[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

```python
#a.) Create class NeuralNetwork():

import numpy as np

class NeuralNetwork:
    def __init__(self, learning_rate):
        np.random.seed(42)
        self.weights = np.random.randn(3, 1)
        self.learning_rate = learning_rate
        self.history = []

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

    def forward_propagation(self, inputs):
        z = np.dot(inputs, self.weights)
        output = self.sigmoid(z)
        return output

    def train(self, inputs_train, labels_train, num_train_iterations):
        for i in range(num_train_iterations):

            output = self.forward_propagation(inputs_train)

            error = labels_train - output

            gradient = np.dot(inputs_train.T, error) / inputs_train.shape[0]

            self.weights += self.learning_rate * gradient

            self.history.append((self.weights.copy(), np.mean(np.square(error))))
```

```python
#create dataset
inputs_train = np.array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
labels_train = np.array([[0], [1], [1], [0]])

#neural network with a learning rate of 0.01
nn = NeuralNetwork(learning_rate=0.01)

#neural network for 1000 iterations
nn.train(inputs_train, labels_train, num_train_iterations=1000)

print("Final weights:", nn.weights)
print("Training cost after the last epoch:", nn.history[-1][1])
```

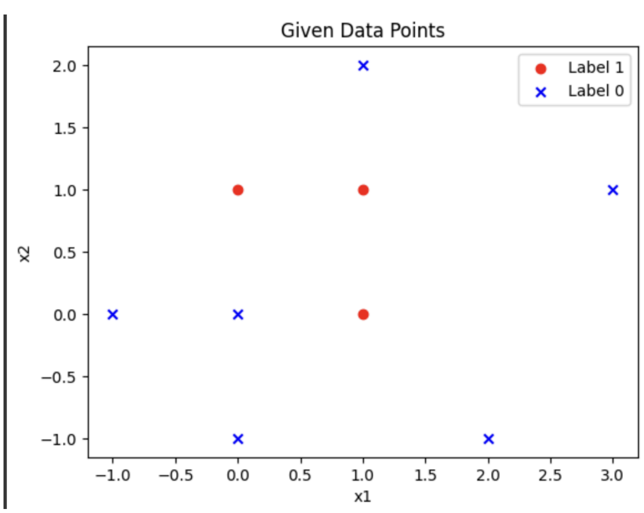
```
Final weights: [[ 1.81468394]
 [-0.45489216]
 [-0.37111334]]
Training cost after the last epoch: 0.09242699536287234
```

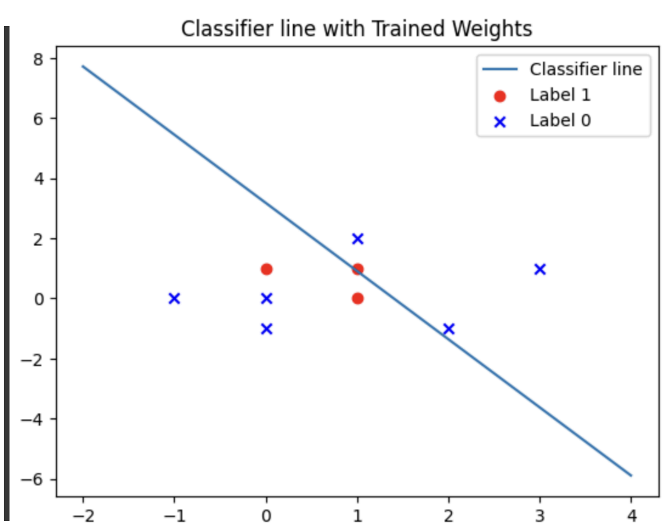
```python
#b.) Use the gradient descent rule to train a single neuron on the datapoints given below:

import numpy as np
import matplotlib.pyplot as plt

class NeuralNetwork:
    def __init__(self, learning_rate):
        np.random.seed(42)
        self.weights = np.random.rand(3, 1)
        self.learning_rate = learning_rate
        self.history = {'weights': [], 'cost': []}

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

    def forward_propagation(self, inputs):
        return self.sigmoid(np.dot(inputs, self.weights))

    def train(self, inputs_train, labels_train, num_train_iterations):
        for epoch in range(num_train_iterations):

            predictions = self.forward_propagation(inputs_train)

            cost = np.mean((predictions - labels_train) ** 2)
            self.history['cost'].append(cost)

            error = predictions - labels_train
            gradient = np.dot(inputs_train.T, error) / len(inputs_train)
            self.weights -= self.learning_rate * gradient

            self.history['weights'].append(np.copy(self.weights))
```

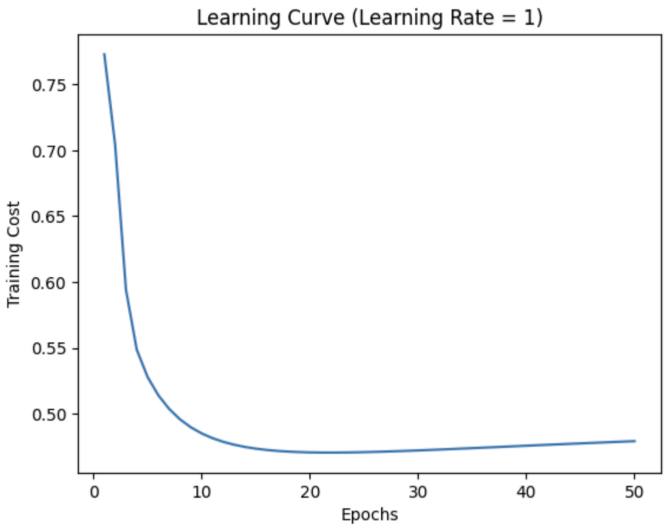
```python
inputs_train = np.array([
    [1, 1],
    [1, 0],
    [0, 1],
    [-1, 0],
    [3, 1],
    [2, -1],
    [0, -1],
    [1, 2],
    [0, 0],
])

labels_train = np.array([[1], [1], [1], [0.5], [-1], [0.7], [2], [1], [0]])

#plot
plt.scatter(inputs_train[:3, 0], inputs_train[:3, 1], c='red', marker='o', label='Label 1')
plt.scatter(inputs_train[3:, 0], inputs_train[3:, 1], c='blue', marker='x', label='Label 0')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Given Data Points')
plt.legend()
plt.show()

#add bias
inputs_train_with_bias = np.hstack((np.ones((inputs_train.shape[0], 1)), inputs_train))

#create and train the neural network with a learning rate of 1
nn = NeuralNetwork(learning_rate=1)
nn.train(inputs_train_with_bias, labels_train, num_train_iterations=50)
```

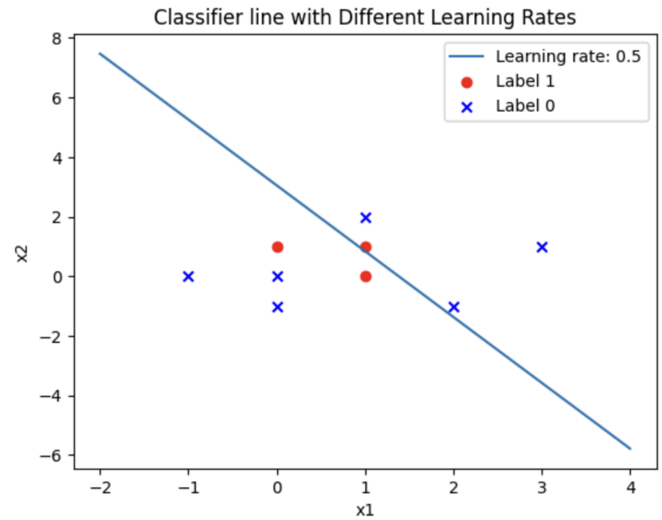
```python
#plot final classifier line using the trained weights
x_line = np.linspace(-2, 4, 100)
y_line = -(nn.weights[0] + nn.weights[1] * x_line) / nn.weights[2]
plt.plot(x_line, y_line, label='Classifier line')
plt.scatter(inputs_train[:3, 0], inputs_train[:3, 1], c='red', marker='o', label='Label 1')
plt.scatter(inputs_train[3:, 0], inputs_train[3:, 1], c='blue', marker='x', label='Label 0')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Classifier line with Trained Weights')
plt.legend()
plt.show()

#plot training cost for all epochs
plt.plot(range(1, 51), nn.history['cost'])
plt.xlabel('Epochs')
plt.ylabel('Training Cost')
plt.title('Learning Curve (Learning Rate = 1)')
plt.show()

#repeat with different learning rates
for learning_rate in [0.5, 0.1, 0.01]:
    nn = NeuralNetwork(learning_rate=learning_rate)
    nn.train(inputs_train_with_bias, labels_train, num_train_iterations=50)
    #plot final classifier line
    y_line = -(nn.weights[0] + nn.weights[1] * x_line) / nn.weights[2]
    plt.plot(x_line, y_line, label=f'Learning rate: {learning_rate}')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Classifier line with Different Learning Rates')
    plt.scatter(inputs_train[:3, 0], inputs_train[:3, 1], c='red', marker='o', label='Label 1')
    plt.scatter(inputs_train[3:, 0], inputs_train[3:, 1], c='blue', marker='x', label='Label 0')
    plt.legend()
    plt.show()
```

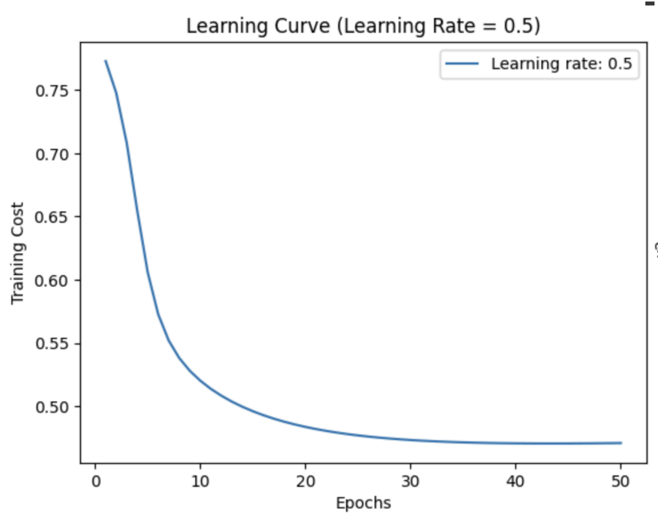
```python
    #plot learning curve
    plt.plot(range(1, 51), nn.history['cost'], label=f'Learning rate: {learning_rate}')
    plt.xlabel('Epochs')
    plt.ylabel('Training Cost')
    plt.title(f'Learning Curve (Learning Rate = {learning_rate})')
    plt.legend()
    plt.show()
```

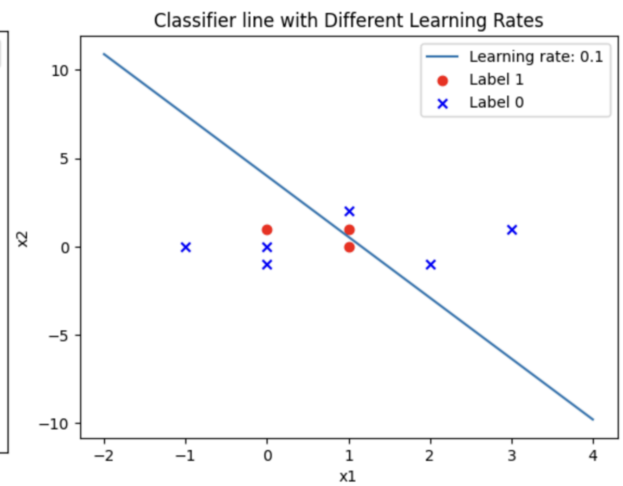


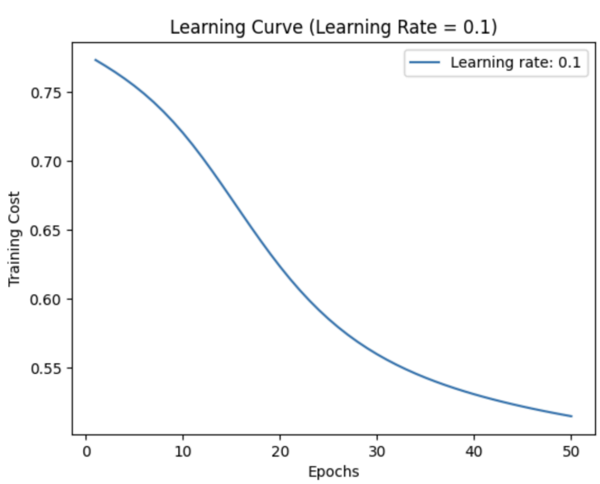


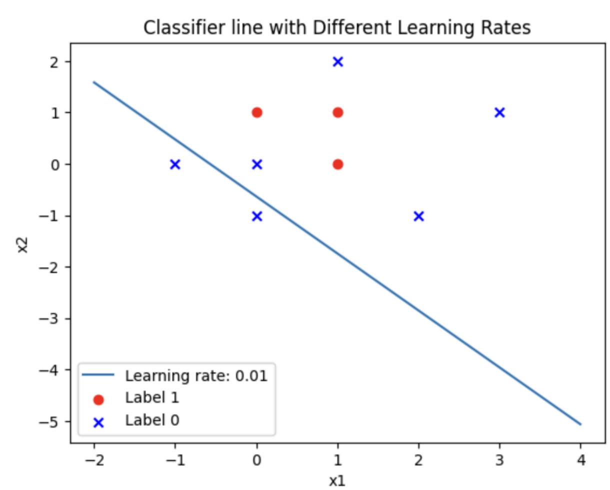

Learning Curve (Learning Rate = 1)
Training Cost
Epochs

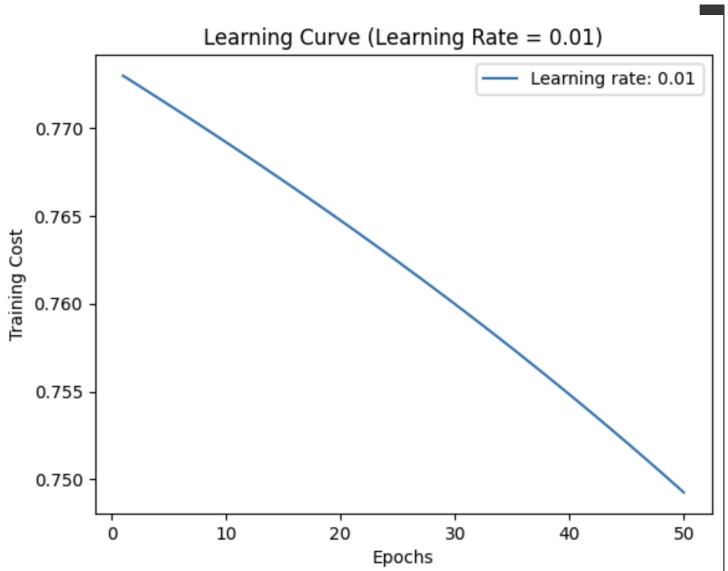

Classifier line with Different Learning Rates
Learning rate: 0.5
Label 1
Label 0
x2
x1


Learning Curve (Learning Rate = 0.5)
Learning rate: 0.5
Training Cost
Epochs


Classifier line with Different Learning Rates
Learning rate: 0.1
Label 1
Label 0
x2
x1


Learning Curve (Learning Rate = 0.1)
Learning rate: 0.1
Training Cost
Epochs


Classifier line with Different Learning Rates
Learning rate: 0.01
Label 1
Label 0
x2
x1



Observations:

Learning Rate = 0.5/1: My observation shows that 0.5 decreases rapidly similar to 1, with a risk of overshooting.

Learning Rate = 0.1: My observation shows that 0.1 decreases steadily with a good balance between speed and stability.

Learning Rate = 0.01: My observation shows that 0.01 decreases gradually, providing a balance between stability and convergence speed.

The best learning rate is Learning rate 0.01 because it is the smoothest and consistent decrease.