evaluate how well our regression-based (model that handles continuous data) has performed .

R squared can vary between 0 and 1 and we can evaluate our model using this metric, while a negative value would mean that the training has not been done properly.

When the R squared value was calculated as above, I obtained the value as 0.5914172730027931 which says that there is a 59.1% possibility that the correct value will be predicted by the model I have developed according to the parameters I have given.

**CONCLUSION:** In this story, we applied the concepts of Recurrent neural network ( RNN) on the Google stock prices dataset. I would recommend trying out other datasets as well.

### c. FAQ's:

1. What's the difference between Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) and in which cases would use each one?
2. How many dimensions must the inputs of an RNN layer have? What does each dimension represent? What about its outputs?
3. What are the main difficulties when training RNNs?
4. What are the uses of using RNN in NLP?

- 5. What's the difference between Traditional Feedforward Networks and Recurrent Neural Networks?**
- 6. How to calculate the output of a Recurrent Neural Network (RNN)?**
- 7. How does LSTM compare to RNN?**

**ASSIGNMENT NO: 5**

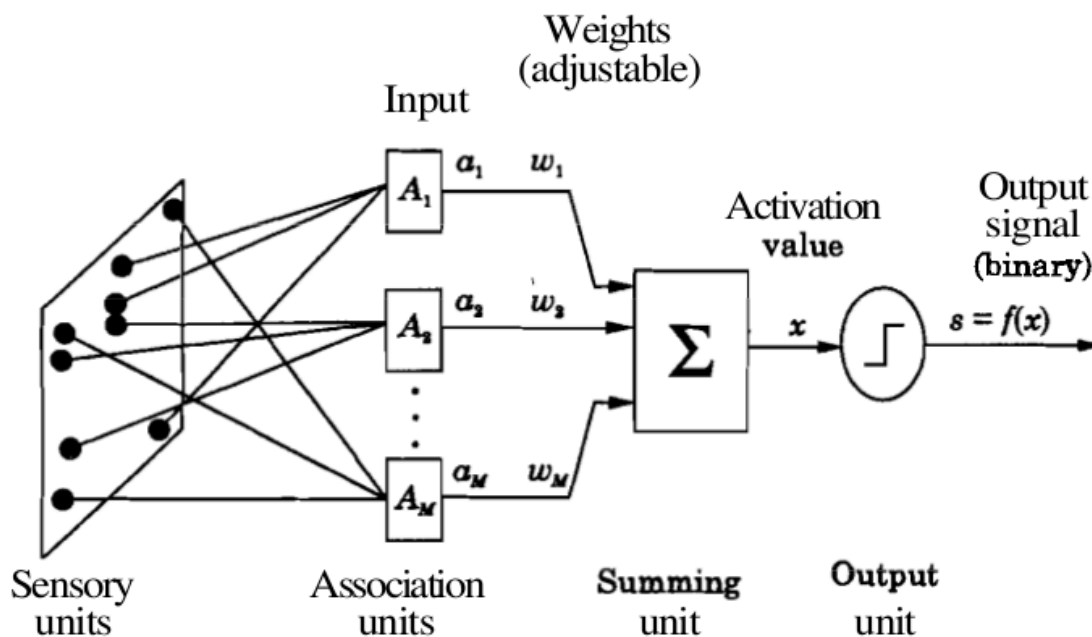
**AIM: Vlab:**To demonstrate the perceptron learning law.

**PROBLEM STATEMENT: Vlab:**To demonstrate the perceptron learning law.

**PREREQUISITES:** Machine Learning

**THEORY:****1. Structure of two-layer feedforward neural network**

A perceptron is a model of a biological neuron. The input to a perceptron is an M-dimensional vector, and each component/dimension of the vector is scaled by a weight. The sum of weighted inputs is computed and compared against a threshold. If the weighted sum exceeds the threshold, the output of the perceptron is '1'. Otherwise, the output of the perceptron is '-1' (or '0'). The output function of a perceptron is a hard-limiting function. Thus the output of the perceptron is binary in nature. The following figure illustrates a perceptron.



**Figure 1:** Perceptron Model.

where  $M$  = number of the elements in the input vector

A two-layer feedforward neural network with hard-limiting output function for the unit in the output layer can be used to perform the task of pattern classification. The number of units in the input layer is equal to the dimension of the input vectors.



The units in the input layer are all linear units, and the input layer merely contributes to fan-out the input to each of the output units. The output layer may consist of one or more perceptrons. The number of perceptron units in the output layer depends on the number of distinct classes in the pattern classification task. If there are only two classes, then one perceptron in the output layer is sufficient. Two perceptrons in the output layer can be used when dealing with four different classes in the pattern classification task. Here, we consider a two-class classification problem, and hence only one perceptron in the output layer. Two-class pattern classification problem.

Let us assume that one subset of input pattern vectors belong to one class, say, class ( $A_1$ ), and the remaining subset of input pattern vectors belong to another class, say, class ( $A_2$ ). Let  $\mathbf{a} = (a_1, a_2, \dots, a_M)$  denote an input pattern vector. The objective in a pattern classification problem is to determine a set of weights  $\mathbf{w} = (w_1, w_2, \dots, w_M)$ , such that the weighted sum

$$\sum_{i=1}^M w_i a_i > \theta, \quad (1) \quad \text{if } (\mathbf{a}) \text{ belongs to } (A_1), \text{ and}$$

$$\sum_{i=1}^M w_i a_i \leq \theta, \quad (2) \quad \text{if } (\mathbf{a}) \text{ belongs to } (A_2).$$

The dividing surface between the two classes is given by

$$\sum_{i=1}^M w_i a_i = \theta. \quad (3)$$

This equation represents a linear hyperplane in the ( $M$ )-dimensional space. For two-dimensional input vectors, this equation represents a straight line. The solution of the classification problem involves determining the weights and the threshold value, such that the resulting hyperplane acts as a dividing surface between the two classes. This is achieved by means of a learning rule, which specifies the manner in which the weights and the threshold value need to be updated.

## 2. Perceptron learning law

The goal of perceptron learning law is to systematically adjust the weights and the threshold in such a manner that a dividing surface between two classes is obtained. The perceptron learning law for a two-class pattern classification problem may be stated as follows:

$$\mathbf{w}^{(m+1)} = \mathbf{w}^{(m)} + \eta \mathbf{a}, \text{ if } \mathbf{a} \in A_1 \text{ and } \mathbf{w}^{(T)} \cdot \mathbf{a} \leq 0, \text{ and}$$

$$\mathbf{w}^{(m+1)} = \mathbf{w}^{(m)} - \eta \mathbf{a}, \text{ if } \mathbf{a} \in A_2 \text{ and } \mathbf{w}^{(T)} \cdot \mathbf{a} > 0. \quad (4)$$

Here ( $m$ ) denotes the index of iteration, or time step. Also,  $\mathbf{a}$  and  $\mathbf{w}$  are augmented input and weight vectors. That is

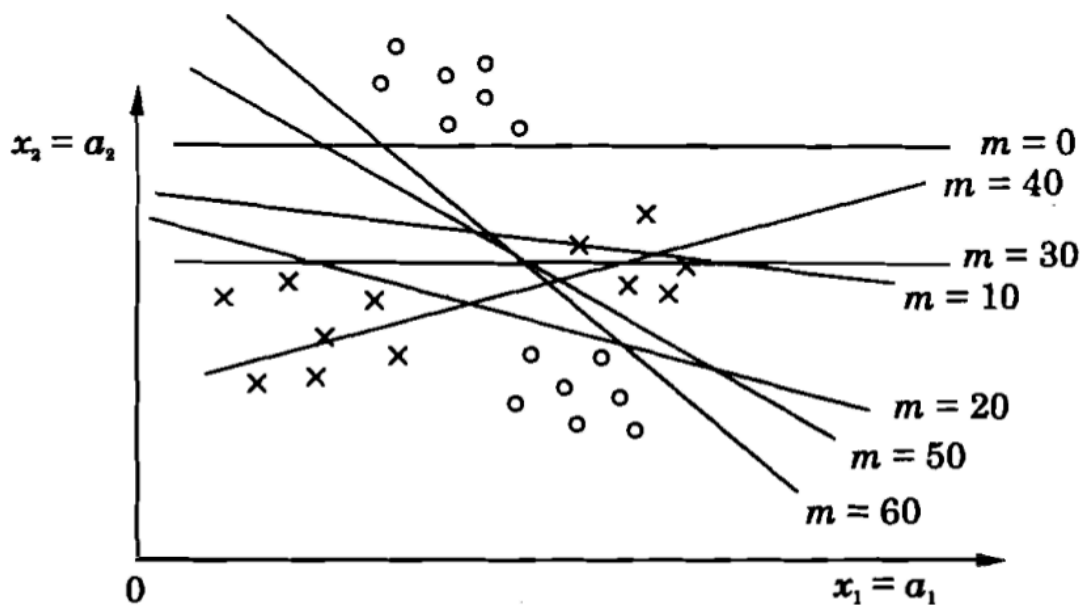
$$\mathbf{a} = (-1, a_1, a_2, \dots, a_M), \text{ and}$$

$$\mathbf{w} = (\theta, w_1, w_2, \dots, w_M).$$

The term ( $\eta$ ) denotes the learning rate, and can be set to a small value (say, 0.1 to 0.5). The value of ( $\eta$ ) can be varied in each learning step, although it is kept constant in perceptron learning. Note that the learning rule modifies the weights only when an input vector is misclassified. When an input vector is classified correctly, there is no adjustment of weights and the threshold. When presenting the input vectors to the network (any neural network in general), we use a term called epoch, which denotes one presentation of all the input pattern vectors to the network. To obtain suitable weights, the learning rule may need to be applied for more than one epoch, typically several epochs. After each epoch, it is verified whether the existing set of weights can correctly classify the input vectors. If so, then the process of updating the weights is terminated. Otherwise the process continues till a desired set of weights is obtained. Note that once a separating hypersurface is achieved, the weights are not modified.

### 3. Perceptron convergence theorem

This theorem states that the perceptron learning law converges to a final set of weight values in a finite number of steps, if the classes are linearly separable. The proof of this theorem first assumes that there exists a set of weights which can correctly classify the input vectors. Then, a bound is obtained on the number of steps used to arrive at the optimum set of weights. This can be illustrated by the figure below, where beginning initially with a set of random weights, the decision boundary finally settles as a valid classifier in a finite number of steps.

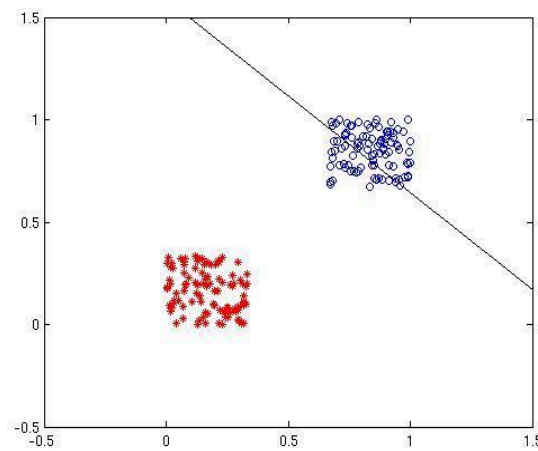


**Figure 2:** Example to illustrate the perceptron convergence in pattern classification problem.

iii.

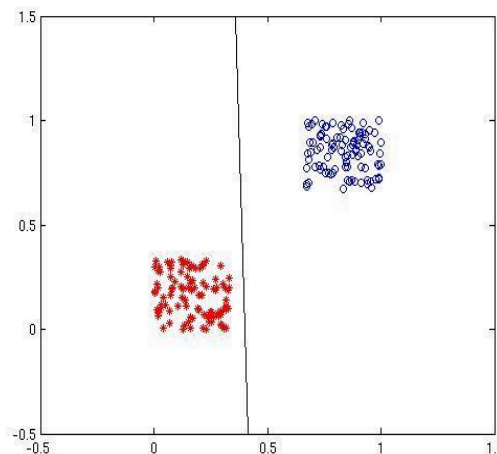
iv. **Illustration of perceptron learning for two-dimensional input**

Consider two linearly separable classes, where each class consists of two-dimensional input pattern vectors. An example of two-class classification problem is shown below. The input vectors belonging to each class are shown in different colors in the following figure. The line is defined by the equation ( $w_1 x + w_2 y = \theta$ .) The straight line is defined by the initial values of the weights and the threshold.



**Figure 1:** *Initial weights and the classes to be separated.*

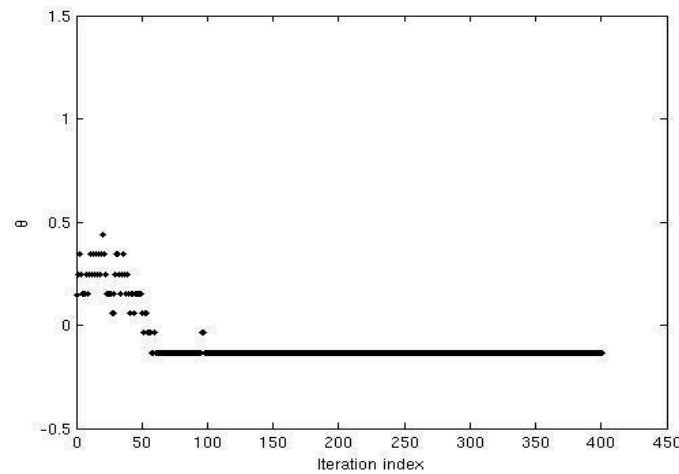
After the convergence of weights, the line separates the two classes. This is shown in the following figure.



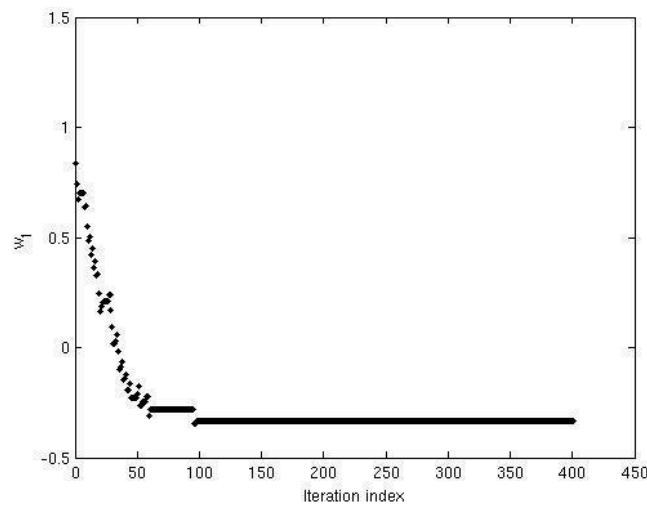
**Figure 2:** *After the weights have converged, the final values of weights determine the line separating the classes.*

### Convergence of weights and threshold value

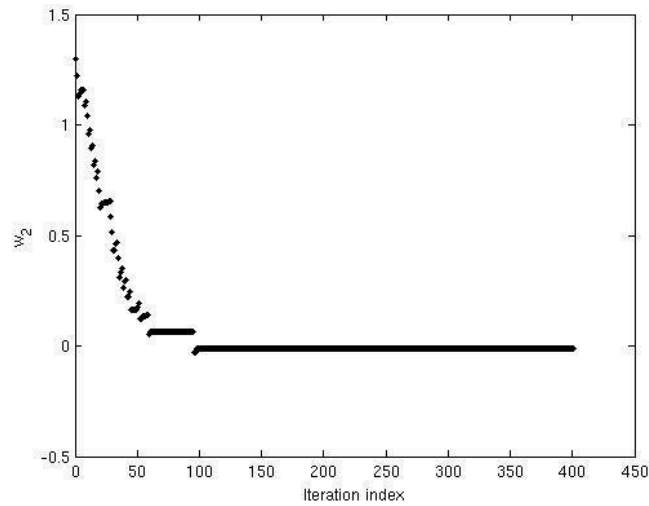
The following figures show the variation of threshold ( $\theta$ ), and the weights ( $w_1$ ) and ( $w_2$ ), as functions of the iteration index. The convergence of ( $\theta$ ,  $w_1$ ) and ( $w_2$ ) can be noted.



**Figure 3:** The variation in values of ( $\theta$ ) with iterations as network reaches convergence.



**Figure 4:** The variation in values of weight ( $w_1$ ) with iterations as the network reaches convergence.



**Figure 5:** *The variation in values of weight ( $w_2$ ) with iterations as the network reaches convergence.*

**CONCLUSION:** In this story, we studied the concepts of Perceptron Learning.