Intro to deep learning
Assignment 4

Jordan Diaz

z23554771

Google Colab Link
↓

https://colab.research.google.com/drive/1vVOrMLS4uWTVmH49frcgmSp-m8a8dJW0?usp=sharing

Problem 1

Gradient descent learning: Consider the following data points.

input x_1	input x_2	desired label
1	1	1
1	0	1
0	1	0
-1	-1	0
-1	0	0
-1	1	0

as the above table shows, the data points are categorized (labeled) in two groups specified by the labels '1' and '0'.

a) use gradient descent learning algorithm to train a single neuron on the data samples given in the table above.
Repeat the gradient descent algorithm for only 3 epochs.
assume the learning rate is 0.1 with initial weight vector of (1 1 1)

a) calculate the loss function (i.e, avg square error) at each epoch.
Plot the learning curve. Is this neuron learning? Explain your answer.
Repeat these tasks using the classifier derived in the last epoch.

b) Draw the schematic of the trained neuron
c) Provide the equation of the trained model
d) Plot the given data points with two different markers for each group
e) Plot the classifier line in figure (d) label your plots.
f) Provide the confusion matrix. Calculate the accuracy, sensitivity, and specificity

assuming a linear activation function
learning rate $\eta = 0.1$, # of data points $N = 6$, $n = 1$

epoch 1

$x(x_0$	x_1	$x_2)$	d	$w(w_0$	w_1	$w_2)$	$v = w^T x$	$y = v$	$(x_0$	x_1	$x_2)(d-y)$
1	1	1	1	1	1	1	3	3	-2	-2	-2
1	1	0	1	1	1	1	2	2	-1	-1	0
1	0	1	0	1	1	1	2	2	-2	0	-2
1	-1	-1	0	1	1	1	-1	-1	1	-1	-1
1	-1	0	0	1	1	1	0	0	0	0	0
1	-1	1	0	1	1	1	1	1	-1	1	-1

$$\Delta W = \frac{\eta}{N}\sum_{n=1}^{N} x^n(d^n - y^n) = \frac{0.1}{6}\left[\begin{matrix}(-2 & -2 & -2) + (-1 & -1 & 0) \\ +(-2 & 0 & -2) + (1 & -1 & -1) + (0 & 0 & 0) + (-1 & 1 & -1)\end{matrix}\right]$$

$$= \frac{0.1}{6}\left[(-5 \quad -3 \quad -6)\right] = (-0.083 \quad -0.05 \quad -0.1)$$

updated weight $= W + \Delta W \Rightarrow (1\ 1\ 1) + (-0.083\ -0.05\ -0.1) = (0.917\ 0.95\ 0.9)$

$$J(\theta) = \frac{1}{2N}\sum_{n=1}^{N}(d^n - y^n)^2$$

$$= \frac{1}{12}\left[(1'-3')^2 + (1'-2')^2 + (0-2)^2 + (0+1)^2 + (0+0)^2 + (0+1)^2\right]$$

$$= \frac{4+1+4+1+0+1}{12} = 0.9167$$

assuming a linear activation function
learning rate $\eta = 0.1$, # of data points $N = 6$, $n = 1$

epoch 2

$x(x_0$	x_1	$x_2)$	d	$w(w_0$	w_1	$w_2)$	v	y	$(x_0$	x_1	$x_2)(d-y)$
1	1	1	1	0.917	0.95	0.9	2.767	2.767	-1.767	-1.767	-1.767
1	1	0	1	0.917	0.95	0.9	1.867	1.867	-0.867	-0.867	0
1	0	1	0	0.917	0.95	0.9	1.817	1.817	-1.817	0	-1.817
1	-1	-1	0	0.917	0.95	0.9	-0.933	-0.933	0.933	-0.933	-0.933
1	-1	0	0	0.917	0.95	0.9	-0.033	-0.033	0.033	-0.033	0
1	-1	1	0	0.917	0.95	0.9	0.867	0.867	-0.867	0.867	-0.867

$$\Delta w = \frac{\eta}{N}\sum_{n=1}^{N} x^n(d^n - y^n) = \frac{0.1}{6}(-4.346 \quad -2.733 \quad -5.317)$$

$$= (-0.0725 \quad -0.0456 \quad -0.0886)$$

updated weight = $w \times \Delta w \Rightarrow (0.8442 \quad 0.9044 \quad 0.8103)$

$$J(\theta) = \frac{1}{2N}\sum_{n=1}^{N}(d^n - y^n)^2 = \frac{1}{12}[(8.49612334)] = 0.7330$$

assuming a linear activation function
learning rate $\eta = 0.1$, # of data points $N = 6$, $n = 1$

epoch 3

$x(x_0$	x_1	$x_2)$	d	$w(w_0$	w_1	$w_2)$	v	y	$(x_0$	x_1	$x_2)(d-y)$
1	1	1	1	0.8442	0.9044	0.8103	2.5589	2.5589	-1.5589	-1.5589	-1.5589
1	1	0	1	0.8442	0.9044	0.8103	1.7486	1.7486	-0.7486	-0.7486	0
1	0	1	0	0.8442	0.9044	0.8103	1.6545	1.6545	-1.6545	0	1.6845
1	-1	-1	0	0.8442	0.9044	0.8103	-0.8705	-0.8705	0.8705	-0.8705	-0.8705
1	-1	0	0	0.8442	0.9044	0.8103	-0.0602	-0.0602	0.0602	-0.0602	0
1	-1	1	0	0.8442	0.9044	0.8103	0.7501	0.7501	-0.7501	0.7501	-0.7501

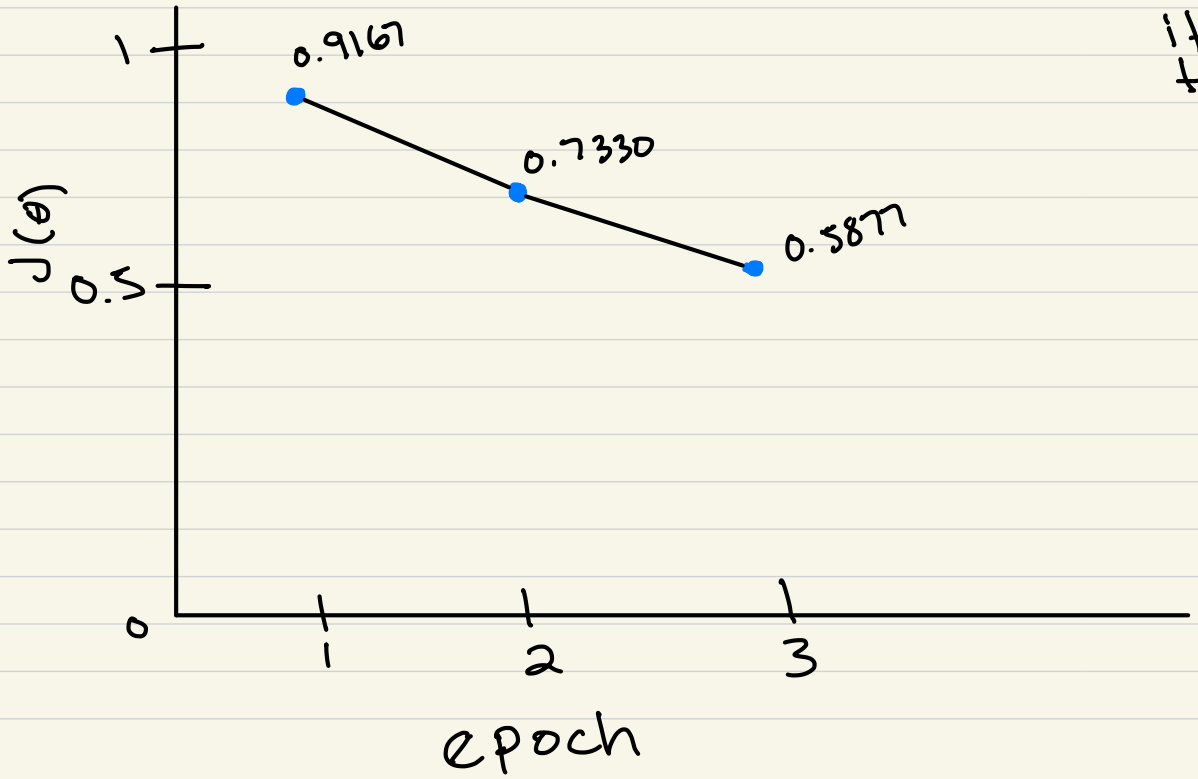
$$\Delta w = \frac{\eta}{N} \sum_{n=1}^{N} x^n (d^n - y^n) = \frac{0.1}{6} (-3.7814 \quad -2.4881 \quad -4.834)$$

$$= (-0.063 \quad -0.0415 \quad 0.0806)$$

updated weight = (0.7812 0.8629 0.7297)

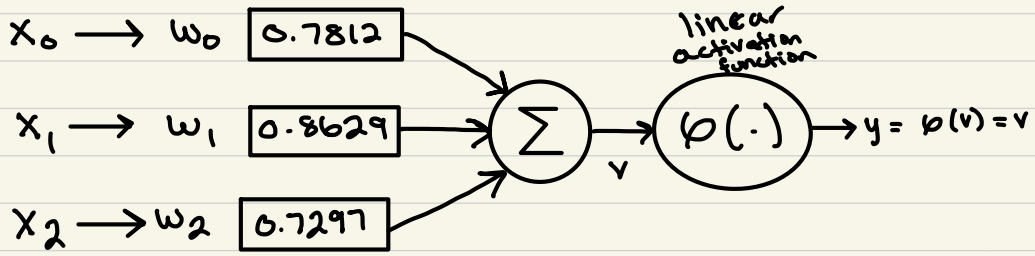
$$J(\theta) = \frac{1}{2N} \sum_{n=1}^{N} (d^n - y^n)^2 = \frac{1}{12} [(7.0519857)] = 0.5877$$

learning Curve



the neuron is learning, it is getting closer to a J(θ) value of 0

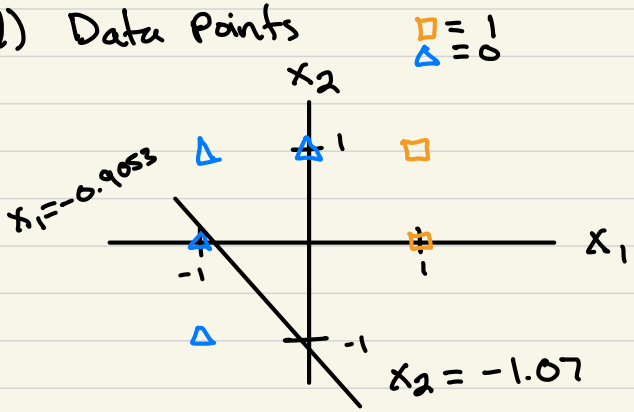
b) Schematic



c) Equation

$$X_0W_0 + X_1W_1 + X_2W_2 = 0$$
$$0.7812 + 0.8629X_1 + 0.7297X_2 = 0$$

d) Data Points



$0.7812 + 0.8629X_1 + 0.7297X_2 = 0$

$X_1 = 0$

$0.7812 + 0.7297X_2 = 0$

$X_2 = -1.07$

$X_2 = 0$

$0.7812 + 0.8629X_1 = 0$

$X_1 = -0.9053$

f) Matrix

Actual \ Predicted	1	0
1	2	2
0	0	2

TP	FN
FP	TN

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

$$\text{Sensitivity} = \frac{TP}{TP+FN} = \frac{2}{4} = 50.00\%$$

$$\text{Specificity} = \frac{TN}{TN+FP} = \frac{2}{2} = 100.00\%$$

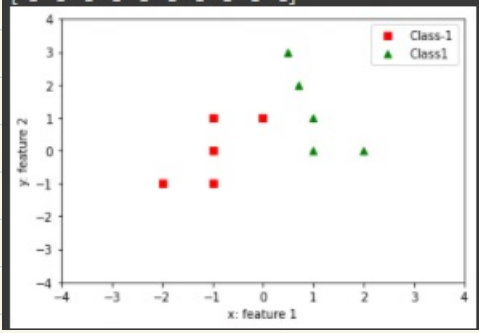
Problem 2

```
from copy import deepcopy
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
from random import randint

class NeuralNetwork(object):
  def __init__(self, learning_r = 1):
    """initializes a 3x1 weight vector randomly and initializes the learning rate to learning_r
    also, it creates a history variable that saves the weights and the training cost after each epoch (iteration)"""
    np.random.seed(1) # using seed to make sure it will generate same weights every run
    self.weight_matrix = 2 * np.random.random((3, 1)) - 1 # 3 x 1 matrix
    self.l_rate = learning_r # set the learning rate to whatever is entered in the function
    self.weight_history = []
    self.training_cost_history = []

  def linear(self, x):
    """linear activation function"""
    return x

  def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))

  def forward_propagation(self, inputs):
    """performs the forward propagation by multiplying the inputs by the neuron weights and then generating the output"""
    outs = np.dot(inputs, self.weight_matrix)
    return self.linear(outs)

  def train(self, inputs_train, labels_train, num_train_epochs=10):
    """performs the gradient descent learning rule for num_train_epochs times using the inputs and labels
       also saves the weights and costs at every epoch to the history variable"""

    N = inputs_train.shape[0]
    cost_func = np.array([])


    for epoch in range(num_train_epochs):
      self.weight_history.append(deepcopy(self.weight_matrix))
      outputs = self.forward_propagation(inputs_train)
      error = labels_train - outputs
      adjustment = (self.l_rate / N) * np.sum(np.multiply(error, inputs_train), axis = 0)

      cost_func = np.append(cost_func, (1/2 * N) * np.sum(np.power(error,2)))
      self.weight_matrix[:,0] += adjustment


      # print every 5 epochs
      if epoch == 0 or (epoch + 1) % 5 == 0:
        print('Iteration #' + str(epoch + 1))
        plot_fun_thr(inputs_train[:,1:3], labels_train[:,0], self.weight_matrix[:,0], classes)

    self.weight_history.append(deepcopy(self.weight_matrix))
    self.training_cost_history = cost_func
    plot_cost_func(cost_func, num_train_epochs)
```

```
  def pred(self, inputs):
    prob = self.forward_propagation(inputs)
    preds = np.int8(prob >= 0.5)
    return preds

def plot_cost_func(J, iterations):
  x = np.arange(iterations, dtype=int)
  y = J
  plt.plot(x,y)
  plt.axis([-1, x.shape[0] + 1, -1, np.max(y) + 1])
  plt.title('Learning Curve')
  plt.xlabel('x: iteration number')
  plt.ylabel('y: J(θ)')
  plt.show()


def plot_fun(features, labels, classes):
    plt.plot(features[labels[:] == classes[0],0], features[labels[:] == classes[0],1], 'rs', features[labels[:] == classes[1], 0], features[labels[:] == classes[1], 1], 'g^')
    plt.axis([-4, 4, -4, 4])
    plt.xlabel('x: feature 1')
    plt.ylabel('y: feature 2')
    plt.legend(['Class'+str(classes[0]), 'Class'+str(classes[1])])
    plt.show()


def plot_fun_thr(features, labels, thre_parms, classes):
    plt.plot(features[labels[:] == classes[0],0], features[labels[:] == classes[0],1], 'rs', features[labels[:] == classes[1], 0], features[labels[:] == classes[1], 1], 'g^')
    plt.axis([-4, 4, -4, 4])
    x1 = np.linspace(-1, 2, 50)
    x2 = -(thre_parms[1]*x1+thre_parms[0])/thre_parms[2]
    plt.plot(x1, x2, '-r')
    plt.xlabel('x: feature 1')
    plt.ylabel('y: feature 2')
    plt.legend(['Class'+str(classes[0]), 'Class'+str(classes[1])])
    plt.show()
```

```
# Create an np array of shape 10x2 that contains the inputs, and another array with a shape 10x1 that contains the labels
features = np.array([[1, 1],[1,0], [0,1], [-1,-1], [0.5, 3], [0.7, 2], [-1, 0], [-1, 1], [2, 0], [-2, -1]])
print(features)
labels = np.array([1, 1, -1, -1, 1, 1, -1, -1, 1, -1])
print(labels)
classes = [-1, 1]

plot_fun(features, labels, classes)
```

```
[[ 1.   1. ]
 [ 1.   0. ]
 [ 0.   1. ]
 [-1.  -1. ]
 [ 0.5  3. ]
 [ 0.7  2. ]
 [-1.   0. ]
 [-1.   1. ]
 [ 2.   0. ]
 [-2.  -1. ]]
[ 1  1 -1 -1  1  1 -1 -1  1 -1]
```



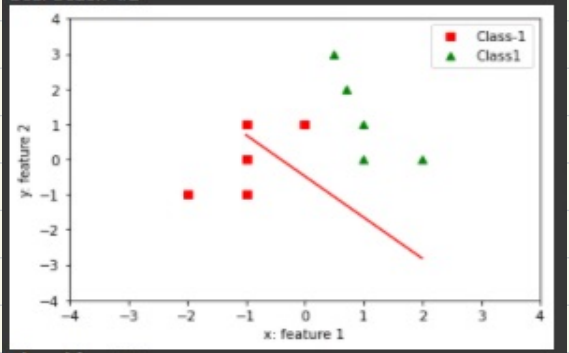
```
# Add the bias to the inputs array to have a 10x3 shape
bias = np.ones((features.shape[0], 1))

print(bias)
print(bias.shape)
features = np.append(bias, features, axis = 1)
print(features)
print(features.shape)
```

```
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]
(10, 1)
[[ 1.   1.   1. ]
 [ 1.   1.   0. ]
 [ 1.   0.   1. ]
 [ 1.  -1.  -1. ]
 [ 1.   0.5  3. ]
 [ 1.   0.7  2. ]
 [ 1.  -1.   0. ]
 [ 1.  -1.   1. ]
 [ 1.   2.   0. ]
 [ 1.  -2.  -1. ]]
(10, 3)
```

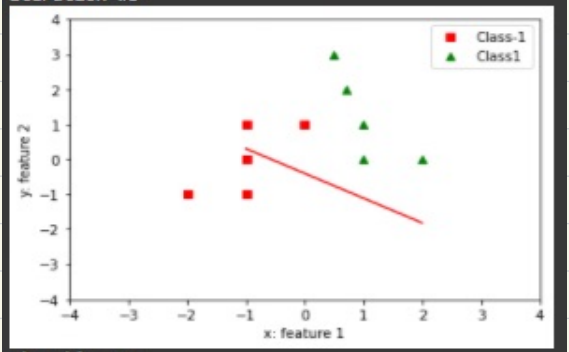
```
# Create the network with one neuron using the class NeuralNetwork() with learning rate of 1 then train it using(inputs, labels, 50) functions
# Learning Rate of 1
neural_network1 = NeuralNetwork(learning_r=1)
print('Random weights at the start of training')
print(neural_network1.weight_matrix)
neural_network1.train(features, np.expand_dims(labels, axis = 1), 50)

print('New weights after training')
print(neural_network1.weight_matrix)

# Test the neural network with the training data points
print("Testing network on training data points - >")
print(neural_network1.pred(features))

# Test the neural network with a new data point
print("Testing network on new examples - >")
print(neural_network1.pred(np.array([1, 1, 1])))
```

```
Random weights at the start of training
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
```

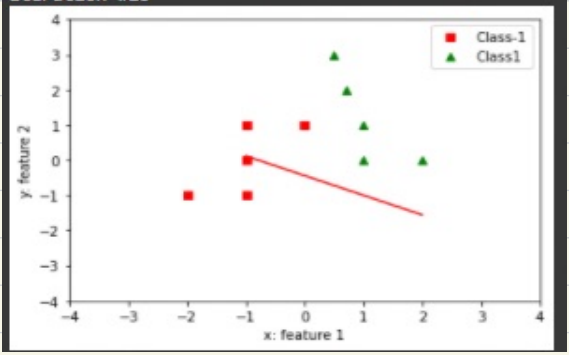
Iteration #1



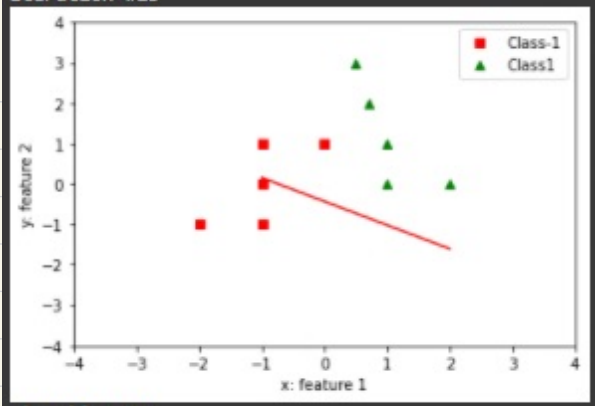
Iteration #5



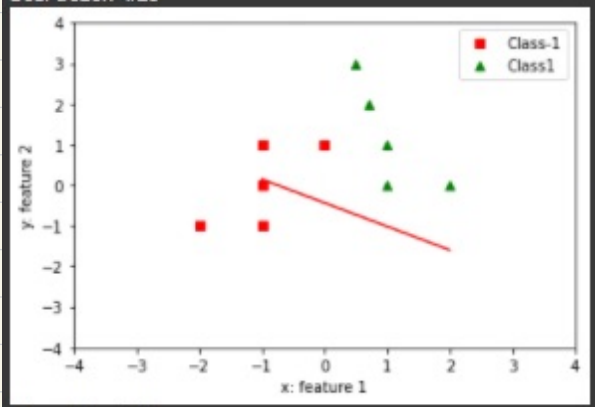
Iteration #10



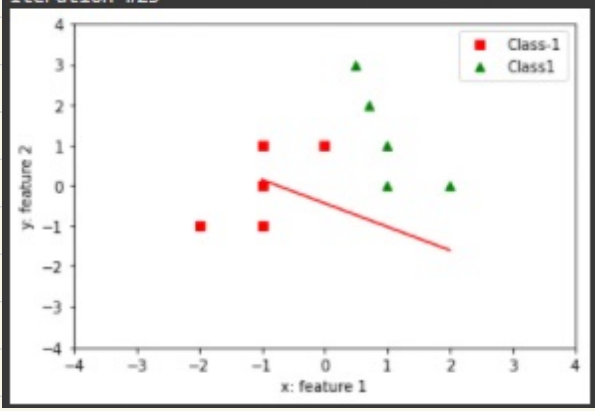
Iteration #15



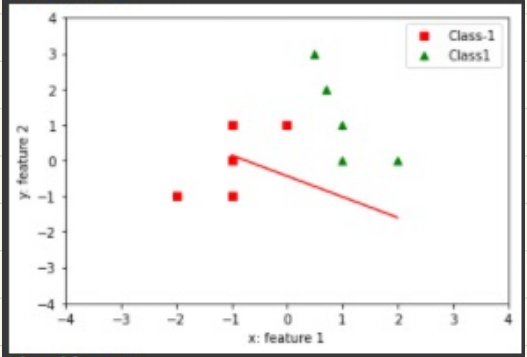
Iteration #20



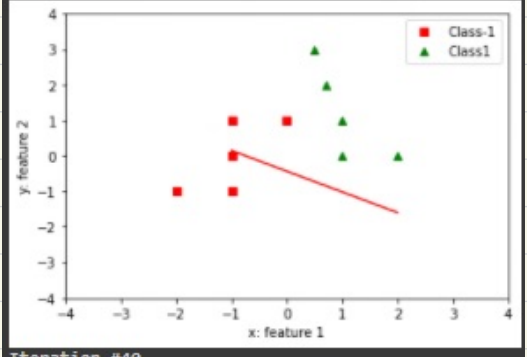
Iteration #25



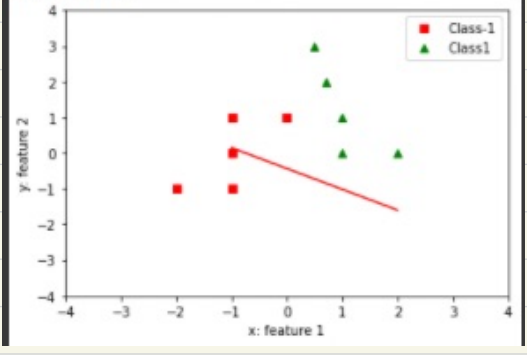
Iteration #30



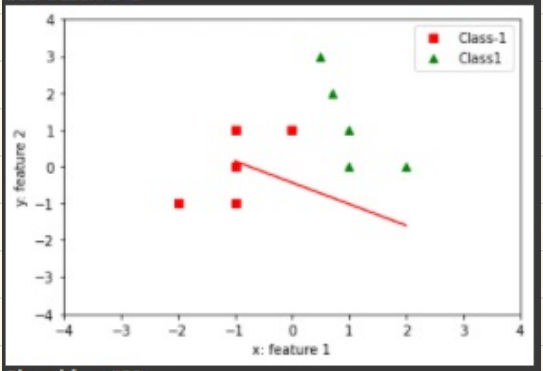
Iteration #35



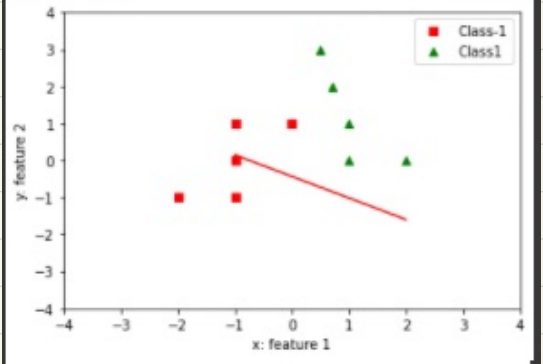
Iteration #40

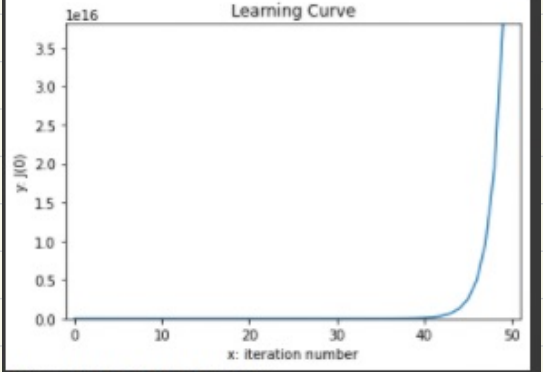


Iteration #45



Iteration #50



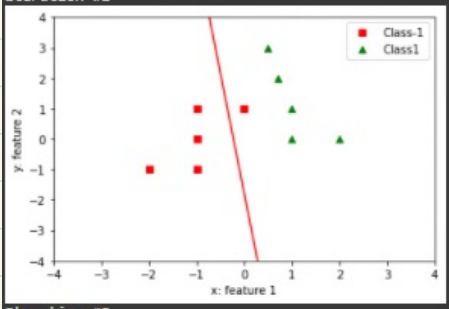



```
New weights after training
[[ -8805326.09888927]
 [-11748558.9750475 ]
 [-20216017.29976815]]
```

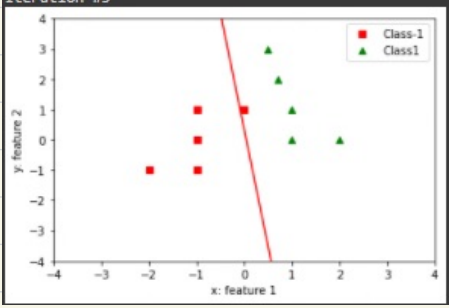
```python
# Learning Rate of 0.5
neural_network2 = NeuralNetwork(learning_r=0.5)
print('Random weights at the start of training')
print(neural_network2.weight_matrix)
neural_network2.train(features, np.expand_dims(labels, axis = 1), 50)

print('New weights after training')
print(neural_network2.weight_matrix)

# Test the neural network with the training data points
print("Testing network on training data points - >")
print(neural_network2.pred(features))

# Test the neural network with a new data point
print("Testing network on new examples - >")
print(neural_network2.pred(np.array([1, 1, 1])))
```

```
Random weights at the start of training
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
```

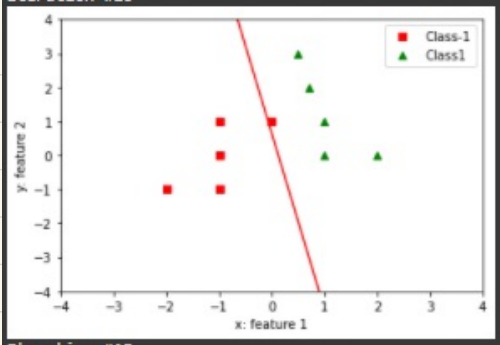
Iteration #1



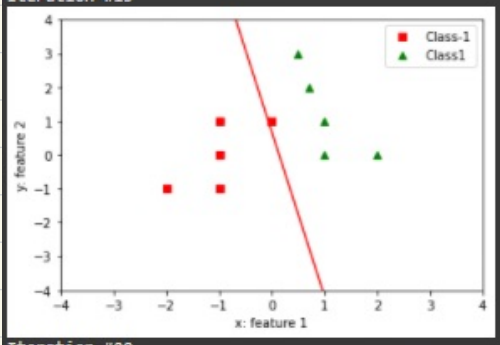
Iteration #5



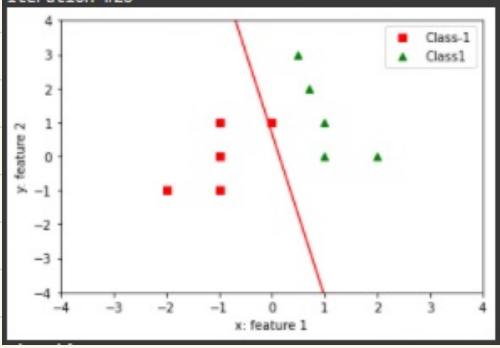
Iteration #10



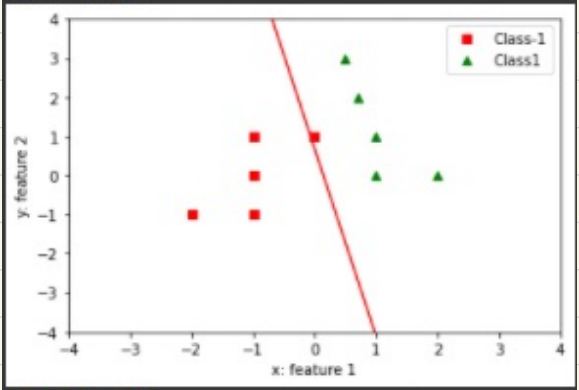
Iteration #15



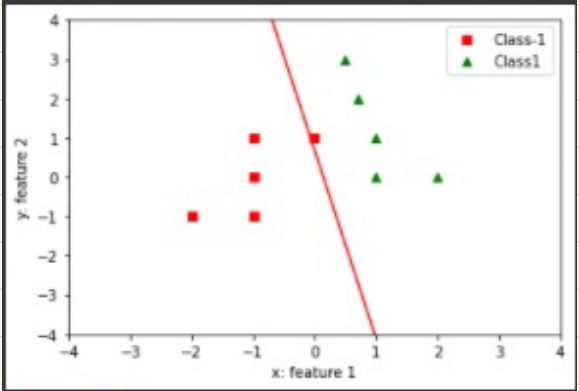
Iteration #20



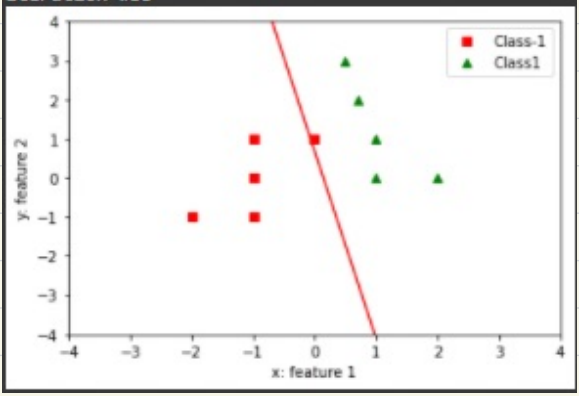
Iteration #25



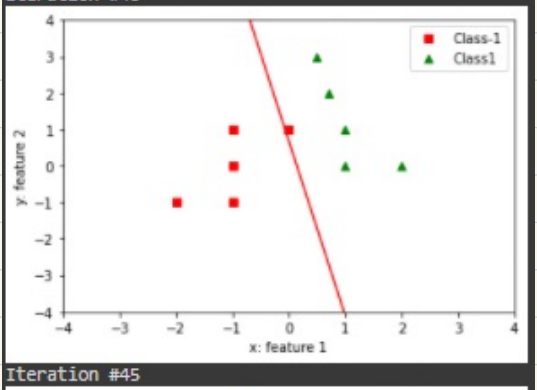
Iteration #30



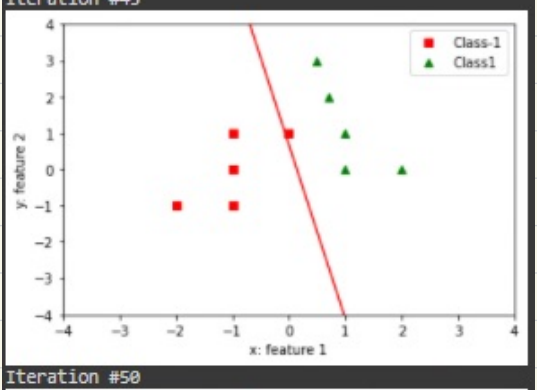
Iteration #35



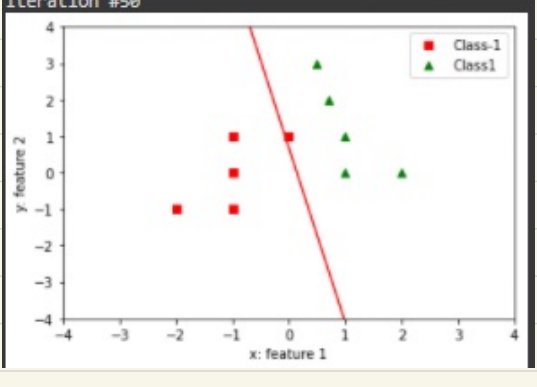
Iteration #40

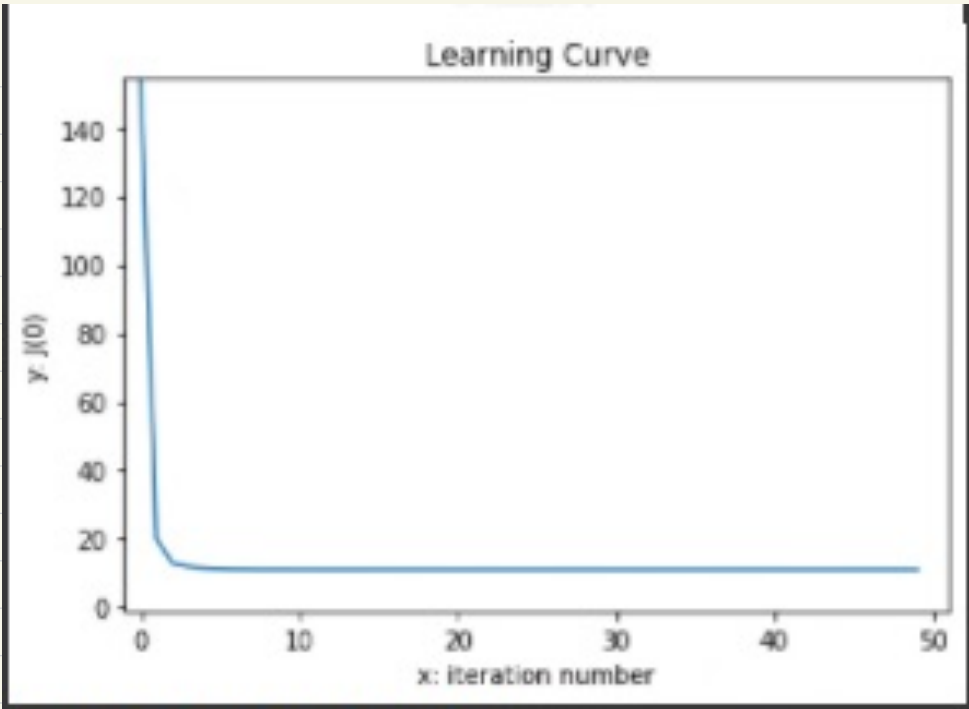


Iteration #45



Iteration #50

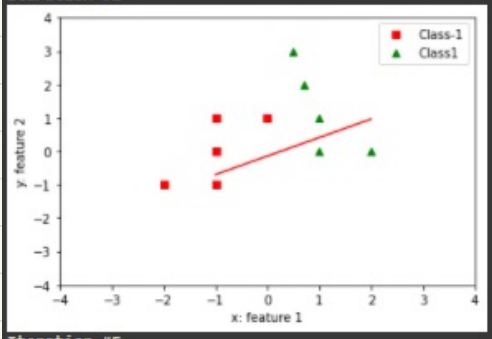




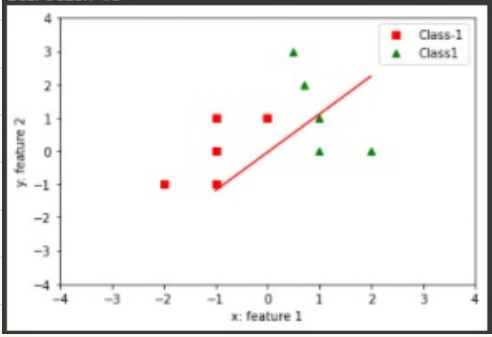

```
New weights after training
[[-0.09927609]
 [ 0.68252328]
 [ 0.1427094 ]]
Testing network on training data points - >
[[1]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]]
Testing network on new examples - >
[1]
```

```
# Learning Rate of 0.05
neural_network3 = NeuralNetwork(learning_r=0.05)
print('Random weights at the start of training')
print(neural_network3.weight_matrix)
neural_network3.train(features, np.expand_dims(labels, axis = 1), 50)

print('New weights after training')
print(neural_network3.weight_matrix)

# Test the neural network with the training data points
print("Testing network on training data points - >")
print(neural_network3.pred(features))

# Test the neural network with a new data point
print("Testing network on new examples - >")
print(neural_network3.pred(np.array([1, 1, 1])))
```

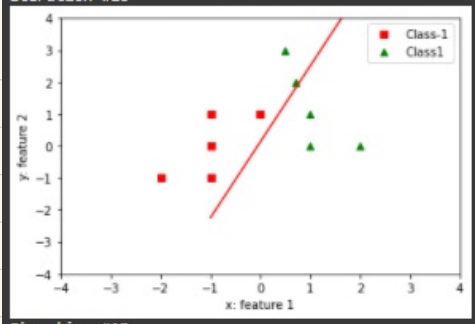
```
Random weights at the start of training
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Iteration #1
```



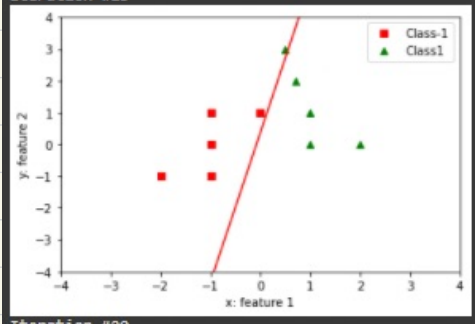
Iteration #5



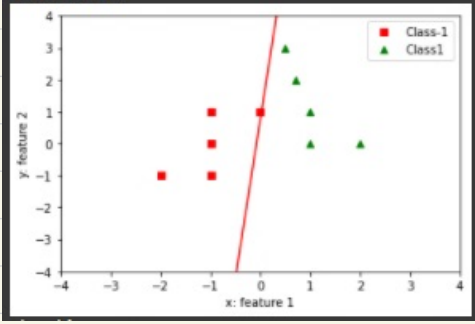
Iteration #10



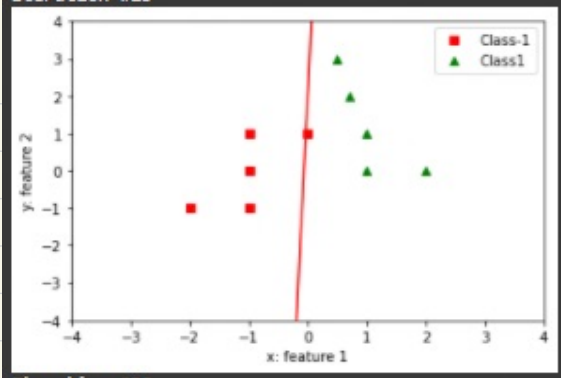
Iteration #15



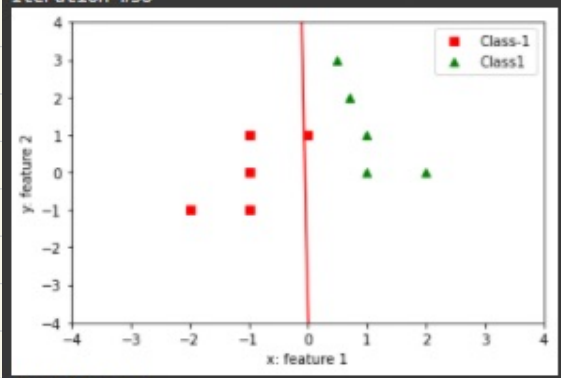
Iteration #20



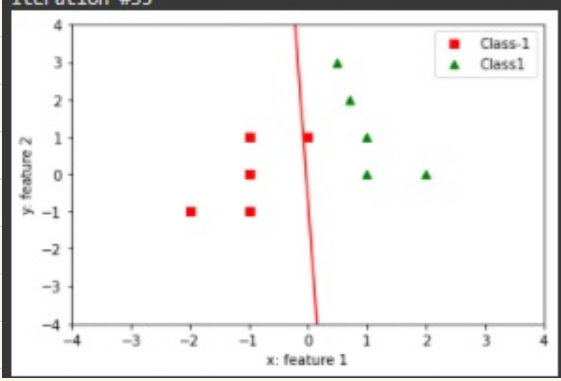
Iteration #25



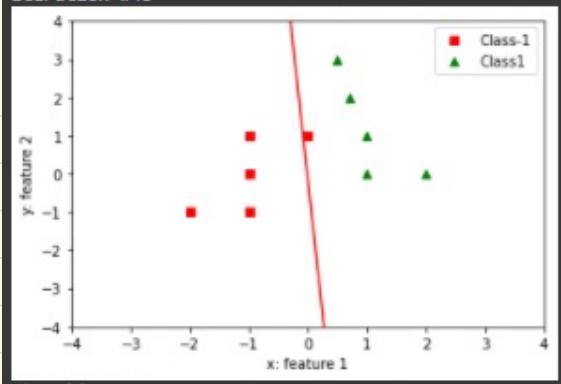
Iteration #30



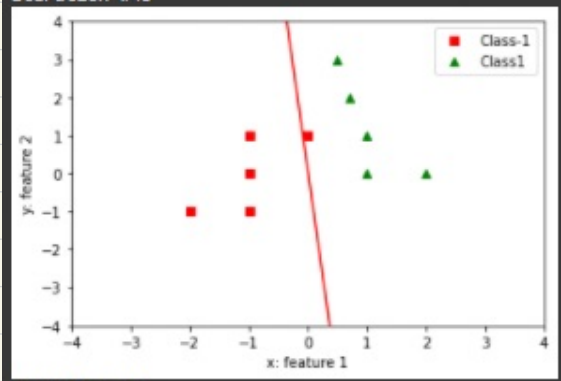
Iteration #35



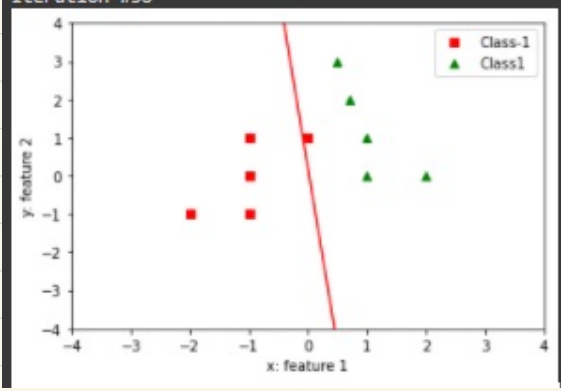
Iteration #40

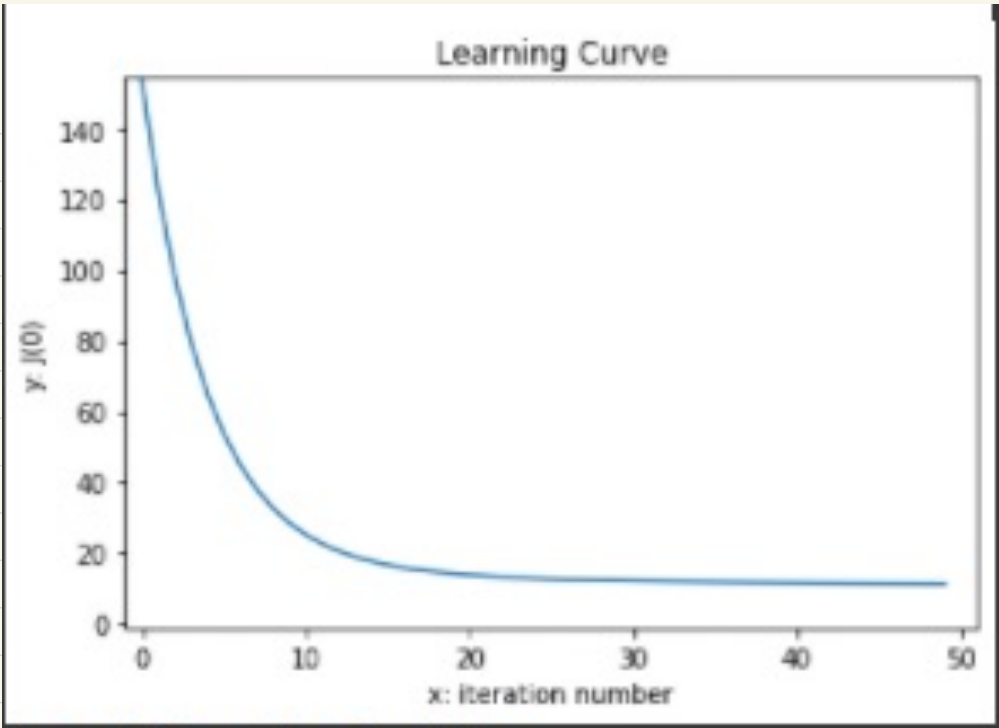


Iteration #45



Iteration #50

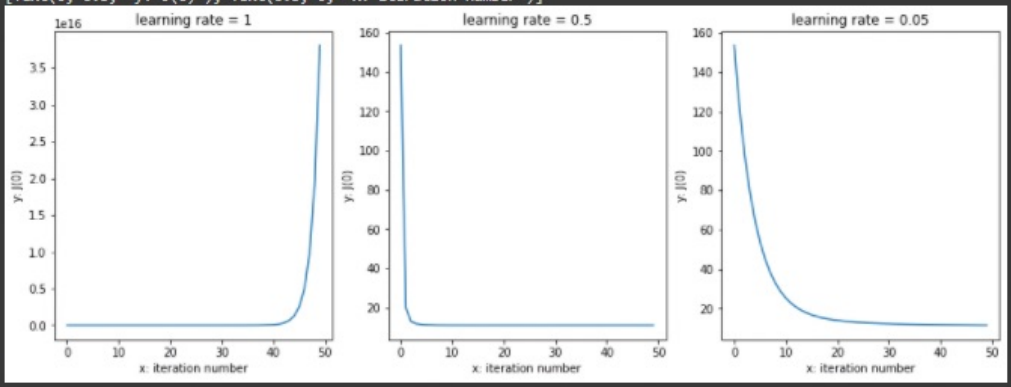





```
New weights after training
[[-0.01755766]
 [ 0.72850045]
 [ 0.07822411]]
Testing network on training data points - >
[[1]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]]
Testing network on new examples - >
[1]
```

```
[7] from IPython.core.pylabtools import figsize
    #Create a subplot with the learning curves of learning rates 1, 0.5, and 0.05. Add titles
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

    x = np.arange(50, dtype=int)

    ax1.plot(x, neural_network1.training_cost_history)
    ax1.set_title("learning rate = 1")
    ax1.set(xlabel = 'x: iteration number', ylabel='y: J(0)')

    ax2.plot(x, neural_network2.training_cost_history)
    ax2.set_title("learning rate = 0.5")
    ax2.set(xlabel = 'x: iteration number', ylabel='y: J(0)')

    ax3.plot(x, neural_network3.training_cost_history)
    ax3.set_title("learning rate = 0.05")
    ax3.set(xlabel = 'x: iteration number', ylabel='y: J(0)')
```

```
[Text(0, 0.5, 'y: J(0)'), Text(0.5, 0, 'x: iteration number')]
```



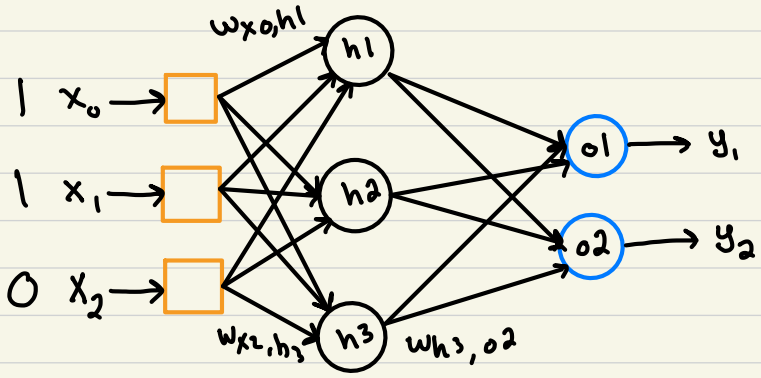
When the learning rate is 1, the weights will oscillate and never reach their goal.

when the learning rate is 0.05 it makes it does not oscillate but it takes a while to get close to 0.

Finally when the learning rate is 0.5 the cost function approaches 0 faster than the other two weights. 0.5 is the most suitable for this case.

Problem 3

Forward-backward propagation. Consider the following forward feed neural network:



Initial weights are given as below. Learn rate is 0.5. The Activation function is sigmoid function with a=1 for all neurons.

$w_{x0,h1} = 0.18$, $w_{x1,h1} = 0.32$, $w_{x2,h1} = 0.42$
$w_{x0,h2} = 0.51$, $w_{x1,h2} = 0.64$, $w_{x2,h2} = 0.12$
$w_{x0,h3} = 0.43$, $w_{x1,h3} = 0.72$, $w_{x2,h3} = 0.33$

$w_{h0,o1} = 0.53$, $w_{h1,o1} = 0.22$, $w_{h2,o1} = 0.19$, $w_{h3,o1} = 0.61$
$w_{h0,o2} = 0.61$, $w_{h1,o2} = 0.38$, $w_{h2,o2} = 0.21$, $w_{h3,o2} = 0.15$

for a training, the inputs of the features ($x_1 = 1$, $x_2 = 0$) and the label of class y_1, Perform the forward-backward steps as instructed below.

Hint: The data sample belongs to class y_1, this means that for this data sample, $y_1 = 1$ and $y_2 = 0$

a) Perform the forward propagation from the left to the right side of the network. Show the detailed calculations in each step. Report the computed outputs in the table below:

for h1

$$v = w_{x0,h1} \cdot x_0 + w_{x1,h1} \cdot x_1 + w_{x2,h1} \cdot x_2$$
$$= 0.18(1) + 0.32(1) + 0.42(0)$$
$$= 0.5$$

sigmoid function

$$y_{h1} = 1/1+e^{-v} = \frac{1}{1+e^{-0.5}} = 0.6225$$

for h2

$$v = w_{x0,h2} \cdot x_0 + w_{x1,h2} \cdot x_1 + w_{x2,h2} \cdot x_2$$
$$= 0.51(1) + 0.64(1) + 0.12(0)$$
$$= 1.15$$

sigmoid function

$$y_{h2} = 1/1+e^{-v} = \frac{1}{1+e^{-1.15}} = 0.7595$$

for h3

$$v = w_{x0,h3} \cdot x_0 + w_{x1,h3} \cdot x_1 + w_{x2,h3} \cdot x_2$$
$$= 0.43(1) + 0.72(1) + 0.33(0)$$
$$= 1.15$$

sigmoid function

$$y_{h3} = 1/1+e^{-v} = \frac{1}{1+e^{-1.15}} = 0.7595$$

for O1

$V = w_{h0,o1} \cdot y_{h0} + w_{h1,o1} \cdot y_{h1} + w_{h2,o1} \cdot y_{h2} + w_{h3,o1} \cdot y_{h3}$

$= 0.53(1) + 0.22(0.6225) + 0.19(0.7595) + 0.61(0.7595)$

$= 1.27455$

Sigmoid function

$y_1 = \frac{1}{1+e^{-V}} = \frac{1}{1 + e^{-1.27455}} = 0.781520637 9$

for O2

$V = w_{h0,o2} \cdot y_{h0} + w_{h1,o2} \cdot y_{h1} + w_{h2,o2} \cdot y_{h2} + w_{h3,o2} \cdot y_{h3}$

$= 0.61(1) + 0.38(0.6225) + 0.21(0.7595) + 0.15(0.7595)$

$= 1.11997$

Sigmoid function

$y_2 = \frac{1}{1+e^{-V}} = \frac{1}{1 + e^{-1.11997}} = 0.7539831517$

y_{h1}	y_{h2}	y_{h3}	y_1	y_2
0.6225	0.7595	0.7595	0.781520637 9	0.7539831517
			0.7815	0.7540

$d_1 = 1$ $d_2 = 0$

because $y_1 = 1$ and $y_2 = 0$

b) Compute the local gradients of each node from the right to the left side of the network. Show the detailed calculations in each step. Report the computed local gradients in the table below.

$$\delta_{o1} = -y_1(1-y_1)(d_1-y_1)$$
$$= -0.7815(1-0.7815)(1-0.7815)$$
$$= -0.0373105684$$

$$\delta_{o2} = -y_2(1-y_2)(d_2-y_2)$$
$$= -0.7540(1-0.7540)(0-0.7540)$$
$$= 0.14497158$$

$$\delta_{h3} = -y_{h3}(1-y_{h3})\left[\sum_{k=1,2} w_{hi,ok}\cdot\delta_{ok} \quad i=1,2,3\right]$$
$$= -0.7595(1-0.7595)\left[(w_{h3,o1}\cdot\delta_{o1})+(w_{h3,o2}\cdot\delta_{o2})\right]$$
$$= -0.7595(1-0.7595)\left[(0.61(-0.0373))+(0.15(0.1450))\right]$$
$$= 0.0001832$$

$$\delta_{h2} = -y_{h2}(1-y_{h2})\left[(w_{h2,o1}\cdot\delta_{o1})+(w_{h2,o2}\cdot\delta_{o2})\right]$$
$$= -0.7595(1-0.7595)\left[(0.19(-0.0373))+(0.21(0.1450))\right]$$
$$= -0.00426747$$

$$\delta_{h1} = -y_{h1}(1-y_{h1})\left[(w_{h1,o1}\cdot\delta_{o1})+(w_{h1,o2}\cdot\delta_{o2})\right]$$
$$= -0.6225(1-0.6225)\left[(0.53(-0.0373))+(0.61(0.1450))\right]$$
$$= -0.0161396$$

δ_{h1}	δ_{h2}	δ_{h3}	δ_{o1}	δ_{o2}
-0.0164	-0.0043	0.0002	-0.0373	0.1450

$\eta = 0.5$ (learning rate)

C. Only for the bias weights of each neuron: Compute the change in the weight and the updated weight. Show the detailed calculations in each step. Report the computed values in the table below:

$$\Delta w_{x0,h1} = -\eta \cdot \delta_{h1} \cdot x_0$$
$$= -0.5(-0.0164)(1)$$
$$= 0.0082$$

$$\Delta w_{x0,h2} = -\eta \cdot \delta_{h2} \cdot x_0$$
$$= -0.5(-0.0043)(1)$$
$$= 0.00215$$

$$\Delta w_{x0,h3} = -\eta \cdot \delta_{h3} \cdot x_0$$
$$= -0.5(0.0002)(1)$$
$$= -1 \times 10^{-4}$$

$$\Delta w_{h0,o1} = -\eta \cdot \delta_{o1} \cdot x_0$$
$$= -0.5(-0.0373)(1)$$
$$= 0.01865$$

$$\Delta w_{h0,o2} = -\eta \cdot \delta_{o2} \cdot x_0$$
$$= -0.5(0.1450)(1)$$
$$= -0.0725$$

updated weights

$$w^{new} = w^{old} + \Delta w$$

$$w_{x0,h1} = 0.18 + 0.0082 = 0.1882$$

$$w_{x0,h2} = 0.51 + 0.00215 = 0.51215$$

$$w_{x0,h3} = 0.43 + (-1 \times 10^{-4}) = 0.4299$$

$$w_{h0,01} = 0.53 + 0.01865 = 0.54865$$

$$w_{h0,02} = 0.61 - 0.0725 = 0.5375$$

$\Delta w_{x0,h1}$	$\Delta w_{x0,h2}$	$\Delta w_{x0,h3}$	$\Delta w_{h0,01}$	$\Delta w_{h0,02}$
0.0082	0.00215	-1×10^{-4}	0.01865	-0.0725

$w_{x0,h1}$	$w_{x0,h2}$	$w_{x0,h3}$	$w_{h0,01}$	$w_{h0,02}$
0.1882	0.51215	0.4299	0.54865	0.5375