Project4-Neural Networks-FishData

Krishna Nadiminti

April 5, 2016

Objective

To develop a program that creates a neural network for the training data and provides the output for the testdata accordingly

Background

Multilayer Perceptron model with feed forward, Back propagated network along with stochaistic gradient has been used to develop this neural network.

Assumptions for programming

The learning rate has been tested across various alphas to find the best learning rate. Alphas being 0.1,0.2,0.5,0.7,1.

Matrix Modeling

Matrix has been modeled accordingly to find the neural network and make the computations look quite feasible.

For example:

A column with three factors like Sunny, Rainy, Cloudy

then the matrix has been modeled as

$$Sunny : 100$$

$$Rainy : 110$$

$$Cloudy : 101$$

Neumerical vectors are left as such.

It is always a good approach to take the standard deviations of the numerical columns and fit it. However, there is no such implimentaion as the data for numerical is relativey small.

Programming approach

The input matrix has been modeled as provided in the earlier statement. However, for bias to be added an extra column of 1s are appended to the input matrix to proceed further.

if input model is

$$100 + 1 = 1001$$

$$010 + 1 = 0101$$

$$001 + 1 = 0011$$

Upon proceeding the first layer to create the first synapse,if the bias added model is 3*4 matrix. Then the first synapse matrix has been randomly generated which has been the weights of the input to proceed into the Neural networks.The random matrix has been generated as 4*hiddenSize. Hidden size is dynamic and can be change as our wish.

The next layer can be addes hiddensize*(number of columns in output) if the layer is converging at that step, else many nymber of layers can added accordingly.

Program

Read the data file into R environment

```
#dataTraining <- read.csv("C:/Krishna/ML-622/Project4/gameData.csv")
dataTraining <- read.csv("C:/Krishna/ML-622/Project4/fishingData.csv")
```

funtion to model the input Matrix

```
inputMatrixModel<- function(dataTraining){

tempmat<-dataTraining[[1]]  # initializes a matrix to bind
matvec1<- model.matrix(~tempmat)
if(is.numeric(tempmat)){ matvec1<-matvec1[,-1]}
for(i in 2:ncol(dataTraining)-1){
  newtemp<- dataTraining[[i]]
  modelmat<- model.matrix(~newtemp)
  if(is.numeric(newtemp))
    modelmat<- modelmat[,-1]
   # models individual columns and binds as a whole matrix
  matvec1<-cbind( matvec1,modelmat)



}


if(is.numeric(dataTraining[[1]])){matvec1<-matvec1[,-1]}
if(!is.numeric(dataTraining[[1]])) {matvec1<-matvec1[,-c(1:length(unique(dataTraining
[[1]])))]}

return(matvec1)
}
```

Modeling the output

```
output<- dataTraining[[ncol(dataTraining)]]
outputMat<- model.matrix(~output)
```

Learning rate vector and Hidden size declaration

```
alphas<- c(0.0001,0.2,0.5,0.7,1)

hiddensize<- 15
```

Functions required for computation.

```
sigmoid<- function(x){

  output<- 1/(1+exp(-x))

  return (output)
}

sigmoid_To_derivative<- function(output){

  return (output*(1-output))

}

dotproduct<- function (x,y){

  return (x %*% y)
}

testingData<- function(testData,synapse_t0,synapse_t1){

  layer_t0<- testData
  layer_t1<- sigmoid(dotproduct(layer_t0,synapse_t0))
  layer_t2<- sigmoid(dotproduct(layer_t1,synapse_t1))
  return(round(layer_t2))
}
```

Adding the bias vector to the input model matrix

```
x<- inputMatrixModel(dataTraining)
y<- outputMat

x<- cbind(x,c(rep(1,nrow(x)))) #Adding Bias Vector
```

Modeled input appended with Bias

```
print(x)
```

```
##    (Intercept) newtempWeak (Intercept) newtempModerate newtempWarm
## 1            1           0           1               0           1
## 2            1           1           1               0           1
## 3            1           0           1               0           1
## 4            1           0           1               1           0
## 5            1           0           1               0           0
## 6            1           1           1               0           0
## 7            1           1           1               0           0
## 8            1           0           1               1           0
## 9            1           0           1               0           0
## 10           1           0           1               1           0
## 11           1           1           1               1           0
## 12           1           1           1               1           0
## 13           1           0           1               0           1
## 14           1           1           1               1           0
##    (Intercept) newtempWarm (Intercept) newtempRainy newtempSunny
## 1            1           1           1            0           1 1
## 2            1           1           1            0           1 1
## 3            1           1           1            0           0 1
## 4            1           1           1            1           0 1
## 5            1           0           1            1           0 1
## 6            1           0           1            1           0 1
## 7            1           0           1            0           1 1
## 8            1           1           1            0           1 1
## 9            1           0           1            0           1 1
## 10           1           0           1            1           0 1
## 11           1           0           1            0           1 1
## 12           1           1           1            0           1 1
## 13           1           0           1            0           1 1
## 14           1           1           1            1           0 1
```

Modeled output

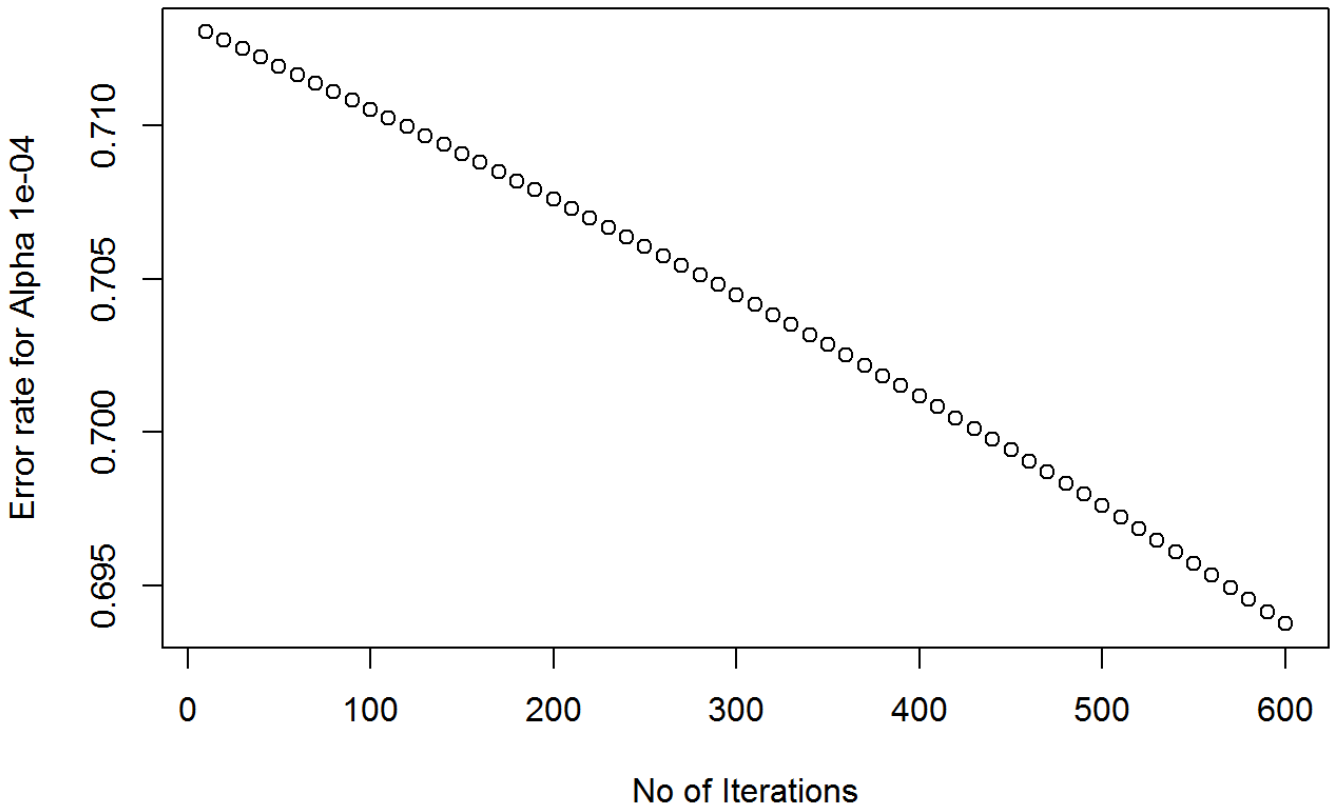
```
print(y)
```

```
##    (Intercept) outputYes
## 1            1         1
## 2            1         0
## 3            1         1
## 4            1         1
## 5            1         0
## 6            1         0
## 7            1         0
## 8            1         1
## 9            1         1
## 10           1         0
## 11           1         1
## 12           1         1
## 13           1         1
## 14           1         0
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$output
## [1] "contr.treatment"
```

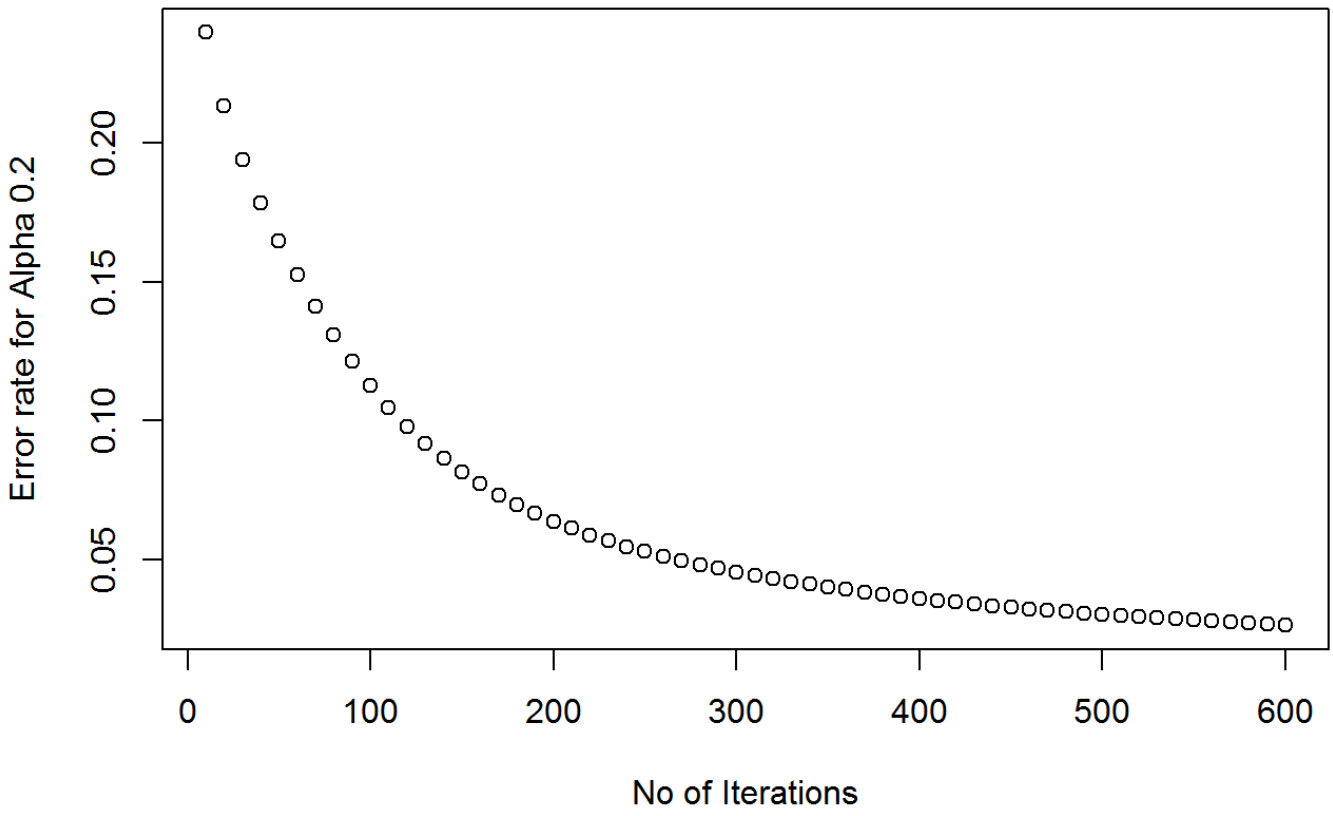
Training the neural network ad estimating the errors

```
set.seed(1)
synapse_0<- matrix(rnorm(ncol(x)*hiddensize*100), ncol(x),hiddensize)
synapse_1<- matrix(rnorm(ncol(x)*hiddensize*100), hiddensize,ncol(y))

plotx<- vector(mode="numeric")
ploty<-vector(mode="numeric")

for (alpha in alphas) ## Loop to estimate the efficiency of each learning rate
{
for(iter in 1:600){

  #feed-forward Network
  layer_0<- x
  layer_1<- sigmoid(dotproduct(layer_0,synapse_0))
  layer_2<- sigmoid(dotproduct(layer_1,synapse_1))

  #change with respect to output
  layer_2_error<- layer_2-y

  layer_2_delta<- layer_2_error*sigmoid_To_derivative(layer_2)

  # print(layer_1_error)
  #Multiply error with the first layer derivative

  layer_1_error<- (layer_2_delta %*% t(synapse_1))
  layer_1_delta<- layer_1_error*sigmoid_To_derivative(layer_1)

  ##Back Propagate the error

  synapse_1_derivative<- (t(layer_1) %*% layer_2_delta)
  synapse_0_derivative<- (t(layer_0) %*% layer_1_delta)



  synapse_1= synapse_1-(alpha*synapse_1_derivative)
  synapse_0= synapse_0-(alpha*synapse_0_derivative)

  if(iter%%10==0){
    error<- mean(abs(layer_2_error))
    plotx<-append(plotx,error)
    ploty<- append(ploty,iter)
  }

}
plot(ploty,plotx,xlab = "No of Iterations",ylab= paste("Error rate for Alpha",alpha) )

print(round(layer_2))

plotx<- vector(mode="numeric")
ploty<-vector(mode="numeric")
```

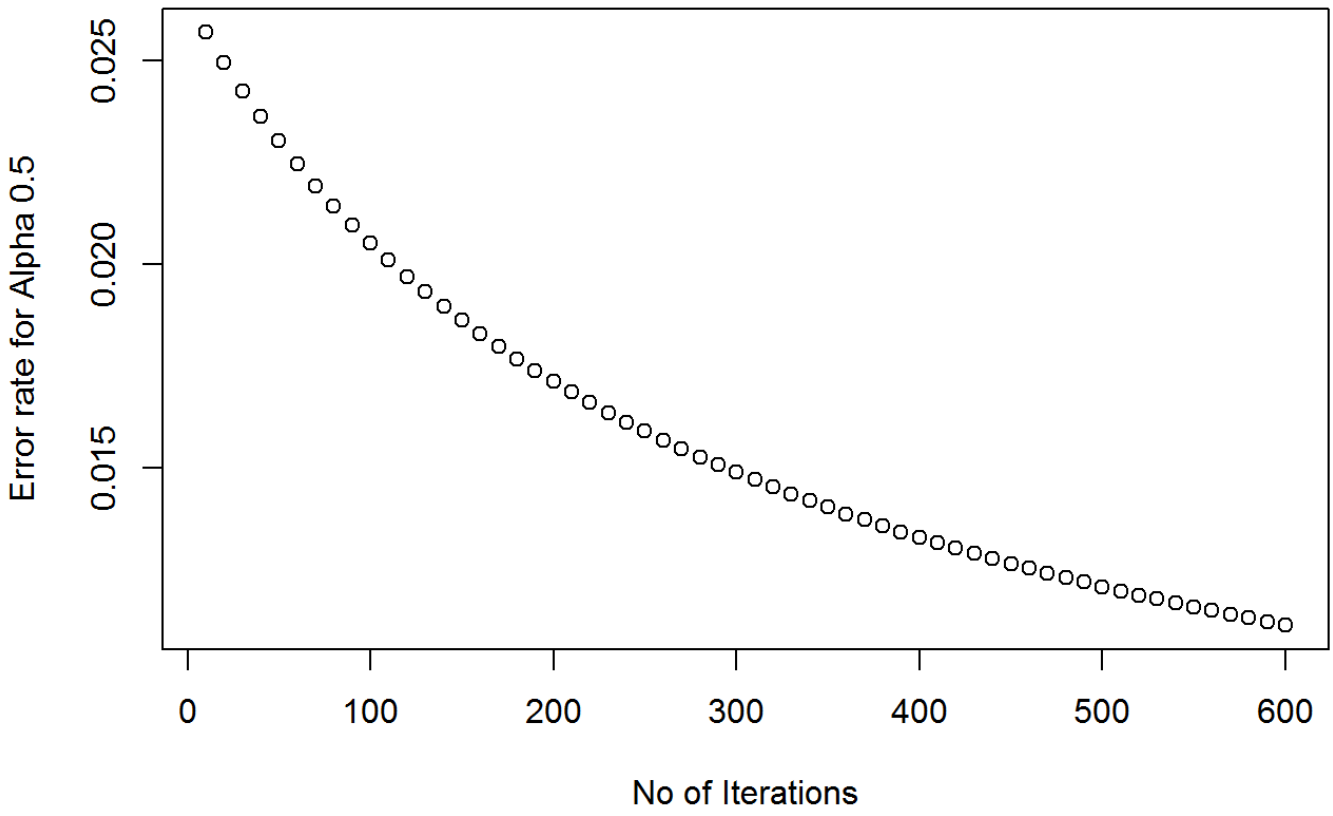
```
}
```



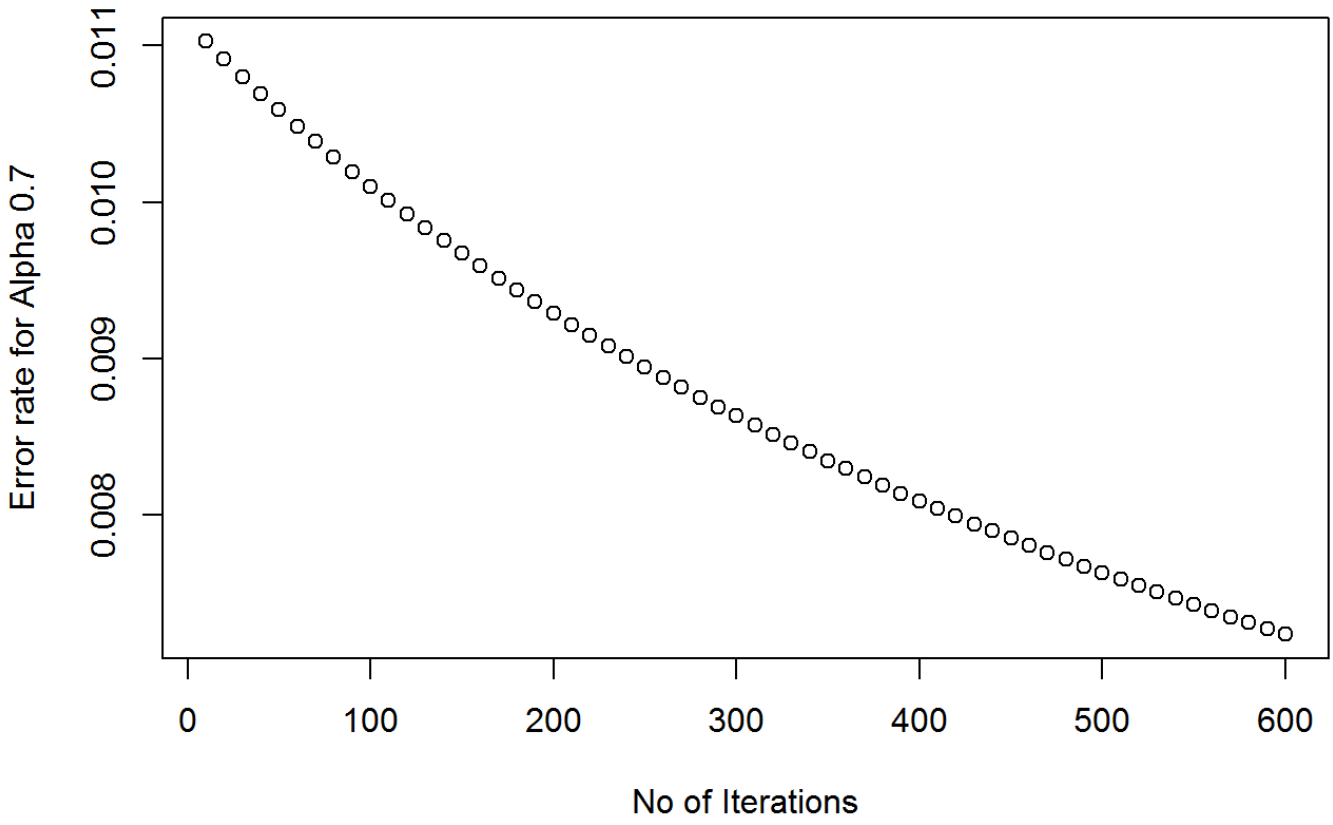

```
##    [,1] [,2]
## 1     0    1
## 2     0    1
## 3     0    1
## 4     0    0
## 5     0    0
## 6     0    0
## 7     0    1
## 8     0    1
## 9     0    0
## 10    0    0
## 11    0    1
## 12    0    1
## 13    0    0
## 14    0    0
```



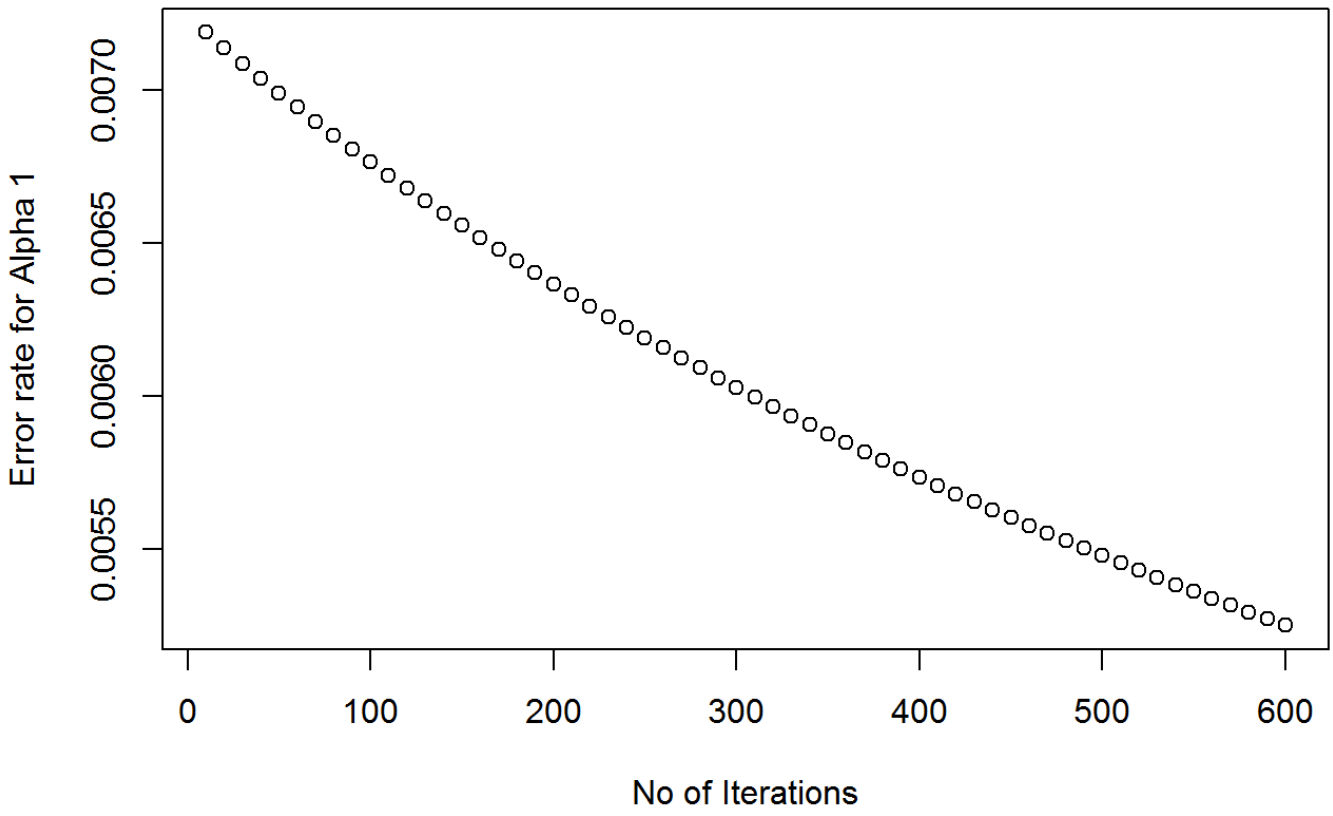

```
##    [,1] [,2]
## 1     1    1
## 2     1    0
## 3     1    1
## 4     1    1
## 5     1    0
## 6     1    0
## 7     1    0
## 8     1    1
## 9     1    1
## 10    1    0
## 11    1    1
## 12    1    1
## 13    1    1
## 14    1    0
```




```
##    [,1] [,2]
## 1     1    1
## 2     1    0
## 3     1    1
## 4     1    1
## 5     1    0
## 6     1    0
## 7     1    0
## 8     1    1
## 9     1    1
## 10    1    0
## 11    1    1
## 12    1    1
## 13    1    1
## 14    1    0
```




```
##    [,1] [,2]
## 1     1    1
## 2     1    0
## 3     1    1
## 4     1    1
## 5     1    0
## 6     1    0
## 7     1    0
## 8     1    1
## 9     1    1
## 10    1    0
## 11    1    1
## 12    1    1
## 13    1    1
## 14    1    0
```




```
##    [,1] [,2]
## 1     1    1
## 2     1    0
## 3     1    1
## 4     1    1
## 5     1    0
## 6     1    0
## 7     1    0
## 8     1    1
## 9     1    1
## 10    1    0
## 11    1    1
## 12    1    1
## 13    1    1
## 14    1    0
```

Output for fishing and gameData

```
## Output For Fishing
## Test Data- Wind,Water,Air,Forecast,Fish
##Strong,Cold,Warm,Sunny

testmodel<- c(1,0,1,0,0,1,1,1,0,1,1)

testingData(t(testmodel),synapse_0,synapse_1)
```

```
##      [,1] [,2]
## [1,]    1    1
```

Output for fishing testdata has been

1. Yes

```
##Output for GameData

#testdataNew<- read.csv("C:/Krishna/ML-622/Project4/gameDataTest.csv",header = FALSE)
#testdataNew<-data.matrix(testdataNew, rownames.force = NA)
#print(testdataNew)
#testingData(testdataNew,synapse_0,synapse_1)
```

The output for gaming test data has been produced as

1.Hide 2.Run 3.Run 4.Hide 5.Run

Discussion:

On observing the grapical representation of error vs iteration.

Alpha value with 1 has shown less error rate and hence, taking learning rate 1 as granted for the neural network.