Midterm_Bipasha

```
library(tidyverse)

## -- Attaching packages ------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ---------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(ggplot2)
library(dplyr)
```

Including Plots

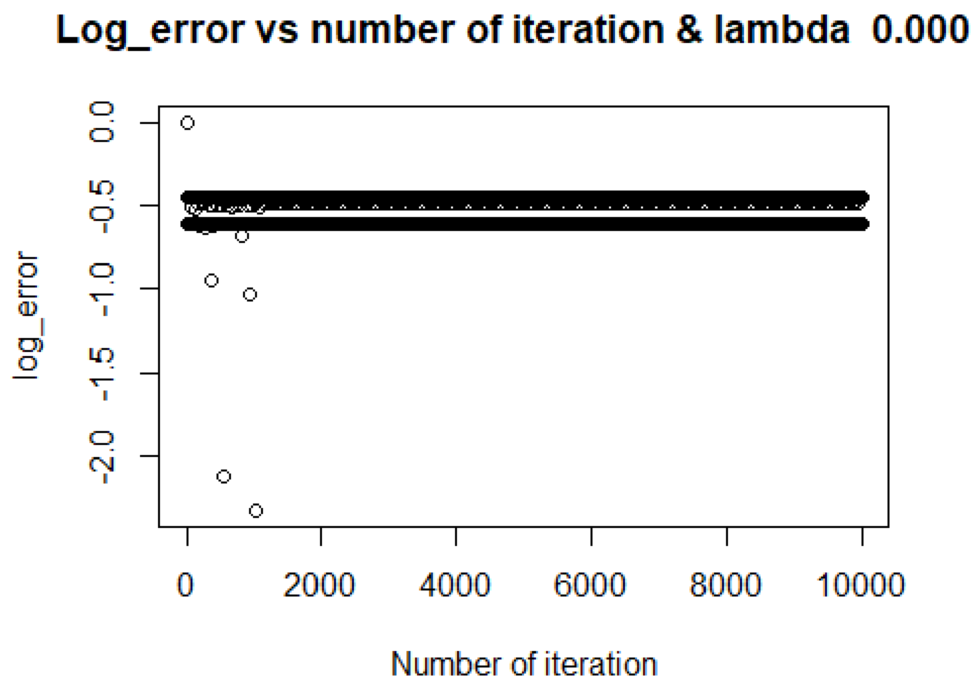
#Activity—- 2(a): #Gradient Descent

```
sigmoid <- function(x) {
   1 / (1 + exp(-x))
}

GD<-function(H,z, x=c(.1,.1), lambda, max_iter){


  tol=.00000001
  error= 1
  errors= matrix(0, max_iter,1)
  num_it=matrix(0,max_iter,1)
   num_iter=0
   while (error > tol && num_iter< max_iter){
     p= sigmoid(H %*% x)
     residual=p-z
     gradient=t(H) %*% residual
     prev= x
     x=x-lambda*gradient

     error=t(residual) %*% residual
     #num_iter=num_iter+1

     #errors(num_iter)=error
     error=norm(prev-x,'2')
     log_error=log10((abs(error)))
```

```
     errors[num_iter,]=log_error
     num_it[num_iter,]=num_iter
     num_iter=num_iter+1
    }
 return (list(x,errors,num_it))


  #errors=errors[1:num_iter]

  }
```

#Activity——–2(b)

```
H<-(matrix(c(1,1,1,2500,535,3000),3,2))
z<-c(0,1,1)

obs_1= GD(H,z, x=c(2,.1),lambda=.001,max_iter=10000)
obs_1[1]

## [[1]]
##           [,1]
## [1,] 4.4123253
## [2,] 0.6331157

obs_2=GD(H,z, x=c(2,.1),lambda=.0001,max_iter=10000)
obs_2[1]

## [[1]]
##            [,1]
## [1,]  2.2354886
## [2,] -0.1074738

error_2 = as.matrix(as.data.frame(obs_2[2]))
it_2 = as.matrix(as.data.frame(obs_2[3]))
plot(it_2,error_2, main="Log_error vs number of iteration & lambda  0.0001",
xlab = "Number of iteration", ylab="log_error")
```




```
obs_3= GD(H,z, x=c(2,.01),lambda=.0000001,max_iter=10000)
obs_3[1]
```

```
## [[1]]
##                  [,1]
## [1,]  1.9999959043
## [2,] -0.0006323291
```

#Activity—-2(c)

```
H<-(matrix(c(1,1,1,2500,535,3000),3,2))
H[,2]<-scale(H[,2])
H
```

```
##      [,1]       [,2]
## [1,]    1  0.3747652
## [2,]    1 -1.1332490
## [3,]    1  0.7584838
```

```
obs1= GD(H,z, x=c(2,.5),lambda=.01,max_iter=10000)
obs1[1]
```

```
## [[1]]
##            [,1]
## [1,]  0.8201937
## [2,] -1.0051149
```

```
obs2= GD(H,z, x=c(2,.5),lambda=.0002,max_iter=10000)
obs2[1]
```

```
## [[1]]
##             [,1]
## [1,]  1.1234766
## [2,] -0.2777514
```

```
obs3= GD(H,z, x=c(2,.5),lambda=.000001,max_iter=10000)
obs3[1]
```

```
## [[1]]
##            [,1]
## [1,] 1.9937903
## [2,] 0.4950948
```

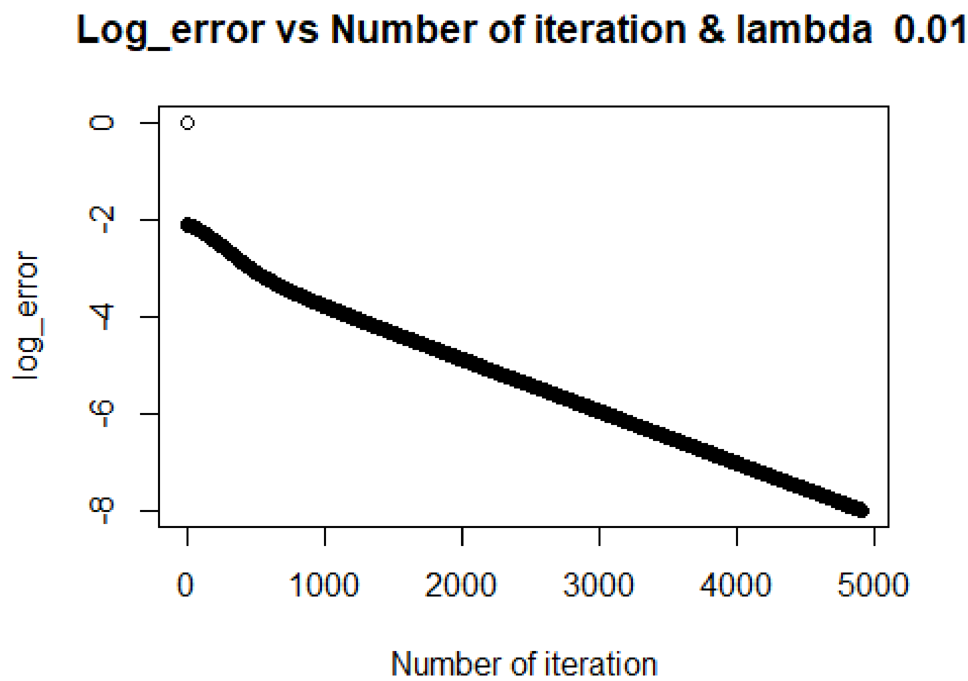
#Activity–2(d)

we need to standardize the inputs, otherwise the network will be ill-conditioned. standardizing is done to have the same range of values for each of the inputs to have stable convergence of weight and biases.Here, after standardizing the input, better value for intercepts and co-efficient was acheived.

#Activity–2(e)

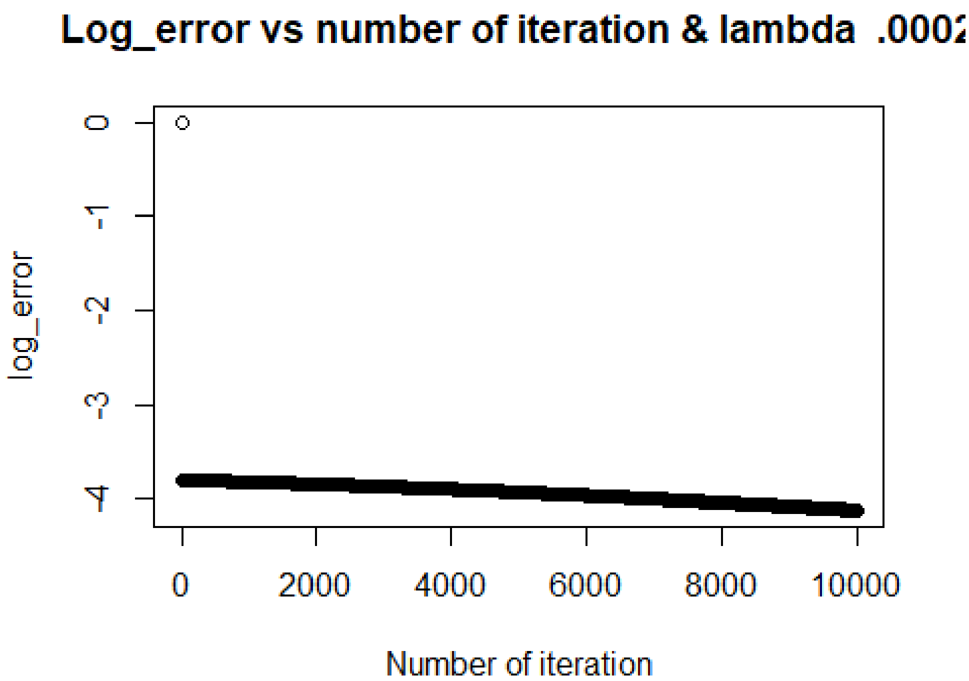
#plotting Log_error as a function of Number of iteration #plot_1

```
error_1 = as.matrix(as.data.frame(obs1[2]))
it_1 = as.matrix(as.data.frame(obs1[3]))
plot(it_1,error_1, main="Log_error vs Number of iteration & lambda  0.01",
xlab = "Number of iteration", ylab="log_error")
```



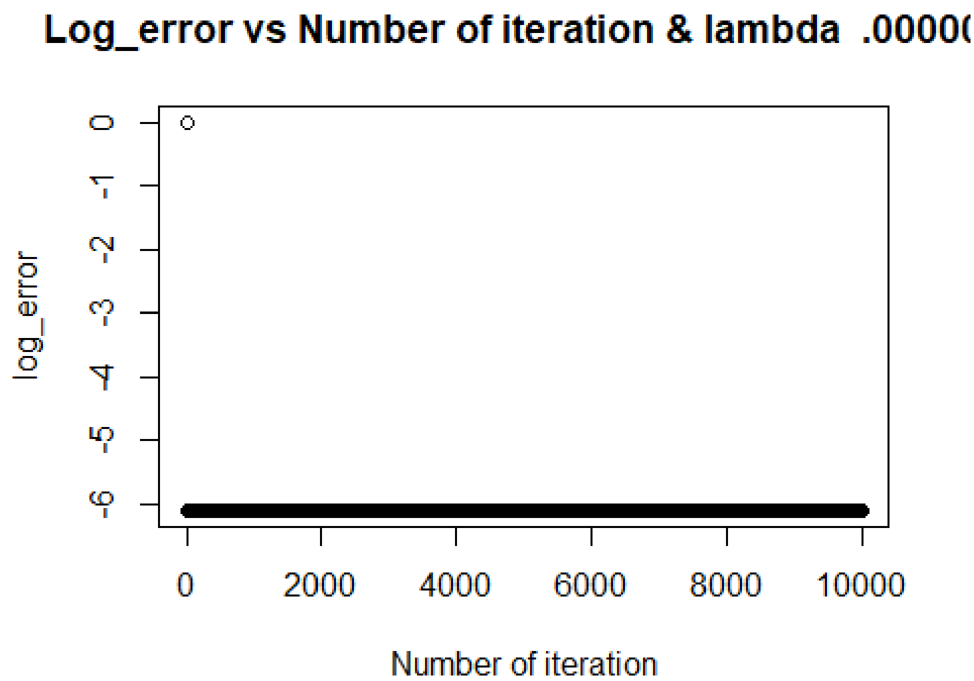
#plot_2

```
error_2 = as.matrix(as.data.frame(obs2[2]))
it_2 = as.matrix(as.data.frame(obs2[3]))
plot(it_2,error_2, main="Log_error vs number of iteration & lambda  .0002",
xlab = "Number of iteration", ylab="log_error")
```



#plot_3

```
error_1 = as.matrix(as.data.frame(obs3[2]))
it_3 = as.matrix(as.data.frame(obs3[3]))
plot(it_3,error_1, main="Log_error vs Number of iteration & lambda  .000001",
xlab = "Number of iteration", ylab="log_error")
```



#Activity 2(f)

From the following plots, it can be concluded that better convergence was acheived from plot2. For plot_1,the convergence was poor(lambda=.01). After decreasing the learning rate(.0000001) better convergence was acheived.So for difference in learing rate, the convergence varies.

#activity–2(g)

Standardizing the features around the center and 0 with a standard deviation of 1 is important when we compare measurements that have different units. Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bais. For example, A variable that ranges between 0 and 10 will outweigh a variable that ranges between 0 and 1.(there is a figure attached in pdf) Using these variables without standardization will give the variable with the larger range weight in the analysis i.e unnormalizing featurescan lead toward an awkward loss function topology which places more emphasis on certain parameter gradients. Transforming the data to comparable scales can prevent this problem.

#Acitivity— 3(a)

```
df<- data.frame(balance=c(2500,535,3000),default=c(0,1,1))
model<- glm(default ~ ., data = df, family = "binomial" )

summary(model)
```

```
##
## Call:
## glm(formula = default ~ ., family = "binomial", data = df)
##
## Deviance Residuals:
##       1        2        3
## -1.3706   0.5136   1.1529
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.3719254  3.6760762   0.645    0.519
## balance     -0.0007714  0.0014624  -0.527    0.598
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3.8191  on 2  degrees of freedom
## Residual deviance: 3.4716  on 1  degrees of freedom
## AIC: 7.4716
##
## Number of Fisher Scoring iterations: 4
```

#Activity-3(b)

```
library(reticulate)

## Warning: package 'reticulate' was built under R version 3.6.3

use_python("C:/Program Files/Python37/python.exe")

import numpy as np
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
X=np.array([[2500],[535],[3000]])
y=np.array([[0],[1],[1]])
logistic_regression = LogisticRegression()
clf = linear_model.LogisticRegression(C=1e40, solver='newton-cg')
fitted_model = clf.fit(X, y)

## C:\Program Files\Python37\lib\site-
packages\sklearn\utils\validation.py:724: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of
y to (n_samples, ), for example using ravel().
##   y = column_or_1d(y, warn=True)

print( fitted_model.intercept_)

## [2.37173882]

print(fitted_model.coef_)

## [[-0.0007713]]
```

#Activity——-5(a)

#Pre_processing Train Data

```
url.train <- "http://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
url.test <- "http://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.test"
download.file(url.train, destfile = "adult_train.csv")
download.file(url.test, destfile = "adult_test.csv")

train <- read.csv("adult_train.csv", header = FALSE,encoding = "latin1")
all_content <- readLines("adult_test.csv")
skip_first <- all_content[-1]
test <- read.csv(textConnection(skip_first), header = FALSE)

Names <- c("Age", "Workclass", "fnlwgt", "Education", "EducationNum",
"MaritalStatus", "occupation", "Relationship","Race", "Sex", "Capital_gain",
"Capital_loss", "hours_per_week", "Native_country", "Income_level")
NROW(train)

## [1] 32561

trainFileName = "adult.data"; testFileName = "adult.test"
if (!file.exists (trainFileName)) download.file (url =
"http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data",
destfile = trainFileName)
if (!file.exists (testFileName)) download.file (url =
"http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test",
destfile = testFileName)

train = read.table (trainFileName, header = FALSE, sep = ",", strip.white =
TRUE, col.names =Names, na.strings = "?", stringsAsFactors = TRUE)
table (complete.cases (train))

##
## FALSE  TRUE
##  2399 30162

NROW(train)

## [1] 32561

myCleanTrain = train [!is.na (train$Workclass) & !is.na(train$occupation), ]
myCleanTrain = myCleanTrain [!is.na (myCleanTrain$Native_country), ]
myCleanTrain$fnlwgt = NULL

NROW(myCleanTrain)

## [1] 30162

head(myCleanTrain)
```

```
##    Age        Workclass Education EducationNum      MaritalStatus
## 1  39         State-gov Bachelors           13      Never-married
## 2  50 Self-emp-not-inc Bachelors           13 Married-civ-spouse
## 3  38           Private   HS-grad            9           Divorced
## 4  53           Private      11th            7 Married-civ-spouse
## 5  28           Private Bachelors           13 Married-civ-spouse
## 6  37           Private   Masters           14 Married-civ-spouse
##          occupation  Relationship  Race    Sex Capital_gain Capital_loss
## 1      Adm-clerical Not-in-family White   Male         2174            0
## 2   Exec-managerial       Husband White   Male            0            0
## 3 Handlers-cleaners Not-in-family White   Male            0            0
## 4 Handlers-cleaners       Husband Black   Male            0            0
## 5    Prof-specialty          Wife Black Female            0            0
## 6   Exec-managerial          Wife White Female            0            0
##   hours_per_week Native_country Income_level
## 1             40  United-States        <=50K
## 2             13  United-States        <=50K
## 3             40  United-States        <=50K
## 4             40  United-States        <=50K
## 5             40           Cuba        <=50K
## 6             40  United-States        <=50K
```

```
df1<-myCleanTrain

X_train<- select(df1,-c('Income_level'))

keeps <- c("Income_level")
ytrain= df1[keeps]

daf<-data.frame(df1)

print(ytrain[1,])

## [1] <=50K
## Levels: <=50K >50K

levels(ytrain$Income_level) <- c(1,0)
```

#Activity—-5(b) #Standardize Data

```
 X_train[]<- lapply(X_train, function(x) if(is.numeric(x)){
                     scale(x, center=TRUE, scale=TRUE)
                      } else x)

#normalize <- function(x) {
#return ((x - min(x)) / (max(x) - min(x)))
#}
```

#Activity—-5(c)

```
library(reticulate)

use_python("C:/Program Files/Python37/python.exe")
```

```r
library(keras)

## Warning: package 'keras' was built under R version 3.6.3

#install_keras()

library(tensorflow)

## Warning: package 'tensorflow' was built under R version 3.6.3
```

#3 layer model without dropout

```r
model1 <- keras_model_sequential()
model1 %>%
layer_dense(units = 5, input_shape = 96, activation ='relu')%>%
layer_dense(units = 1, activation = 'sigmoid')

summary(model1)

## Model: "sequential"
##
______________________________________________________________________________
___
## Layer (type)                        Output Shape                     Param
#
##
==============================================================================
===
## dense (Dense)                       (None, 5)                        485
##
______________________________________________________________________________
___
## dense_1 (Dense)                     (None, 1)                        6
##
==============================================================================
===
## Total params: 491
## Trainable params: 491
## Non-trainable params: 0
##
______________________________________________________________________________
___
```

#3 Layer Model with dropout

```r
model2 <- keras_model_sequential()
model2 %>%
#layer_dropout(rate = 0.2) %>%
layer_dense(units = 5, input_shape = 96, activation ='relu')%>%
layer_dropout(rate = 0.2) %>%
layer_dense(units = 1, activation = 'sigmoid')
```

```
summary(model2)

## Model: "sequential_1"
##
________________________________________________________________________________
___
## Layer (type)                        Output Shape                        Param
#
##
================================================================================
===
## dense_2 (Dense)                     (None, 5)                           485
##
________________________________________________________________________________
___
## dropout (Dropout)                   (None, 5)                           0
##
________________________________________________________________________________
___
## dense_3 (Dense)                     (None, 1)                           6
##
================================================================================
===
## Total params: 491
## Trainable params: 491
## Non-trainable params: 0
##
________________________________________________________________________________
___
```

#Model with four layer without drop_out

```
model3 <- keras_model_sequential()
model3 %>%
layer_dense(units = 10, input_shape = 96, activation ='relu')%>%
layer_dense(units = 5, activation = 'relu') %>%
#layer_dropout(rate = 0.5) %>%
layer_dense(units = 1, activation = 'sigmoid')

summary(model3)

## Model: "sequential_2"
##
________________________________________________________________________________
___
## Layer (type)                        Output Shape                        Param
#
##
================================================================================
===
## dense_4 (Dense)                     (None, 10)                          970
```

```
##
________________________________________________________________________________
___
## dense_5 (Dense)                      (None, 5)                       55
##
________________________________________________________________________________
___
## dense_6 (Dense)                      (None, 1)                       6
##
================================================================================
===
## Total params: 1,031
## Trainable params: 1,031
## Non-trainable params: 0
##
________________________________________________________________________________
___
```

#Model with 4 layer with dropout

```
model4 <- keras_model_sequential()
model4 %>%
layer_dense(units = 10, input_shape = 96, activation ='relu')%>%
layer_dropout(rate = 0.2) %>%
layer_dense(units = 5, activation = 'relu') %>%
#layer_dropout(rate = 0.2) %>%
layer_dense(units = 1, activation = 'sigmoid')

summary(model4)

## Model: "sequential_3"
##
________________________________________________________________________________
___
## Layer (type)                         Output Shape                    Param
#
##
================================================================================
===
## dense_7 (Dense)                      (None, 10)                      970
##
________________________________________________________________________________
___
## dropout_1 (Dropout)                  (None, 10)                      0
##
________________________________________________________________________________
___
## dense_8 (Dense)                      (None, 5)                       55
##
________________________________________________________________________________
___
```

```
## dense_9 (Dense)                     (None, 1)                       6
##
================================================================================
===
## Total params: 1,031
## Trainable params: 1,031
## Non-trainable params: 0
##
________________________________________________________________________________
___
```

#Final model with own choice

```
model5 <- keras_model_sequential()
model5 %>%
layer_dense(units = 10, input_shape = 96, activation ='relu')%>%
#layer_dropout(rate = 0.2) %>%
layer_dense(units = 10, activation = 'relu') %>%
#layer_dropout(rate = 0.2) %>%
layer_dense(units = 5, activation = 'relu') %>%
#layer_dropout(rate = 0.5) %>%
layer_dense(units = 1, activation = 'sigmoid')

summary(model5)
```

```
## Model: "sequential_4"
##
________________________________________________________________________________
___
## Layer (type)                        Output Shape                    Param
#
##
================================================================================
===
## dense_10 (Dense)                    (None, 10)                      970
##
________________________________________________________________________________
___
## dense_11 (Dense)                    (None, 10)                      110
##
________________________________________________________________________________
___
## dense_12 (Dense)                    (None, 5)                       55
##
________________________________________________________________________________
___
## dense_13 (Dense)                    (None, 1)                       6
##
================================================================================
===
## Total params: 1,141
```

```
## Trainable params: 1,141
## Non-trainable params: 0
##
________________________________________________________________________________
___
```

```
library(caret)

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:tensorflow':
##
##     train

## The following object is masked from 'package:purrr':
##
##     lift

#X_train_new[] <-dummyVars(~.,data=X_train,levelsOnly=TRUE)
#head(predict(X_train_new,X_train))
#X_train_new <-dummyVars(~.,data=X_train,levelsOnly=TRUE)
#head(predict(X_train_new,X_train))
dmy <- dummyVars(~., data=X_train, fullRank=TRUE)
X_train_new <- data.frame(predict(dmy, newdata=X_train))
X_train_new1<-as.matrix(X_train_new)

#View(X_train_new)
typeof(X_train_new1)

## [1] "double"

#y_train<-as.data.frame(y_train)
y_train<- as.matrix(ytrain)

#final variable
y_train<-as.numeric(y_train)
typeof(y_train)

## [1] "double"

#nrow(X_train_new1)
ncol(X_train_new1)

## [1] 96
```

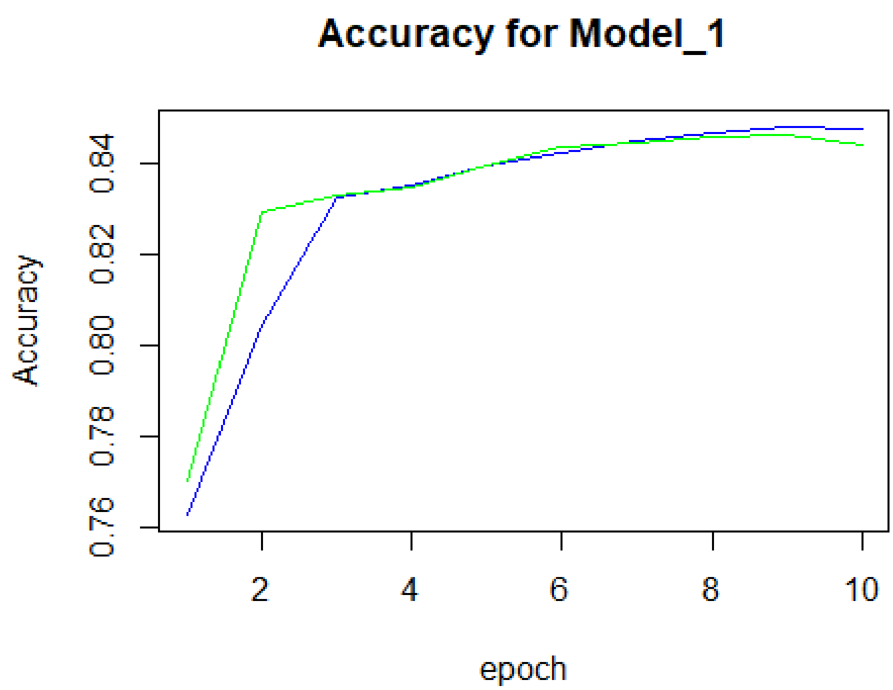
#Activity 5(d) #Plotting the training and the validation accuracy

```
model1 %>% compile ( loss = 'binary_crossentropy', metrics = 'accuracy',
optimizer = optimizer_sgd(lr = 0.01))
```

```
history1<-model1 %>%
  fit( X_train_new1, y_train, epochs = 10, batch_size = 32, validation_split
= 0.2 )
```

#Model1

```
plot(history1$metrics$acc, main="Accuracy for Model_1", xlab = "epoch",
ylab="Accuracy", col="blue", type="l")
lines(history1$metrics$val_acc, col="green")
```

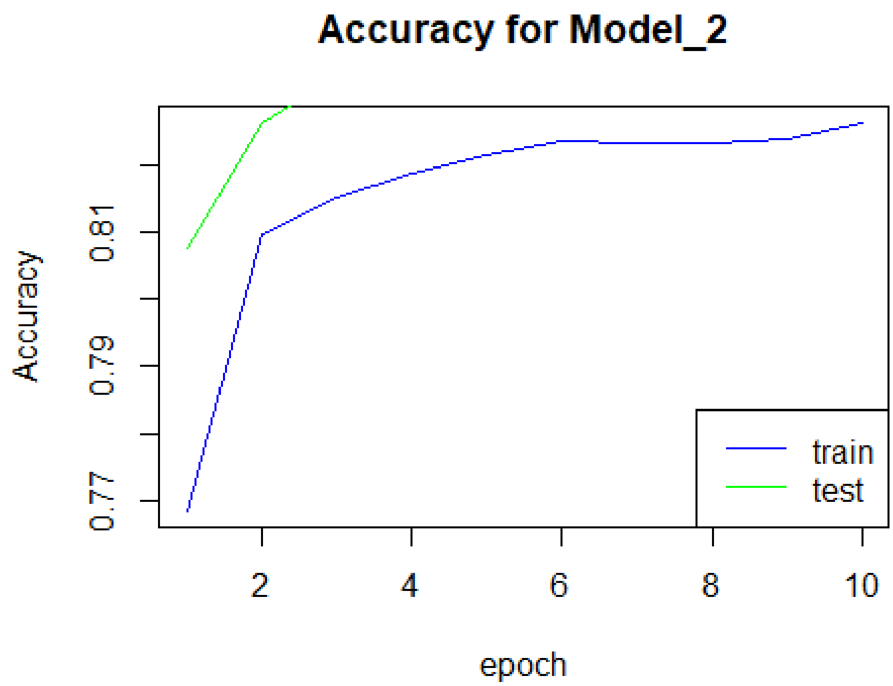


#model 2

```
model2 %>% compile ( loss = 'binary_crossentropy', metrics = 'accuracy',
optimizer = optimizer_sgd(lr = 0.01))

history2<-model2 %>%
  fit( X_train_new1, y_train, epochs = 10, batch_size = 32, validation_split
= 0.2 )
```

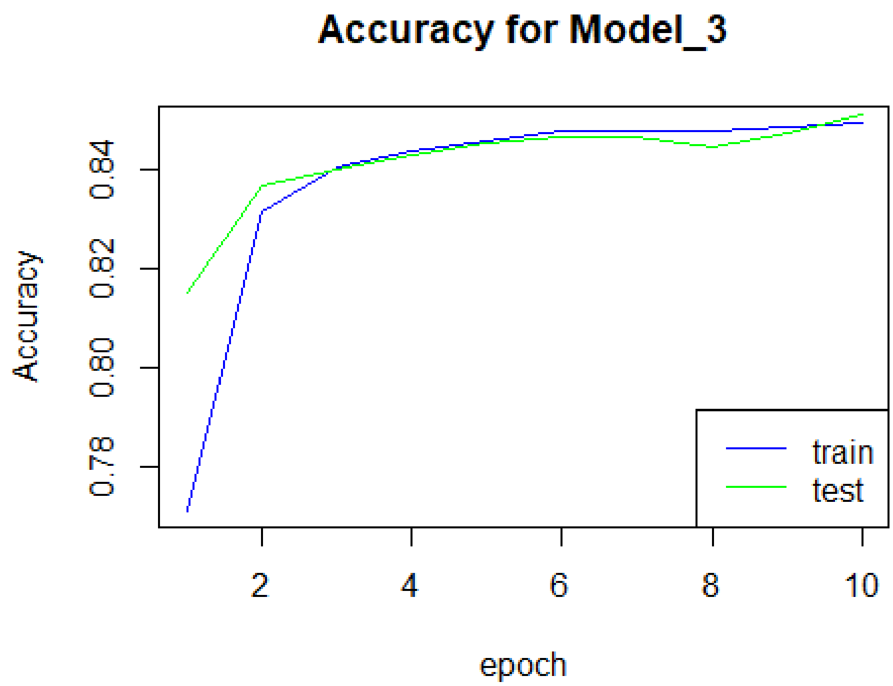
Plotting of training and validation accuracy

```
plot(history2$metrics$accuracy, main="Accuracy for Model_2", xlab = "epoch",
ylab="Accuracy", col="blue", type="l")
lines(history2$metrics$val_accuracy, col="green")
legend("bottomright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```



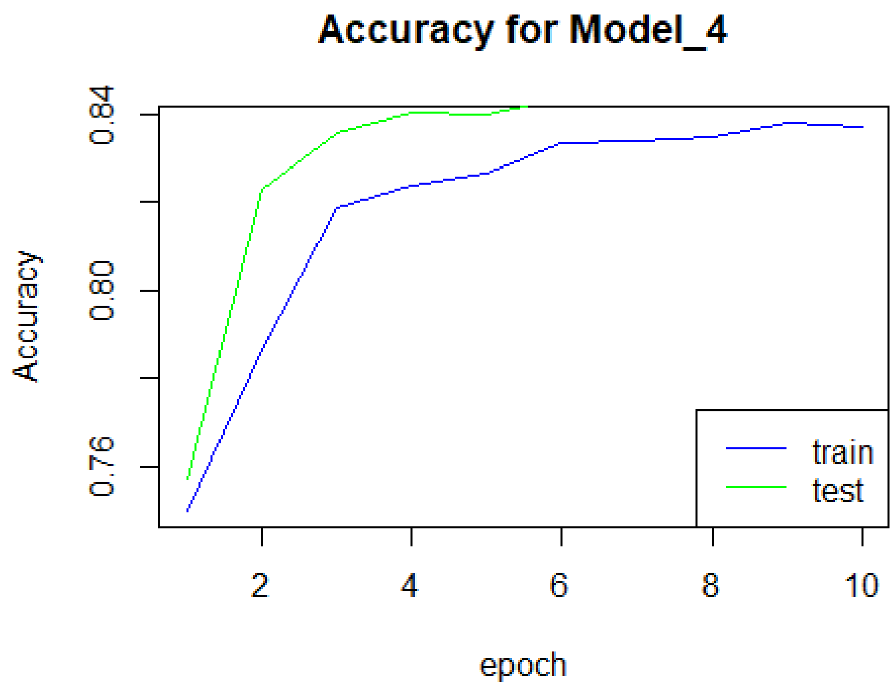
#Model 3

```
model3 %>% compile ( loss = 'binary_crossentropy', metrics = 'accuracy',
optimizer = optimizer_sgd(lr = 0.01))

history3<-model3 %>%
  fit( X_train_new1, y_train, epochs = 10, batch_size = 32, validation_split
= 0.2)

plot(history3$metrics$acc, main="Accuracy for Model_3", xlab = "epoch",
ylab="Accuracy", col="blue", type="l")
lines(history3$metrics$val_acc, col="green")
legend("bottomright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```



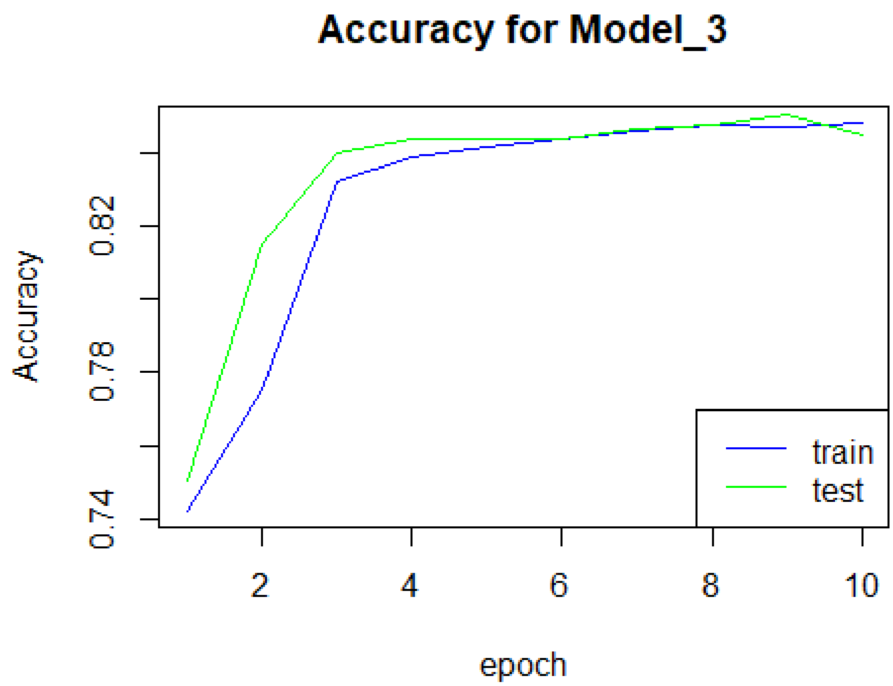
#Model 4

```
model4 %>% compile ( loss = 'binary_crossentropy', metrics = 'accuracy',
optimizer = optimizer_sgd(lr = 0.01))

history4<-model4 %>%
  fit( X_train_new1, y_train, epochs = 10, batch_size = 32, validation_split
= 0.2 )

plot(history4$metrics$acc, main="Accuracy for Model_4", xlab = "epoch",
ylab="Accuracy", col="blue", type="l")
lines(history4$metrics$val_acc, col="green")
legend("bottomright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```



#Model 5

```
model5 %>% compile ( loss = 'binary_crossentropy', metrics = 'accuracy',
optimizer = optimizer_sgd(lr = 0.01))

history5<-model5 %>%
  fit( X_train_new1, y_train, epochs = 10, batch_size = 32, validation_split
= 0.2 )

plot(history5$metrics$acc, main="Accuracy for Model_3", xlab = "epoch",
ylab="Accuracy", col="blue", type="l")
lines(history5$metrics$val_acc, col="green")
legend("bottomright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```



Highest Accuracy for model 1 (without droop_out)=84.82%

Highest Accuracy for model 2(with drop_out) =83.72%

Highest Accuracy for model 3(without drop_out) =85.38%

Highest Accuracy for model 4 =82.4%(.5) and 83.95% using drop_out(.2) just after the input layer.

Highest Accuracy for model 5 =84.4%(with drop_out) 84.87%(without dropout)

#Activity—-5(e) From the plot above,we can see that dropout layer doesn't improve the accuracyfor both cases.Dropout is used to prevent overfitting.For the model 3, may be the capacity is already low so thats why by using drop out is hurting the performance of the network.Moreover,using lower rate(less than .25) improves the performance of the model. This model was trained using .5 and the accuracy was lower.A large network with more training and the use of a weight constraint might improve the accuracy while using dropout.

Increasing the number of layers improves the accuracy.Single layer Neural Networks can only learn solutions to problems that are linearly separable. So having more layers can generlise the data.We can see that from above plot also that increasing the layer is increasing the accuracy.

Model3 is the best architecture.we used four layers and got the highest performance.

#Activity—–5(f)Part#1

From above 5 Architectures, we will test the model using model3 which is a four layer Model.

#Preparing the test file

```
test <- read.csv("adult_test.csv", header = FALSE,encoding = "latin1")

names(test) <- c("Age", "Workclass", "fnlwgt", "Education", "EducationNum",
"MaritalStatus", "occupation", "Relationship","Race", "Sex", "Capital_gain",
"Capital_loss", "hours_per_week", "Native_country", "Income_level")

#Names <- c("Age", "Workclass", "fnlwgt", "Education", "EducationNum",
"MaritalStatus", "occupation", "Relationship","Race", "Sex", "Capital_gain",
"Capital_loss", "hours_per_week", "Native_country", "Income_level")
#NROW(train)

testFileName = "adult.test"
if (!file.exists (testFileName)) download.file (url =
"http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test",
destfile = testFileName)
NROW(test)

## [1] 16282

ncol(test)

## [1] 15

test = read.csv(testFileName, header = FALSE, sep = ",", strip.white = TRUE,
col.names =Names, na.strings = "?", stringsAsFactors = TRUE)
table (complete.cases (test))

##
## FALSE  TRUE
##  1222 15060

nrow(test)

## [1] 16282

#head(test)
```

#removing the first row as there were 15 missing elements and was showing error

```
#test <- test[-c(1),]
test<-test[-c(1),]

nrow(test)

## [1] 16281

#test = test [!is.na (test$Workclass) & !is.na(test$occupation), ]
#test = test [!is.na (test$Native_country) & !is.na(test$Income_level), ]
```

```
#test$fnlwgt = NULL
#test<- select(test,-c(fnlwgt))
nrow(test)

## [1] 16281

ncol(test)

## [1] 15

test <- test [!is.na (test$Workclass) & !is.na(test$occupation), ]
test<- test [!is.na (test$Native_country), ]
test$fnlwgt <- NULL
nrow(test)

## [1] 15060

library(plyr)

## ---------------------------------------------------------------------------
----

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first,
then dplyr:
## library(plyr); library(dplyr)

## ---------------------------------------------------------------------------
----

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following object is masked from 'package:purrr':
##
##     compact

test$Income_level <- revalue(test$Income_level, c("<=50K."= 1))
test$Income_level <- revalue(test$Income_level, c(">50K."= 0))
test$Workclass <- revalue(test$Workclass, c("Federal-gov" = "State-gov"))
test$Education <- revalue(test$Education, c("10th" = "11th"))
test$MaritalStatus <- revalue(test$MaritalStatus, c("Divorced" =
"Separated"))
test$occupation <- revalue(test$occupation, c("Adm-clerical" = "Armed-
Forces"))
test$Relationship <- revalue(test$Relationship, c("Husband" = "Wife"))
test$Race <- revalue(test$Race, c("Amer-Indian-Eskimo" = "Black"))
test$Sex <- revalue(test$Sex, c("Male" = "Female"))
```

```r
ncol(test)

## [1] 14
```

#preparing X_test and y_test

```r
X_test<- select(test,-c("Income_level"))

keeps <- c("Income_level")
y_test= test[keeps]
head(y_test)

##   Income_level
## 2            1
## 3            1
## 4            0
## 5            0
## 7            1
## 9            0

typeof(y_test)

## [1] "list"

#nrow(X_test)
#nrow(y_test)
#library(plyr)
#y_test$Income_level <- revalue(y_test$Income_level, c("<=50K."=1))
#y_test$Income_level<- revalue(y_test$Income_level, c(">50K."=0))
#print(y_test)

head(y_test)

##   Income_level
## 2            1
## 3            1
## 4            0
## 5            0
## 7            1
## 9            0

unique(X_test$Relationship, incomparables = FALSE)

## [1] Own-child      Wife           Not-in-family  Unmarried      Other-
relative
## Levels:  Wife Not-in-family Other-relative Own-child Unmarried

#print(X_test[1,])

unique(X_train$Workclass, incomparables = FALSE)
```

```
## [1] State-gov        Self-emp-not-inc Private          Federal-gov
## [5] Local-gov        Self-emp-inc     Without-pay
## 8 Levels: Federal-gov Local-gov Never-worked Private ... Without-pay

X_test$Age <- as.numeric(X_test$Age)

X_test[]<- lapply(X_test, function(x) if(is.numeric(x)){
                     scale(x, center=TRUE, scale=TRUE)
                     } else x)
```

#Dummy variable

```
dmy_test <- dummyVars(~., data=X_test, fullRank=TRUE)
X_test <- data.frame(predict(dmy_test, newdata=X_test))
X_test<-as.matrix(X_test)

#X_test_new<-as.matrix(y_test)
y_test<-as.matrix(y_test)

nrow(X_test)

## [1] 15060

nrow(y_test)

## [1] 15060

ncol(X_test)

## [1] 96

y_test<-as.numeric(y_test)

typeof(y_test)

## [1] "double"

model3 %>% evaluate(X_test, y_test, verbose=0)

## $loss
## [1] 0.4365585
##
## $accuracy
## [1] 0.8061089

model5 %>% evaluate(X_test, y_test, verbose=0)

## $loss
## [1] 0.5672241
##
## $accuracy
## [1] 0.784263
```

#Activity—5(f) part_2

From HW2, for Decision Tree train_accuracy=86.9% and test_accuracy was 87.5%. Where as for Neural Network(model3), train_accuracy= 85.38% and test_accuracy=79.58%.

The performance of Decisn trees are better than Neural Network. Decisn Trees outperforms Neural Network when the data are semi-structured/unstructured. And neural-network are outperformed by tree-based algorithms when structured data is being considered.Here the dataset is structued. That might be the cause of performing better than Neural Network.