Intro to deep learning

Assignment 3

Jordan Diaz

223554771

Google Colab Link

↓

https://colab.research.google.com/drive/1KRsYVPw6rDs_QaH9x2hOedIovJe46lTk?usp=sharing

Problem 1

Consider the following set of 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 the perceptron learning rule to train a single neuron perceptron on the data points given above. show all the steps in the iteration
Assuming the learning rate is 1 (i.e. $\eta=1$) and use the hard-limiter activation function (if $v \geq 0$ output 1; otherwise output 0), ie:

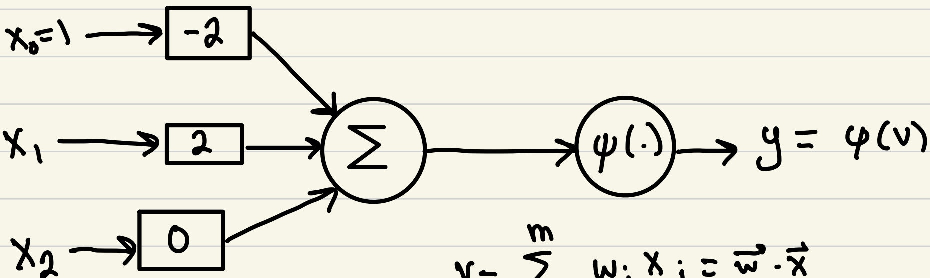
$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

Hint: if you start with a weight vector (0, 0, 0), you must Complete 3 iterations until the perceptron learning converges

i	X (x_0, x_1, x_2)	d	W (w_0, w_1, w_2)	v	y	update(y/n)	ΔW ($\Delta w_0, \Delta w_1, \Delta w_2$)	updated w
1	1 1 1	1	0 0 0	0	1	no	0 0 0	0 0 0
2	1 1 0	1	0 0 0	0	1	no	0 0 0	0 0 0
3	1 0 1	0	0 0 0	0	1	yes	-1 0 -1	-1 0 -1
4	1 -1 -1	0	-1 0 -1	0	1	yes	-1 1 1	-2 1 0
5	1 -1 0	0	-2 1 0	-3	0	no	0 0 0	-2 1 0
6	1 -1 1	0	-2 1 0	-3	0	no	0 0 0	-2 1 0
1	1 1 1	1	-2 1 0	-1	0	yes	1 1 1	-1 2 1
2	1 1 0	1	-1 2 1	1	1	no	0 0 0	-1 2 1
3	1 0 1	0	-1 2 1	0	1	yes	-1 0 -1	-2 2 0
4	1 -1 -1	0	-2 2 0	-4	0	no	0 0 0	-2 2 0
5	1 -1 0	0	-2 2 0	-4	0	no	0 0 0	-2 2 0
6	1 -1 1	0	-2 2 0	-4	0	no	0 0 0	-2 2 0
1	1 1 1	1	-2 2 0	0	1	no	0 0 0	-2 2 0
2	1 1 0	1	-2 2 0	0	1	no	0 0 0	-2 2 0
3	1 0 1	0	-2 2 0	-2	0	no	0 0 0	-2 2 0
4	1 -1 -1	0	-2 2 0	-4	0	no	0 0 0	-2 2 0
5	1 -1 0	0	-2 2 0	-4	0	no	0 0 0	-2 2 0
6	1 -1 1	0	-2 2 0	-4	0	no	0 0 0	-2 2 0

$w_0 = -2$, $w_1 = 2$, $w = 0$

b) Draw the schematics of the learned perception labeling the learned weights.



$$v = \sum_{j=0}^{m} w_j x_j = \vec{w} \cdot \vec{x}$$

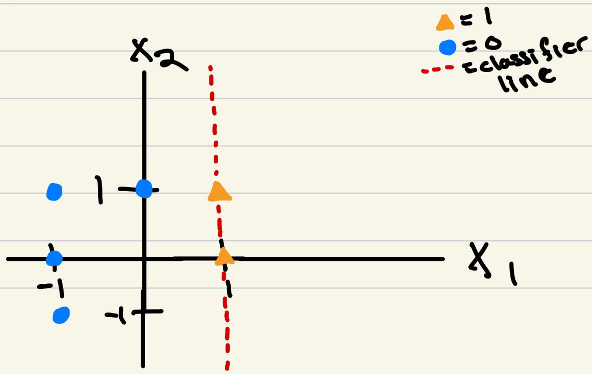
c) Provide the equation of the trained model (i.e. classifier line)

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$

$$-2 + 2x_1 = 0$$

D) Plot the given data points with two different markers for each group.

e) Plot the classifier line to the plot in (D)



when $x_2 = 1$

$$-2 + 2x_1 = 0$$

$$x_1 = 1$$

$$(1, 1)$$

$x_1 \; x_2$

when $x_2 = 0$

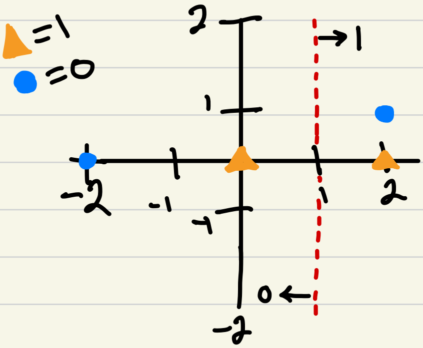
$$-2 + 2x_1 = 0$$

$$x_1 = 1$$

$$(1, 0)$$

$x_1 \; x_2$

f) Use the trained perceptron and classify the test data points given in the table below. Show your work for each data point. Provide the predicted label (0 or 1) in a table like below



Input x_1	Input x_2	desired label	local field (v)	predicted label (y)
2	0	1	2	1
2	1	0	2	1
0	0	1	-2	0
-2	0	0	-6	0

$$v = xw$$

$$= (1\ 2\ 0)(-2, 2, 0) = -2 + 4 + 0 = 2 \quad y = 1$$

$$= (1\ 2\ 1)(-2, 2, 0) = -2 + 4 + 0 = 2 \quad y = 1$$

$$= (1\ 0\ 0)(-2, 2, 0) = -2 + 0 + 0 = -2 \quad y = 0$$

$$= (1\ {-2}\ 0)(-2, 2, 0) = -2 - 4 + 0 = -6 \quad y = 0$$

g) Provide the confusion matrix for part (f) and calculate the accuracy. Consider class "1" as the positive class.

Actual \ Predicted	1	0
1	1	1
0	1	1

TP=1
FN=1
FP=1
TN=1

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

$$\text{Accuracy} = \frac{2}{4}$$

Accuracy = 50.00%

[1]

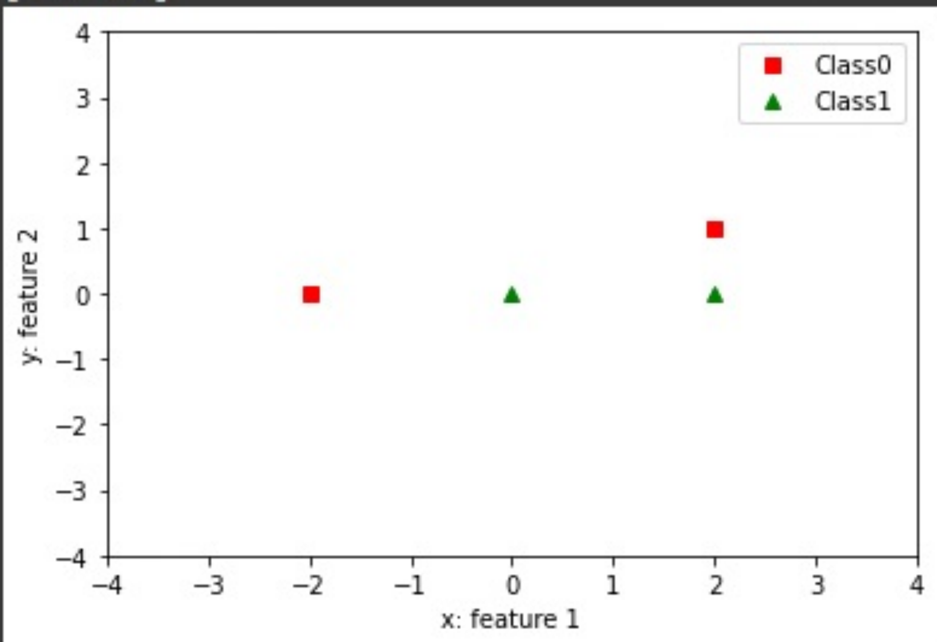
```python
import numpy as np
import matplotlib.pyplot as plt

class NeuralNetwork(object):
    def __init__(self, num_params=2):
        """ initializes a 3x1 weight vector randomly and initializes the learning rate to 1"""
        self.weight_matrix = 2 * np.random.random((num_params+1, 1)) -1   #random weights between -1 and 1
        self.learning_rate = 1

    def hard_limiter(self, x):
        """ performs the hard-limiter activation on the nx1 vector x"""
        outs = np.zeros(x.shape)
        outs[x > 0] = 1
        return outs

    def forward_propagation(self, inputs):
        """performs the forward propagation by multiplying the inputs by the neuron weights and passing the output through the hard_limiter activation function"""
        outs = np.dot(inputs, self.weight_matrix)
        return self.hard_limiter(outs)

    def train(self, inputs, labels, num_train_iterations=10):
        """performs the perceptron learning rule for num_train_iterations times using the inputs and labels"""

        for iteration in range(num_train_iterations):
            for i in range(inputs.shape[0]):
                pred_i = self.pred(inputs[i,:])
                if pred_i != labels[i]:
                    output = self.forward_propagation(inputs[i,:])
                    error = labels[i] - output
                    adjustment = self.learning_rate * error * inputs[i]
                    self.weight_matrix[:,0] += adjustment
                    print('Iteration #' + str(iteration))
                    plot_fun_thr(inputs[:,1:3], labels, self.weight_matrix[:,0], classes)

    def pred(self, inputs):
        """classifies the inputs to either class 0 or 1 by multiplying them by the neuron weights, passing the output through the hard_limiter activation function and thresholding"""
        preds = self.forward_propagation(inputs)
        return preds
```

```python
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()
```


```python
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()
```

[2]
```
#if __name__ == "__main__":
features = np.array([[2, 0],[2,1], [0,0], [-2,0]])
print(features)
labels = np.array([1, 0, 1, 0])
print(labels)
classes = [0, 1]

plot_fun(features, labels, classes)
```

```
[[ 2  0]
 [ 2  1]
 [ 0  0]
 [-2  0]]
[1 0 1 0]
```



```
[3]  bias = np.ones((features.shape[0], 1))

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

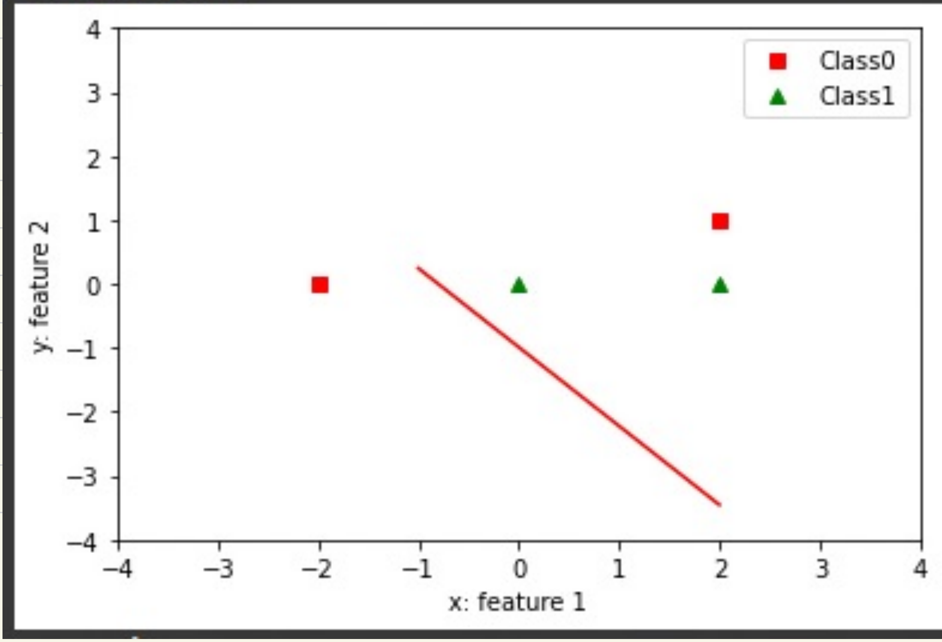
```
[[1.]
 [1.]
 [1.]
 [1.]]
(4, 1)
[[ 1.  2.  0.]
 [ 1.  2.  1.]
 [ 1.  0.  0.]
 [ 1. -2.  0.]]
(4, 3)
```

```
[4]  neural_network = NeuralNetwork()
     print('Random weights at the start of training')
     print(neural_network.weight_matrix)
```

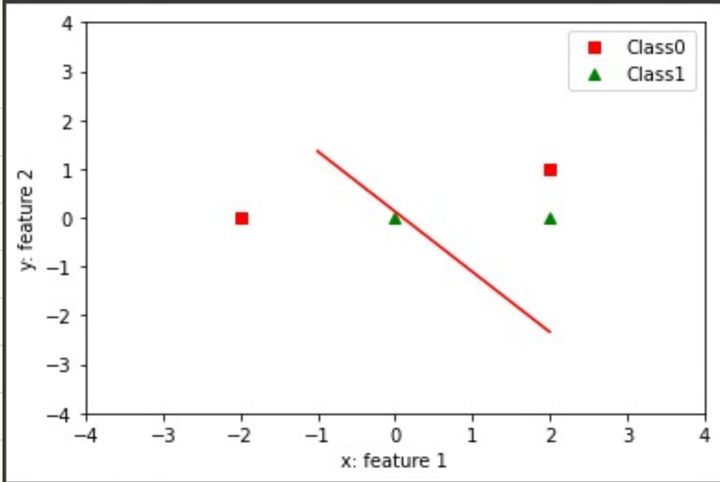
```
Random weights at the start of training
[[-0.26423872]
 [-0.26392681]
 [-0.56819898]]
```

```python
neural_network = NeuralNetwork()
print('Random weights at the start of training')
print(neural_network.weight_matrix)
neural_network.train(features, labels, 10)

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

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

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

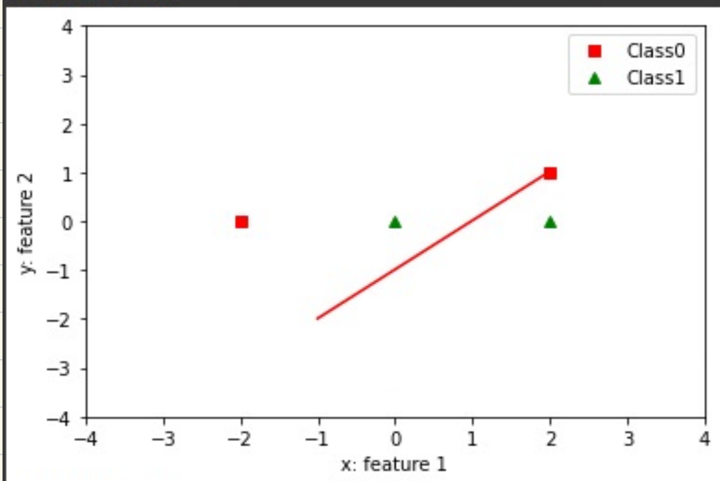
```
Random weights at the start of training
[[0.1148941 ]
 [0.89700206]
 [0.10495134]]
Iteration #0
```



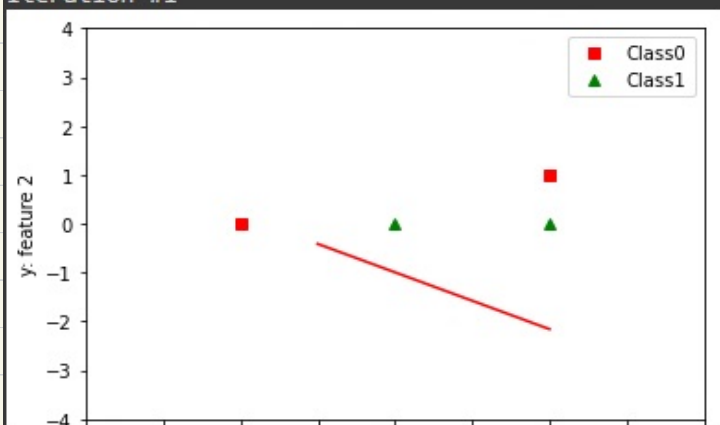
Iteration #0



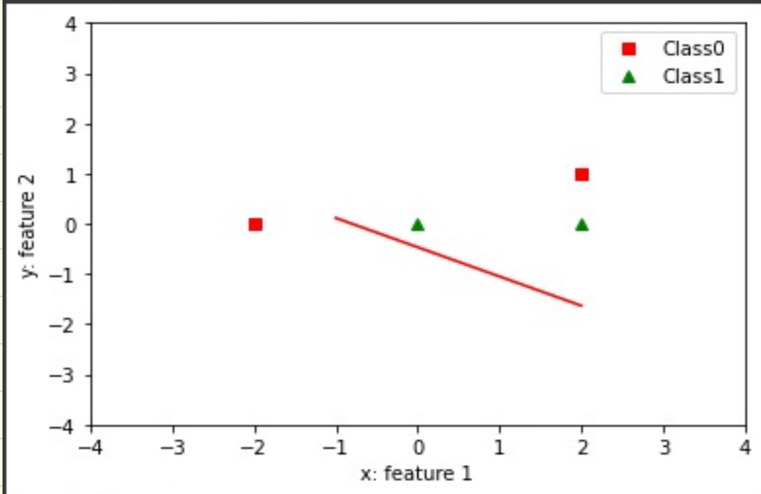
Iteration #0



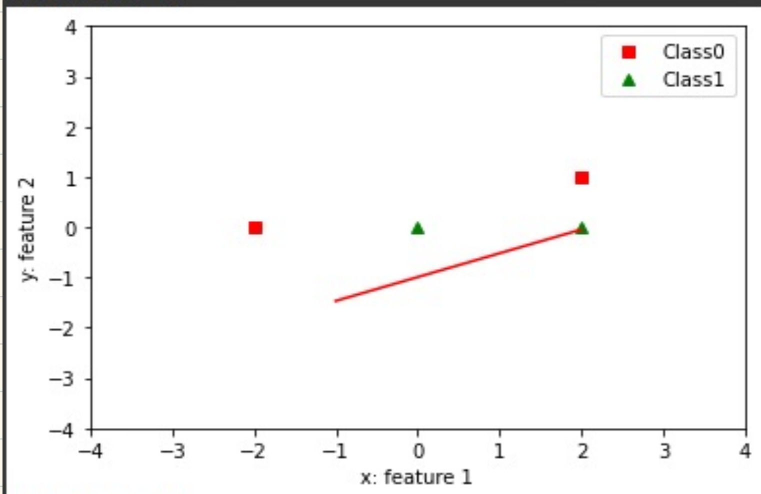
Iteration #1



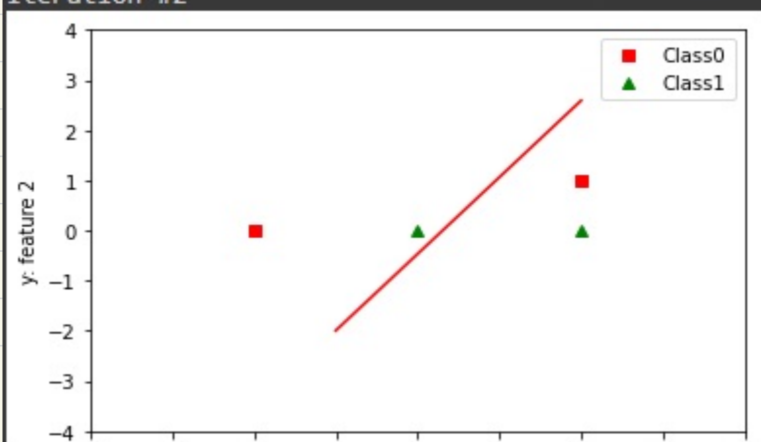
Iteration #1



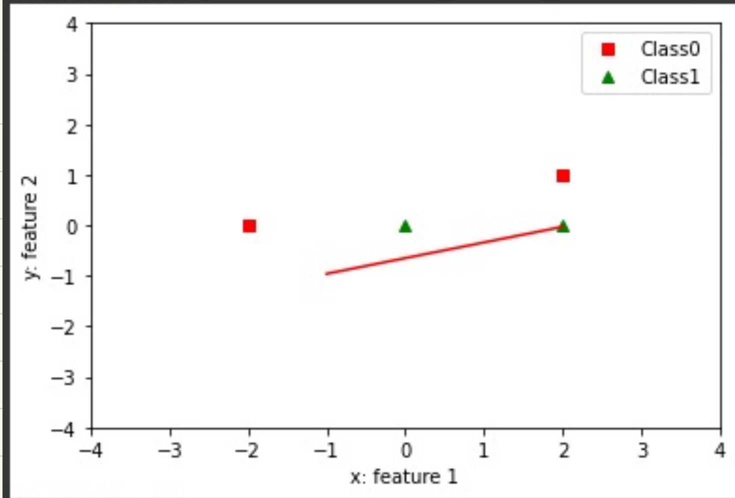
Iteration #1



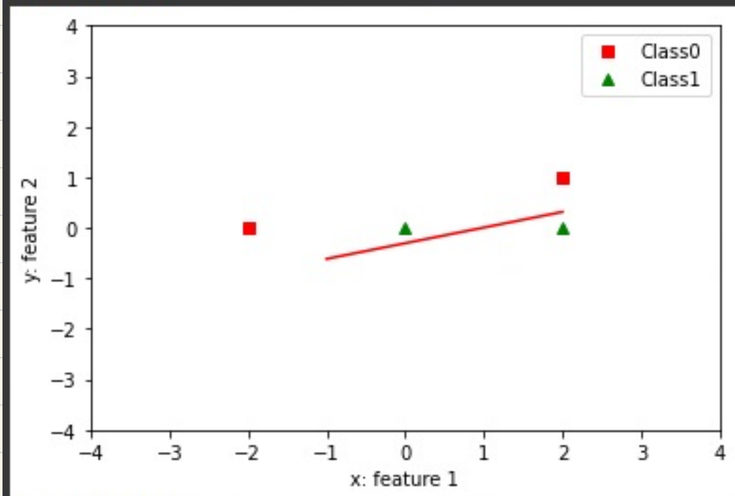
Iteration #2



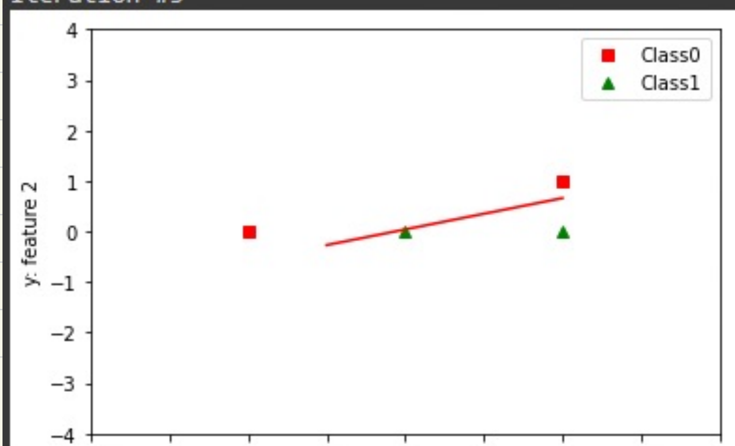
Iteration #2



Iteration #2



Iteration #3



```
New weights after training
[[ 0.1148941 ]
 [ 0.89700206]
 [-2.89504866]]
Testing network on training data points ->
[[1.]
 [0.]
 [1.]
 [0.]]
Testing network on new examples ->
[0.]
```