Aaron P. Mills Z-23547104

Dr. Ghoraani CAP 4613 Intro to Deep Learning

Assignment 4 20 March 2022

https://colab.research.google.com/drive/1G_kz8Q5rskoiPo7hGVHSKZ2fgbHly6gP?usp=sharing

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

Problem 1)

a) Answer: ↓

$N = \#samples, \eta = learning\ rate, d = desired\ output, y = predicted\ output,$

$x = input\ (attribute), weight\ update\ value = \Delta W = \frac{\eta}{N}\sum_{n=1}^{N} x^n(d^n - y^n)$

$v = X \bullet W$ (dot product), $f(v) = y$=Adaline

$J(\theta) = \frac{1}{2N}\sum_{n=1}^{N}((d_n - y_n)^2)$ =loss function • definitions

Note: I only rounded for presentation purposes, I did NOT round during calculations.

Note2: I used Adaline activation function, for instructions did not specify

• for gradient descent learning, the weights are changed at the end of each epoch

Epoch #1

i	x_0	x_1	x_2	d	w_0	w_1	w_2	v	y	$X_n(d_n - y_n)$
1	1	1	1	1	1	1	1	3	3	[-2,-2,-2]
2	1	1	0	1	1	1	1	2	2	[-1,-1,0]
3	1	0	1	0	1	1	1	2	2	[-2,0,-2]
4	1	-1	-1	0	1	1	1	-1	-1	[1,-1,-1]
5	1	-1	0	0	1	1	1	0	0	[0,0,0]
6	1	-1	1	0	1	1	1	1	1	[-1,1,-1]

$$\Delta W = \frac{\eta}{N}\sum_{n=1}^{N} x^n(d^n - y^n) = \frac{0.1}{6}[-5,-3,-6] = \frac{1}{60}[-5,-3,-6] = \left[-\frac{5}{60}, -\frac{3}{60}, -\frac{6}{60}\right]$$

$$W = W + \Delta W = [1,1,1] + \left[-\frac{5}{60}, -\frac{3}{60}, -\frac{6}{60}\right] = \left[\frac{11}{12}, \frac{19}{20}, \frac{9}{10}\right]$$

$$\sum_{n=1}^{N}((d_n - y_n)^2) = 4 + 1 + 4 + 1 + 0 + 1 = 11$$

$$J(\theta) = \frac{1}{2N}\sum_{n=1}^{N}((d_n - y_n)^2) = \frac{1}{12}(11) = \frac{11}{12}$$

Epoch #2

i	x_0	x_1	x_2	d	w_0	w_1	w_2	v	y	$X_n(d_n - y_n)$
1	1	1	1	1	11/12	19/20	9/10	83/30	83/30	≈[-1.77,-1.77,-1.77]

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

2	1	1	0	1	11/12	19/20	9/10	28/15	28/15	≈[-0.87,-0.87,0]
3	1	0	1	0	11/12	19/20	9/10	109/60	109/60	≈[-1.82,0,-1.82]
4	1	-1	-1	0	11/12	19/20	9/10	-14/15	-14/15	≈[0.93,-0.93,-0.93]
5	1	-1	0	0	11/12	19/20	9/10	-1/30	-1/30	≈[0.03,-0.03,0]
6	1	-1	1	0	11/12	19/20	9/10	13/15	13/15	≈[-0.87,0.87,-0.87]

$\Delta W = \frac{1}{60}[-4.35, -2.73333, -5.38333] = [-0.0725, -0.04556, -0.08972]$

$W += \Delta W = \left[\frac{11}{12}, \frac{19}{20}, \frac{9}{10}\right] + [-0.0725, -0.04556, -0.08972] = \left[\frac{65}{77}, \frac{407}{450}, \frac{205}{253}\right]$

$\sum_{n=1}^{N}((d_n - y_n)^2) = 8.795833333$

$J(\theta) = \frac{1}{12}(8.795833333) \approx 0.732986111$

<u>Epoch #3</u>

i	x_0	x_1	x_2	d	w_0	w_1	w_2	v	y	$X_n(d_n - y_n)$
1	1	1	1	1	65/77	407/450	205/253	1369/535	1369/535	≈[-1.56,-1.56,-1.56]
2	1	1	0	1	65/77	407/450	205/253	313/179	313/179	≈[-0.75,-0.75,0]
3	1	0	1	0	65/77	407/450	205/253	541/327	541/327	≈[-1.65,0,-1.65]
4	1	-1	-1	0	65/77	407/450	205/253	-417/479	-417/479	≈[0.87,-0.87,-0.87]
5	1	-1	0	0	65/77	407/450	205/253	-46/763	-46/763	≈[0.06,-0.06,0]
6	1	-1	1	0	65/77	407/450	205/253	3/4	3/4	≈[-0.75,0.75,-0.75]

$\Delta W = \frac{1}{60}[-3.78104398, -2.488343058, -4.833862852]$

$= [-0.063018519, -0.041472222, -0.080564815]$

$W += \Delta W = \left[\frac{65}{77}, \frac{407}{450}, \frac{205}{253}\right] + [-0.0630174, -0.041472384, -0.080564381]$

$= \left[\frac{721}{923}, \frac{296}{343}, \frac{27}{37}\right] \approx [0.781148148, 0.862972222, 0.729712963]$

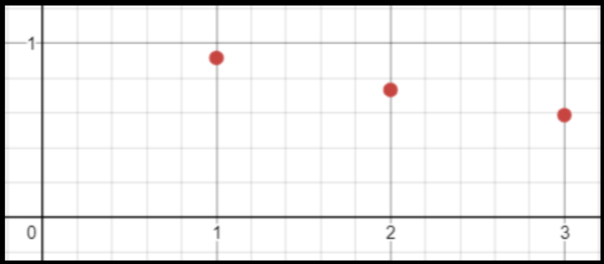
$\sum_{n=1}^{N}((d_n - y_n)^2) = 7.051739972$

$J(\theta) = \frac{1}{12}(7.051739972) \approx 0.587637293$

<u>a)</u> Answer: Yes, the neuron is learning because the loss function has a negative slope, implying error rate is decreasing for each epoch.

$J(\theta) \approx \left[\frac{11}{12}, 0.732986111, 0.587637293\right]$

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com



b) Answer: ↓

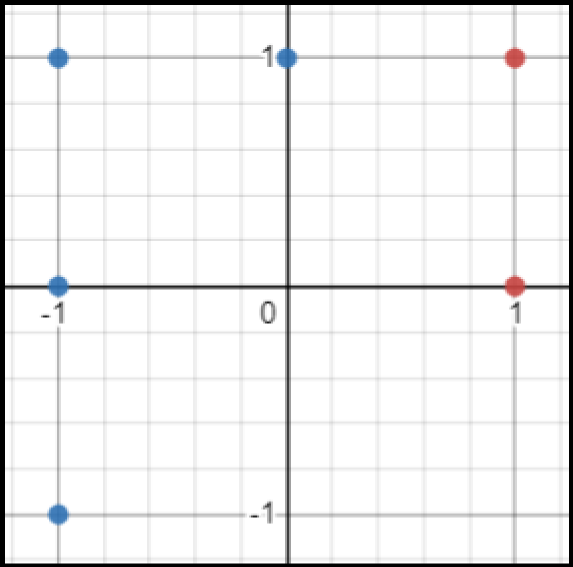
$$\begin{array}{l} x_0 \nearrow \frac{721}{923} \searrow \\ x_1 \Rightarrow \frac{296}{343} \Rightarrow \sum x_i w_i \Rightarrow g(v) \Rightarrow v = y \\ x_2 \searrow \frac{27}{37} \nearrow \end{array}$$

where $W = \left[\frac{439}{562}, \frac{296}{343}, \frac{27}{37}\right]$

c) Answer: $0 = \frac{721}{923} + \frac{296}{343}x_1 + \frac{27}{37}x_2$

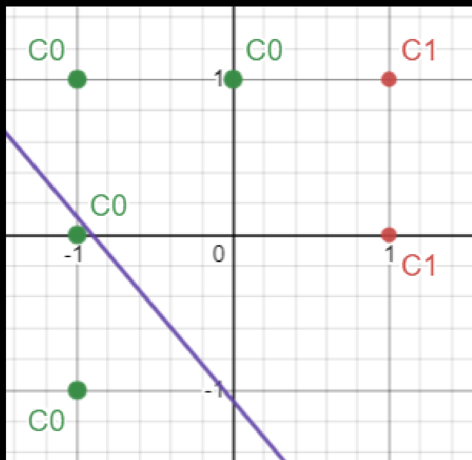
$$classifier\ line: v = x_0 w_0 + x_1 w_1 + x_2 w_2 \rightarrow 0 = \frac{721}{923} + \frac{296}{343}x_1 + \frac{27}{37}x_2$$

d) Answer: ↓



e) Answer: ↓

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com



f) Answer: ↓ ; Accuracy≈66.67%, Sensitivity=100%, Specificity=50%

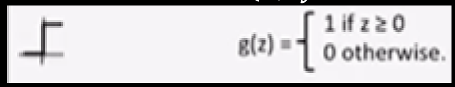
Label	Class 1	Class 0
Class 1	2	0
Class 0	2	2

$$Accuracy = \frac{\text{TP+TN}}{\text{TP+FP+FN+TN}} = \frac{2+2}{2+2+2} = \frac{4}{6} = \frac{2}{3} \approx 66.67\%$$

$$Sensitivity = \frac{TP}{TP+FN} = \frac{2}{2+0} = 100\%$$

$$Specificity = \frac{TN}{TN+FP} = \frac{2}{2+2} = 50\%$$

• we use thresholding to find $class = \begin{cases} 1, & y \geq 0 \\ 0, & y < 0 \end{cases}$, but NOT to be mistaken as unit step

activation function: 

Problem 3)

Note: all chart values are rounded to 4 decimal places

a) Answer: ↓

y_{h_1}	y_{h_2}	y_{h_3}	y_1	y_2
0.6225	0.7595	0.7595	0.7815	0.7540

$v_{hj} = \sum_{i=\{0,\ldots,\#\}}(w_{x_i h_j} x_i)$ • definitions

$y_{h_j} = f(v)$ where $f(v)$ is the activation function

α=1, $x_0 \to 1, x_1 = 1, x_2 = 0, w_{x_i h_j} = \{\ldots\}, f(v) = \frac{1}{1+e^{-v}}$ • given

$$v_{hj} = \sum_{i=\{0,\ldots,\#\}}(w_{x_i, h_j} x_i)$$

$$v_{h1} = (w_{x_0,h_1})x_0 + (w_{x_1,h_1})x_1 + (w_{x_2,h_1})x_2 = (0.18)1 + (0.32)1 + 0 = 0.50$$

$$v_{h2} = (w_{x_0,h_2})x_0 + (w_{x_1,h_2})x_1 + (w_{x_2,h_2})x_2 = (0.51)1 + (0.64)1 + 0 = 1.15$$

$$v_{h3} = (w_{x_0,h_3})x_0 + (w_{x_1,h_3})x_1 + (w_{x_2,h_3})x_2 = (0.43)1 + (0.72)1 + 0 = 1.15$$

$$v_{O1} = v(h_i, O1) = (w_{h_0,O1})y_{h_0} + (w_{h_1,O1})y_{h_1} + (w_{h_2,O1})y_{h_2} + (w_{h_3,O1})y_{h_3} = 1.274549786 \downarrow$$

$$= (0.53)1 + (0.22)0.622459331 + (0.19)0.759510917 + (0.61)0.759510917 = 1.274549786$$

$$v_{O2} = v(h_i, O2) = (w_{h_0,O2})y_{h_0} + (w_{h_1,O2})y_{h_1} + (w_{h_2,O2})y_{h_2} + (w_{h_3,O2})y_{h_3} = 1.119958476 \downarrow$$

$$= (0.61)1 + (0.38)0.622459331 + (0.21)0.759510917 + (0.15)0.759510917 = 1.119958476$$

$$y_{h1} = \frac{1}{1+e^{-0.5}} = 0.622459331 \,||\, \mathrm{y_{h2}} = \frac{1}{1+e^{-1.15}} = 0.759510917 \,||\, \mathrm{y_{h3}} = \frac{1}{1+e^{-1.15}} = 0.759510917$$

$$y_{O1} = y_1 = \frac{1}{1+e^{-1.274549786}} = 0.781520601 \,||\, \mathrm{y_{O2}} = y_2 = \frac{1}{1+e^{-1.119958476}} = 0.753981014$$

<u>b)</u> Answer: ↓

δ_{h_1}	δ_{h_2}	δ_{h_3}	δ_{O_1}	δ_{O_2}
-0.0106	-0.0041	0.0003	-0.0373	0.1399

$\delta_{O_k} = -y_k(1 - y_k)(d_k - y_k)$ • definitions

$\delta_{h_1} = -y_{h_i}(1 - y_{h_i}) \sum_{k\in\{1,\dots,\#)}(w_{h_i,O_k}\delta_{O_k})$

$y_{h1} = 0.622459331, \mathrm{y_{h2}} = 0.759510917, \mathrm{y_{h3}} = 0.759510917$ • from part a)

$\mathrm{y_1} = 0.781520601, y_2 = 0.753981014$

$d_1 = 1, d_2 = 0$ • given

$$\delta_{O_1} = -y_1(1 - y_1)(d_1 - y_1) = -0.781520601(1 - 0.781520601)(1 - 0.781520601)$$

$$= -0.037304516$$

$$\delta_{O_2} = -y_2(1 - y_2)(d_2 - y_2) = -0.753981014(1 - 0.753981014)(0 - 0.753981014)$$

$$= 0.139858686$$

$$h_{i=1} \to \sum_{k\in\{1,2)}(w_{h_1,O_k}\delta_{O_k}) = (w_{h_1,O_1})O_1 + (w_{h_1,O_2})O_2 = 0.044939307 \downarrow$$

$$(0.22)(-0.037304516) + (0.38)0.139858686 = 0.044939307$$

$$h_{i=2} \to \sum_{k\in\{1,2)}(w_{h_2,O_k}\delta_{O_k}) = (w_{h_2,O_1})O_1 + (w_{h_2,O_2})O_2 = 0.022282466 \downarrow$$

$$(0.19)(-0.037304516) + (0.21)0.139858686 = 0.022282466$$

$$h_{i=3} \to \sum_{k\in\{1,2)}(w_{h_3,O_k}\delta_{O_k}) = (w_{h_3,O_1})O_1 + (w_{h_3,O_2})O_2 = -0.001776952 \downarrow$$

$(0.61)(-0.037304516) + (0.15)0.139858686 = -0.001776952$

$\delta_{h_1} = -y_{h_1}(1 - y_{h_1}) \sum_{k\in\{1,2)} (w_{h_1,O_k}\delta_{O_k}) = -0.622459331(1 - 0.622459331)0.044939307$

$\delta_{h_2} = -y_{h_2}(1 - y_{h_2}) \sum_{k\in\{1,2)} (w_{h_2,O_k}\delta_{O_k}) = -0.759510917(1 - 0.759510917)0.022282466$

$\delta_{h_3} = -y_{h_3}(1 - y_{h_3}) \sum_{k\in\{1,2)} (w_{h_3,O_k}\delta_{O_k}) = -0.759510917(1 - 0.759510917)(-0.001776952)$

$\delta_{h_1} = -0.010560904,\ \delta_{h_2} = -0.004069983,\ \delta_{h_3} = 0.000324568$

c) Answer: ↓

$\Delta w_{x_0,h_1}$	$\Delta w_{x_0,h_2}$	$\Delta w_{x_0,h_3}$	$\Delta w_{h_0,O_1}$	$\Delta w_{h_0,O_2}$
0.0053	0.0020	-0.0002	0.0187	-0.0699
w_{x_0,h_1}	w_{x_0,h_2}	w_{x_0,h_3}	w_{x_0,O_1}	w_{x_0,O_2}
0.1853	0.5120	0.4298	0.5487	0.5401

$\Delta w_{i,j} = -\eta\delta_j x_{i,j}, w_{i,j}^{new} = w_{i,j}^{old} + \Delta w_{i,j}$ • definitions

$\eta = 0.5, w_{i,j}^{old} = \{\dots\}$ • given

$\Delta w_{x_0,h_1} = -0.5(\delta_{h_1})x_0 = -0.5(-0.010560904)1 = 0.005280452$

$\Delta w_{x_0,h_2} = -0.5(\delta_{h_2})x_0 = -0.5(-0.004069983)1 = 0.002034992$

$\Delta w_{x_0,h_3} = -0.5(\delta_{h_3})x_0 = -0.5(0.000324568)1 = -0.000162284$

$\Delta w_{h_0,O_1} = -0.5(\delta_{O_1})h_0 = -0.5(-0.037304516)1 = 0.018652258$

$\Delta w_{h_0,O_2} = -0.5(\delta_{O_2})h_0 = -0.5(0.139858686)1 = -0.069929343$

$w_{x_0,h_1}^{new} = 0.18 + 0.005280452 = 0.185280452$

$w_{x_0,h_2}^{new} = 0.51 + 0.002034992 = 0.512034992$

$w_{x_0,h_3}^{new} = 0.43 + -0.000162284 = 0.429837716$

$w_{h_0,O_1}^{new} = 0.53 + 0.018652258 = 0.548652258$

$w_{h_0,O_2}^{new} = 0.61 + -0.069929343 = 0.540070657$

Aaron P. Mills / Z-23547104 / Dr. Ghoraani Intro to Deep Learning / CAP 4613 / Assignment 4 / 20 March 2022 https://colab.research.google.com/drive/1G_kz8Q5rskoiPo7hGVHSKZ2fgbHly6gP?usp=sharing

Note: I placed my hand-written answers here because Colab would otherwise place it in the middle of the graphs, rendering them illegible. h) answer the question: I have observed that the learning rate of η=1 is way too large for this scenario, as it does not cause a decreasing learning curve. η=0.5 and η=0.05 are more appropriate. I believe this to be the case because those rates allow for the learning costs to decline. If I had to chose which individual learning rate was best or most suitable, which I do, I would chose η=0.05, for it generates the most "steady" curve: decreasing rapidly in the beginning, and slowing down near the end of the training.

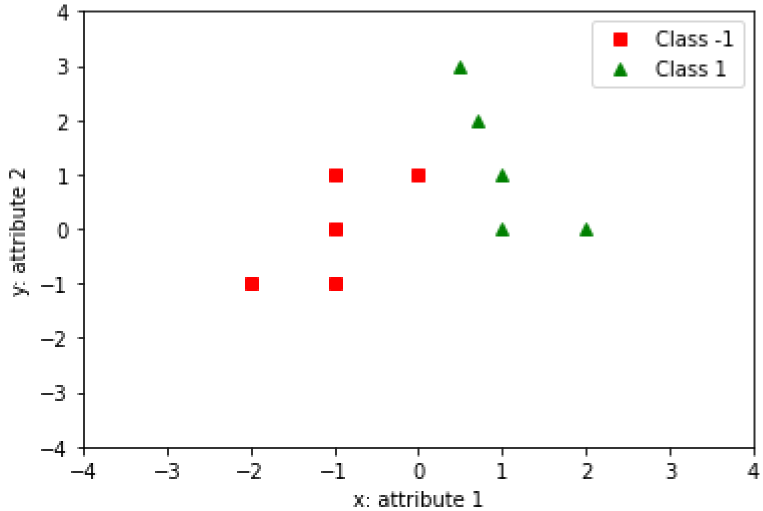
```
# Aaron P. Mills /                Z-23547104 /
# Dr. Ghoraani
# Intro to Deep Learning  /       CAP 4613 /
# Assignment 4             /      20 March 2022
# Discription:
# https://colab.research.google.com/drive/1G_kz8Q5rskoiPo7hGVHSKZ2fgbHly6gP?usp=sharing
###########################################################################################
#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,learning_r):
        #3x1 random matrix
        self.weight_matrix = 2 * np.random.random((3,1))-1
        # np.random.random = matrix of size (x,y) of random values between (0,1); 2*,-1 =
        self.l_rate = learning_r                           #learning rate = 1
        self.history_weights = np.array([ [self.weight_matrix] ])              #history v
        self.history_costs   = np.array([[0]])                                 #history v

    #{adaline} activation_function; note exercise did not specify
    def activation_function(self,x):
        return x                                   #θ(v)=v=y

    #a)ii. forward propogation: generats input(dot)weight, returns product after applying
    #note: I assume the activation is Adaline, θ(v)=v=y, since the exercise did NOT specif
    def forward_propagation(self, inputs):
        outs = np.dot(inputs, self.weight_matrix)   #dot product of inputs & weights = v
        return self.activation_function(outs)       #returning y, the predicted class

```

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

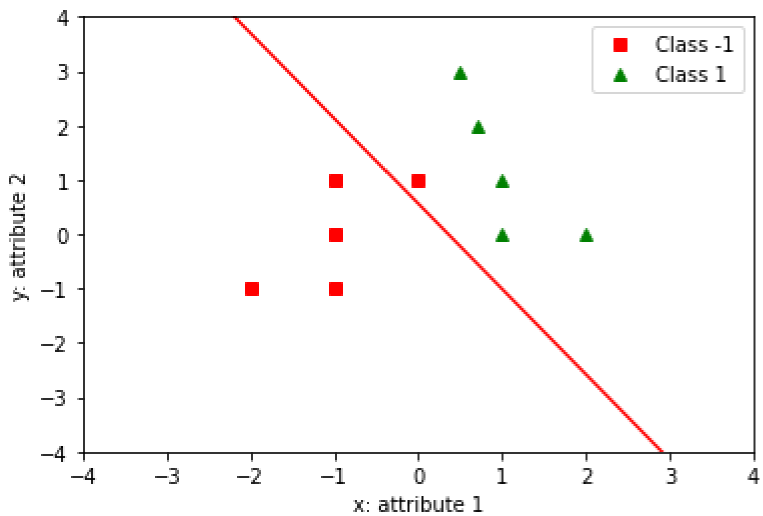
```
    #c)plot given data points
    def plotter(self,inputs,labels,thre_parms=0,classes=[-1,1],liner='yes'):
        #plotting data points
        plt.plot(inputs[labels[:,0]==classes[0],0], inputs[labels[:,0]==classes[0],1], 'rs
                inputs[labels[:,0]==classes[1],0], inputs[labels[:,0]==classes[1],1],'g^')
        plt.axis([-4,4,-4,4])
        #d) plotting separate line
        if liner=='yes':
            x1 = np.linspace(-5,5,50)                           #line goes from x=-5 to x=
            if (thre_parms[2]==0):                              #recall c+ax+by=0, c=0 cau
                x2=0                                            #when c is 0, y = 0
                plt.axvline(x=-thre_parms[0]/thre_parms[1])     #thus the new equation of
            else:
                x2 = -(thre_parms[1]*x1+thre_parms[0])/thre_parms[2]    #otherwise y=-(ax+
                plt.plot(x1,x2,'-r')                                    #display line
        plt.xlabel('x: attribute 1')                                #labels
        plt.ylabel('y: attribute 2')                                #^
        plt.legend( ['Class '+str(classes[0]),'Class '+str(classes[1])] )   #legend
        plt.show()

    #f) plot the costs of every epoch
    def plot_costs(self):
        print(f"Learning curve")
        #plotting data points
        for e in range(1,self.history_costs.shape[0]):
            plt.plot(e,self.history_costs[e,:], 'rs')
        plt.axis([0,self.history_costs.shape[0],0,self.history_costs.max()])
        plt.title('Learning Curve')
        plt.xlabel('x: epochs')                                 #labels
        plt.ylabel('y: cost')                                    #^
        plt.show()

    #g) repeat step b) with learning rates 1,0.5, 0.05; graph learning curves in subplot
    def subplot_costs(self,inputs_train, labels_train, num_train_epochs=10,grapher=0,learn
        plt.figure()
        fig, axes = plt.subplots(nrows=1, ncols=3)
        fig.tight_layout()                                #figure is the box that holds images
        #plotting data points
        for i in range(len(learning_rates)):
            plt.subplot(1,3,i+1)                #2 rows, 5 cols, i+1 position in grid matr
            self.weight_matrix = 2 * np.random.random((3,1))-1
            self.l_rate = learning_rates[i]
            self.history_weights = np.array([ [self.weight_matrix] ])                  #h
            self.history_costs   = np.array([[0]])                                     #h
            self.train(inputs_train, labels_train, num_train_epochs,grapher)
            plt.title(f'η={learning_rates[i]}')#use corresponding label in tittle name
            for e in range(1,self.history_costs.shape[0]):
                plt.plot(e,self.history_costs[e,:], 'rs')
            plt.axis([0,self.history_costs.shape[0],0,self.history_costs.max()])
            plt.xlabel('x: epochs')                                 #labels
            plt.ylabel('y: cost')                                    #^
```

```
        plt.show()

    #a)iii. train: trains the neural network
    def train(self, inputs_train, labels_train, num_train_epochs=10,grapher=1):
        N = inputs_train.shape[0]                                   #num of inputs
        classes = [-1,1]                                            #for plotting labels
        for epoch in range(num_train_epochs):
            if (grapher == 1) and (epoch == 0):
                print("Starting line")
                self.plotter(inputs_train[:,1:3],labels,self.weight_matrix[:,0],classes)
            # print(self.weight_matrix)
            outputs = self.forward_propagation(inputs_train)                         #t
            error   = labels_train - outputs                                         #d
            updater = (self.l_rate/N)*np.sum( np.multiply(error,inputs_train), axis=0 ) #Δ
            self.weight_matrix[:,0] += updater                                       #W
            learning_cost = (1/(2*N))*np.sum( error*error )                          #J
            self.history_weights = np.concatenate( (self.history_weights,[[self.weight_mat
            self.history_costs = np.concatenate((self.history_costs,[[learning_cost]]), ax
            if (grapher==1):
                #d) plot the classifier line for every 5 epochs
                #e) plot the final classifier line (the final "or" statement found below ↓
                if (epoch == 0) or ( (epoch+1)%5==0) or (epoch+1==num_train_epochs):
                    if (epoch+1 == num_train_epochs):
                        print("Final epoch")
                    else:
                        print(f"Epoch #{epoch+1}")
                    self.plotter(inputs_train[:,1:3],labels,self.weight_matrix[:,0],classe

    #display learning history
    def history(self):
        for x in range(self.history_costs.shape[0]):
            print(f"Epoch #{x}******************************************************")
            print(f"Weights: {self.history_weights[x,:]}")
            print(f"Cost: {self.history_costs[x,:]}")
```

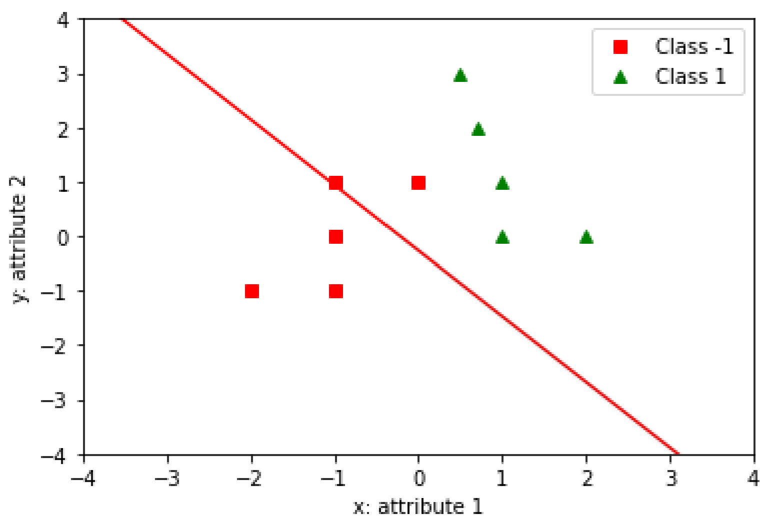
```
#b)i. create array for input & labels
inputs = np.array([[1,1],[1,0],[0,1],[-1,-1],[0.5,3],[0.7,2],[-1,0],[-1,1],[2,0],[-2,-1]])
labels = np.array([[1],[1],[-1],[-1],[1],[1],[-1],[-1],[1],[-1]])
#b)ii. create & append bias to input array
bias = np.ones((inputs.shape[0],1))                         #bias=matrix of rows=inputs, 1
inputs = np.append(bias,inputs,axis=1)                      #appending bias to inputs
#b)iii. create & train network
neural_network = NeuralNetwork(1)                               #b)iii. creating object(le
neural_network.plotter(inputs[:,1:3],labels,0,[-1,1],'no')      #c) plot the data points
neural_network.train(inputs,labels,50)                          #b)iii. train the one perc
neural_network.plot_costs()                                     #f): plot learning curve
neural_network.subplot_costs(inputs, labels,50,0,[1,0.5,0.05])  #g) repeat b with differen
```



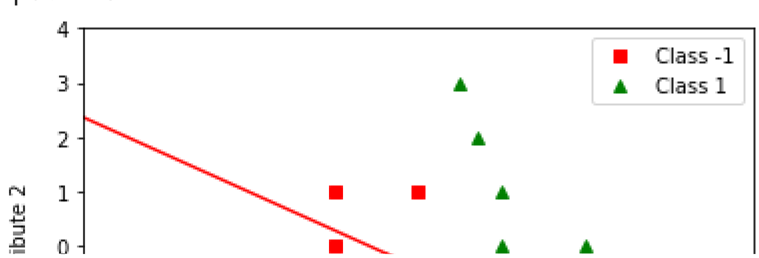
Starting line

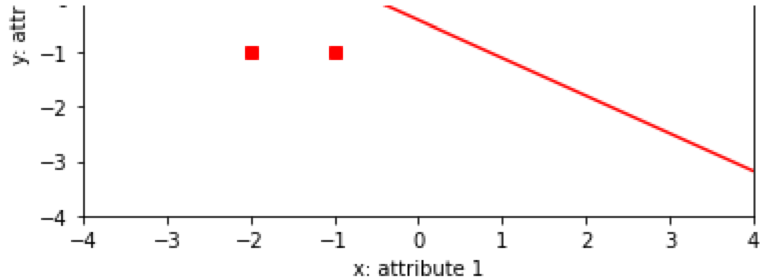


Epoch #1

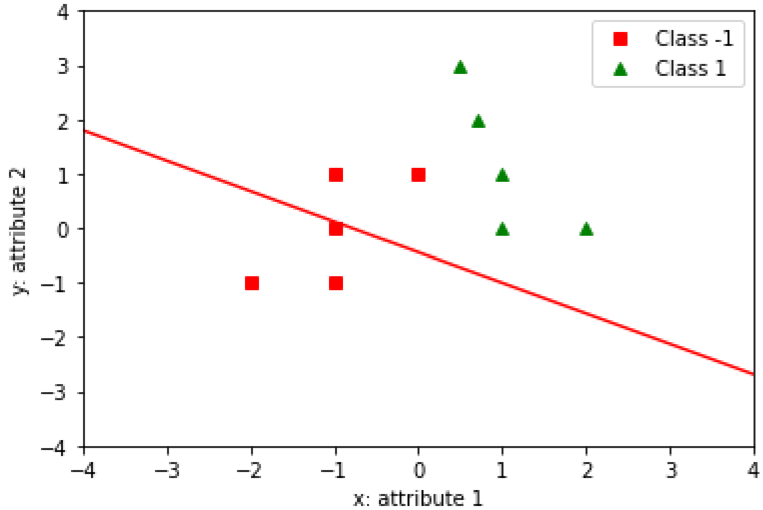


Epoch #5

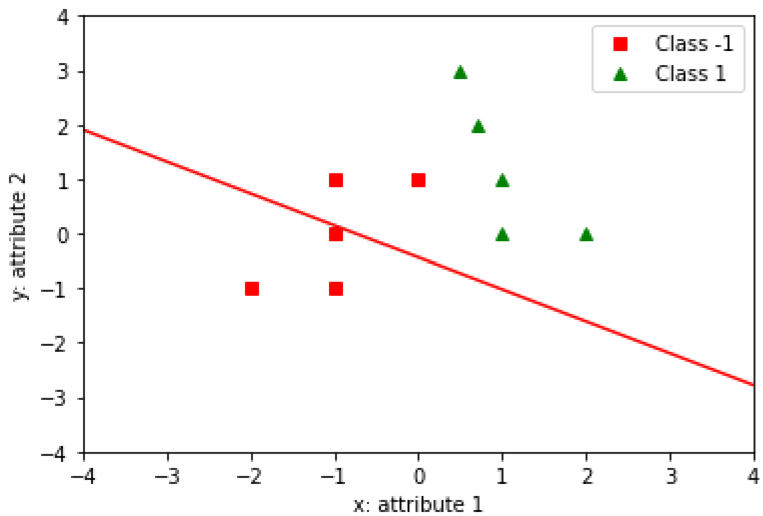




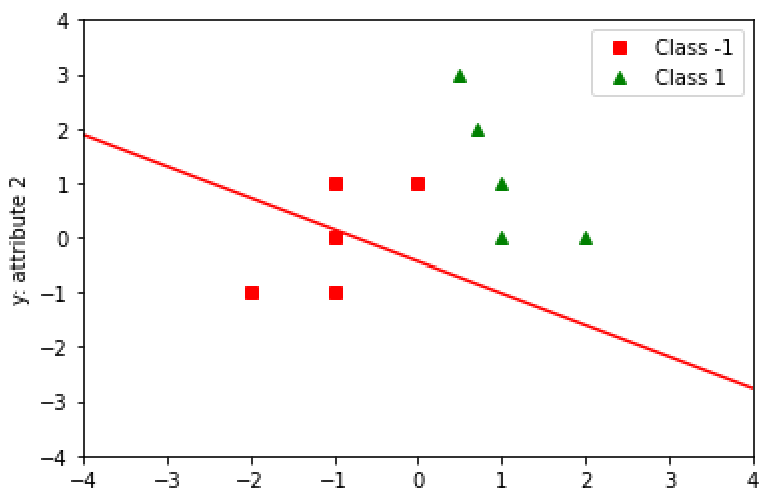
Epoch #10



Epoch #15



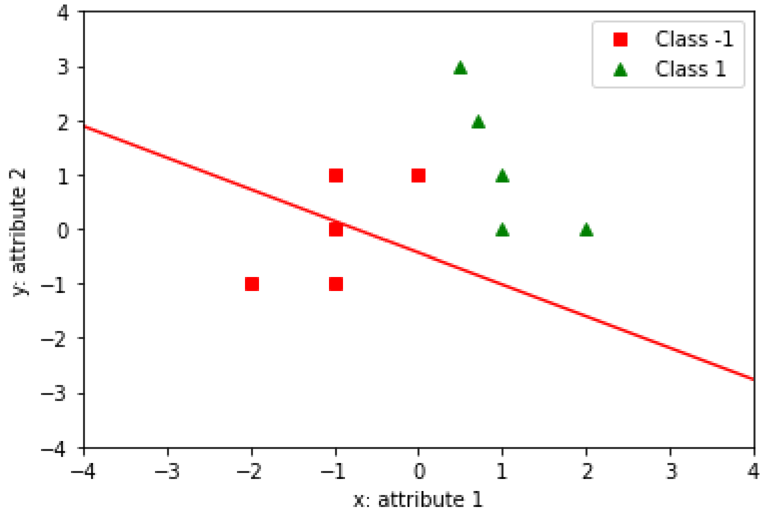
Epoch #20



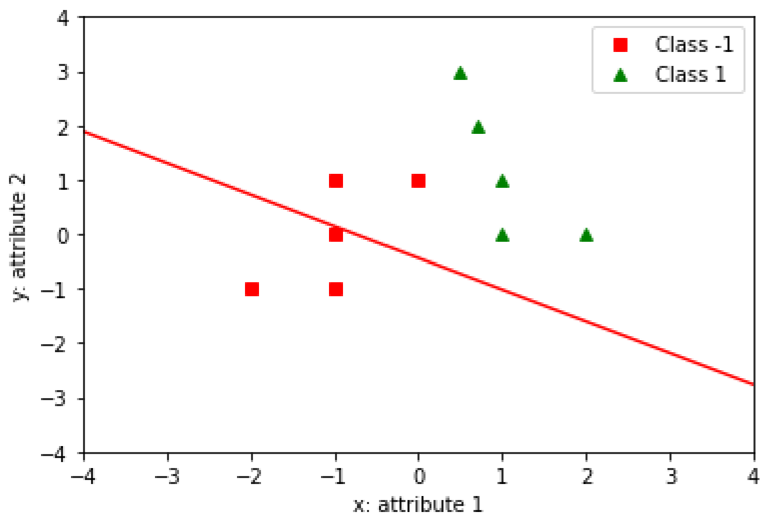
PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

x: attribute 1

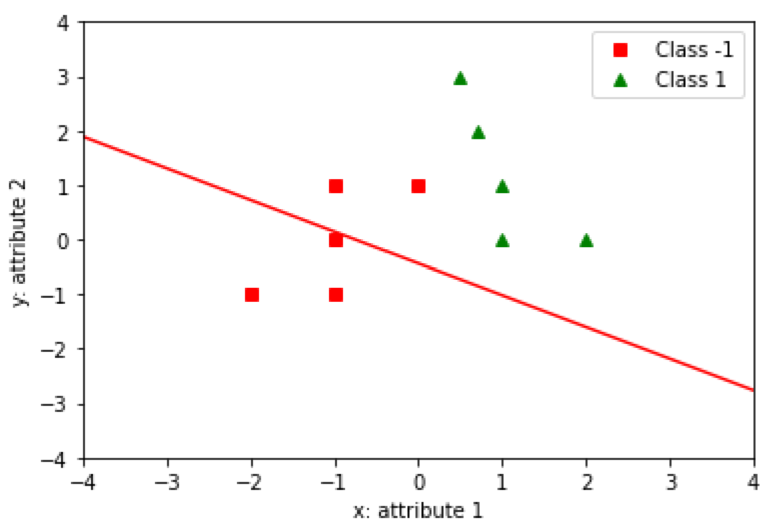
Epoch #25



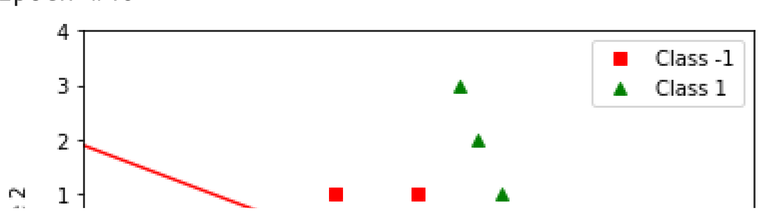
Epoch #30

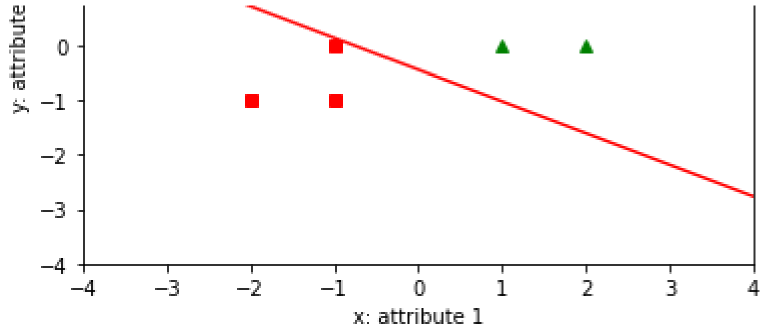


Epoch #35

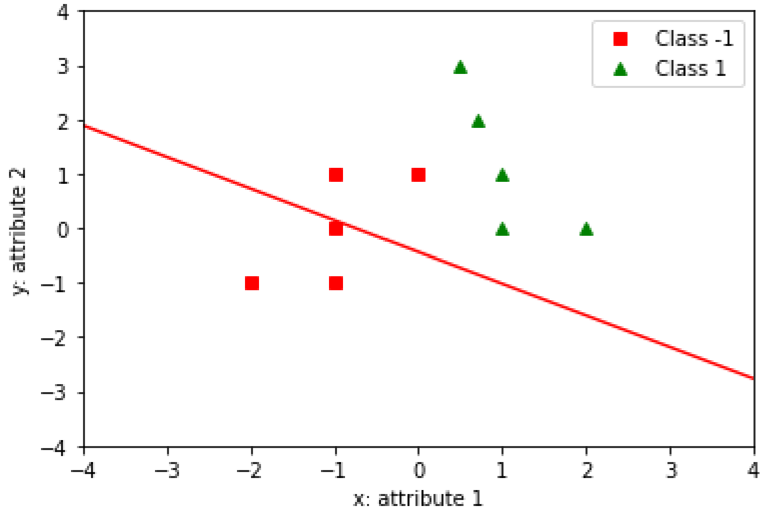


Epoch #40

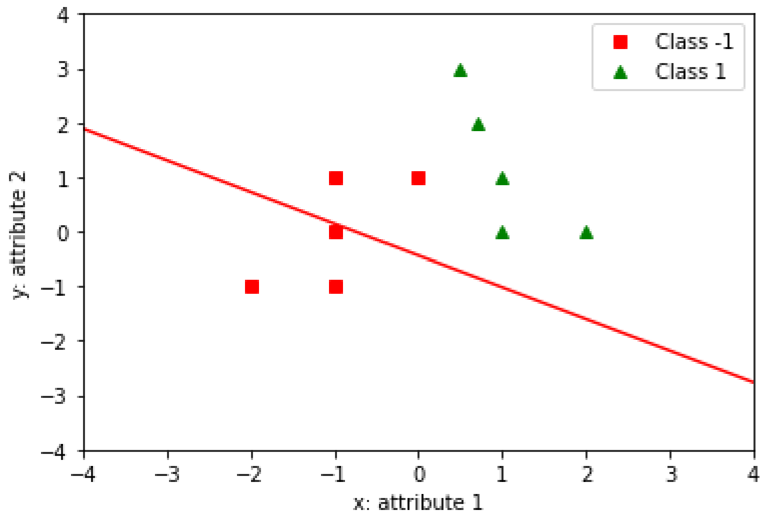




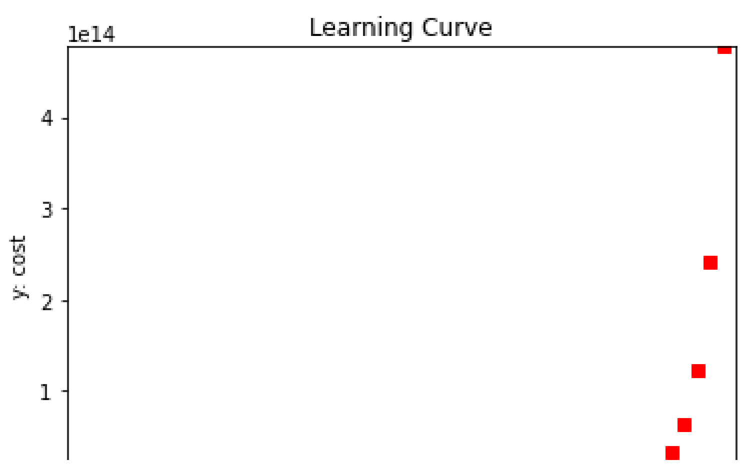
Epoch #45



Final epoch



Learning curve



PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

<Figure size 432x288 with 0 Axes>