University of Cape Town

STA4010W

Analytics assignment 2

Author:
Yovna Junglee
Chongo Nkalamo

Student Number:
JNGYOV001
NKLCHO001

April 8, 2019

Contents

List of Figures

Using a Neural Network to predict breast cancer

Number of parameters in a neural network

A neural network constitutes of an input layer, output layer and hidden layer(s) of which each layer has nodes. The parameters of the network are made up of the weights and biases. A count of the number of parameters in a neural network with **p** input nodes, **q** output nodes and **d** hidden layers is given by :

$$Total = \sum_{l}^{N} d_{l-1} \times d_l + \sum_{l=1}^{N} d_l$$

where
d_l = the number of nodes in layer l.
$d_0 = p$, $d_N = q$ and $\sum_{l=1}^{N} d_l$ = number of bias vectors from hidden layer 1 to output layer.

An R-function to calculate the above relationship is given by:

```
get_params <- function(p,q,d){

  nodes <- c(p,d,q) # Order all nodes

  total <- 0 # Initialise total

  n <- length(nodes) # Length of nodes
  for (i in 1:(n-1)){ # update total using d_l * d_(l+1)
    total = total + (nodes[i]*nodes[i+1])
  }


  # Add bias vectors
  b <- sum(d)+q

  total <- b+total

  return(total) # Return total
}
```

Matrix form of updating equation

The updating equation can be written in matrix form to evaluate **N** observations at a time as

$$\mathbf{a_l} = \sigma_l(\mathbf{W^l})^T\mathbf{a^{l-1}} + \mathbf{b^l}$$

where

$$\mathbf{W_l} = (W_{ij}^l : i = 1...d_{l-1}, j = 1...d_l)_{d_{l-1}Xd_l}$$

$$\mathbf{a}_{l-1} = (a_{ij} : i = 1...d_{l-1}, j = 1...N)_{d_{l-1}XN}$$

$$\mathbf{a}_0 = \mathbf{X^T}$$

$$\mathbf{b_l} = (b_{ij} : i = 1...d_l, j = 1...N)_{d_lXN}$$

(bias vectors of layer l repeated in N columns)

R-function to evaluate a feed-forward Neural Network

The R-function can be found below:

```
neural_net = function(X,Y,theta,d,lambda)
{

  d1 <- d[1] # Get number of nodes in 1st hidden layer
  d2 <- d[2] # Get number of nodes in 2nd hidden layer
  # X- Feature matrix (input/variables)
  #Y - Response matrix
  # theta - Concatenate all of the weights and biases into par vector
  # lambda - Regularization hyperparameter


  # input and output dimensions
  p = dim(X)[2]
  q = dim(Y)[2]


  # populate weight matrices and bias vector
  index  = 1:(p*d1)
  W1     = matrix(theta[index],p,d1) # Input to hidden layer 1 weights
  index  = max(index)+1:(d1*d2)
  W2     = matrix(theta[index], d1 ,d2) # Hidden layer 1 to Hidden layer 2 weights
  index  = max(index)+1:(d2*q)
  W3    =  matrix(theta[index],d2,q) # hidden layer 2 to output weights
  index  = max(index)+1:d1
  b1     = matrix(theta[index],d1,1) # bias vector for the hidden layer 1 nodes
  index  = max(index)+1:d2
  b2     = matrix(theta[index],d2,1) # bias vector for the hidden layer 2 nodes
  index  = max(index)+1:q
  b3     = matrix(theta[index],q,1) # bias vector for the output layer nodes


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

  ## Matrix form

  # X <- Nxp
  # W_L <- D(L)*D(L-1)
  # a_l <- d(l) * N
  # b_l  : d(l) x N

  n <- dim(X)[1]
  a0 <- t(X)
  a1 <- sigma(t(W1)%*%a0 + matrix(rep(b1, n), d1 ,n))
  a2 <- sigma(t(W2)%*% a1 + matrix(rep(b2,n),d2,n))
  a3 <- sigma(t(W3)%*%a2 + matrix(rep(b3,n),q,n))
  out <- a3

  err <- -1*sum((cost(Y,t(out)))) # Error cost function
```

```
  reg <- lambda*(sum(W1^2) + sum(W2^2)+ sum(W3^2)) # Regularisation

  err.pen <- err + reg # Penalized cost function

  out <- list(Predicted=out, Error= err.pen, a1=a1, a2=a2 )
  return (out)
}
```

Pseudo code

X: Input matrix (N x p) dimension
Y: Output matrix (q x N) dimension
theta : concacenated weight matrices
d = c(d1,d2) : d1 and d2 : Number of nodes in hidden layer 1 and 2
lambda = regularization parameter

1. For each layer l, set weight matrix $\mathbf{W}^l\}$ and bias vectors.
2. For each layer l until l=N, apply the updating equation:

$$\mathbf{a_l} = \sigma_l(\mathbf{W^l})^T\mathbf{a^{l-1}} + \mathbf{b^l}$$

   where σ_l is the activation function in layer l.
3. Use the relevant *cost function* C to evaluate the errors.
4. Apply regularization parameter.

5. Calculate the penalized cost function.

Regularization and Cost functions

The task at hand is to use a neural network to predict whether given certain attributes, breast cancer is present. The response scale is thus binary and hence the task at hand is a classification problem. With this in mind, selection of the appropriate activation function(s) and cost function is vital in order to allow the network learn the relationship between the response and predictor variables.

The choice of activation function(s) for both the output and hidden layer is the sigmoid activation function. In particular, the output activation has to be one that ensures the output layer behaves like a distribution. So that the resulting outputs can then be treated as probabilities and the predictions may correspond to the node with the highest predicted probability. The sigmoid function does just this.

Selecting an appropriate cost function again depends on what problem or task is at hand. That is, a regression or classification problem. Reasons as to why we want to have cost functions is that we want the network to be able to replicate the data we have well. For classification problems, it is suggested in the literature that the use of the **cross-entropy error** cost function as a cost function is appropriate. The **cross-entropy error** cost function is given as:

$$C_{CE} = -\sum_{i}^{N}(y_i log(\widehat{y}_i) + (1-y_i)log(1-\widehat{y}_i))$$

Cost functions essentially measure how far away a prediction a model gives out to the actual observed value. If this distance is still significantly large, the network will undergo the learning process again until this distance is minimized. However, we do not want a model that is too complex as this leads to over-fitting out-of-sample and from a practical aspect, might be out of budget. To this end, we apply a regularization mechanism in addition to the cost function which avoids over-fitting by controlling the complexity of the model. It does so by subjecting the cost function to a new constraint. The regularization control is given by:

$$\lambda \sum_{ij} \omega_{ij}^2$$

In this case, the regularization parameter λ controls for model complexity. A large λ value gives a model with high bias but low variance whereas low λ value gives the opposite. In this problem we considered a sequence of lambda values ranging from 0.005 to 1 by increments of 0.01.

As can be seen in Figure 1, different combinations of λ with different m values (number of nodes in the two layers) are plotted against training and validation cross-entropy errors. In particular, if we look at the validation error plot, we see that the combination of m and λ that gives the lowest cross-entropy error was the combination of m = 2 and $\lambda = 0.005$ for the given λ values we have attempted to use. To this end, we select a neural network with two hidden layers each with two nodes as an appropriate model to tackle this problem given a penalized parameter. However, this might not be the case if we consider a different sequence of lambda values thus it is not entirely clear as to what an optimal number of nodes for the hidden layers would be from different λ values.

Fit neural network to data

A neural network was fit to the breast cancer data provided using an 80% training and 20% validation split. The neural network was fit with two hidden layers with an equal number of nodes in each of the layers. We fit 5 different models considering different values, **m**, the number of nodes in each hidden layer. This was carried out for m ranging from 2 to 5 nodes and the purpose of this was to use the validation errors obtained from fitting the five models to select the most appropriate number of nodes for each hidden layer that will give the best predictions. Again, we refer to Figure 1 to view these error plots.

As indicated in the previous section, we decided to choose a neural network comprising of two hidden layers each with two nodes as this particular combination gave the lowest validation error.

Predictions

Predictions where done on a second file that contained only predictor labels. We did this using the best model obtained from the previouis section that comprised of m = 2 and $\lambda = 0.005$. A prediction accuracy of 96% on the validation set was obtained. We then proceeded to retrained this model on the full dataset as to get optimal weight and bias values. Thereafter, prediction on new data set was carried out and these predictions were uploaded to vula.

Under the Hood

A neural network with two hidden layers was fit to the data set provided with an 80% training 20% Validation split as done previously. The hidden layers consisted of six and two nodes respectively. Training of the network was done as before (first section) where we incorporated different regularization penalizing parameters λ.

Figure 2 shows the penalized cross-entropy error for the training and validation set. It can be seen that the error is at its minimum for $\lambda \approx 0.0025$ in the validation error plot, after which it increases. Thus the model we consider is a neural network (with regularization parameter $\lambda = 0.0025$.) with two hidden layers with six and two nodes in the first and second hidden layers respectively.

Activity in hidden and output layers.

If we consider the first node in the first hidden layer, the weight from the first example(input) feeding into this node is $\omega_{11} = 4.14$ and that from the second example, X2 is $\omega_{21} = -3.40$. This implies that as the value

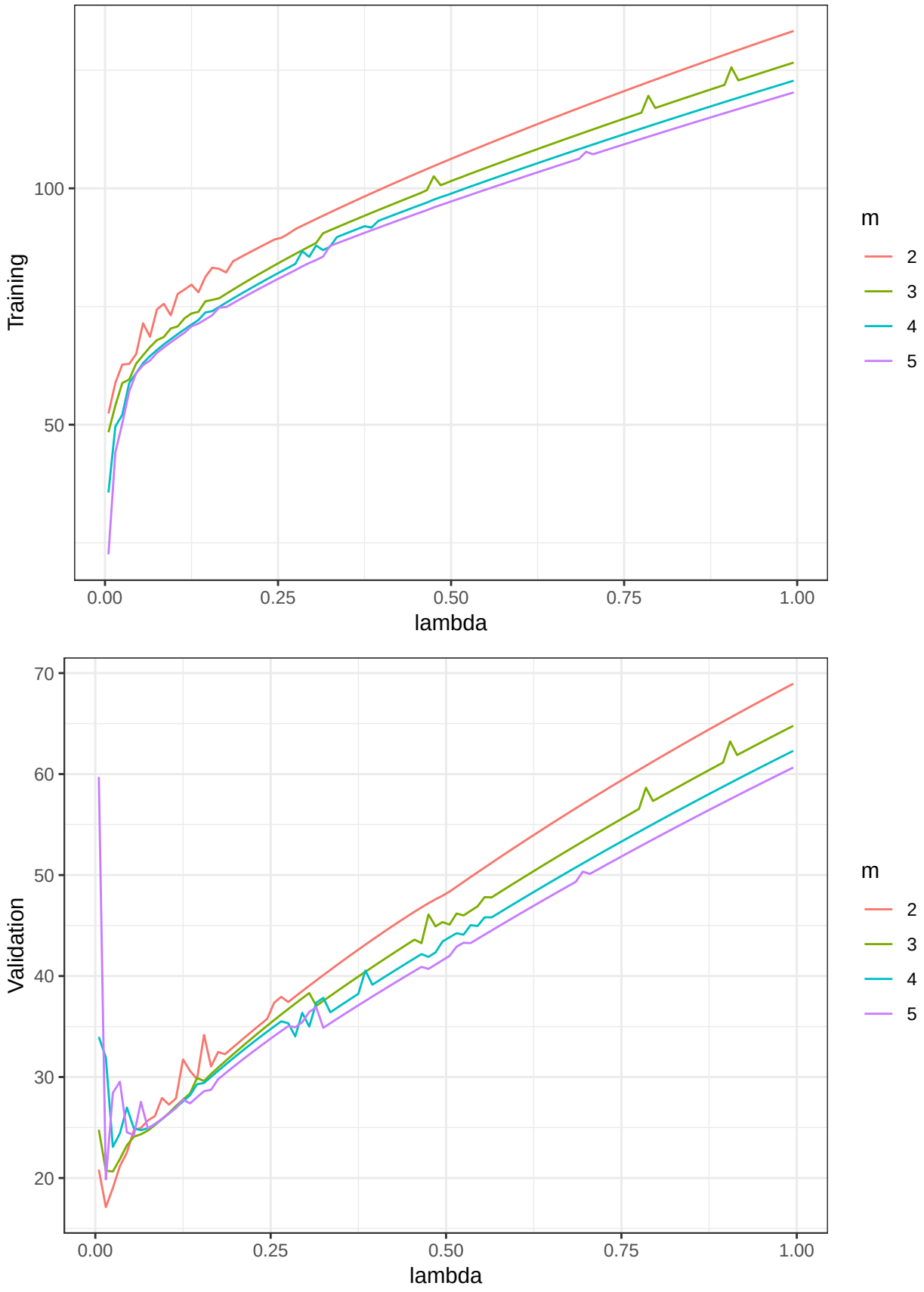

Figure 1: Training Error plot(top) and Validation error plot(bottom)

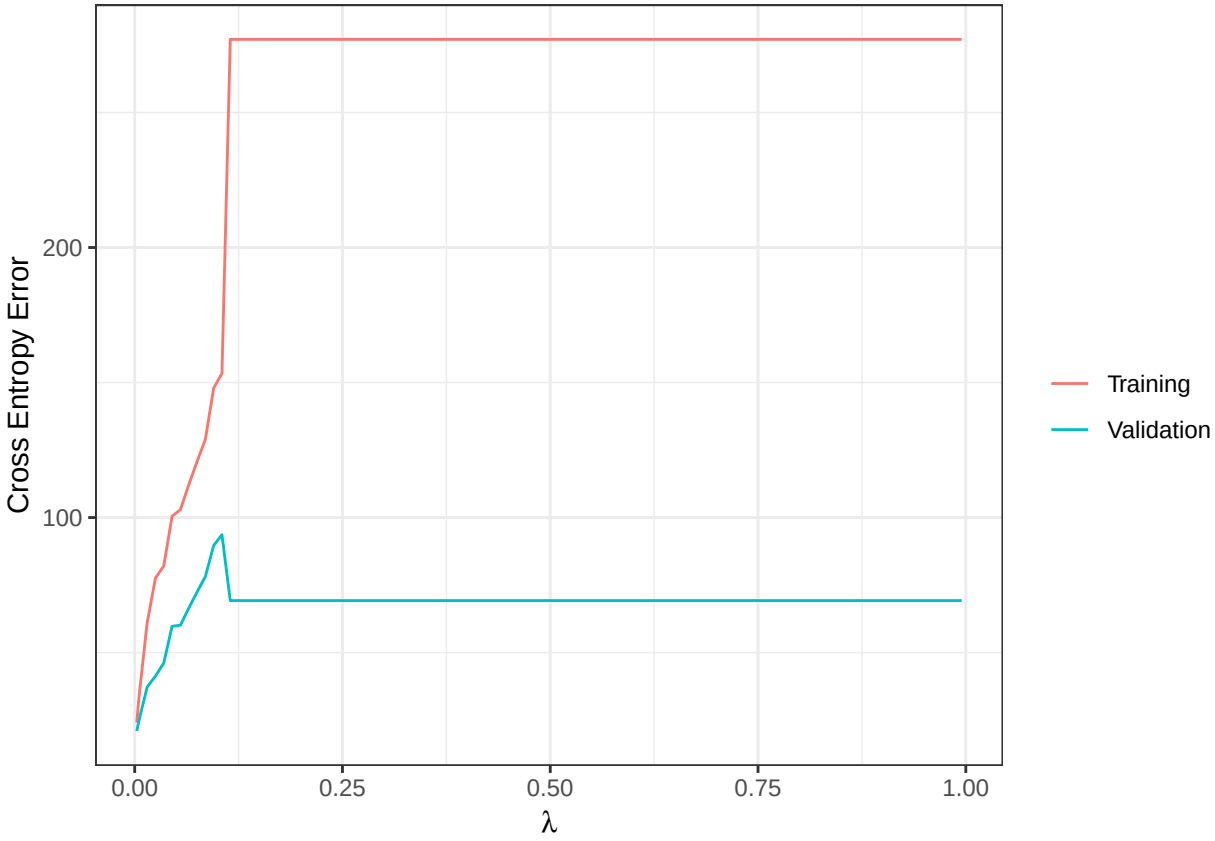

Figure 2: Cross entropy error

X1 increases, the intensity will increase as the sign of the weight is positive whereas as X2 increases, the intensity decreases. Since the $|\omega_{11}| > |\omega_{21}|$, more weight is assigned to X1 values than X2 values, which explains the first diagram obtained in Figure 3. The weight matrices can be found in the appendix.

As seen in Figure 3 the intensity scale increases the darker the image gets (i.e from yellow to orange) thus justifying our claim. The same can be said for the remaining nodes. Note that from the second hidden layer onwards, the intensity/output of the previous layer become the new inputs for each of the nodes in the layer. Thus, we expect to see a different intensity mapping. Given that this is the immediate layer before the output layer, we expect to start seeing the intensity we expect to see in the output layer. This is seen in Figure 4.

Just as in convolutional neural networks, this is the stage where, given an output image such as an image of a Zebra, this layer would recognize the stripes or eyes. Thus this layer is attempting to learn more about some of the features of what the output image produces.

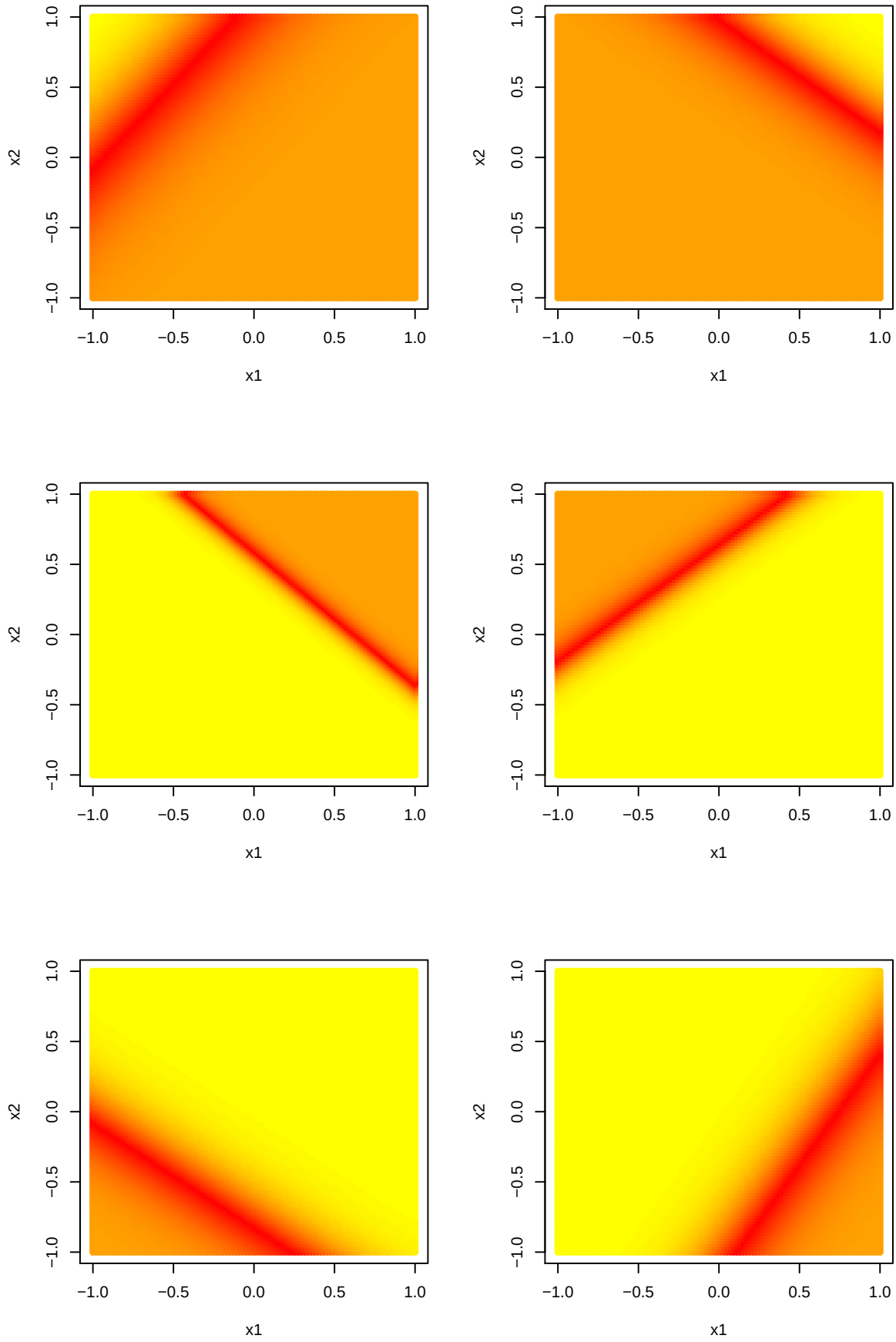

Figure 3: First hidden layer nodes

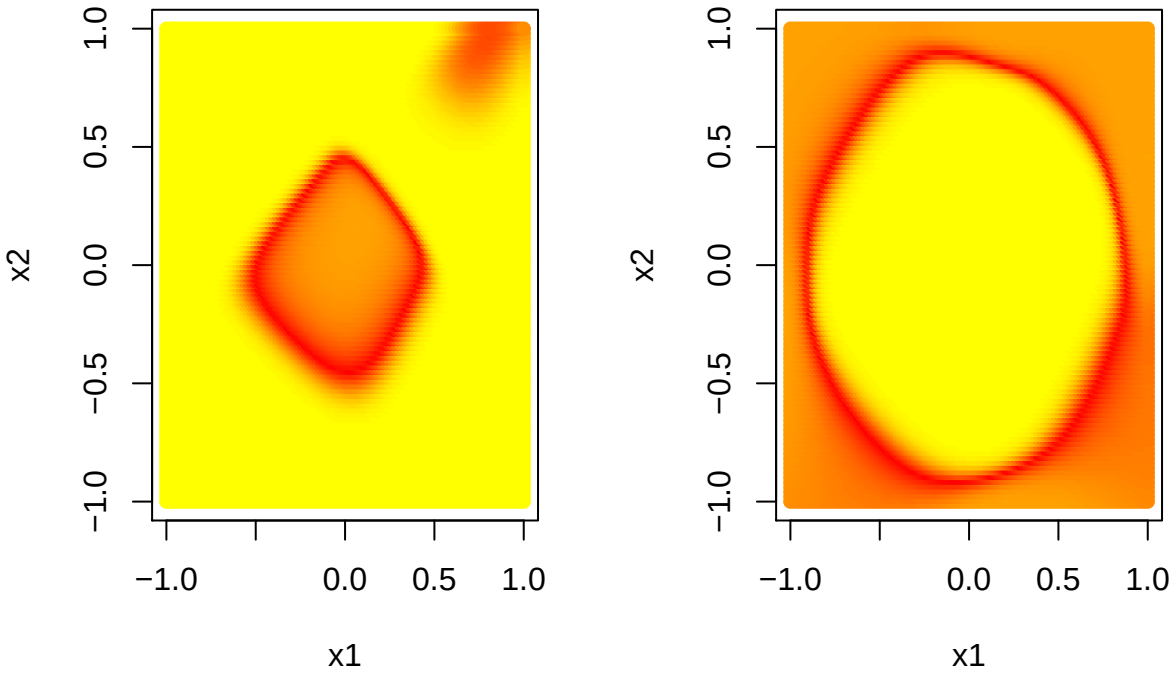

Figure 4: Second hidden layer nodes

Figure 5, in particular the left image, shows what is taking place in the output layer and node. What we see in the right image of this figure is a representation of what the training data looks like. The image on the left gives a clearer representation of what the output looks like taking into account unseen data. Hence, the more data the fed into the network, the better our neural network predicted the desired outcome.

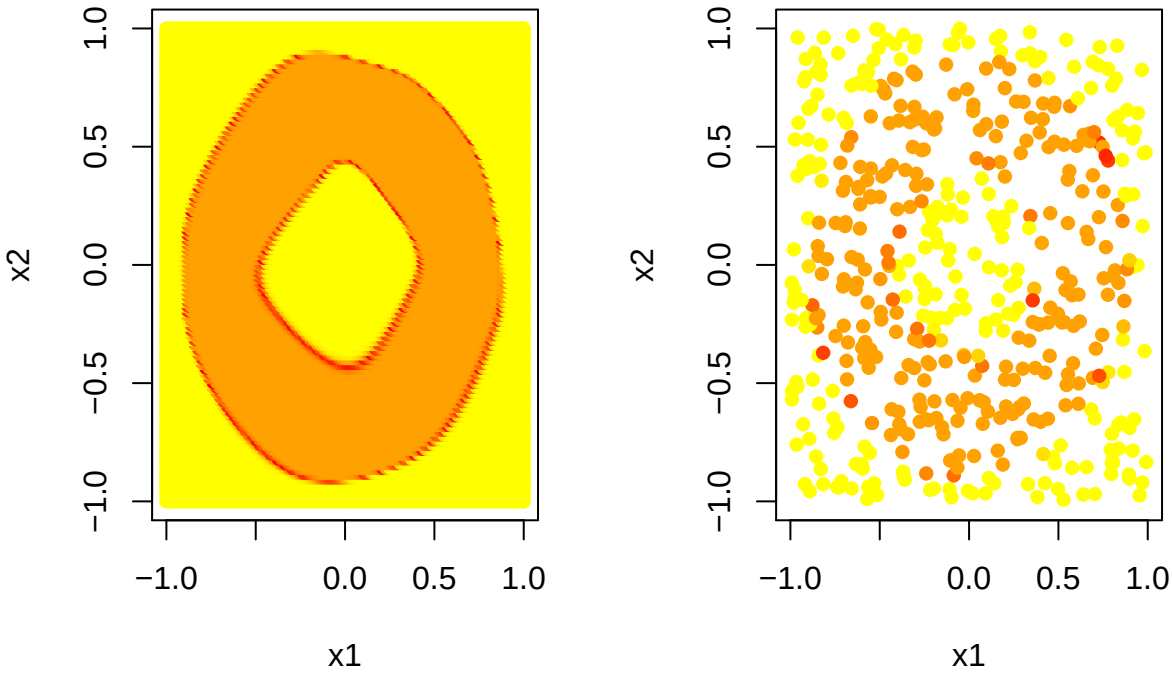

Figure 5: Output layer nodes: Grid output(left) vs Hood output(right)

1 Appendix

:

Weight matrices for question 2b.

Weight matrix from input layer to the first hidden layer of the network.

$$\begin{pmatrix} 4.135847 & -4.864914 & 16.45329 & -9.941573 & -4.384425 & 6.428131 \\ -3.401062 & -6.065791 & 17.38817 & 11.983790 & -5.916650 & -4.061842 \end{pmatrix}$$

Weight matrix from first hidden layer to the second hidden layer of the network.

$$\begin{pmatrix} 6.077391 & -17.160559 \\ -7.820891 & -20.516080 \\ -9.969226 & -0.503557 \\ -13.807941 & 2.965691 \\ -18.412988 & 16.703526 \\ -17.738437 & 16.523484 \end{pmatrix}$$

Weight matrix from second hidden layer to the output layer of the network.

$$\begin{pmatrix} 33.74045 \\ -35.16449 \end{pmatrix}$$

R Code

```
get_params <- function(p,q,d){

  nodes <- c(p,d,q) # Order all nodes

  total <- 0 # Initialise total

  n <- length(nodes) # Length of nodes
  for (i in 1:(n-1)){ # update total using d_l * d_(l+1)
    total = total + (nodes[i]*nodes[i+1])
  }


  # Add bias vectors
  b <- sum(d)+q

  total <- b+total

  return(total) # Return total
}

neural_net = function(X,Y,theta,d,lambda)
{

  d1 <- d[1] # Get number of nodes in 1st hidden layer
  d2 <- d[2] # Get number of nodes in 2nd hidden layer
  # X- Feature matrix (input/variables)
  #Y - Response matrix
  # theta - Concatenate all of the weights and biases into par vector
  # lambda - Regularization hyperparameter


  # input and output dimensions
  p = dim(X)[2]
  q = dim(Y)[2]


  # populate weight matrices and bias vector
  index  = 1:(p*d1)
  W1     = matrix(theta[index],p,d1)
  # Input to hidden layer 1 weights
  index  = max(index)+1:(d1*d2)
  W2     = matrix(theta[index], d1 ,d2)
```

```
  # Hidden layer 1 to Hidden layer 2 weights
  index  = max(index)+1:(d2*q)
  W3    =  matrix(theta[index],d2,q)
  # hidden layer 2 to output weights
  index  = max(index)+1:d1
  b1     = matrix(theta[index],d1,1)
  # bias vector for the hidden layer 1 nodes
  index  = max(index)+1:d2
  b2     = matrix(theta[index],d2,1)
  # bias vector for the hidden layer 2 nodes
  index  = max(index)+1:q
  b3     = matrix(theta[index],q,1)
  # bias vector for the output layer nodes


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

  ## Matrix form

  # X <- Nxp
  # W_L <- D(L)*D(L-1)
  # a_l <- d(l) * N
  # b_l  : d(l) x N

  n <- dim(X)[1]
  a0 <- t(X)
  a1 <- sigma(t(W1)%*%a0 + matrix(rep(b1, n), d1 ,n))
  a2 <- sigma(t(W2)%*% a1 + matrix(rep(b2,n),d2,n))
  a3 <- sigma(t(W3)%*%a2 + matrix(rep(b3,n),q,n))
  out <- a3

  err <- -1*sum((cost(Y,t(out)))) # Error cost function


  reg <- lambda*(sum(W1^2) + sum(W2^2)+ sum(W3^2)) # Regularisation

  err.pen <- err + reg # Penalized cost function

  out <- list(Predicted=out, Error= err.pen, a1=a1, a2=a2 )
  return (out)
}
set.seed(1)
# Sigmoid activation function
sigma <- function(x){
```

```
    (1/(1+exp(-x)))

}

# Cross entropy as classification problem
cost <- function(x, xhat){
  c <- (x*log(xhat))+((1-x)*(log(1-xhat)))

  return(c)
}



# Load data

dat1 <- read.table("Breast Cancer Dataset A.txt", header=T)
samp <- sample(1:nrow(dat1), .80*nrow(dat1)) #Randomize training set

X <- dat1[,1:5]
X<- as.matrix(X)
X <- scale(X)

Y<- dat1[,6]
Y <- as.matrix(Y)

train <- dat1[samp,] # Training set
val <- dat1[-samp,] # Validation set

Xtrain <- train[,1:5]
Ytrain <- train[,6]
Xtrain <- as.matrix(Xtrain)
Ytrain <- as.matrix(Ytrain)

Xtrain <- scale(Xtrain)


# Validation set

XVal <- as.matrix(val[,1:5])
XVal <- scale(XVal)
YVal <- as.matrix(val[,6])


# Parameters
```

```r
# Parameters

# m=2

p <- 5
d2 <- c(2,2)
q <- 1
npars2 <- get_params(p,q,d2)

mod.train2.err <- numeric(0)
val.train2.err <- numeric(0)


lambda <- seq(0.005,1,by=0.01)

for (i in 1:length(lambda)){


    obj2 <- function(pars){
        d= c(2,2)
        lambda <- lambda[i]
        res =  neural_net(Xtrain,Ytrain, pars,d,lambda)
        res$Error
      }

    res2 = nlm(obj2,runif(npars2,-0.5,0.5), iterlim = 1000)

    mod.train2 <- neural_net(Xtrain, Ytrain,
    res2$estimate, d=c(2,2),lambda[i])
    mod.train2.err[i] <- mod.train2$Error

    val.train2 <- neural_net(XVal, YVal, res2$estimate,
    d=c(2,2),lambda[i])
    val.train2.err[i] <- val.train2$Error

}



# m= 3

p <- 5
d3 <- c(3,3)
q <- 1
```

```
npars3 <- get_params(p,q,d3)




mod.train3.err <- numeric(0)
val.train3.err <- numeric(0)


for (i in 1:length(lambda)){
          obj3 <- function(pars){
            d= c(3,3)
            lambda <- lambda[i]
            res =  neural_net(Xtrain,Ytrain, pars,d,lambda)
            res$Error
          }

          res3 = nlm(obj3,runif(npars3, -0.5,0.5), iterlim = 1000)
          mod.train3 <- neural_net(Xtrain, Ytrain,
          res3$estimate, d=c(3,3),lambda[i])
          mod.train3.err[i] <- mod.train3$Error

          val.train3 <- neural_net(XVal, YVal,
          res3$estimate, d=c(3,3),lambda[i])
          val.train3.err[i]<- val.train3$Error

}


# m= 4


p <- 5
d4 <- c(4,4)
q <- 1
npars4 <- get_params(p,q,d4)

val.train4.err <- numeric(0)
mod.train4.err <- numeric(0)

for (i in 1:length(lambda)){
        obj4 <- function(pars){
          d= c(4,4)
          lambda <- lambda[i]
```

```
          res =  neural_net(Xtrain,Ytrain, pars,d,lambda)
          res$Error
        }

        res4 = nlm(obj4,runif(npars4, -0.5,0.5), iterlim =1000)

        mod.train4 <- neural_net(Xtrain, Ytrain,
        res4$estimate, d=c(4,4),lambda[i])
        mod.train4.err[i] <- mod.train4$Error

        val.train4 <- neural_net(XVal, YVal,
        res4$estimate, d=c(4,4),lambda[i])
        val.train4.err[i]<- val.train4$Error

}
# m= 5


p <- 5
d5 <- c(5,5)
q <- 1
npars5 <- get_params(p,q,d5)

val.train5.err <- numeric(0)
mod.train5.err <- numeric(0)

for (i in 1:length(lambda)){
      obj5 <- function(pars){
        d= c(5,5)
        lambda <- lambda[i]
        res =  neural_net(Xtrain,Ytrain, pars,d,lambda)
        res$Error
      }

      res5 = nlm(obj5,runif(npars5, -0.5,0.5), iterlim =1000)

      mod.train5 <- neural_net(Xtrain, Ytrain,
      res5$estimate, d=c(5,5),lambda[i])
      mod.train5.err[i]<- mod.train5$Error

      val.train5 <- neural_net(XVal, YVal,
      res5$estimate, d=c(5,5),lambda[i])
      val.train5.err[i]<- val.train5$Error
}
```

```
###

#cbind(res2£minimum,res3£minimum,res4£minimum,res5£minimum)
## Choose res 4
#cbind(res2£iterations,res3£iterations,res4£iterations,
res5$iterations) ## all less than 1000
# so convergence

m <-rep(c(2,3,4,5), each=length(lambda))
train.errors <- c(mod.train2.err,mod.train3.err,
mod.train4.err,mod.train5.err)
train.errors <- as.data.frame(cbind(m,lambda,Training=train.errors))
train.errors$m <- as.factor(m)
val.errors <- c(val.train2.err,val.train3.err,
val.train4.err,val.train5.err)
val.errors <- as.data.frame(cbind(m,lambda,Validation=val.errors))
val.errors$m<- as.factor(m)
p1<- ggplot(data=train.errors, aes(x=lambda, y=Training, color = m))
+ geom_line()+
  theme_bw()


p2 <- ggplot(data=val.errors, aes(x=lambda, y=Validation, color = m))
+ geom_line()+
  theme_bw()

grid.arrange(p1,p2)

data.b <- read.table("Breast Cancer Dataset B.txt", header=T)
X.test<- as.matrix(data.b)
X.test <- scale(X.test)
y.dummy <- cbind(rep(1,nrow(xlab)))
#dummy column vector to feed the neural network

YVal <- YVal[1:69,] #just to feed the neural net
YVal <- cbind(YVal)

#use function to get weights and biases from training
num <- get_params(ncol(xlab),ncol(YVal),c(2,2))
obj.pred<- function(pars){
  d= c(2,2)
  lambda <- 0.005
  #Training from the whole dataset
  res =  neural_net(X,Y, pars,d,lambda)
```

```
}

res.pred = nlm(obj.pred,runif(num,-.5,.5), iterlim =1000)
pred <- neural_net(X.test, YVal, res.pred$estimate, d=c(2,2),0.005)
pre.predicted <- t(pred$Predicted)
predicted<- ifelse(pre.predicted >= 0.5,1,0)
predicted <- as.data.frame(cbind(predicted))
#View(predicted)
colnames(predicted)=c("Predictions")
write.table(predicted, file="Cancer_Pred_NKLCHO001.csv"
,row.names=F, quote=F, sep=',')
set.seed(1)
# Load data
dat2 <- read.table("UnderTheHood.txt", header=T)
datX <- as.matrix(dat2[,1:2])
datY <- as.matrix(dat2[,3])

samp1 <- sample(1:500, 400)
datX.train <- as.matrix(datX[samp1,])
datY.train <- as.matrix(datY[samp1,])

datX.test <- as.matrix(datX[-samp1,])
datY.test <- as.matrix(datY[-samp1,])



# Set parameters

# Set parameters
p1 <- 2
d.hood <- c(6,2)
q <- 1
npars.2 <- get_params(p1,q,d.hood)

lambda1 <- c(0.0025, seq(0.005,1,by=0.01))


hood.train <- numeric(length(lambda1))
hood.test <- numeric(length(lambda1))

for (i in 1:length(lambda1))  {
  obj.fit <- function(pars){
    d= d.hood
    lambda <- lambda1[i]
```

```
    res =  neural_net(datX.train,datY.train, pars,d,lambda)
    res$Error
  }

  # Fit model
  res.fit <- nlm(obj.fit, runif(npars.2,-1,1),iterlim=1000)

  mod.fit <- neural_net(datX.train, datY.train,
  theta=res.fit$estimate, d=c(6,2),lambda1[i])
  hood.train[i] <- mod.fit$Error

  test.fit <- neural_net(datX.test, datY.test,
  theta=res.fit$estimate, d=c(6,2),lambda1[i])
  hood.test[i] <- test.fit$Error
}

err.hood <- cbind(lambda1, Training=hood.train,Validation=hood.test)
err.hood <- as.data.frame(err.hood)
err.hood <- cbind(lambda1,melt(err.hood[,2:3]))



ggplot(data=err.hood, aes(x=lambda1,y=value,color=variable))
+geom_line()+
  labs(x=expression(lambda), y= "Cross Entropy Error", color="")
  +theme_bw()
###

# Generate grid of inputs
obj.fit <- function(pars){
  d= d.hood
  lambda <- 0.0025
  res =  neural_net(datX,datY, pars,d,lambda)
  res$Error
}

      # Fit model
res.fit <- nlm(obj.fit, runif(npars.2,-1,1),iterlim=1000)
mod.fit <- neural_net(datX, datY, theta=res.fit$estimate,
d=c(6,2),0.0025)
#hood.train[i] <- mod.fit£Error
# Generate grid of inputs
x1<- seq(-1,1,length=100)
x2 <- seq(-1,1,length=100)
```

```r
Xin <- as.matrix(expand.grid(x1,x2)) # Create matrix of inputs
y <- as.matrix(seq(-1,1,length=100*100))
# Arbitrary y val so neural net works


# Colour function
colfunc = function(x, n=100){
  xx = seq(min(x), max(x), length=n)
  ii = findInterval(x,xx)
  colr <- colorRampPalette((c('yellow','red','orange')))
  cols = colr(n)
  return(cols[ii])
}

# Fit neural net to input grid
mod.new <- neural_net(Xin, y, theta=res.fit$estimate,
d=c(6,2),0.0025)

a1 <- mod.new$a1 # Hidden layer 1 output
a2 <- mod.new$a2 # Hidden layer 2 output
out <- mod.new$Predicted # Final layer output

# Layer 1

layout(matrix(c(1,2,3,4,5,6), 3, 2, byrow = TRUE))
plot(NULL, xlim=c(-1,1), ylim=c(-1,1), ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a1[1,]), pch=16)
plot(NULL, xlim=c(-1,1), ylim=c(-1,1), ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a1[2,]), pch=16)
plot(NULL, xlim=c(-1,1), ylim=c(-1,1), ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a1[3,]), pch=16)
plot(NULL, xlim=c(-1,1), ylim=c(-1,1), ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a1[4,]), pch=16)
plot(NULL, xlim=c(-1,1), ylim=c(-1,1), ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a1[5,]), pch=16)
plot(NULL, xlim=c(-1,1), ylim=c(-1,1), ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a1[6,]), pch=16)

# Layer 2

layout(matrix(c(1,2), 1, 2, byrow = TRUE))
plot(NULL, xlim=c(-1,1), ylim=c(-1,1),ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a2[1,]), pch=16)
```

```
plot(NULL, xlim=c(-1,1), ylim=c(-1,1), ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(a2[2,]), pch=16)

# Output layer
layout(matrix(c(1,2), 1, 2, byrow = TRUE))
plot(NULL, xlim=c(-1,1), ylim=c(-1,1),ylab="x2", xlab="x1")
points(Xin[,1],Xin[,2], col=colfunc(out), pch=16)
plot(NULL, xlim=c(-1,1), ylim=c(-1,1),ylab="x2", xlab="x1")
points(datX[,1],datX[,2], col=colfunc(mod.fit$Predicted), pch=16)

W1 <- matrix(res.fit$estimate[1:12],2,6); W1 #layer 1 weights
W2 <- matrix(res.fit$estimate[13:24],6,2);W2 #layer 2 weights
W3 <- matrix(res.fit$estimate[25:26],2,1);W3 # output layer weights
```