Paul Beeman

November 17, 2017

Task 1

Test Data and Scaling:

```
# split between training and testing data
set.seed(1)
n <- dim(iris)[1]
rows <- sample(1:n, 0.8*n)
train <- iris[rows,]
test <- iris[-rows,]

# write your code here to scale the data
iris$Sepal.Length <- iris$Sepal.Length/max(iris$Sepal.Length)
iris$Sepal.Width <- iris$Sepal.Width/max(iris$Sepal.Width)
iris$Petal.Length <- iris$Petal.Length/max(iris$Petal.Length)
iris$Petal.Width <- iris$Petal.Width/max(iris$Petal.Width)
```

Parameters:

$$W^{(1)} = 12\ parameters$$

$$B^{(1)} = 3\ parameters$$

$$W^{(2)} = 9\ parameters$$

$$B^{(2)} = 3\ parameters$$

Task 2 Turn Outputs into 0s and 1s:

To express the categories correctly, we need to turn the factor labels in species column into vectors of 0s and 1s. For example, an iris of species *setosa* should be expressed as `1 0 0`. Write some code that will do this. Hint: you can use `as.integer()` to turn a factor into numbers, and then use a bit of creativity to turn those values into vectors of 1s and 0s.

```
species_number <- as.integer(iris$Species)
setosa_num <- ifelse(species_number == 1, 1, 0)
versicolor_num <-ifelse(species_number == 2, 1, 0)
virginica_num <-ifelse(species_number == 3, 1, 0)
Species <- iris$Species
iris <- cbind(iris[,1:4], setosa_num, versicolor_num, virginica_num, Species)
head(iris)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width setosa_num
## 1    0.6455696   0.7954545    0.2028986        0.08          1
## 2    0.6202532   0.6818182    0.2028986        0.08          1
## 3    0.5949367   0.7272727    0.1884058        0.08          1
## 4    0.5822785   0.7045455    0.2173913        0.08          1
```

```
## 5    0.6329114   0.8181818    0.2028986        0.08          1
## 6    0.6835443   0.8863636    0.2463768        0.16          1
##   versicolor_num virginica_num Species
## 1              0             0  setosa
## 2              0             0  setosa
## 3              0             0  setosa
## 4              0             0  setosa
## 5              0             0  setosa
## 6              0             0  setosa

train <- iris[rows,]
test <- iris[-rows,]
```

Task 3: Forward Propogation Formula

$$f(t) = \frac{1}{1 + e^{-t}}$$

$$a^{(2)} = f(XW^{(1)} + B^{(1)})$$

$$\hat{y} = f(a^{(2)}W^{(2)} + B^{(2)})$$

Task 4: Forward Propogation as R code

```
#necessary functions
sigmoid <- function(Z){
    1/(1 + exp(-Z))
}

sigmoidprime <- function(z){
    exp(-z)/((1+exp(-z))^2)
}

cost <- function(y,y_hat){
    0.5*sum((y - y_hat)^2)
}

# define the size our our neural network
input_layer_size <- 4
output_layer_size <- 3
hidden_layer_size <- 3

set.seed(1)
# set some initial weights
W_1 <- matrix(runif(input_layer_size * hidden_layer_size)-.5, nrow = input_la
yer_size, ncol = hidden_layer_size)
W_2 <- matrix(runif(hidden_layer_size * output_layer_size)-.5, nrow = hidden_
layer_size, ncol = output_layer_size)

# biases matrix
B_1 <- matrix(runif(hidden_layer_size), ncol = 1)
```

```r
B_2 <- matrix(runif(output_layer_size), ncol = 1)

#X and Y matrices
X <- as.matrix(train[,1:4])
Y <- as.matrix(train[,5:7])

#Forward Propogation
Z_2 <- X%*%W_1
A_2 <- sigmoid(Z_2 + t( B_1 %*% rep(1,120) ) )
Z_3 <- A_2%*%W_2
Y_hat <- sigmoid(Z_3 + t( B_2 %*% rep(1,120) ) )
```

Back Propogation

Task 5: Latex formulas for partial derivatives

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial W^{(2)}}$$

$$= -(y - \hat{y})\frac{\partial \hat{y}}{\partial Z^{(3)}}\frac{\partial Z^{(3)}}{\partial W^{(2)}}$$

$$= -(y - \hat{y})\frac{e^{-Z^{(3)}-B^{(2)}}}{(1 + e^{-Z^{(3)}-B^{(2)}})^2}A^{(2)}$$

$$\frac{\partial J}{\partial W^{(1)}} = \delta^{(3)}\frac{\partial Z^{(3)}}{\partial W^{(1)}}$$

$$= \delta^{(3)}\frac{\partial Z^{(3)}}{\partial A^{(2)}}\frac{\partial A^{(2)}}{\partial W^{(1)}}$$

$$= \delta^{(3)}W^{(2)}\frac{Xe^{-XW^{(1)}-B^{(1)}}}{(1 + e^{-XW^{(1)}-B^{(1)}})^2}$$

$$\frac{\partial J}{\partial B^{(1)}} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial B^{(1)}}$$

$$= -(y - \hat{y})\frac{\partial \hat{y}}{\partial Z^{(3)}}\frac{\partial Z^{(3)}}{\partial B^{(1)}}$$

$$= \delta^{(3)}\frac{\partial Z^{(3)}}{\partial A^{(2)}}\frac{\partial A^{(2)}}{\partial B^{(1)}}$$

$$= \delta^{(3)}W^{(2)}\frac{e^{-Z^{(2)}-B^{(1)}}}{(1 + e^{-Z^{(1)}-B^{(1)}})^2}$$

$$= \delta^{(2)}$$

$$\frac{\partial J}{\partial B^{(2)}} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial B^{(2)}}$$

$$= -(y - \hat{y})\frac{e^{-Z^{(3)}-B^{(2)}}}{(1 + e^{-Z^{(3)}-B^{(2)}})^2}$$

$$= \delta^{(3)}$$

Task 6: R code for partial derivatives

We can see in the above derivatives that there are some issues with dimensionality if we try and multiply some of our matrices together. Turning our derivatives into R code involves manipulating some of the matrices to make sure our dimensions are suited for matrix multiplication and addition.

```
delta_3 <- delta_3 <- ( -(Y - Y_hat) * sigmoidprime(Z_3 + t( B_2 %*% rep(1,12
0) ) ) )
djdw2 <- t(A_2) %*% delta_3

delta_2 <- delta_3 %*% t(W_2) * sigmoidprime(Z_2 + t( B_1 %*% rep(1, 120) ) )
djdw1 <- t(X) %*% delta_2

djdb2 <- rep(1, 120) %*% delta_3

djdb1 <- rep(1, 120) %*% delta_2

#resulting partials
djdw2

##      setosa_num versicolor_num virginica_num
## [1,]   4.554800       4.609546      3.890419
## [2,]   5.879802       6.288232      5.413074
## [3,]   2.668985       3.478291      3.480871

djdw1

##                    [,1]       [,2]      [,3]
## Sepal.Length 0.15620053 0.40070193 1.6569276
## Sepal.Width  0.03909035 0.50408416 1.6406212
## Petal.Length 0.27900213 0.10934802 1.1036671
## Petal.Width  0.31629684 0.02361399 0.9322625

djdb2

##      setosa_num versicolor_num virginica_num
## [1,]    7.62335       8.585331      7.699915
```

```
djdb1
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.1098867 0.6549913 2.315455
```

Task 7: Numerical Gradient Checking

```
# set some initial weights
set.seed(1)
W_1 <- matrix(runif(input_layer_size * hidden_layer_size)-.5, nrow = input_la
yer_size, ncol = hidden_layer_size)
W_2 <- matrix(runif(hidden_layer_size * output_layer_size)-.5, nrow = hidden_
layer_size, ncol = output_layer_size)
B_1 <- matrix(runif(hidden_layer_size), ncol = 1)
B_2 <- matrix(runif(output_layer_size), ncol = 1)

i = 1
X <- as.matrix(train[,1:4])
Y <- as.matrix(train[,5:7])

Z_2 <- X%*%W_1
A_2 <- sigmoid(Z_2 + t( B_1 %*% rep(1,120) ) )
Z_3 <- A_2%*%W_2
Y_hat <- sigmoid(Z_3 + t( B_2 %*% rep(1,120) ) )
currentcost <- cost(Y,Y_hat)  # Current cost

e <- 1e-4  # size of perturbation



# place holder for our numeric gradients
numgrad_w_1 <- matrix(0, nrow = input_layer_size, ncol = hidden_layer_size)
elements <- input_layer_size * hidden_layer_size

for(i in 1:elements){  # calculate the numeric gradient for each value in the
W matrix
    set.seed(1)
    W_1 <- matrix(runif(input_layer_size * hidden_layer_size)-.5, nrow = inpu
t_layer_size, ncol = hidden_layer_size)
    W_2 <- matrix(runif(hidden_layer_size * output_layer_size)-.5, nrow = hid
den_layer_size, ncol = output_layer_size)
    B_1 <- matrix(runif(hidden_layer_size), ncol = 1)
    B_2 <- matrix(runif(output_layer_size), ncol = 1)

    W_1[i] <- W_1[i] + e # apply the perturbation

    Z_2 <- X%*%W_1
    A_2 <- A_2 <- sigmoid(Z_2 + t( B_1 %*% rep(1,120) ) )
    Z_3 <- A_2%*%W_2
    Y_hat <- sigmoid(Z_3 + t( B_2 %*% rep(1,120) ) )
```

```
    numgrad_w_1[i] <- (cost(Y,Y_hat) - currentcost)/e # change in cost over p
erturbation = slope
}
numgrad_w_1

##            [,1]       [,2]      [,3]
## [1,] 0.15619870 0.40069782 1.6569430
## [2,] 0.03908928 0.50407668 1.6406344
## [3,] 0.27900029 0.10934884 1.1036776
## [4,] 0.31629490 0.02361544 0.9322722

djdw1

##                    [,1]       [,2]      [,3]
## Sepal.Length 0.15620053 0.40070193 1.6569276
## Sepal.Width  0.03909035 0.50408416 1.6406212
## Petal.Length 0.27900213 0.10934802 1.1036671
## Petal.Width  0.31629684 0.02361399 0.9322625
```

After performing numerical gradient checking, I feel pretty good about my derivatives.

Task 8: Gradient Descent

```
set.seed(1)
W_1 <- matrix(runif(input_layer_size * hidden_layer_size)-.5, nrow = input_la
yer_size, ncol = hidden_layer_size)
W_2 <- matrix(runif(hidden_layer_size * output_layer_size)-.5, nrow = hidden_
layer_size, ncol = output_layer_size)
B_1 <- matrix(runif(hidden_layer_size), ncol = 1)
B_2 <- matrix(runif(output_layer_size), ncol = 1)

# for cost tracking
cost_hist <- rep(NA, 18000)

scalar <- .2
for(i in 1:18000){
    # this takes the current weights and calculates y-hat
    Z_2 <- X%*%W_1
    A_2 <- A_2 <- sigmoid(Z_2 + t( B_1 %*% rep(1,120) ) )
    Z_3 <- A_2%*%W_2
    Y_hat <- sigmoid(Z_3 + t( B_2 %*% rep(1,120) ) )
    cost_hist[i] <- cost(Y, Y_hat)

    # this part calculates the gradient at the current y-hat
    delta_3 <- delta_3 <- ( -(Y - Y_hat) * sigmoidprime(Z_3 + t( B_2 %*% rep(
1,120) ) ) )
    djdw2 <- t(A_2) %*% delta_3

    delta_2 <- delta_3 %*% t(W_2) * sigmoidprime(Z_2 + t( B_1 %*% rep(1, 120)
) )
    djdw1 <- t(X) %*% delta_2
```

```
    djdb2 <- rep(1, 120) %*% delta_3

    djdb1 <- rep(1, 120) %*% delta_2

    # this updates the weights based on the gradient
    W_1 <- W_1 - scalar * djdw1
    W_2 <- W_2 - scalar * djdw2
    B_1 <- B_1 - scalar * t(djdb1)
    B_2 <- B_2 - scalar * t(djdb2)

    # repeat
}

# the results
W_1
```

```
##                    [,1]       [,2]       [,3]
## Sepal.Length   1.594066  -6.172546 -0.4429642
## Sepal.Width    4.459171 -20.262024  5.1440814
## Petal.Length -24.176362  15.257919 -8.5167462
## Petal.Width    4.740471  52.620150 -9.7209369
```

```
W_2
```

```
##      setosa_num versicolor_num virginica_num
## [1,]   1.107234       8.038447     -7.922875
## [2,]  -5.566673     -16.782368     16.576038
## [3,]  12.025418     -13.163996     -5.159708
```

```
B_1
```

```
##            [,1]
## [1,]  10.870779
## [2,] -30.949324
## [3,]   2.677496
```

```
B_2
```

```
##                     [,1]
## setosa_num     -6.948822
## versicolor_num -2.677947
## virginica_num   2.654854
```

```
Y_hat
```

```
##       setosa_num versicolor_num virginica_num
## 40  9.972793e-01   5.262060e-04  3.514855e-05
## 56  2.495756e-03   9.719175e-01  2.656635e-02
## 85  2.603227e-03   9.844832e-01  1.555299e-02
## 134 1.532731e-03   6.559966e-01  3.437799e-01
```

```
## 30  9.969711e-01   5.918084e-04  3.681796e-05
## 131 4.194082e-06   6.603548e-09  1.000000e+00
## 137 6.834963e-06   3.071226e-07  9.999996e-01
## 95  2.902346e-03   9.883252e-01  1.078113e-02
## 90  3.018453e-03   9.910511e-01  8.269341e-03
## 9   9.968668e-01   6.143211e-04  3.735338e-05
## 29  9.973607e-01   5.090188e-04  3.468978e-05
## 25  9.967940e-01   6.294117e-04  3.775278e-05
## 143 7.781593e-06   4.600311e-07  9.999995e-01
## 53  2.132737e-03   9.449379e-01  5.561471e-02
## 105 4.752561e-06   2.209236e-08  1.000000e+00
## 68  3.724117e-03   9.823723e-01  1.087589e-02
## 97  3.038639e-03   9.895070e-01  9.320383e-03
## 132 7.059832e-06   3.376386e-08  1.000000e+00
## 51  2.512610e-03   9.767729e-01  2.260437e-02
## 102 7.781593e-06   4.600311e-07  9.999995e-01
## 122 8.931238e-06   1.821265e-06  9.999979e-01
## 28  9.973565e-01   5.098799e-04  3.471534e-05
## 84  1.097326e-03   4.249200e-01  5.745212e-01
## 16  9.974949e-01   4.807715e-04  3.391587e-05
## 34  9.977240e-01   4.327542e-04  3.254581e-05
## 49  9.974859e-01   4.825951e-04  3.397245e-05
## 2   9.969732e-01   5.914940e-04  3.679949e-05
## 48  9.972151e-01   5.398600e-04  3.550272e-05
## 107 1.845341e-04   2.136582e-02  9.777393e-01
## 42  9.941218e-01   1.226198e-03  4.898794e-05
## 58  5.860756e-03   9.881688e-01  4.352561e-03
## 72  3.210757e-03   9.920126e-01  6.963058e-03
## 59  2.449625e-03   9.691147e-01  2.937634e-02
## 22  9.968605e-01   6.158266e-04  3.737776e-05
## 96  3.288198e-03   9.878812e-01  9.446305e-03
## 77  2.068693e-03   9.288409e-01  7.103998e-02
## 91  2.529434e-03   9.680941e-01  2.894939e-02
## 13  9.973231e-01   5.168646e-04  3.490701e-05
## 82  4.272555e-03   9.890005e-01  6.207029e-03
## 46  9.964069e-01   7.140665e-04  3.961788e-05
## 114 7.541428e-06   6.004617e-07  9.999993e-01
## 71  7.745011e-04   6.274913e-01  3.751932e-01
## 148 8.024819e-06   6.869390e-07  9.999992e-01
## 60  3.109419e-03   9.928828e-01  6.613197e-03
## 57  2.546224e-03   9.839832e-01  1.638020e-02
## 83  3.458462e-03   9.913569e-01  6.734254e-03
## 3   9.973051e-01   5.208048e-04  3.500209e-05
## 50  9.972883e-01   5.243450e-04  3.509703e-05
## 75  2.924606e-03   9.882734e-01  1.071345e-02
## 70  3.517948e-03   9.894171e-01  7.754464e-03
## 123 3.718260e-06   3.706004e-09  1.000000e+00
## 86  2.801896e-03   9.906397e-01  9.419457e-03
## 43  9.973169e-01   5.182966e-04  3.493641e-05
## 24  9.932575e-01   1.425994e-03  5.197545e-05
```

```
## 7   9.970748e-01   5.698286e-04  3.625879e-05
## 10  9.973147e-01   5.185931e-04  3.495732e-05
## 146 8.500329e-06   1.484719e-06  9.999983e-01
## 125 5.399758e-06   5.199003e-08  9.999999e-01
## 61  4.000830e-03   9.904299e-01  6.044219e-03
## 38  9.976500e-01   4.481465e-04  3.300021e-05
## 136 4.279891e-06   1.038981e-08  1.000000e+00
## 27  9.960162e-01   7.997446e-04  4.141960e-05
## 41  9.972648e-01   5.293911e-04  3.522364e-05
## 140 6.853311e-06   2.932098e-07  9.999997e-01
## 149 7.780500e-06   7.798870e-07  9.999991e-01
## 117 3.324691e-05   1.011314e-05  9.999885e-01
## 88  2.401242e-03   9.716148e-01  2.813919e-02
## 64  2.252693e-03   9.564475e-01  4.305838e-02
## 116 8.130130e-06   1.074093e-06  9.999988e-01
## 109 4.756580e-06   1.094304e-08  1.000000e+00
## 129 5.068720e-06   3.495046e-08  1.000000e+00
## 67  2.616607e-03   9.850646e-01  1.499053e-02
## 80  5.612372e-03   9.882077e-01  4.594669e-03
## 26  9.966343e-01   6.643502e-04  3.853010e-05
## 37  9.974868e-01   4.824453e-04  3.396453e-05
## 79  2.591145e-03   9.848273e-01  1.536720e-02
## 142 9.276166e-06   2.786482e-06  9.999968e-01
## 87  2.458450e-03   9.780343e-01  2.219656e-02
## 99  6.290159e-03   9.882077e-01  3.956240e-03
## 69  2.064117e-03   9.580933e-01  4.329863e-02
## 31  9.968206e-01   6.241226e-04  3.759588e-05
## 103 4.427402e-06   1.304333e-08  1.000000e+00
## 78  1.437281e-03   8.339637e-01  1.694612e-01
## 108 6.271797e-06   1.928780e-08  1.000000e+00
## 81  3.575482e-03   9.903365e-01  7.055208e-03
## 14  9.975413e-01   4.709578e-04  3.364636e-05
## 111 1.076162e-05   2.860724e-06  9.999967e-01
## 8   9.972835e-01   5.253253e-04  3.512572e-05
## 127 9.935805e-05   2.746968e-03  9.970814e-01
## 141 6.426965e-06   1.970136e-07  9.999998e-01
## 15  9.977231e-01   4.329484e-04  3.255001e-05
## 4   9.969847e-01   5.889546e-04  3.674290e-05
## 110 5.001021e-06   3.208530e-08  1.000000e+00
## 66  2.865843e-03   9.893070e-01  1.021165e-02
## 44  9.930665e-01   1.470827e-03  5.259797e-05
## 119 3.695906e-06   3.599110e-09  1.000000e+00
## 139 2.914369e-04   7.327784e-02  9.251853e-01
## 145 6.491388e-06   2.119449e-07  9.999997e-01
## 98  2.917486e-03   9.879415e-01  1.099172e-02
## 101 4.868080e-06   2.640935e-08  1.000000e+00
## 33  9.977583e-01   4.255677e-04  3.233637e-05
## 18  9.971586e-01   5.519549e-04  3.580692e-05
## 113 5.974561e-06   1.122451e-07  9.999999e-01
## 47  9.974882e-01   4.820765e-04  3.396117e-05
```

```
## 94  5.413737e-03   9.890502e-01  4.536185e-03
## 138 5.585929e-05   5.223196e-05  9.999415e-01
## 6   9.968216e-01   6.241573e-04  3.757925e-05
## 21  9.970480e-01   5.753338e-04  3.641295e-05
## 39  9.971285e-01   5.583260e-04  3.597273e-05
## 54  2.910624e-03   9.903547e-01  9.207539e-03
## 150 8.255553e-05   6.526742e-04  9.992928e-01
## 73  1.649635e-03   7.928367e-01  2.091227e-01
## 65  3.777580e-03   9.927730e-01  5.194440e-03
## 62  2.937282e-03   9.922195e-01  7.647222e-03
## 55  2.432666e-03   9.782075e-01  2.227691e-02
## 1   9.974339e-01   4.935605e-04  3.427211e-05
## 128 2.254266e-04   2.746194e-02  9.716108e-01
## 118 3.831752e-06   4.576659e-09  1.000000e+00
## 130 9.035575e-04   8.046782e-02  9.175063e-01
## 20  9.973351e-01   5.144643e-04  3.483308e-05
```

Y

```
##     setosa_num versicolor_num virginica_num
## 40           1              0             0
## 56           0              1             0
## 85           0              1             0
## 134          0              0             1
## 30           1              0             0
## 131          0              0             1
## 137          0              0             1
## 95           0              1             0
## 90           0              1             0
## 9            1              0             0
## 29           1              0             0
## 25           1              0             0
## 143          0              0             1
## 53           0              1             0
## 105          0              0             1
## 68           0              1             0
## 97           0              1             0
## 132          0              0             1
## 51           0              1             0
## 102          0              0             1
## 122          0              0             1
## 28           1              0             0
## 84           0              1             0
## 16           1              0             0
## 34           1              0             0
## 49           1              0             0
## 2            1              0             0
## 48           1              0             0
## 107          0              0             1
## 42           1              0             0
```

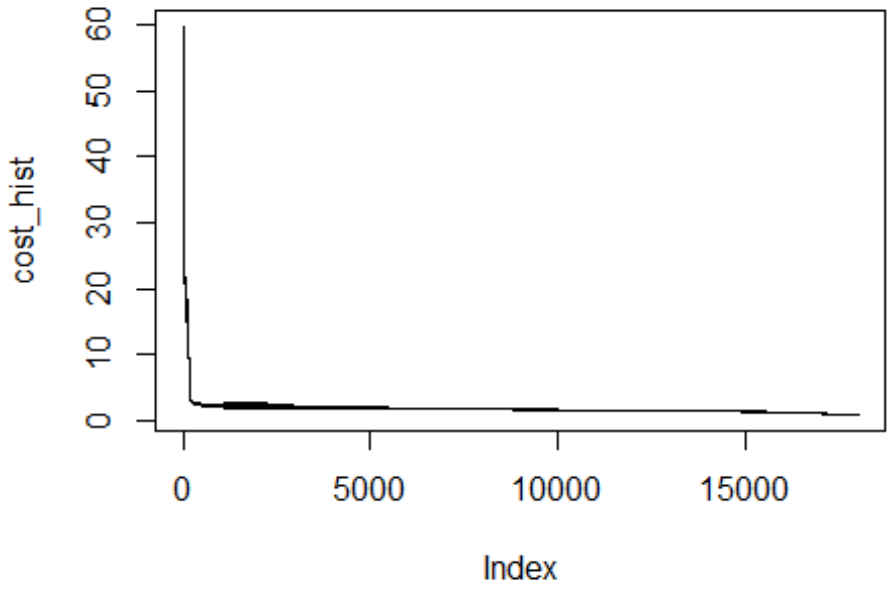
```
## 58           0              1              0
## 72           0              1              0
## 59           0              1              0
## 22           1              0              0
## 96           0              1              0
## 77           0              1              0
## 91           0              1              0
## 13           1              0              0
## 82           0              1              0
## 46           1              0              0
## 114          0              0              1
## 71           0              1              0
## 148          0              0              1
## 60           0              1              0
## 57           0              1              0
## 83           0              1              0
## 3            1              0              0
## 50           1              0              0
## 75           0              1              0
## 70           0              1              0
## 123          0              0              1
## 86           0              1              0
## 43           1              0              0
## 24           1              0              0
## 7            1              0              0
## 10           1              0              0
## 146          0              0              1
## 125          0              0              1
## 61           0              1              0
## 38           1              0              0
## 136          0              0              1
## 27           1              0              0
## 41           1              0              0
## 140          0              0              1
## 149          0              0              1
## 117          0              0              1
## 88           0              1              0
## 64           0              1              0
## 116          0              0              1
## 109          0              0              1
## 129          0              0              1
## 67           0              1              0
## 80           0              1              0
## 26           1              0              0
## 37           1              0              0
## 79           0              1              0
## 142          0              0              1
## 87           0              1              0
## 99           0              1              0
## 69           0              1              0
```

```
## 31           1              0              0
## 103          0              0              1
## 78           0              1              0
## 108          0              0              1
## 81           0              1              0
## 14           1              0              0
## 111          0              0              1
## 8            1              0              0
## 127          0              0              1
## 141          0              0              1
## 15           1              0              0
## 4            1              0              0
## 110          0              0              1
## 66           0              1              0
## 44           1              0              0
## 119          0              0              1
## 139          0              0              1
## 145          0              0              1
## 98           0              1              0
## 101          0              0              1
## 33           1              0              0
## 18           1              0              0
## 113          0              0              1
## 47           1              0              0
## 94           0              1              0
## 138          0              0              1
## 6            1              0              0
## 21           1              0              0
## 39           1              0              0
## 54           0              1              0
## 150          0              0              1
## 73           0              1              0
## 65           0              1              0
## 62           0              1              0
## 55           0              1              0
## 1            1              0              0
## 128          0              0              1
## 118          0              0              1
## 130          0              0              1
## 20           1              0              0
```

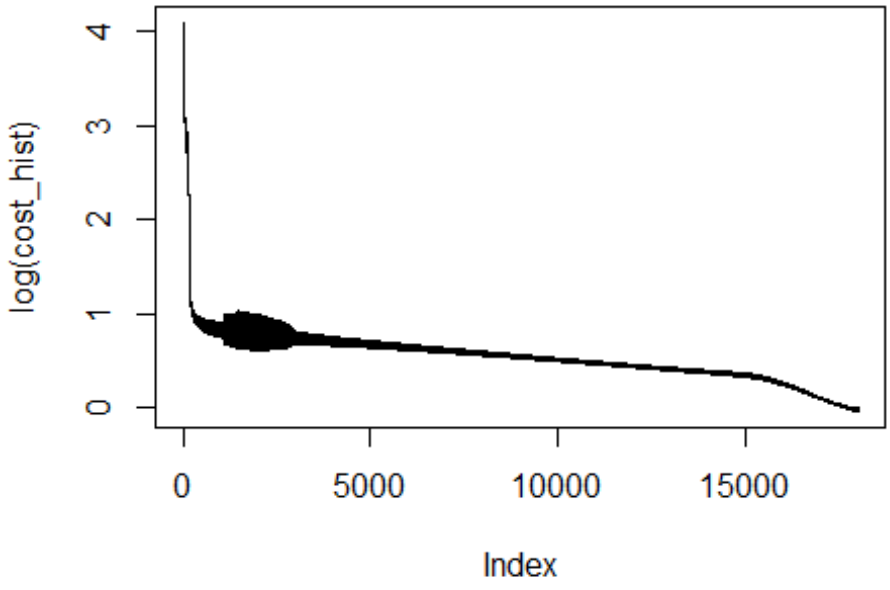
```
cost(Y,Y_hat)
```

```
## [1] 1.005635
```

```
plot(cost_hist, type="l") # plot the history of our cost function
```




```
plot(log(cost_hist), type="l") # plotting the log of the cost emphasizes the change
```



At this point, I feel fairly confident in my model. The cost funtion shows that we have a fairly close fit and while I could continue messing with the scalar and number of training rounds, I am willing to move on and see how the model does with the training data.

Task 9: Testing our Trained Model

```
## use test data

X_test <- as.matrix(test[,1:4])
Y_test <- as.matrix(test[,5:7])

Z_2 <- X_test%*%W_1
A_2 <- A_2 <- sigmoid(Z_2 + t( B_1 %*% rep(1,30) ) )
Z_3 <- A_2%*%W_2
Y_hat <- sigmoid(Z_3 + t( B_2 %*% rep(1,30) ) )
guess <- round(Y_hat)
guess
```

```
##     setosa_num versicolor_num virginica_num
## 5            1              0             0
## 11           1              0             0
## 12           1              0             0
## 17           1              0             0
## 19           1              0             0
## 23           1              0             0
## 32           1              0             0
## 35           1              0             0
## 36           1              0             0
## 45           1              0             0
## 52           0              1             0
## 63           0              1             0
## 74           0              1             0
## 76           0              1             0
## 89           0              1             0
## 92           0              1             0
## 93           0              1             0
## 100          0              1             0
## 104          0              0             1
## 106          0              0             1
## 112          0              0             1
## 115          0              0             1
## 120          0              0             1
## 121          0              0             1
## 124          0              0             1
## 126          0              0             1
## 133          0              0             1
## 135          0              0             1
## 144          0              0             1
## 147          0              0             1
```

```
Y_test
```

```
##     setosa_num versicolor_num virginica_num
## 5            1              0             0
## 11           1              0             0
## 12           1              0             0
## 17           1              0             0
## 19           1              0             0
## 23           1              0             0
## 32           1              0             0
## 35           1              0             0
## 36           1              0             0
## 45           1              0             0
## 52           0              1             0
## 63           0              1             0
## 74           0              1             0
## 76           0              1             0
## 89           0              1             0
## 92           0              1             0
## 93           0              1             0
## 100          0              1             0
## 104          0              0             1
## 106          0              0             1
## 112          0              0             1
## 115          0              0             1
## 120          0              0             1
## 121          0              0             1
## 124          0              0             1
## 126          0              0             1
## 133          0              0             1
## 135          0              0             1
## 144          0              0             1
## 147          0              0             1
```

```
table(guess%*%matrix(1:3),Y_test%*%matrix(1:3))
```

```
##
##      1  2  3
##   1 10  0  0
##   2  0  8  0
##   3  0  0 12
```

My model guessed correctly 30/30 times as we can see by the contigency table. I could probably fiddle with the scalar and number of itterations to try and get a more optimal, less computationally heavy model that would also get a perfect prediction, but I will leave that to the black box algorithm. Interestingly, I was able to lower my cost function by adding higher number of iterations, but I did not always get perfect guesses and the computation time (several agaonizing seconds) was not worth it to me so I settled on this model.

Task 10: Black Box Code

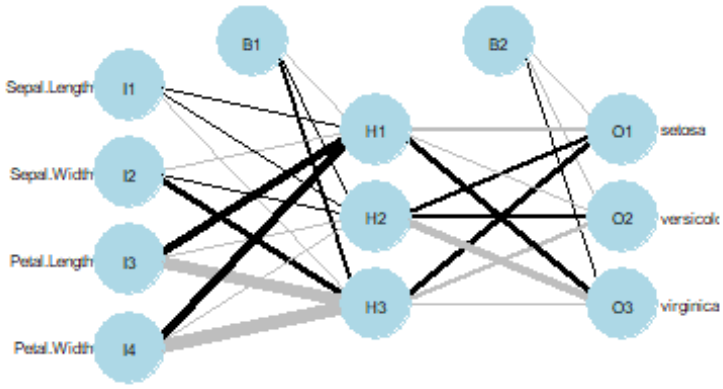
```
set.seed(1)
n <- dim(iris)[1]
```

```r
rows <- sample(1:n, 0.8*n)
train <- iris[rows,]

library(nnet)
library(NeuralNetTools)
irismodel <- nnet(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal
.Width, size=3, data = train)
```

```
## # weights:  27
## initial  value 140.610848
## iter  10 value 12.646902
## iter  20 value 5.941742
## iter  30 value 5.812551
## iter  40 value 5.748340
## iter  50 value 5.730385
## iter  60 value 5.724673
## iter  70 value 5.720554
## iter  80 value 5.708326
## iter  90 value 5.697913
## iter 100 value 5.675483
## final  value 5.675483
## stopped after 100 iterations
```

```r
plotnet(irismodel, cex=.5) # a plot of our network
```




```r
results <- predict(irismodel, iris[-rows,])
data.frame(round(results), actual = iris[-rows, 8])
```

```
##     setosa versicolor virginica     actual
## 5        1          0         0     setosa
## 11       1          0         0     setosa
## 12       1          0         0     setosa
## 17       1          0         0     setosa
## 19       1          0         0     setosa
## 23       1          0         0     setosa
## 32       1          0         0     setosa
## 35       1          0         0     setosa
## 36       1          0         0     setosa
## 45       1          0         0     setosa
## 52       0          1         0 versicolor
## 63       0          1         0 versicolor
## 74       0          1         0 versicolor
## 76       0          1         0 versicolor
## 89       0          1         0 versicolor
## 92       0          1         0 versicolor
## 93       0          1         0 versicolor
## 100      0          1         0 versicolor
## 104      0          0         1  virginica
## 106      0          0         1  virginica
## 112      0          0         1  virginica
## 115      0          0         1  virginica
## 120      0          0         1  virginica
## 121      0          0         1  virginica
## 124      0          0         1  virginica
## 126      0          0         1  virginica
## 133      0          0         1  virginica
## 135      0          0         1  virginica
## 144      0          0         1  virginica
## 147      0          0         1  virginica
```

```
table(round(results)%*%matrix(1:3),Y_test%*%matrix(1:3))
```

```
##
##      1  2  3
##   1 10  0  0
##   2  0  8  0
##   3  0  0 12
```

```
# we can see that the predicted probability of each class matches the actual
label
```

As I suspected the black box code was able to predict the test data with 100% accuracy in an efficient manner.