Neural Networks

Anna Yeaton

Fall 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

$MeanSquaredError = \frac{1}{m}\sum_{i=1}^{m}(output - target)^2$

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
```

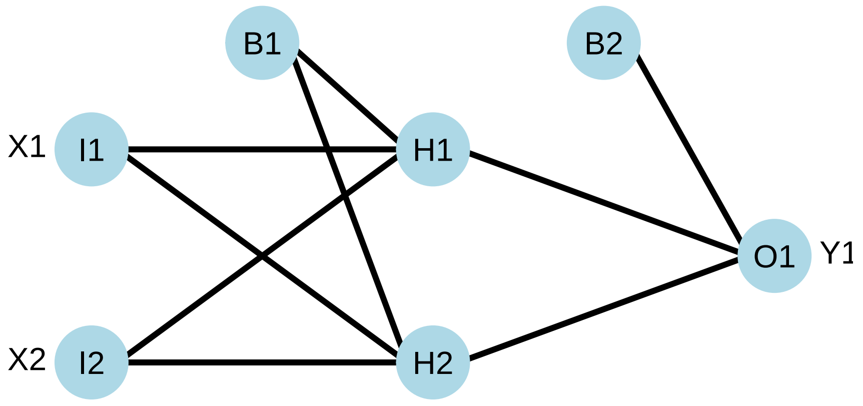
```
##   b
## 1 1
## 2 1
```

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
```



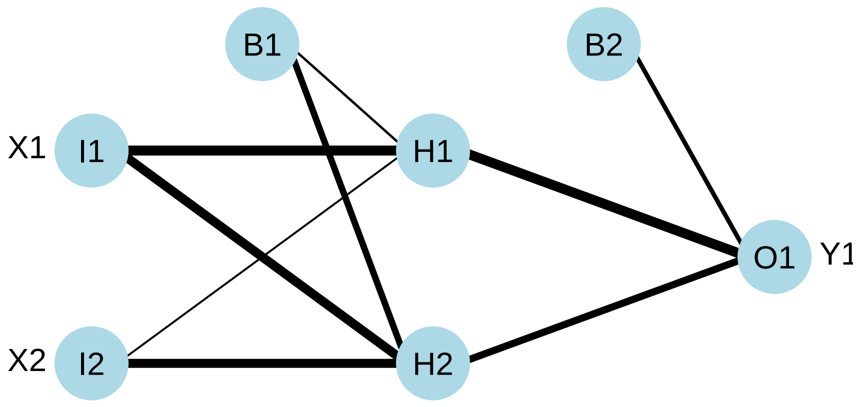
```
weights <- plot.nnet(wts.in,struct=struct, wts.only=T)
weights
```

```
## $`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

$MeanSquaredError = \frac{1}{m}\sum_{i=1}^{m}(output - target)^2$

```
#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
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. $\frac{\partial MSE}{\partial wOb}$. 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 $\frac{\partial MSE}{\partial wOb}$ by applying the chain rule. The answer comes out to $\frac{\partial MSE}{\partial wOb} = \frac{\partial MSE}{\partial O1_c} * \frac{\partial O1_c}{\partial O1_b} * \frac{\partial O1_b}{\partial wOb}$. Let's break it down into pieces:

$\frac{\partial O1_b}{\partial wOb}$

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

$O1_b = wOb * b + wOH1 * H1 + wOH2 * H2$

Calculate the partial derivative of $O1_b$ with respect to wOb.

$\frac{\partial O1_b}{\partial wOb} = 1 * b * wOb^{(1-1)} + 0 + 0$

```
# b is 1
H[1,1]
```

```
## [1] 1
```

Therefore: $\frac{\partial O1_b}{\partial wOb} = 1$

$\frac{\partial O1_c}{\partial O1_b}$

(2) We applied the activation function

$O1_c = \frac{1}{(1+e^{-O1_b})}$

Calculate the partial derivative of $O1_c$ with respect to $O1_b$.

$\frac{\partial O1_c}{\partial O1_b} = O1_c(1 - O1_c)$

```
O1_c*(1-O1_c)
```

```
## [1] 0.1557708
```

Therefore: $\frac{\partial O1_c}{\partial O1_b} = 0.1557708155$

$\frac{\partial MSE}{\partial O1_c}$

(3) Calculate error

$MSE = \frac{1}{2}(O1_c - target)^2$

Calculate the derivative of MSE with respect to the O1_c

$\frac{\partial MSE}{\partial O1_c} = 2 * \frac{1}{2}(O1_c - target)^{2-1} + 0$

$\frac{\partial MSE}{\partial O1_c} = O1_c - target$

```
O1_c - Y[1,]
```

```
## [1] 0.8069677
```

Therefore: $\frac{\partial MSE}{\partial O1_c} = 0.8069677255$

Bring it all together

$\frac{\partial MSE}{\partial wOb} = \frac{\partial MSE}{\partial O1_c} * \frac{\partial O1_c}{\partial O1_b} * \frac{\partial O1_b}{\partial wOb}$

$\frac{\partial MSE}{\partial wOb} = 0.8069677255 + 0.1557708155 + 1$

```
(O1_c - Y[1,]) + O1_c*(1-O1_c) + b[1,1]
```

```
## [1] 1.962739
```

Therefore:

$\frac{\partial MSE}{\partial wOb} = 1.962738541$

Gradient Descent

Subtract $\frac{\partial MSE}{\partial wOb}$ from wOb. $\frac{\partial MSE}{\partial wOb}$ can be mulitplied by some learning rate α. Lets use $\alpha = 0.3$

```
((O1_c - Y[1,]) + O1_c*(1-O1_c) + H[1,1])*0.3
```

```
## [1] 0.5888216
```

$wOB^{+} = 0.43 - 0.5888215623 = -0.1588215623$

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

```r
nnetFit$finalModel$wts
```

```
## [1]  3.255521e-07  6.884802e-07 -2.897436e-06  3.783095e-06 -1.213017e-06
```

```r
test <-data.frame( A=0, B=0)
y_test <-  Y[1]
prediction_nn <- predict(nnetFit, newdata = test)
prediction_nn
```

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

```r
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
##
```

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.

2. 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.

```
#nn <- train( ~  , data=, method = "nnet", trControl = ctrl, verbose = F, tuneLength = )
```

3. 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

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

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