Lab5

Shuyang Deng

10/27/2018

# Lab Section

In this lab, we will go over neural networks.

# Neural Networks

## Step 1. Create model architecture

## Step 2. Initialize weights

## Step 3. Forward Propagation

## Step 4. Calculate error

## Step 5. Back Propagation / Gradient Descent

## Repeat Steps 3-5

## Step 1.

We will use a Neural network with two input nodes, two hidden nodes, and one output node. We will train this network on the XOR function.

Define activation function

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

XOR

X <- data.frame(A = c(0,0,1,1), B = c(0,1,1,0))  
Y <- data.frame(Y = c(0,1,0,1))  
b <- data.frame(b = c(1,1))  
X

## A B  
## 1 0 0  
## 2 0 1  
## 3 1 1  
## 4 1 0

Y

## Y  
## 1 0  
## 2 1  
## 3 0  
## 4 1

b

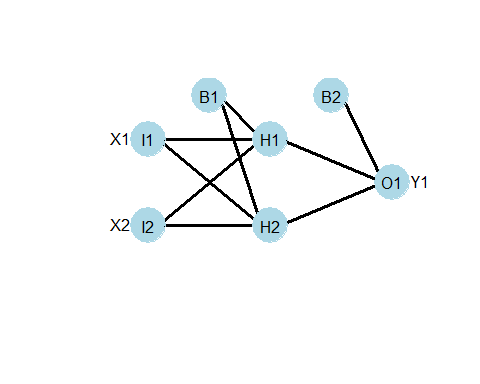
## b  
## 1 1  
## 2 1

Neural Network Architecture

wts.in <- c(1,1,1,1,1,1,1,1,1)  
struct <- c(2,2,1) #two inputs, two hidden, one output   
plot.nnet(wts.in,struct=struct)

## Loading required package: scales

## Loading required package: reshape



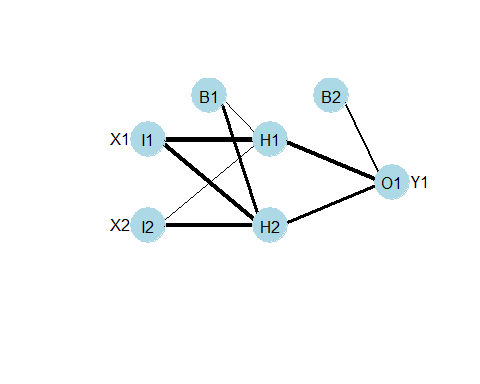
weights <- plot.nnet(wts.in,struct=struct, wts.only=T)  
weights

## $`hidden 1 1`  
## [1] 1 1 1  
##   
## $`hidden 1 2`  
## [1] 1 1 1  
##   
## $`out 1`  
## [1] 1 1 1

## Step 2.

Randomly initialize weights

set.seed(342)  
wts.in <- round(runif(9, 0, 1), digits = 2)  
plot.nnet(wts.in,struct=struct)



weights <- plot.nnet(wts.in,struct=struct, wts.only=T)  
weights

## $`hidden 1 1`  
## [1] 0.22 0.99 0.18  
##   
## $`hidden 1 2`  
## [1] 0.63 0.90 0.89  
##   
## $`out 1`  
## [1] 0.43 0.98 0.70

Input layer

I <- cbind(b,X)  
I

## b A B  
## 1 1 0 0  
## 2 1 0 1  
## 3 1 1 1  
## 4 1 1 0

## Step 3. Forward Pass: Hidden Layer

#multiply the weights by the inputs for each Hidden node  
H1\_a <- weights$`hidden 1 1` \* I[1,]  
#sum these values together  
H1\_b <- sum(H1\_a)  
# apply activation function   
H1\_c <- sigmoid(H1\_b)  
  
#multiply weights  
H2\_a <- weights$`hidden 1 2` \* I[1,]  
#sum values  
H2\_b <- sum(H2\_a)  
#apply activation function  
H2\_c <- sigmoid(H2\_b)

Hidden layer values

H <- data.frame(b = 1, H1 = H1\_c, H2 = H2\_c)  
H

## b H1 H2  
## 1 1 0.5547792 0.6524895

## Step 3. Forward Pass: Output Layer

#mulitply the weights by the values of the nodes of the hidden layer  
O1\_a <- weights$`out 1` \* H  
#sum these values together  
O1\_b <- sum(O1\_a)  
#apply activation function  
O1\_c <- sigmoid(O1\_b)  
print(O1\_c)

## [1] 0.8069677

## Step 4. Calculate error

#only one output so no need to sum  
MSE <- 0.5\*(O1\_c - Y[1,])^2  
MSE

## [1] 0.3255985

## Step 5. Backpropagation: Output layer to Hidden layer

# we want to update these weights   
weights$`out 1`

## [1] 0.43 0.98 0.70

# to improve our model  
  
# Start with the weight corresponding to the bias. Lets call it w\_Ob for weight between output and bias  
wOb <- weights$`out 1`[1]  
wOH1 <- weights$`out 1`[2]  
wOH2 <- weights$`out 1`[3]

How does a change in each weight affect the MSE? We figure it out by calculating the partial derivative of the MSE with repect to each weight. . Then adjust the weights accordingly.

First let's remember what operations we did on wOb to get to MSE: (1) we multiplied the weights by the values of the nodes of the hidden layer and summed these values together, (2) we applied the activation function, (3) we calculated the error.

We can solve by applying the chain rule. The answer comes out to . Let's break it down into pieces:

1. We multiplied the weights by the values of the nodes of the hidden layer and added these values together

Calculate the partial derivative of with respect to .

# b is 1  
H[1,1]

## [1] 1

Therefore:

1. We applied the activation function

Calculate the partial derivative of with respect to .

O1\_c\*(1-O1\_c)

## [1] 0.1557708

Therefore:

1. Calculate error

Calculate the derivative of MSE with respect to the O1\_c

O1\_c - Y[1,]

## [1] 0.8069677

Therefore:

Bring it all together

(O1\_c - Y[1,]) + O1\_c\*(1-O1\_c) + b[1,1]

## [1] 1.962739

Therefore:

## Gradient Descent

Subtract from wOb. can be mulitplied by some learning rate . Lets use

((O1\_c - Y[1,]) + O1\_c\*(1-O1\_c) + H[1,1])\*0.3

## [1] 0.5888216

## Repeat backprop and gradient descent wOH1 and wOH2.

## Using the caret package to create a 1 layer neural network on the XOR function

Get log loss from model

fit neural net to XOR

Test on one data point

test <-data.frame( A=0, B=0)  
y\_test <- Y[1]  
prediction\_nn <- predict(nnetFit, newdata = test)  
prediction\_nn

## [1] X1  
## Levels: X0 X1

confusionMatrix(prediction\_nn, reference = y\_test)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X0 X1  
## X0 0 0  
## X1 1 0  
##   
## Accuracy : 0   
## 95% CI : (0, 0.975)  
## No Information Rate : 1   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0   
## Mcnemar's Test P-Value : 1   
##   
## Sensitivity : 0   
## Specificity : NA   
## Pos Pred Value : NA   
## Neg Pred Value : NA   
## Prevalence : 1   
## Detection Rate : 0   
## Detection Prevalence : 0   
## Balanced Accuracy : NA   
##   
## 'Positive' Class : X0   
##

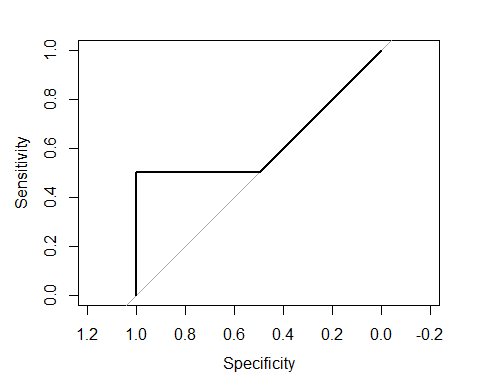
help(trainControl)

# Homework

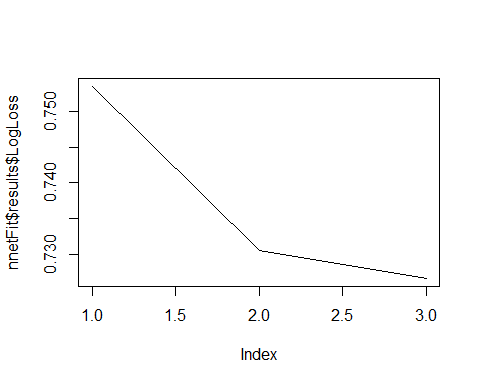
# <https://cran.r-project.org/web/packages/nnet/nnet.pdf>

1. Train a neural network on the XOR function. Try increasing the data set, playing with the number of nodes, and other hyperparameters. Plot ROC and loss.

prediction\_nn <- predict(nnetFit, newdata = X,type="prob")  
rocobj <- roc(Y~as.numeric(prediction\_nn[[1]]))  
  
#  
plot(rocobj)



#loss  
plot(nnetFit$results$LogLoss,type='l')



1. Use the iris dataset and train a neural net to classify the species. Be sure to split into training and test set, use cross validation, and plot the ROC curves and loss. Draw the structure of your Neural network, including the position of the activating functions.

library(RSNNS)

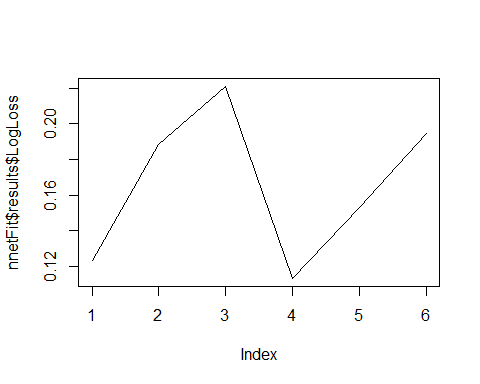
## Warning: package 'RSNNS' was built under R version 3.4.4

## Loading required package: Rcpp

##   
## Attaching package: 'RSNNS'

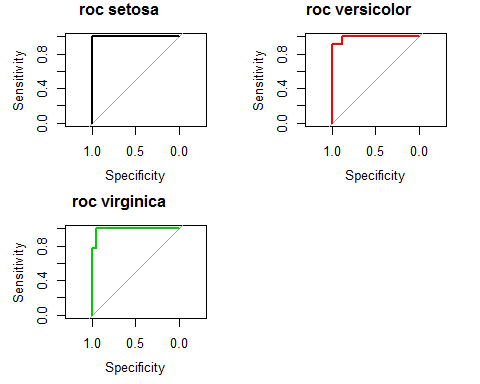
## The following objects are masked from 'package:caret':  
##   
## confusionMatrix, train

plot(nnetFit$results$LogLoss,type='l')

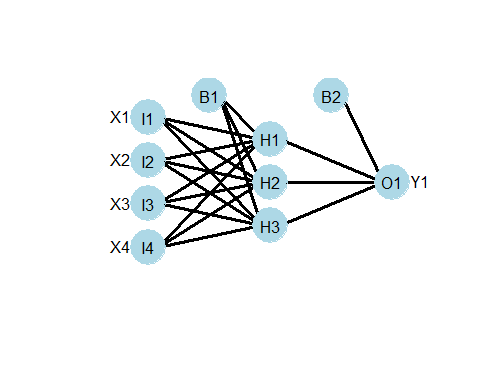


par(mar=c(0.1,0.1,1,0.1),mfrow=c(2,2))  
par(mar=c(0.1,0.1,1,0.1),mfrow=c(2,2))  
irisTargets<- decodeClassLabels(Y[test])  
prediction\_nn <- predict(nnetFit, newdata = X[test,],type="prob")  
for(i in 1:3)  
{  
  
   
 k=roc(irisTargets[,i],as.numeric(prediction\_nn[[i]]),plot=T,col=i);  
 title(main=paste("roc",colnames(irisTargets)[i],collapse =" "))  
}  
  
  
  
#loss  
  
  
prediction\_nn <- predict(nnetFit, newdata = X[test,])  
  
table(prediction\_nn, Y[test])

##   
## prediction\_nn setosa versicolor virginica  
## setosa 9 0 0  
## versicolor 0 11 0  
## virginica 0 1 9



wts.in <- rep(1,19)  
struct <- c(4,3,1) #four rinputs, tree hidden, one output   
plot.nnet(wts.in,struct=struct)



1. Finish the lab work: Update O1\_B2, O1\_I1, and O1\_I2. Bonus: Update H1\_I1, H1\_I2, H2\_b1, H2\_I1, H2\_I2, H2\_b1

nnetFit1$finalModel$wts

## [1] -1.657174e-06 -8.426081e-07 -1.533149e-06 1.257305e-06 4.746762e-07  
## [6] 1.003918e-06 1.808084e-07 -1.899722e-08 3.371332e-08

<http://www.emergentmind.com/neural-network>

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>