Chapter 10: Widrow-Hoff Learning

Brandon Morgan

1/18/2021

E10.4

We are given the following input and expected output patterns:

$$p_1 = \begin{bmatrix}1\\1\end{bmatrix}, t_1 = 1 \quad p_2 = \begin{bmatrix}-1\\-1\end{bmatrix}, t_2 = -1$$

1

Here we have the mean square performance index:

$$F(X) = c - 2X^T h + X^T R X$$

where $c = E[t^2]$; because we have assumed each pattern occurs with equal probability, then there is a 50% chance either p_1 or p_2 is chosen: $c = E[t^2] = t_1^2\theta_1 + t_2^2\theta_2 = (1)^2(0.5) + (-1)^2(.05) = 1$, where θ denotes probability.

Next, we calculate the cross-correlation between the input and the target output:

$$h = E(z) = \theta_1 t_1 p_1 + \theta_2 t_2 p_2 = (0.5)(1)\begin{bmatrix}1\\1\end{bmatrix} + (0.5)(-1)\begin{bmatrix}-1\\-1\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}$$

Lastly, the correlation matrix R can be calculated to be $R = E[zz^T] = p_1 p_1^T \theta_1 + p_2 p_2^T \theta_2$:

$$R = 0.5\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1 & 1\end{bmatrix} + 0.5\begin{bmatrix}-1\\-1\end{bmatrix}\begin{bmatrix}-1 & -1\end{bmatrix} = \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$$

Therefore, the mean square error performance index is $F(X) = c - 2X^T h + X^T R X$:

$$F(X) = 1 - 2X^T\begin{bmatrix}1\\1\end{bmatrix} + X^T\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}X$$

which reduces down to:

$$F(X) = 1 - 2x_1 - 2x_2 + x_1^2 + 2x_1x_2 + x_2^2$$

The Hessian matrix of our square error performance index is equal to $H = 2R$, $H = \begin{bmatrix}2 & 2\\2 & 2\end{bmatrix}$.

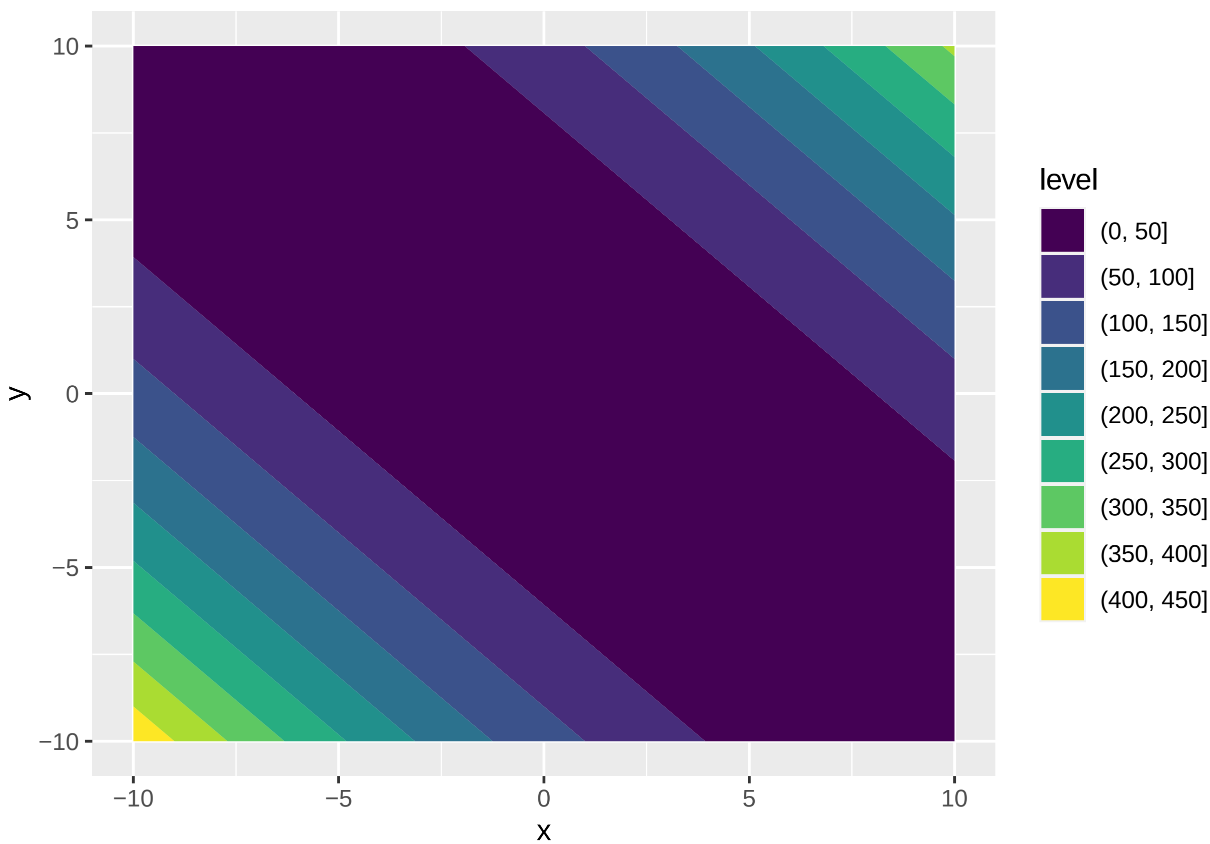
```r
library(purrr) # used for map2_dbl function
library(plotly) # used for 3D plotting
library(ggplot2) # used for plotting
```

```r
fun = function(x, y) {
  1-2*x-2*y+x^2+2*x*y+y^2
}
```

Here we have the contour plot of our function:

```r
# BOUNDS USED FOR CONTOUR
pointsX = seq(-10, 10, length=200)# create a 200x1 vector of values for x axis
pointsY = seq(-10, 10, length=200) # create a 200x1 vector of values for y axis
myGrid = expand.grid(pointsX, pointsY) # create 200x200 grid of points
z = map2_dbl(myGrid$Var1, myGrid$Var2, ~fun(.x, .y)) # maps all possible combinations
# of the grid through the function
x = myGrid$Var1
y =  myGrid$Var2
myPlot = plot_ly(x=~x, y=~y, z=~z, type="mesh3d", intensity=~z,
        colors = colorRamp(c("black","red","yellow","chartreuse3")))
myGrid$z = z

v = ggplot( myGrid,aes(x, y, z=z)) +geom_contour_filled()
v
```



2

The maximum stable learning rate is given by $\alpha < \frac{1}{\lambda_{max}}$ where λ_{max} is the largest positive eigenvalue of the correlation matrix R; or, $\alpha < \frac{2}{\lambda_{max}}$ where λ_{max} is the largest positive eigenvalue of the Hessian matrix H.

```
R = matrix(c(1, 1, 1, 1), ncol=2)
H = 2*R
eigen(R)
```

```
## eigen() decomposition
## $values
## [1] 2 0
##
## $vectors
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

```
eigen(H)
```

```
## eigen() decomposition
## $values
## [1] 4 0
##
## $vectors
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

Thus, the largest eigen value for R is 2;therefore $\alpha < \frac{1}{2} = 0.5$. The largest eigen value for H is 4;therefore $\alpha < \frac{2}{4} = 0.5$.

3

Instead of MATLAB, R code will be used to implement the LMS algorithm. The LMS function defined below takes in an *init* set of weights, an initial *bias*, a constant $learning_rate$, the input vectors P, the expected outcome T, maximum number of iterations, a convergence tolerance, an output mask boolean, and *useBias* boolean to signify if the algorithm should use the bias during calculations.

```
LMS = function(init, bias, learning_rate, input, expect, maxIter, tol, mask, useBias) {

  n = ncol(input)
  w = init
  b = bias

  for(i in 1:maxIter) { # loop over iterations

    if(!mask) { # if the user does want to print out statements
      print(sprintf("     Iteration %d", i))
    }
```

```r
for(j in 1:n) { # loop over input vectors

  if(useBias) { # does the user want to include a bias
    a = w%*%input[,j]+b
  }
  else {
    a = w%*%input[,j]
  }


  e = expect[j] - a

  w = w + 2*learning_rate*e%*%t(input[,j])

  if(useBias) {
    b = b + 2*learning_rate*e
  }

  if(!mask) { # if the user does want to print out statements
    print(sprintf("INPUT VECTOR: %d", j))
    print("a value: ")
    print(a)
    print("error value: ")
    print(e)
    print("new weight value: ")
    print(w)
    print("new bias value: ")
    print(b)
  }
}


error = 0

for(j in 1:n) {

  if(useBias) { # does the user want to include a bias
    a = w%*%input[,j]+b
  }
  else {
    a = w%*%input[,j]
  }

  e = expect[j] - a

  error = error + e^2

}

# if the square root of the sum of the square
# errors is greater than the tol, then algo
# has not converged
converged = 1
```

```
    if(sqrt(error)>tol) {
      converged = 0
    }

    if(converged) {
      print("    Algorithm CONVERGED:")
      print("Current weight is: ")
      print(w)
      if(useBias) { # does the user want to include a bias
        print("Current bias is: ")
        print(b)
      }

      print(sprintf("Iterations taken: %d", i))
      if(useBias) { # does the user want to include a bias
        return(list(w=w, b=b))
      }
      return(list(w=w))
    }


  }

  error = 0

  for(j in 1:n) {

    a = w%*%input[,j]+b

    e = expect[j] - a

    error = error + abs(e)

  }

  print("MAXIMUM ITERATIONS REACHED")
  print("Current weight is: ")
  print(w)
  if(useBias) { # does the user want to include a bias
        print("Current bias is: ")
        print(b)
  }
  print(sprintf("Iterations taken: %d", i))
  print("ERROR: ")
  print(error)
  if(useBias) { # does the user want to include a bias
        return(list(w=w, b=b))
  }
  return(list(w=w))

}
```

Here we will call our algorithm with an initial weight matrix of zeroes and learning rate of 0.2.

```r
init = matrix(c(0, 0), ncol=2)
input = matrix(c(1, 1, -1, -1), ncol=2, byrow=FALSE)
expected = c(1, -1)
val = LMS(init, 0, 0.20, input, expected, 100, 1e-10, 1, 1)
```

```
## [1] "    Algorithm CONVERGED:"
## [1] "Current weight is: "
##      [,1] [,2]
## [1,]  0.5  0.5
## [1] "Current bias is: "
##              [,1]
## [1,] 3.192195e-11
## [1] "Iterations taken: 15"
```

After running our algorithm, it convered in fifteen iterations, yielding the weight is $w = [0.5, 0.5]$ with bias $b = 0$.

It was determined earlier that the maximum stable learning rate was $\alpha < 0.5$; however, any value grater than or equal to $\alpha = 0.3$ diverged. This is extremely interesting…

```r
val = LMS(init, 0, 0.3, input, expected, 100, 1e-10, 1, 1)
```

```
## [1] "MAXIMUM ITERATIONS REACHED"
## [1] "Current weight is: "
##      [,1] [,2]
## [1,]  0.5  0.5
## [1] "Current bias is: "
##              [,1]
## [1,] 9.420945e-11
## [1] "Iterations taken: 100"
## [1] "ERROR: "
##              [,1]
## [1,] 2.783767e-10
```

Here we have our algorithm again excpe this time with initial inputs of ones.

```r
init = matrix(c(1,1), ncol=2)
val = LMS(init, 1, 0.2, input, expected, 100, 1e-10, 1, 1)
```

```
## [1] "    Algorithm CONVERGED:"
## [1] "Current weight is: "
##      [,1] [,2]
## [1,]  0.5  0.5
## [1] "Current bias is: "
##               [,1]
## [1,] -7.398161e-12
## [1] "Iterations taken: 15"
```

Even after changing the initial points for the weights and bias, the same weight and bias is converged. From our LMS algorithm, we get the following values:

$$Wp = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = 0.5p_1 + 0.5p_2$$

We can now solve the decision boundary by setting this equation to zero,

$$Wp = 0.5p_1 + 0.5p_2 = 0 \;\therefore\; p_1 = -p_2$$

We end up getting a line with a negative slope centered about the origin, $y = -x$. Here we have our input patterns classified by color along with our decision boundary created by our LMS algorithm:

```
data =data.frame(p1=c(1, -1), p2=c(1,-1), t=c(1, -1))
ggplot(data=data, aes(x=p1, y=p2, color=t))+geom_point(aes(size=2))+xlab("X")+
  ylab("Y")+ggtitle("Plot of p1, p2, and decision boundary")+
  geom_hline(yintercept=0)+geom_vline(xintercept=0)+
  geom_abline(intercept=0, slope=-1)
```