Jakob Lovato
Will Bliss
STAT 760
HW 7

1. When considering a Neural Network, the activation function $\sigma(v)$ is usually chosen to be the sigmoid

$$\sigma(v) = \frac{1}{1+e^{-v}}$$

(ESL pg. 392). Let $Z_m = \sigma(\alpha_{0m} + \alpha_m^T X)$. Note that $\alpha_{0m} + \alpha_m^T X$ is a linear combination of X, where X is the input. When α_m are nearly 0, $\alpha_m^T X$ are also 0 since $0 \cdot X = 0$. Thus Z_m is nearly linear when α_m is close to 0. Let $t_k = \beta_{0k} + \beta_k^T Z$. Then t_k is nearly linear since Z is nearly linear. Then we apply the identity function, which can be seen from eqn 11.5 (ESL pg. 392), giving us

$$f_k(X) = g_k(t) = t$$

Hence the resulting model is nearly linear in the inputs.

2. Observe the Cross-entropy function

$$R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log(f_k(x_i)). \quad \text{(11.10 from ESL pg. 395)}$$

For forward propogation, we have the equations 11.5 from ESL pg 392:

$$Z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i), \quad m = 1, \ldots, M$$
$$T_k = \beta_{0k} + \beta_k^T X, \quad k = 1, \ldots, K$$
$$f_x(X) = g_k(T), \quad k = 1, \ldots, K.$$

Then

$$R(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(-y_{ik} \log(f_k(x_i))\right)$$

which has partial derivatives

$$\frac{\partial R_i}{\partial \beta_{km}} = -\frac{y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i) z_{mi}$$

and

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} \frac{y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_{0m} + \alpha_m^T x_i) x_{il}$$

Then we update using gradient descent. Let γ_r be the learning rate. It follows that

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}}$$

and

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}}.$$

Let δ_{ki} be the errors of the current models output and s_{mi} be the errors of the hidden layer. Then

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}$$

and

$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}.$$

Thus, we have the backward-propagation equation

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}.$$

3. Let

$$T_k = \beta_{0k} + \beta_k^T x \quad , k = 1, \dots, K. \quad \text{(Eqn 11.5)}$$

Let

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}. \quad \text{(Eqn 11.6)}$$

Then

$$f_k(x) = g_k(x) = \frac{e^{\beta_{0k} + \beta_k^T x}}{\sum_{l=1}^{K} e^{\beta_{0l} + \beta_l^T x}}. \quad (1)$$

Observe the cross-entropy function

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log(f_k(x_i)). \quad \text{(Eqn 1.10)}$$

To optimize the NN, we need to minimize the error of $R(\theta)$. Equivalently, we can maximize the multinomial logistic model

$$\ell(\beta) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log(p_k(x_i; \beta))$$

Which was studied in Ex 4.4 (HW 6 Ex 1).
We get

$$P(G = i \mid X = x) = \frac{e^{\beta_{0i} + \beta_i^T x}}{\sum_{l=1}^{K} e^{\beta_{0l} + \beta_l^T x}}.$$

Then

$$\begin{aligned} P(G = K \mid X = x) &= 1 - \sum_{i=1}^{K-1} P(G = i \mid X = x) \\ &= 1 - \sum_{i=1}^{K-1} \left(\frac{e^{\beta_{0i} + \beta_i^T x}}{\sum_{l=1}^{K} e^{\beta_{0l} + \beta_l^T x}} \right) \\ &= \frac{e^{\beta_{0K} + \beta_K^T x}}{\sum_{l=1}^{K} e^{\beta_{0l} + \beta_l^T x}} \end{aligned}$$

Which is equivalent to $f_k(x) = g_k(x)$ from (1), as required.

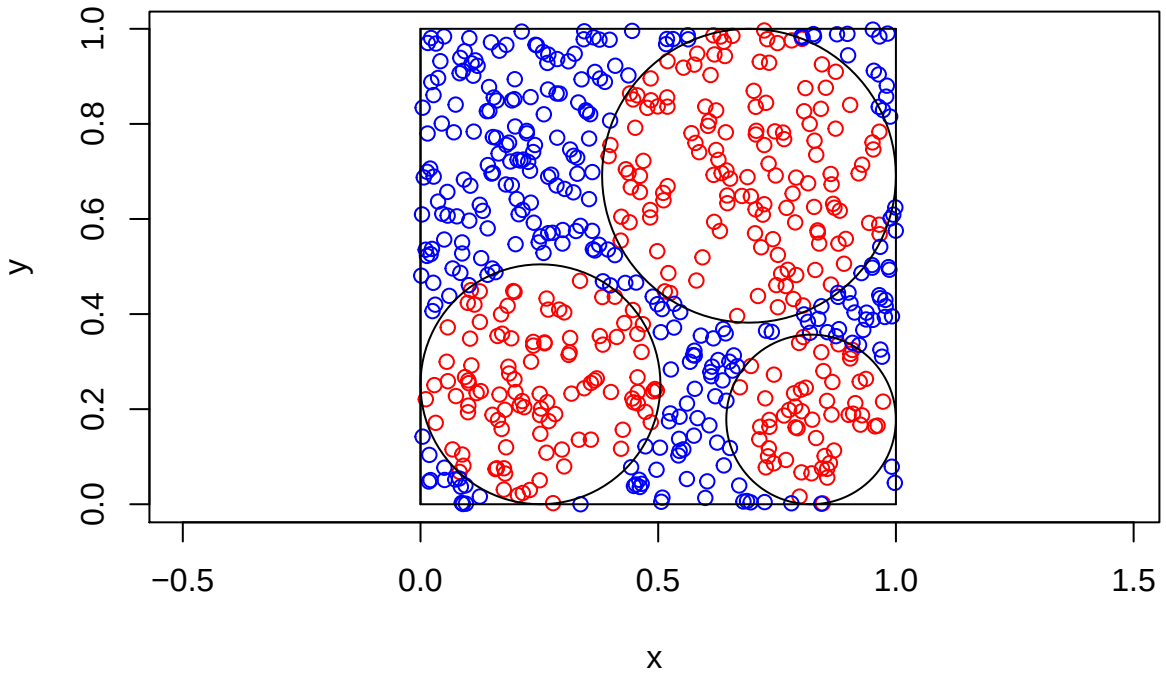
Stat 760 Homework 7 Question 4

Will Bliss and Jakob Lovato

3/18/2022

```r
#Generate data

#Radius of circles containing 10%, 20%, and 30% of area of unit square
R1 <- sqrt(0.1 / pi)
R2 <- sqrt(0.2 / pi)
R3 <- sqrt(0.3 / pi)

#Centers to use for each circle
c1 <- c(1 - R1, R1)
c2 <- c(R2, R2)
c3 <- c(1 - R3, 1 - R3)

#Generate points inside circles
insidex <- c()
insidey <- c()
while(TRUE){
  randx <- runif(1, min = 0, max = 1)
  randy <- runif(1, min = 0, max = 1)
  if((randx - c1[1]) ^ 2 + (randy - c1[2]) ^2 < (R1 ^ 2)){
    insidex <- c(insidex, randx)
    insidey <- c(insidey, randy)
  }
  if((randx - c2[1]) ^ 2 + (randy - c2[2]) ^2 < (R2 ^ 2)){
    insidex <- c(insidex, randx)
    insidey <- c(insidey, randy)
  }
  if((randx - c3[1]) ^ 2 + (randy - c3[2]) ^2 < (R3 ^ 2)){
    insidex <- c(insidex, randx)
    insidey <- c(insidey, randy)
  }
  if(length(insidex) >= 300){break}
}
inside <- data.frame(x = insidex, y = insidey)
plot(inside, asp = 1, col = "red")

#Generate points outside of circles
outsidex <- c()
outsidey <- c()
while(TRUE){
  randx <- runif(1, min = 0, max = 1)
  randy <- runif(1, min = 0, max = 1)
  if(((randx - c1[1]) ^ 2 + (randy - c1[2]) ^2 > (R1 ^ 2))
```

```
    & (randx - c2[1]) ^ 2 + (randy - c2[2]) ^2 > (R2 ^ 2)
    & (randx - c3[1]) ^ 2 + (randy - c3[2]) ^2 > (R3 ^ 2)){
    outsidex <- c(outsidex, randx)
    outsidey <- c(outsidey, randy)
  if(length(outsidex) >= 300){break}
  }
}
outside <- data.frame(x = outsidex, y = outsidey)
points(outside, asp = 1, col = "blue")

#Overlay circles
draw.circle(c3[1], c3[2], R3)
draw.circle(c1[1], c1[2], R1)
draw.circle(c2[1], c2[2], R2)
lines(c(0,0), c(0,1))
lines(c(0,0), c(1,0))
lines(c(1,1), c(1,0))
lines(c(0,1), c(1,1))
lines(c(0,1), c(0,0))
```




```
#combine data
#class A = 0, class B = 1
outside <- cbind(outside, "class" = 0)
inside <- cbind(inside, "class" = 1)
data <- rbind(outside, inside)
```

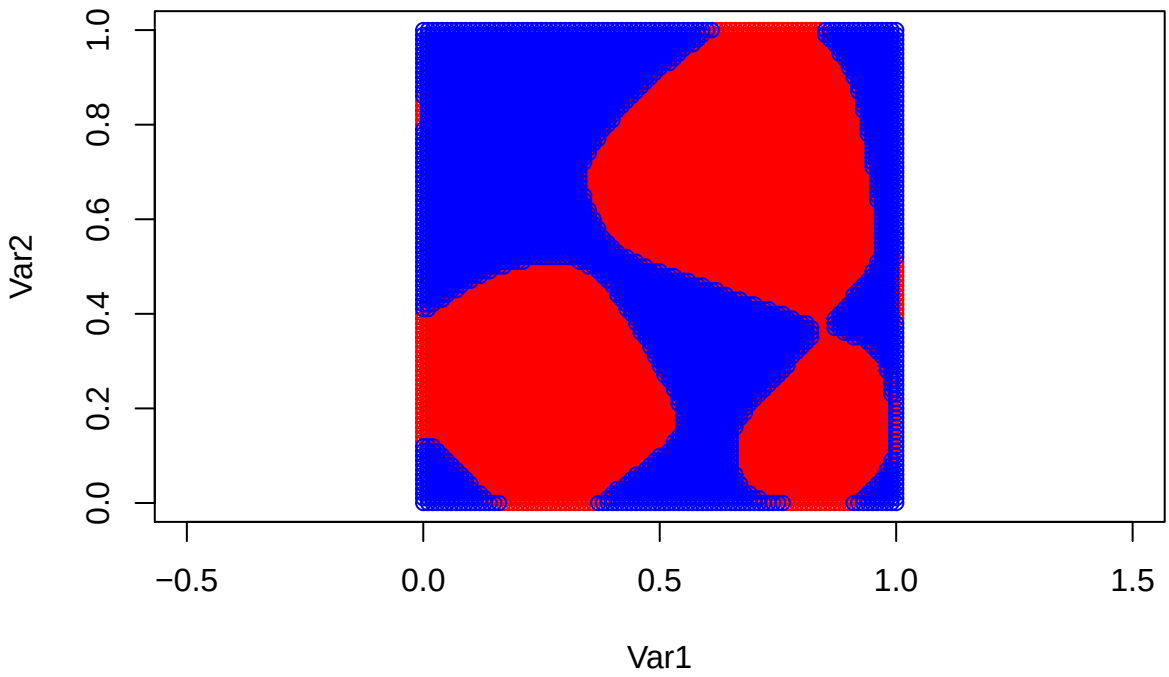
```
#Generatre sigmoid function (activation function)
 sigmoid <- function(x){
   1 / (1 + (exp(-x)))
 }

 derivative <- function(x){
   sigmoid(x) * (1 - sigmoid(x))
```

```
}

#Generate random initial weights
randWeights <- matrix(c(runif(2 * 10)), nrow = 2,
                      ncol = 10, byrow = TRUE)

#Store current step of neural network in training
nn <- list(
  input = cbind(data[,-3], "ones" = rep(1, 600)),
  w1 = rbind(randWeights, rep(0, ncol(randWeights))),
  w2 = rbind(matrix(runif(10), ncol = 1, byrow = TRUE), 0),
  target = data[,3],
  preds = matrix(rep(0, nrow(data)), ncol = 1, byrow = TRUE),
  hidden = rep(0, 6001)
)
```

```
gamma <- 0.75

#train the model
for(count in 1:10000){
  #keep track of error
  errors <- c()
  #keep track of predictions
  predictions <- c()
  #update for every data point
  for(i in 1:600){
    #feed forward
    nn$hidden <- c(sigmoid(as.matrix(nn$input[i,]) %*% as.matrix(nn$w1)), 1)
    nn$preds <- c(sigmoid(nn$hidden %*% nn$w2), 1)
    predictions <- c(predictions, nn$preds[1])
    #backprop
    e <- nn$preds[-length(nn$preds)] - nn$target[i]
    errors <- c(errors, e)
    D1 <- diag(as.vector(nn$hidden[-length(nn$hidden)]
                         * (1 - as.vector(nn$hidden[-length(nn$hidden)]))))
    D2 <- as.vector(nn$preds[-length(nn$preds)]
                         * (1 - as.vector(nn$preds[-length(nn$preds)])))
    delta2 <- D2 * e
    delta1 <- (D1 %*% t(t(nn$w2[-11]))) * delta2
    nn$w2 <- nn$w2 + (-gamma * delta2 * as.matrix(nn$hidden))
    nn$w1 <- nn$w1 + (-gamma * t(delta1 %*% as.matrix(nn$input[i,])))
  }
}
```

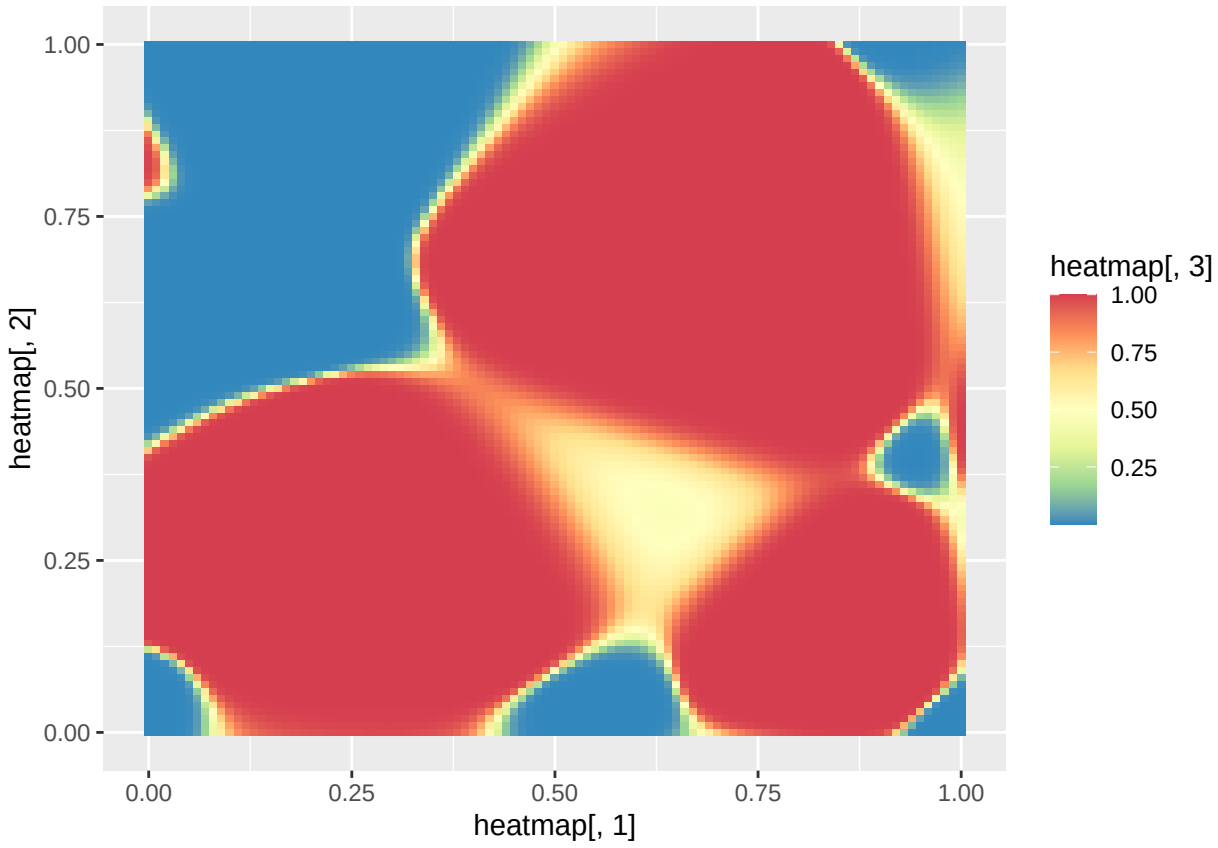
```
#test neural network on 100x100 grid
pixels <- expand.grid(seq(0, 1, by = .01), seq(0, 1, by = .01))
pixels <- cbind(pixels, rep(1, nrow(pixels)))
classifiedPixels <- cbind(pixels[,-3], class = 0)
pixelPredictions <- c()
for(i in 1:nrow(pixels)){
  o1 <- cbind(sigmoid(as.matrix(pixels[i,]) %*% as.matrix(nn$w1)), 1)
  o2 <- sigmoid(o1 %*% as.matrix(nn$w2))
  pixelPredictions <- c(pixelPredictions, o2)
  #Define boundary as 0.95
```

```
  if(o2 < 0.95){
    classifiedPixels[i,3] <- 0
  }
  else{
    classifiedPixels[i,3] <- 1
  }
}
plot(classifiedPixels[which(classifiedPixels[,3] == 1), -3], col = "red", asp = 1)
points(classifiedPixels[which(classifiedPixels[,3] == 0), -3], col = "blue")
```



We also created a heatmap to see areas where classification is more difficult for the Neural Network. As expected, the expected values are further from 0 and 1 around the edges of the circles.

```
#Create Heatmap
heatmap <- cbind(pixels[,-3], pixelPredictions)
ggplot(heatmap, aes(heatmap[,1], heatmap[,2], fill = heatmap[,3])) +
  geom_tile() +
  scale_fill_distiller(palette = "Spectral")
```


1.00
0.75
0.50
0.25
0.00
heatmap[, 2]
0.00
0.25
0.50
0.75
1.00
heatmap[, 1]
heatmap[, 3]
1.00
0.75
0.50
0.25