Jenna Leali Assignment 7

Imports

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

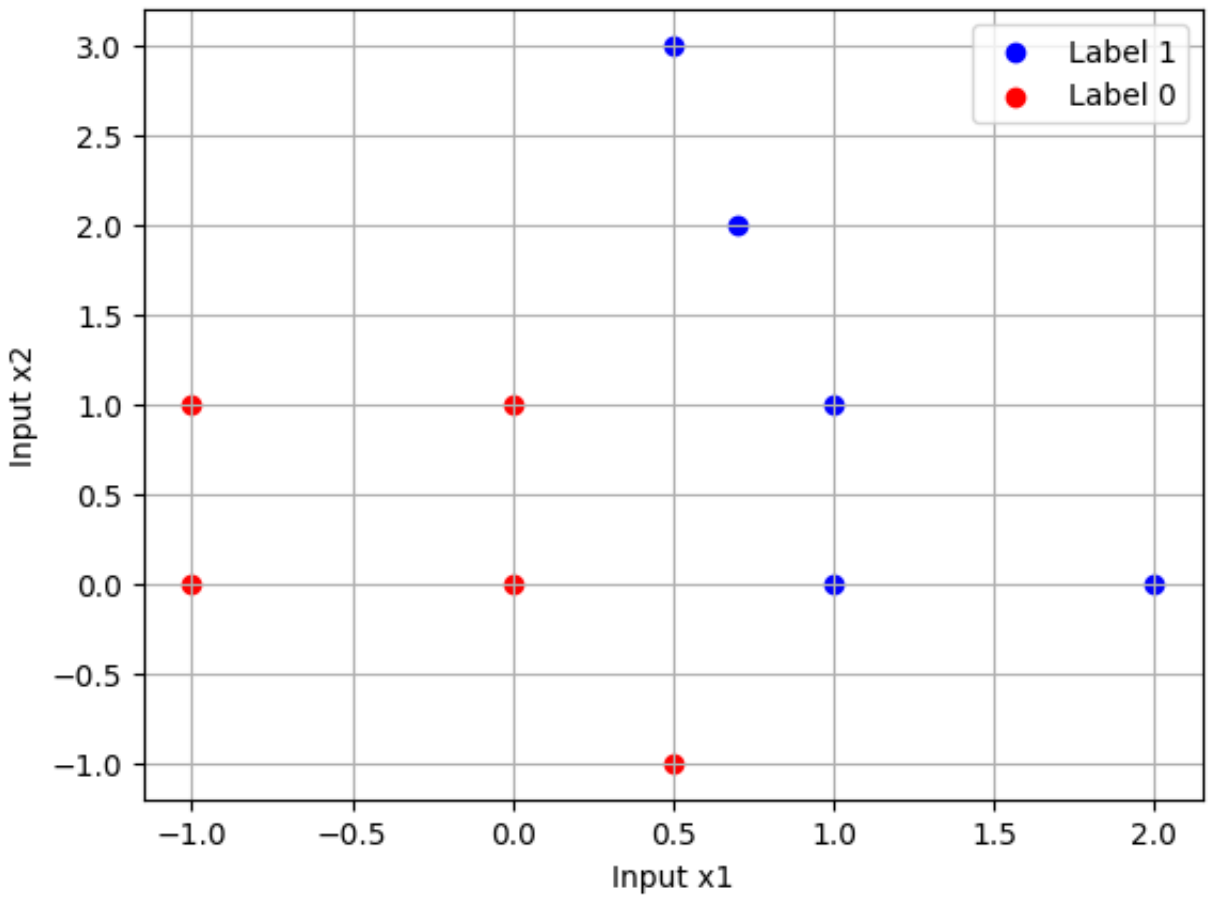
A) Create class NeuralNetwork(): that creates a single neuron with a linear activation, train it using gradient descent learning. This class should have the following function:

```
class NeuralNetwork:

# i. def __init__(self, learning_r): that initializes a 3x1 weight vector randoml
    def __init__(self, learning_r):
        np.random.seed(1)
        self.weight_matrix = 2 * np.random.rand(3, 1) - 1
        self.learning_rate = learning_r
        self.history = {'weights': [], 'cost': []}

# ii. def sigmoid(self, x): that takes an input x, and applies the sigmoid functi
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

# iii. def forward_propagation(self, inputs): that performs forward propagation b
    def forward_propagation(self, inputs):
        return self.sigmoid(np.dot(inputs, self.weight_matrix))

# iv. def train(self, inputs_train, labels_train, num_train_iterations): that per
    def train(self, inputs_train, labels_train, num_train_iterations):
        for iteration in range(num_train_iterations):
            outputs = self.forward_propagation(inputs_train)
            error = labels_train - outputs
            adjustments = self.learning_rate * np.dot(inputs_train.T, error * (ou
            self.weight_matrix += adjustments
            cost = np.mean((error ** 2))
            self.history['weights'].append(self.weight_matrix.copy())
            self.history['cost'].append(cost)
            if iteration % 10 == 0:
                print(f'Iteration {iteration} cost: {cost}')
```

B) Use the gradient descent rule to train a single neuron on the datapoints given below:

```
# i. Create an np array of a shape 10x2 that contains the inputs, and another arr
inputs = np.array([
    [1, 1],
    [1, 0],
    [0, 1],
    [0.5, -1],
    [0.5, 3],
    [0.7, 2],
    [-1, 0],
    [-1, 1],
    [2, 0],
```

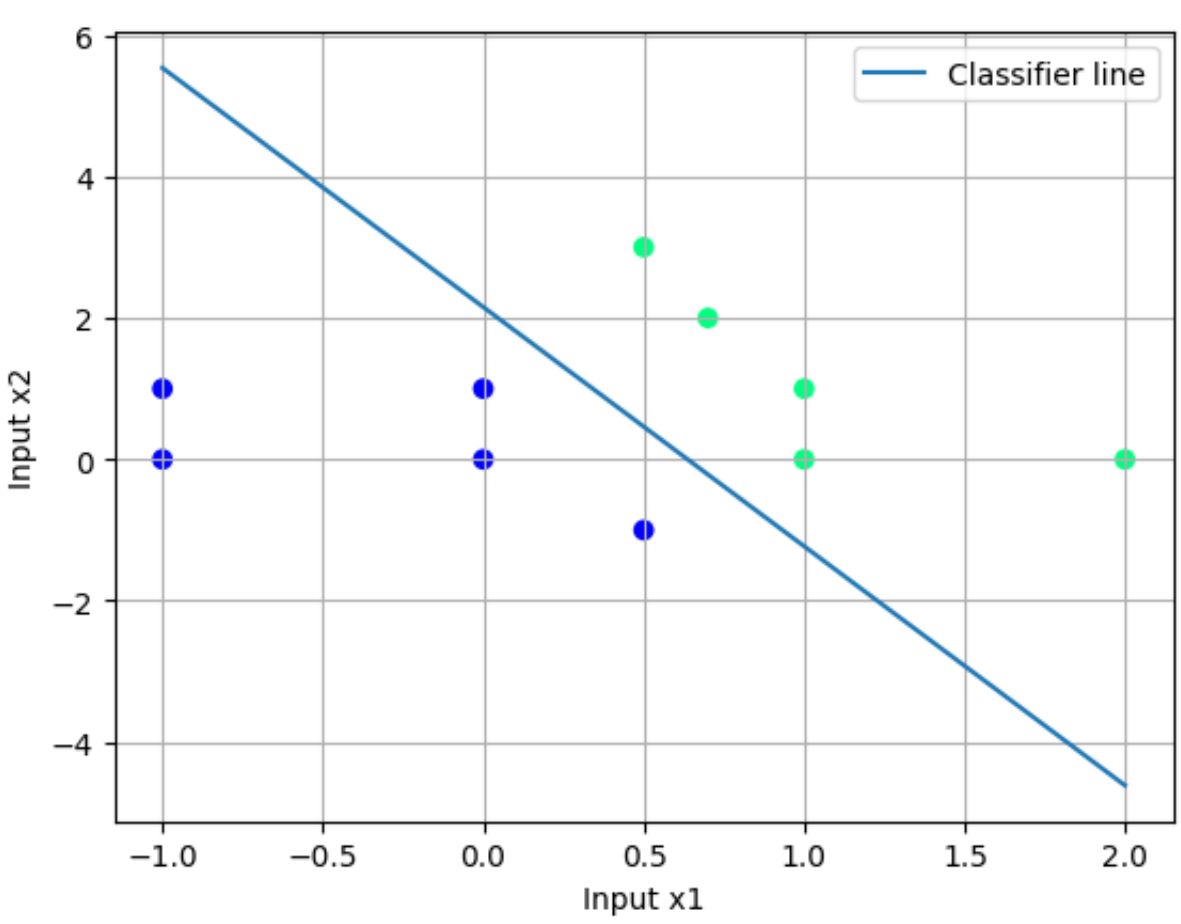
```
    [0, 0]
])

labels = np.array([
    [1],
    [1],
    [0],
    [0],
    [1],
    [1],
    [0],
    [0],
    [1],
    [0]
])

# ii. Plot the given data points with two different markers for each group.
group1 = inputs[labels.flatten() == 1]
group2 = inputs[labels.flatten() == 0]
plt.scatter(group1[:, 0], group1[:, 1], color='blue', label='Label 1')
plt.scatter(group2[:, 0], group2[:, 1], color='red', label='Label 0')
plt.xlabel('Input x1')
plt.ylabel('Input x2')
plt.legend()
plt.grid(True)
plt.show()

# iii. Add the bias to the inputs array to have a 10x3 shape.
bias = np.ones((inputs.shape[0], 1))
inputsbias = np.append(bias, inputs, axis=1)

# iv. Create the network with one neuron using the class NeuralNetwork() with lea
nn = NeuralNetwork(learning_r=1)
nn.train(inputsbias, labels, num_train_iterations=50)
print('Weights after training:')
print(nn.weight_matrix)
```




```
Iteration 0 cost: 0.3367987174255354
Iteration 10 cost: 0.03984351716837888
Iteration 20 cost: 0.02473922530502223
Iteration 30 cost: 0.017869964793316224
Iteration 40 cost: 0.013904776864459473
Weights after training:
[[-2.57419947]
 [ 4.04134309]
 [ 1.19482093]]
```

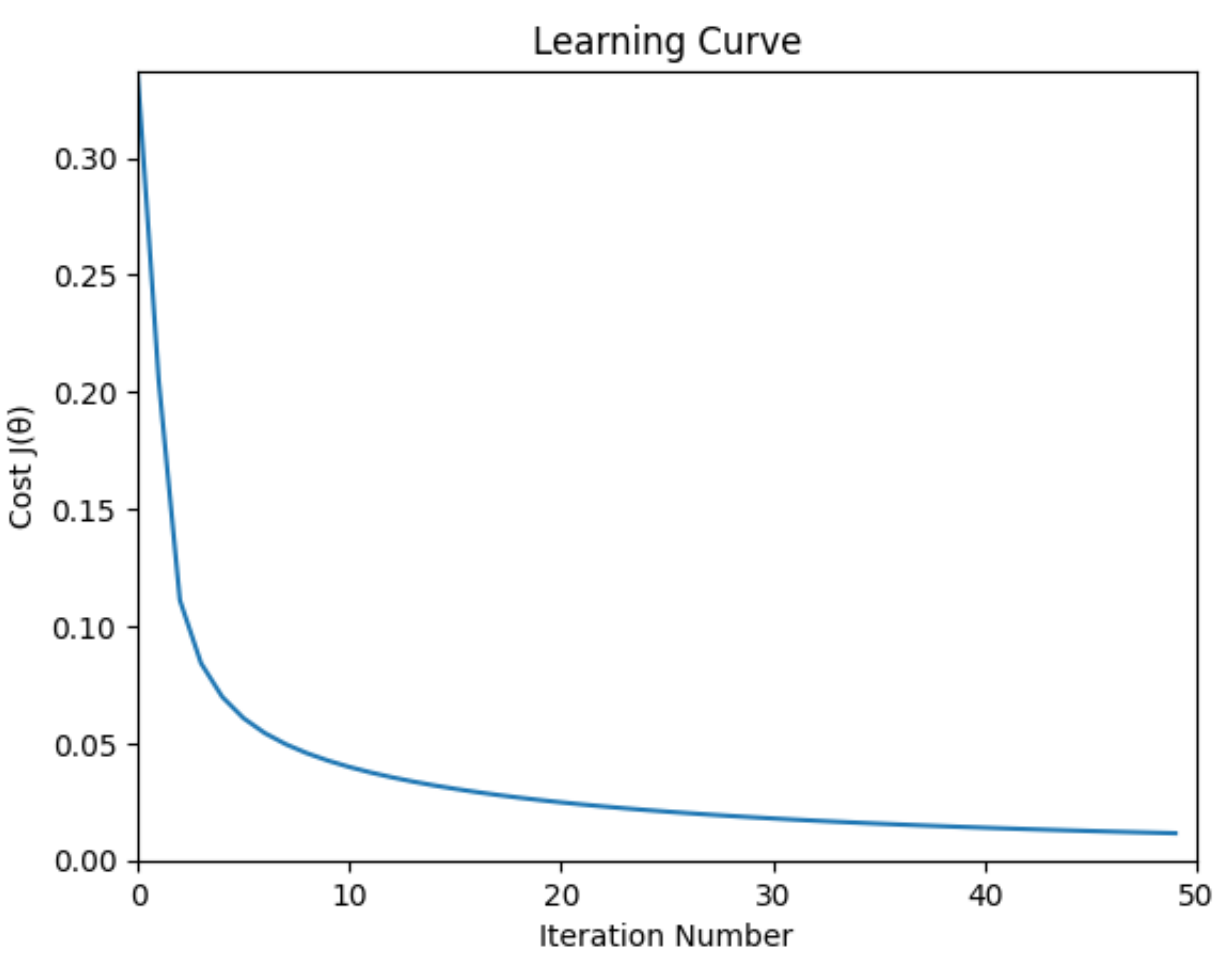
C) Use the trained weights and plot the final classifier line.

```
def plot_classifier(weights, inputs):
    bias = weights[0]
    w1 = weights[1]
    w2 = weights[2]

    x1_values = np.linspace(np.min(inputs[:,1]), np.max(inputs[:,1]), 100)
    x2_values = (-w1 * x1_values - bias) / w2

    plt.plot(x1_values, x2_values, label='Classifier line')
    plt.scatter(inputs[:,1], inputs[:,2], c=labels.flatten(), cmap=plt.cm.winter)
    plt.xlabel('Input x1')
    plt.ylabel('Input x2')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_classifier(nn.weight_matrix.flatten(), inputsbias)
```



D) Plot the training cost (i.e., the learning curve) for all the epochs.

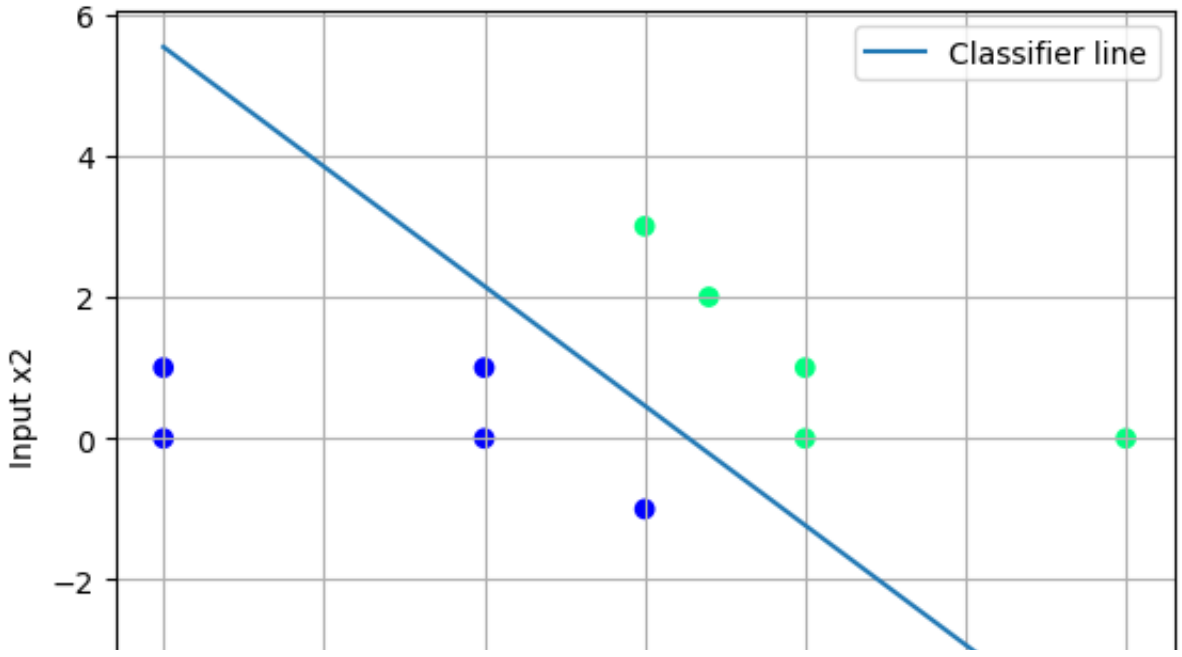
```
def plot_cost_func(history):
    iterations = range(len(history['cost']))
    costs = history['cost']

    plt.plot(iterations, costs)
    plt.axis([0, len(history['cost']), 0, np.max(history['cost'])])
    plt.title('Learning Curve')
    plt.xlabel('Iteration Number')
    plt.ylabel('Cost J(θ)')
    plt.show()

plot_cost_func(nn.history)
```

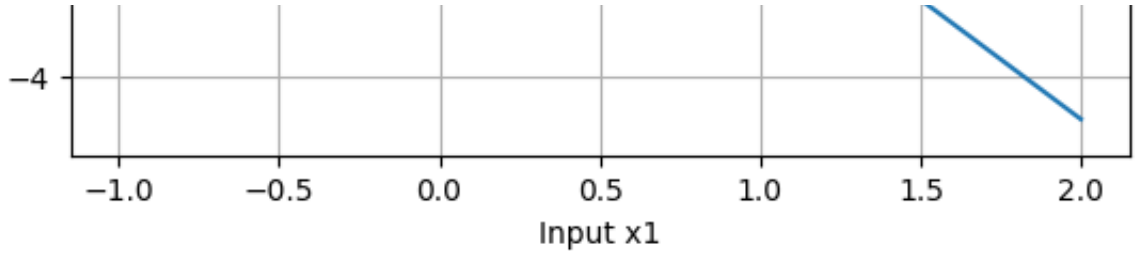


E) Repeat step (b.iv) with the learning rates of 0.5, 0.1, and 0.01. Plot the final classifier line and the learning curve for each learning rate.

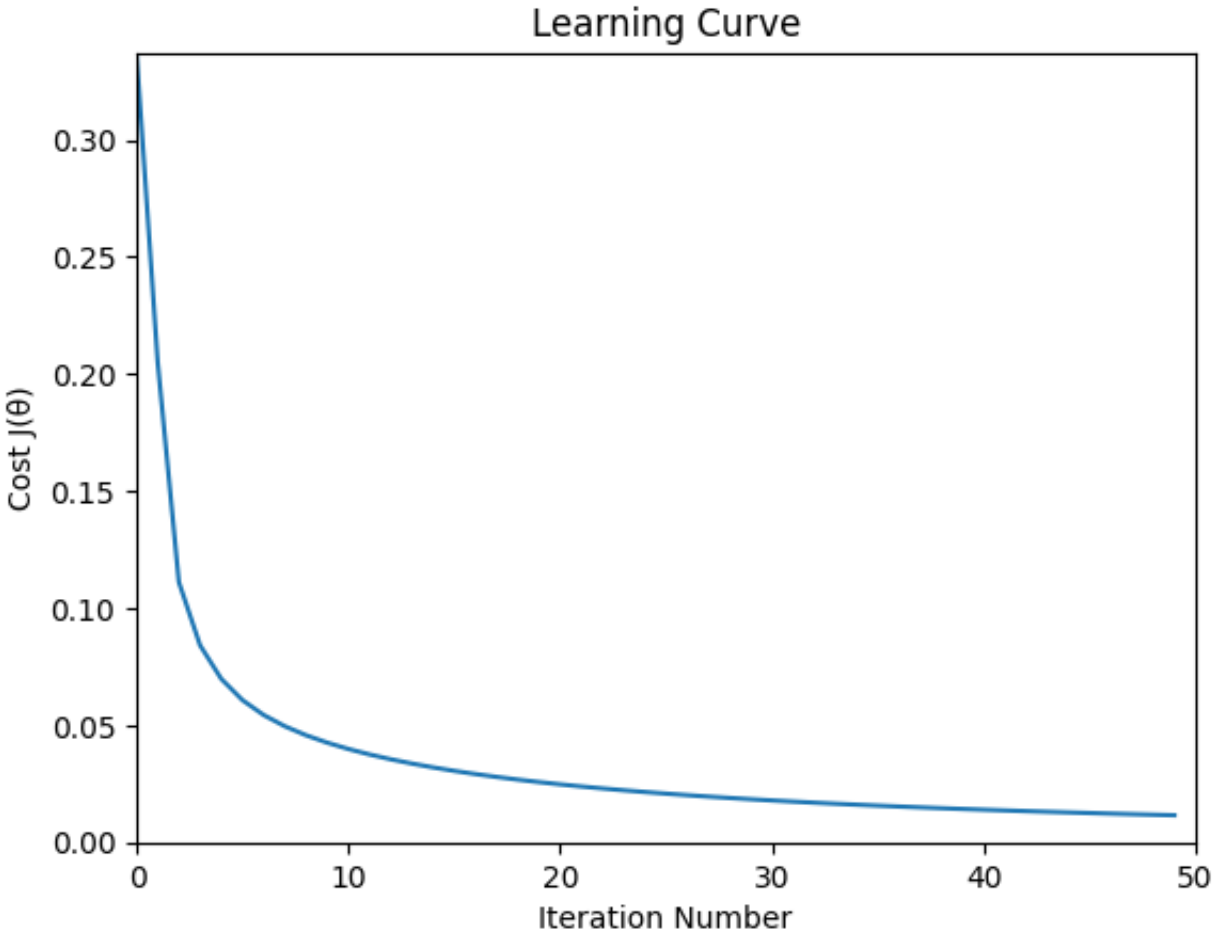
```
learning_rates = [1, 0.5, 0.1, 0.01]

for lr in learning_rates:
    nn = NeuralNetwork(learning_r=lr)
    nn.train(inputsbias, labels, num_train_iterations=50)

    print(f'Weights after training with learning rate {lr}:')
    print(nn.weight_matrix)

    print(f'Final classifier line with learning rate {lr}:')
    plot_classifier(nn.weight_matrix.flatten(), inputsbias)

    print(f'Learning curve with learning rate {lr}:')
    plot_cost_func(nn.history)
```

```
Iteration 0 cost: 0.3367987174255354
Iteration 10 cost: 0.03984351716837888
Iteration 20 cost: 0.02473922530502223
Iteration 30 cost: 0.017869964793316224
Iteration 40 cost: 0.013904776864459473
Weights after training with learning rate 1:
[[-2.57419947]
 [ 4.04134309]
 [ 1.19482093]]
Final classifier line with learning rate 1:
```

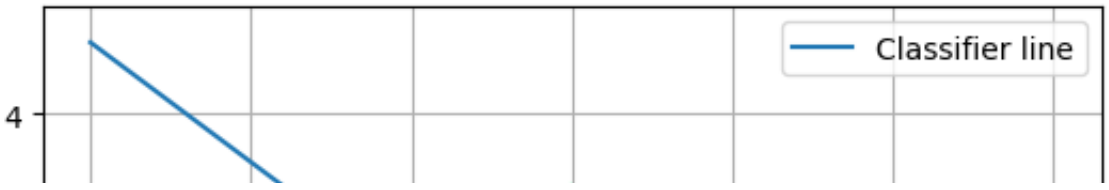


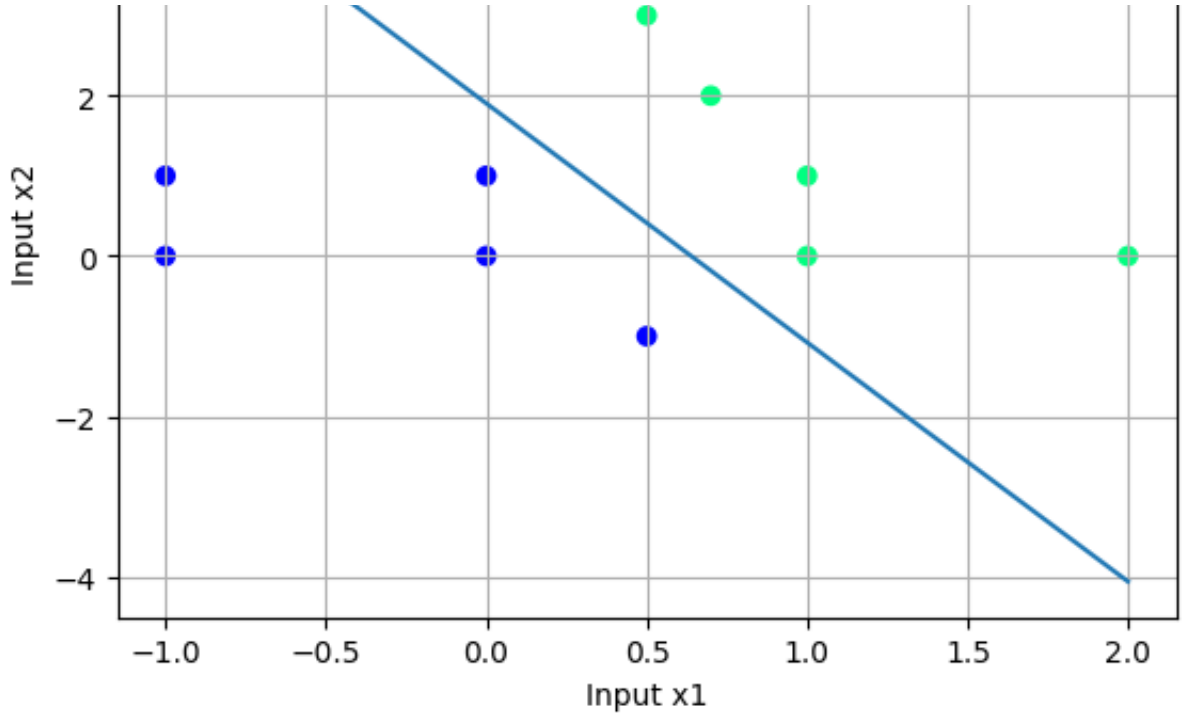



```
Learning curve with learning rate 1:
```

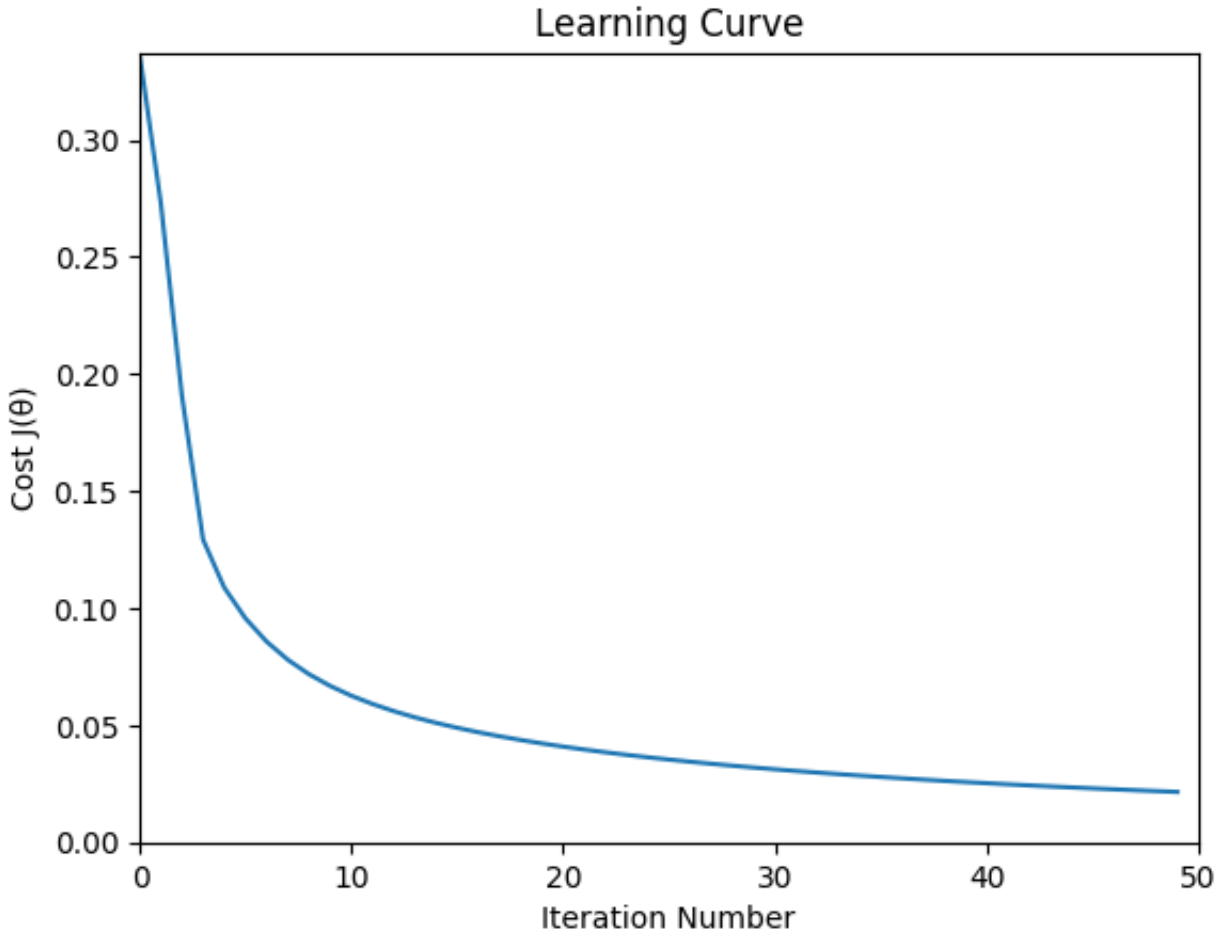



```
Iteration 0 cost: 0.3367987174255354
Iteration 10 cost: 0.06258971942654346
Iteration 20 cost: 0.04085991416626246
Iteration 30 cost: 0.03116557913105823
Iteration 40 cost: 0.025233077315549378
Weights after training with learning rate 0.5:
[[-1.98717681]
 [ 3.11091067]
 [ 1.04356904]]
Final classifier line with learning rate 0.5:
```



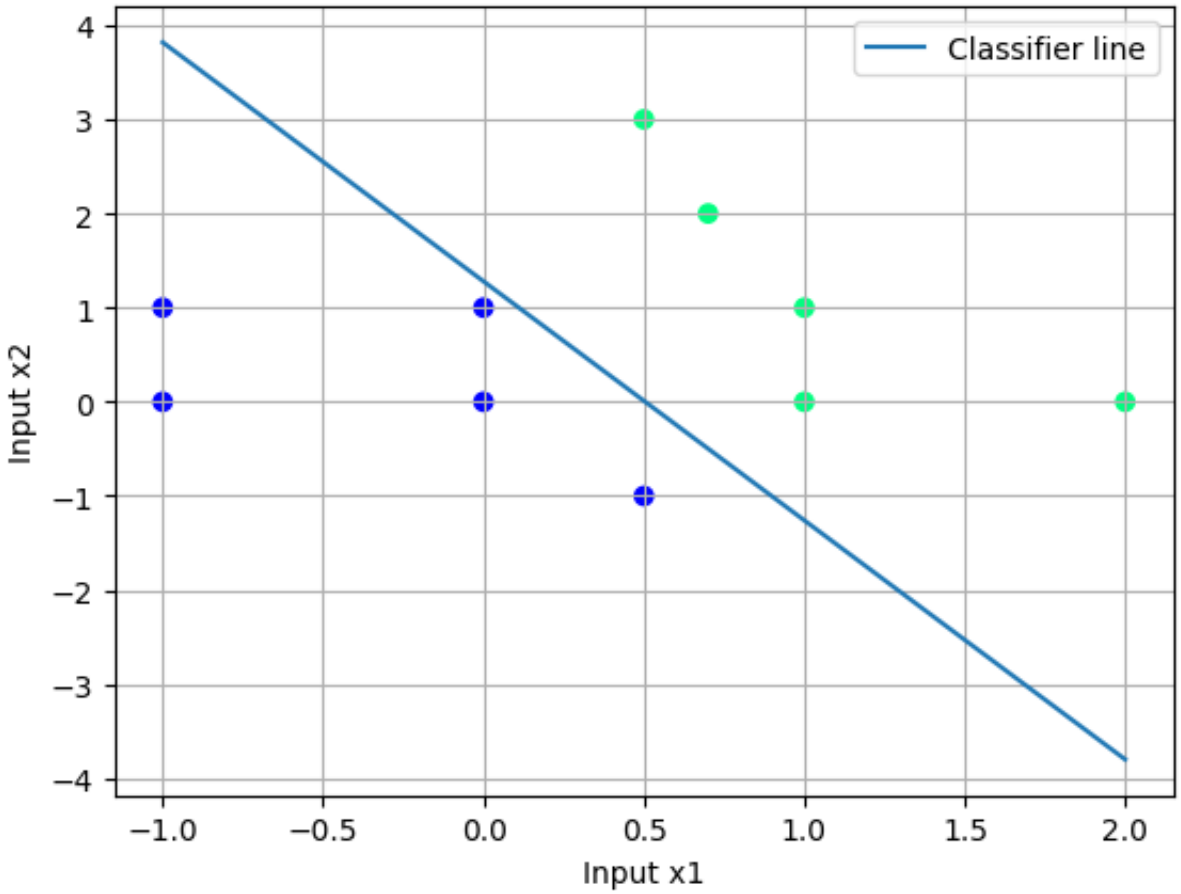



```
Learning curve with learning rate 0.5:
```

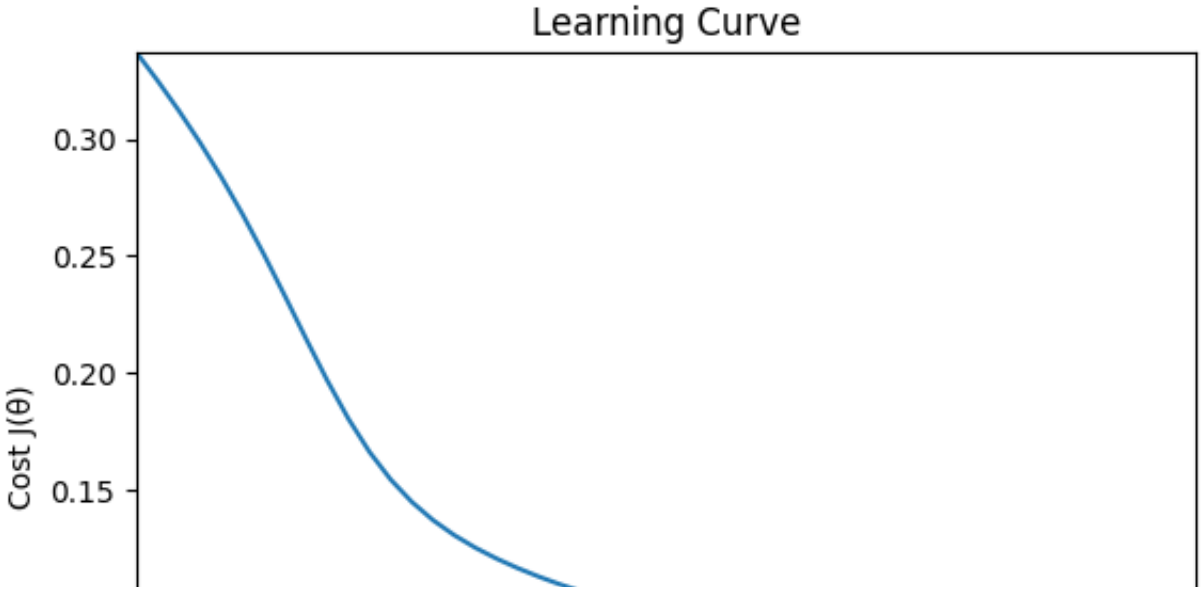


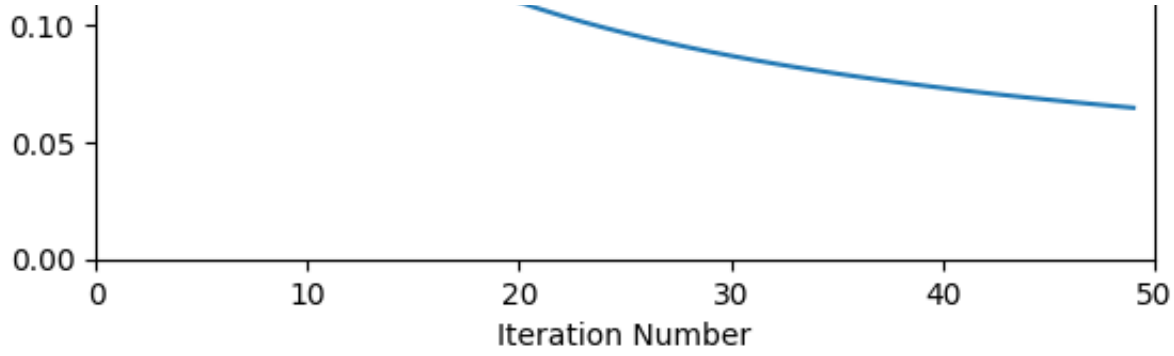

```
Iteration 0 cost: 0.3367987174255354
Iteration 10 cost: 0.18046171743895206
Iteration 20 cost: 0.10967536250041891
```

```
Iteration 30 cost: 0.08675658504628328
Iteration 40 cost: 0.073041352429266
Weights after training with learning rate 0.1:
[[-0.80785306]
 [ 1.60288217]
 [ 0.63140304]]
Final classifier line with learning rate 0.1:
```

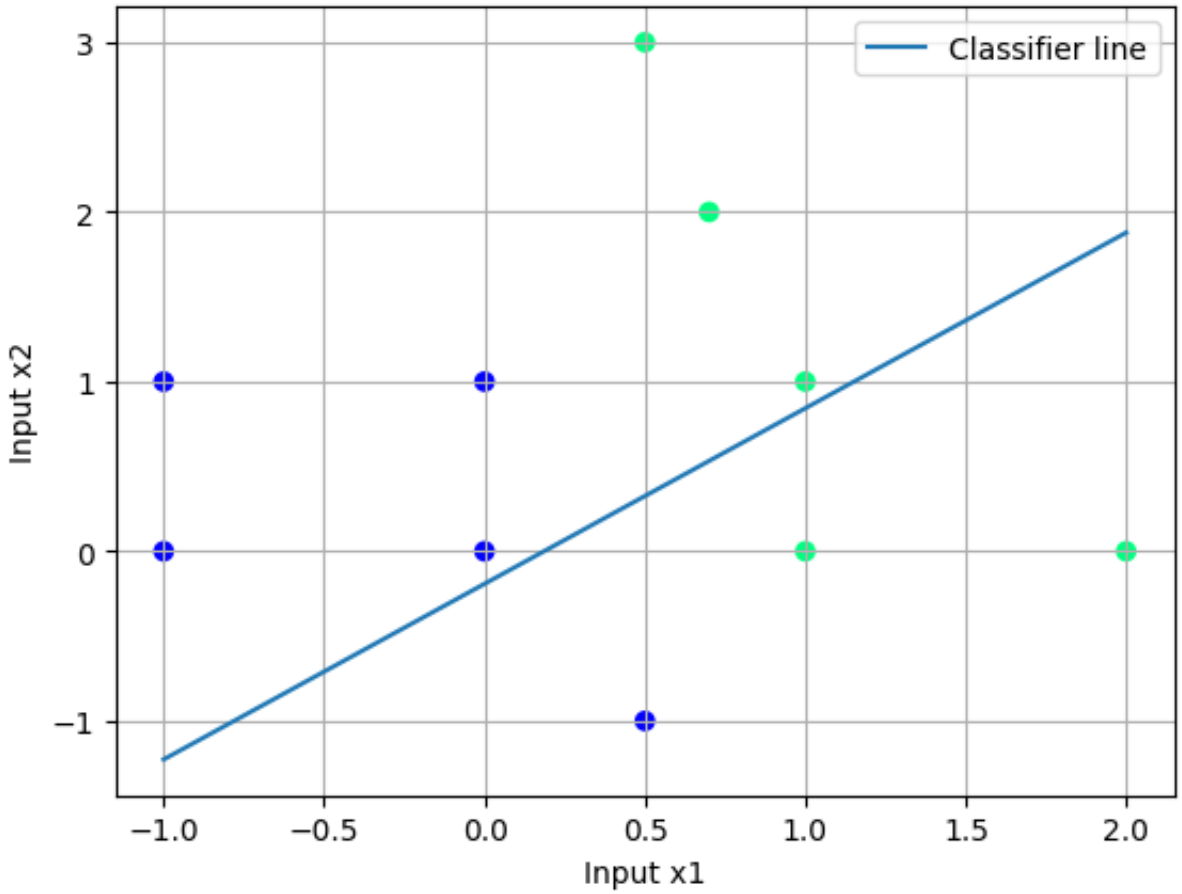



```
Learning curve with learning rate 0.1:
```

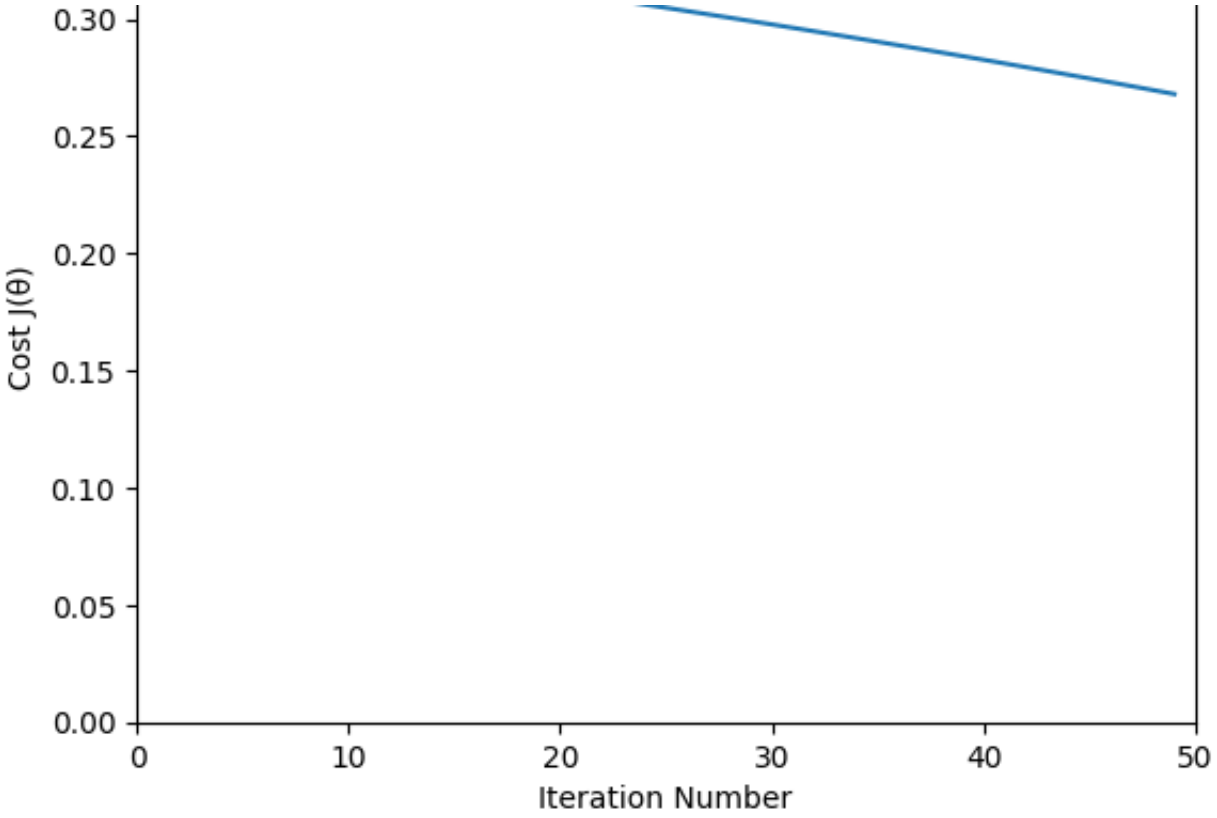





```
Iteration 0 cost: 0.3367987174255354
Iteration 10 cost: 0.32457688224832043
Iteration 20 cost: 0.3115792393084414
Iteration 30 cost: 0.297621256638606
Iteration 40 cost: 0.28253936772552957
Weights after training with learning rate 0.01:
[[-0.12744521]
 [ 0.68828451]
 [-0.66498412]]
Final classifier line with learning rate 0.01:
```




```
Learning curve with learning rate 0.01:
```



F) What behavior do you observe from the learning curves with the different learning rates? Explain your observations. Which learning rate is more suitable? Explain.

Learning rate 1:

In the beginning the cost drops quickly as seen by the steep decrease, but then slows down as it approaches lower values. This shows that the learning rate is high enough, but not too high. Meaning, it can make quick changes, but doesn't get to a point where it stays around the minimum becuase of large updates. Overall, I would consider 1 to be a good learning rate for the problem.

Learning rate 0.5:

The cost decreases more steadily and less sharply compared to the learning rate of 1. This learning rate is a bit more conservative and can lead to a more stable convergence, but it might require more iterations to reach the same cost as a higher learning rate.

Learning rate 0.1:

The cost starts to decrease slowly, showing that the steps taken towards the minimum are smaller. This rate may avoid the risks of overshooting the minimum but will take longer to converge. If the cost is still decreasing after 50 iterations, it may need more iterations to reach an optimum.

Learning rate 0.01:

The change in cost is very gradual, indicating very slow convergence. In cases where the landscape of the cost function is complex with many local minima, such a small learning rate might be beneficial as it carefully probes the landscape. However, in a simple problem, it may be unnecessarily slow and inefficient.

Double-click (or enter) to edit