Aaron P. Mills Z-23547104

Dr. Ghoraani CAP 4613 Intro to Deep Learning

Assignment 3 20 February 2022

https://colab.research.google.com/drive/1pNqUSkL_jt5qb6_6mBB42U9j8fGBpY3r?usp=sharing

Note: The code PDF is below, after the handwork solutions.

Problem 1)

a) Answer: ↓

N = 6 (#samples), η = 1 (learning rate), W = 0 (initialize weights to [0,0,0])

$$\theta(v) = y = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases} = predicted\ output, update(y\ or\ n) = \begin{cases} y, & y == d \\ n, & y \neq d \end{cases}$$

$$v = XW = \sum x_i w_i, \Delta W = \eta(d_i - y_i)X_i$$

Iteration 1:

| $n_i$ | $X(x_0, x_1, x_2)$ | $d$ | $W(w_0, w_1, w_2)$ | $v$ | $y$ | update(y/n)? | $\Delta W(\Delta w_1, \Delta w_2, \Delta w_3)$ |
|---|---|---|---|---|---|---|---|
| 1 | [1,1,1] | 1 | [0,0,0] | 0 | 1 | n | [0,0,0] |
| 2 | [1,1,0] | 1 | [0,0,0] | 0 | 1 | n | [0,0,0] |
| 3 | [1,0,1] | 0 | [0,0,0] | 0 | 1 | y | [-1,0,1] |
| 4 | [1,-1,-1] | 0 | [-1,0,1] | -2 | 0 | n | [0,0,0] |
| 5 | [1,-1,0] | 0 | [-1,0,1] | -1 | 0 | n | [0,0,0] |
| 6 | [1,-1,1] | 0 | [-1,0,1] | 0 | 1 | y | [-1,1-1] |

i=1| $v = 1(0) + 1(0) + 1(0) = 0 \geq 0 \rightarrow y = 1 = d \rightarrow update = no \rightarrow W = W$

i=2| $v = 1(0 + 1(0) + 0(0) = 0 \geq 0 \rightarrow y = 1 = d \rightarrow update = no \rightarrow W = W$

i=3| $v = 1(0) + 0(0) + 1(0) = 0 \geq 0 \rightarrow y = 1 \neq d = 0 \rightarrow update = yes \rightarrow W = W + \Delta W$

↑ | $\Delta W_1 = 1(0 - 1)1 = -1; \Delta W_2 = 1(0 - 1)0 = 0; \Delta W_3 = 1(0 - 1)1 = -1]$

↑ | $\Delta W = [-1,0,1] \rightarrow W = [0,0,0] + [-1,0,1] = [-1,0,1]$

i=4| $v = 1(-1) + -1(0) + -1(1) = -2 < 0 \rightarrow y = 0 = d \rightarrow update = no \rightarrow W = W$

i=5| $v = 1(-1) + -1(0) + 0(1) = -1 < 0 \rightarrow y = 0 = d \rightarrow update = no \rightarrow W = W$

i=6| $v = 1(-1) + -1(0) + 1(1) = 0 \geq 0 \rightarrow y = 1 \neq d = 0 \rightarrow update = yes \rightarrow W = W + \Delta W$

↑ | $\Delta W_1 = 1(0 - 1)1 = -1, \Delta W_2 = 1(0 - 1)(-1) = 1, \Delta W_3 = 1(0 - 1)1 = -1$

↑ | $\Delta W = [-1,1,-1] \rightarrow W = [-1,0,1] + [-1,1,-1] = [-2,1,0]$

Iteration 2:

| $n_i$ | $X(x_0, x_1, x_2)$ | $d$ | $W(w_0, w_1, w_2)$ | $v$ | $y$ | update(y/n)? | $\Delta W(\Delta w_1, \Delta w_2, \Delta w_3)$ |
|---|---|---|---|---|---|---|---|
| 1 | [1,1,1] | 1 | $[-2,1,0]$ | -1 | 0 | y | [1,1,1] |
| 2 | [1,1,0] | 1 | [-1,2,1] | 1 | 1 | n | [0,0,0] |
| 3 | [1,0,1] | 0 | [-1,2,1] | 0 | 1 | y | [-1,0,-1] |
| 4 | [1,-1,-1] | 0 | [-2,2,0] | -4 | 0 | n | [0,0,0] |
| 5 | [1,-1,0] | 0 | [-2,2,0] | -4 | 0 | n | [0,0,0] |
| 6 | [1,-1,1] | 0 | [-2,2,0] | -4 | 0 | n | [0,0,0] |

i=1| $v = 1(-2) + 1(1) + 1(0) = -1 < 0 \rightarrow y = 0 \neq d = 1 \rightarrow update = yes \rightarrow W = W + \Delta W$

$\uparrow$ | $\Delta W_1 = 1(1-0)1, \Delta W_2 = 1(1-0)1, \Delta W_3 = 1(1-0)1 \rightarrow \Delta W = [1,1,1]$

$\uparrow$ | $W = [-2,1,0] + [1,1,1] = [-1,2,1]$

i=2| $v = -1(1) + 1(2) + 0(1) = 1 \geq 0 \rightarrow y = 1 = d \rightarrow update = no \rightarrow W = W$

i=3| $v = 1(-1) + 0(2) + 1(1) = 0 \geq 0 \rightarrow y = 1 \neq d = 0 \rightarrow update = no \rightarrow W = W + \Delta W$

$\uparrow$ | $\Delta W_1 = 1(0-1)1, \Delta W_2 = 1(0-1)0, \Delta W_3 = 1(0-1)1 \rightarrow \Delta W = [-1,0,-1]$

$\uparrow$ | $W = [-1,2,1] + [-1,0,-1] = [-2,2,0]$

i=4| $v = 1(-2) + -1(2) + -1(0) = -4 < 0 \rightarrow y = 0 = d \rightarrow W = W$

i=5| $v = 1(-2) + -1(2) + 0(0) = -4 < 0 \rightarrow y = 0 = d \rightarrow W = W$

i=6| $v = 1(-2) + -1(2) + 1(0) = -4 < 0 \rightarrow y = 0 = d \rightarrow W = W$

Iteration 3:

| $n_i$ | $X(x_0, x_1, x_2)$ | $d$ | $W(w_0, w_1, w_2)$ | $v$ | $y$ | update(y/n)? | $\Delta W(\Delta w_1, \Delta w_2, \Delta w_3)$ |
|---|---|---|---|---|---|---|---|
| 1 | [1,1,1] | 1 | $[-2,2,0]$ | 0 | 1 | n | [0,0,0] |
| 2 | [1,1,0] | 1 | $[-2,2,0]$ | 0 | 1 | n | [0,0,0] |
| 3 | [1,0,1] | 0 | $[-2,2,0]$ | -2 | 0 | n | [0,0,0] |
| 4 | [1,-1,-1] | 0 | $[-2,2,0]$ | -4 | 0 | n | [0,0,0] |
| 5 | [1,-1,0] | 0 | $[-2,2,0]$ | -4 | 0 | n | [0,0,0] |
| 6 | [1,-1,1] | 0 | $[-2,2,0]$ | -4 | 0 | n | [0,0,0] |

i=1| $v = 1(-2) + 1(2) + 1(0) = 0 \geq 0 \rightarrow y = 1 = d \rightarrow update = no \rightarrow W = W$

i=2| $v = 1(-2) + 1(2) + 0(0) = 0 \geq 0 \rightarrow y = 1 = d \rightarrow update = no \rightarrow W = W$

i=3| $v = 1(-2) + 0(2) + 1(0) = -2 < 0 \rightarrow y = 0 = d \rightarrow update = no \rightarrow W = W$

i=4| $v = 1(-2) + -1(2) + -1(0) = -4 < 0 \rightarrow y = 0 = d \rightarrow update = no \rightarrow W = W$

i=5| $v = 1(-2) + -1(2) + 0(0) = -4 < 0 \rightarrow y = 0 = d \rightarrow update = no \rightarrow W = W$

i=6| $v = 1(-2) + -1(2) + 1(0) = -4 < 0 \rightarrow y = 0 = d \rightarrow update = no \rightarrow W = W$ ✓

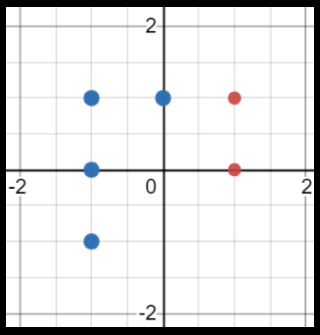
b) Answer: ↓

$x_0 \Rightarrow -2 \searrow$
$x_1 \Rightarrow 2 \Rightarrow \sum x_i w_i \Rightarrow Th \Rightarrow y = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$
$x_2 \Rightarrow 0 \nearrow$

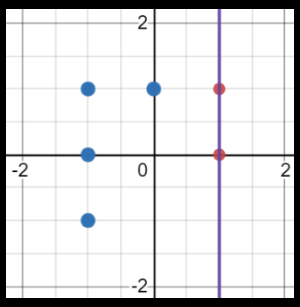
c) Answer: $0 = -2 + 2x_1 + 0x_2$

$classifier\ line = v = x_0 w_0 + x_1 w_1 + x_2 w_2 \rightarrow 0 = -2 + 2x_1 + 0x_2$

d) Answer: ↓ Red = labeled 1; Blue = labeled 0




e) Answer: ↓ Red = labeled 1; Blue = labeled 0




f) Answer: $x_i = inputs, d = desired\ label, v = local\ field, y = predicted\ label$

| $x_1$ | $x_2$ | $d$ | $v$ | y |
|---|---|---|---|---|
| 2 | 0 | 1 | 2 | 1 |
| 2 | 1 | 0 | 2 | 1 |
| 0 | 0 | 1 | -2 | 0 |
| -2 | 0 | 0 | -6 | 0 |

$v = -2 + 2x_1 + 0x_2$ • local field equation

$v_1 = -2 + 2(2) + 0(0) = 2 \geq 0 \rightarrow y = 1$

$v_2 = -2 + 2(2) + 0(1) = 2 \geq 0 \rightarrow y = 1$

$v_3 = -2 + 2(0) + 0(0) = -2 < 0 \rightarrow y = 0$

$v_4 = -2 + 2(-2) + 0(0) = -6 < 0 \rightarrow y = 0$

g) Answer: ↓

| Label | Class 1 | Class 0 |
|---|---|---|
| Class 1 | 1 | 1 |
| Class 0 | 1 | 1 |

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} = \frac{1+1}{1+1+1+1} = \frac{2}{4} = 50\%$$

Aaron P. Mills / Z-23547104 / Dr. Ghoraani

Intro to Deep Learning / CAP 4613 / Assignment 3 / 20 February 2022

https://colab.research.google.com/drive/1pNqUSkL_jt5qb6_6mBB42U9j8fGBpY3r?usp=sharing

```
# Aaron P. Mills /               Z-23547104 /
# Dr. Ghoraani
# Intro to Deep Learning  /       CAP 4613 /
# Assignment 3             /      20 February 2022
# Discription: replicate the perceptron learning that was done by hand in problem 1.
# https://colab.research.google.com/drive/1pNqUSkL_jt5qb6_6mBB42U9j8fGBpY3r?usp=sharing
#################################################################################################
#header block   - includes heading, imports, functions, classes, ect.
import math as mth
import numpy as np
import matplotlib.pyplot as plt

#a): create a nueral network with single neuron
class NeuralNetwork(object):
    #a)i. initializing the network
    def __init__(self,num_params=1):
        self.weight_matrix = 2 * np.random.random((num_params+1,1))-1
        #np.random.random = matrix of size (x,y) of random values between (0,1); 2*,-1 = t
        self.l_rate = 1     #learning rate = 1

    #a)ii. hard limiter as activation function for nx1 vector x
    def hard_limiter(self,x):
            outs = np.zeros(x.shape)    #zero matrix with same size as x
            outs[x>=0] = 1              #for all ind in x where entry >= 0, insert 1 into
            return outs                 #^TA says this should be >=0, NOT >0; return matri

    #a)iii. forward propogation, generats input(dot)weight, passes to hard limiter activat
    def forward_propagation(self,inputs):
        outs = np.dot(inputs, self.weight_matrix)   #dot product of inputs & weights = v
        return self.hard_limiter(outs)              #returning y, the predicted class

    #a)v. classifies inputs as 0 or 1 by multiplying by neuron weights, passing that to ac
    def pred(self,inputs):
        preds = self.forward_propagation(inputs)    #perform dot product, pass to activati
        return preds                                #return y^

    #a)iv. training that applies perceptron learning rule for num_train_iterations times
    def train(self, inputs, labels, num_train_iterations=10):
        for iteration in range(num_train_iterations):   #set number of iterations
            for i in range(inputs.shape[0]):            #for each iteration, go through ea
                #ΔW=n(d-y)x->ΔW=update value;n=learning rate;d=desired output;y=predicted
                pred_i = self.pred(inputs[i,:])         #generate predictions based on eac
                if pred_i != labels[i]:                 #if y != d (predicted != desired)
                    output = self.forward_propagation(inputs[i,:])  #get y (predicted outp
                    error = labels[i] - output                      #d - y
```

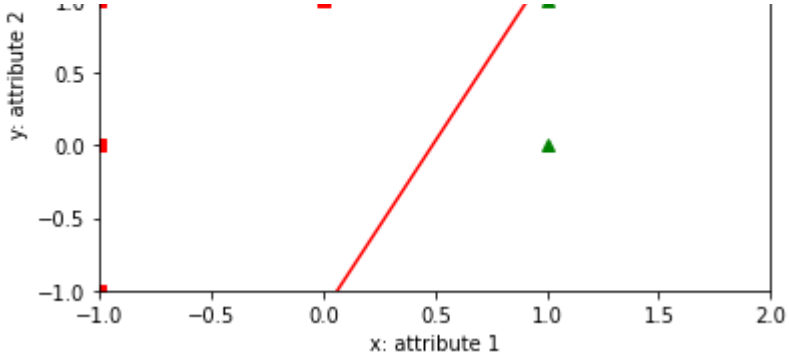
```
                error = labels[i] - output                      #d = y
                adjustment = self.l_rate*error*inputs[i]        #ΔW = n(d-y)x
                self.weight_matrix[:,0] += adjustment                #W = W+ΔW
                # print("Iteration #"+str(iteration))

#c+d) plot the points
def plotter(inputs,labels,  thre_parms,classes):
    #plotting data points
    plt.plot(inputs[labels[:,0]==classes[0],0], inputs[labels[:,0]==classes[0],1], 'rs',
             inputs[labels[:,0]==classes[1],0], inputs[labels[:,0]==classes[1],1],'g^')
    plt.axis([-1,2,-1,2])
    #d) plotting separate line
    x1 = np.linspace(-1,2,50)
    if (thre_parms[2]==0):                                  #recall c+ax+by=0, c=0 causes ille
        x2=0                                                #when c is 0, y = 0
        plt.axvline(x=-thre_parms[0]/thre_parms[1])         #thus the new equation of the line
    else:
        x2 = -(thre_parms[1]*x1+thre_parms[0])/thre_parms[2]    #otherwise y=-(ax+c)/b
        plt.plot(x1,x2,'-r')
    plt.xlabel('x: attribute 1')                            #labels
    plt.ylabel('y: attribute 2')                            #^
    plt.legend( ['Class '+str(classes[0]),'Class '+str(classes[1])] )   #legend
    plt.show()
```

```
#b) Use the perceptron learning rule to train a single neuron on the data points given in
#b)i. Create an np array of shape 6x2 containing inputs,& another with shape of 6x1 contai
inputs = np.array([[1,1],[1,0],[0,1],[-1,-1],[-1,0],[-1,1]])#attributes for each data
labels = np.array([[1],[1],[0],[0],[0],[0]])                #labels/classification of each
#b)ii. Add the bias to the input array to have a 6x3 shape
bias = np.ones((inputs.shape[0],1))                         #bias=matrix of rows=inputs, 1
inputs = np.append(bias,inputs,axis=1)                      #appending bias to inputs
#b)iii. Create network with one perceptron using class NeuralNetwork(), train it using tra
neural_network = NeuralNetwork(2)       #creating object(paramaters=attributes=2)
neural_network.train(inputs,labels,100) #train the one perceptron network

#c+d) Plot the given data points & classifier line with two different markers for each gro
weights = neural_network.weight_matrix              #weights...
classes = [0,1]                                     #classes...
plotter(inputs[:,1:3],labels,weights[:,0],classes)  #plotter...

#e) Use trained perceptron & classify the provided test data samples by calling the pred()
test_data = np.array([[2,0],[2,1],[0,0],[-2,0]])                    #this is the test data
test_bias = np.ones((test_data.shape[0],1))                         #bias=matrix of rows=i
test_inputs = np.append(test_bias,test_data,axis=1)                 #appending bias to inp
test_pred = neural_network.pred(test_inputs)                        #generate prediction
print(f"Predicted Output y for test_data: \n{test_pred}")
```




```
Predicted Output y for test_data:
[[1.]
 [1.]
 [0.]
 [0.]]
```

✓ 0s completed at 8:16 PM