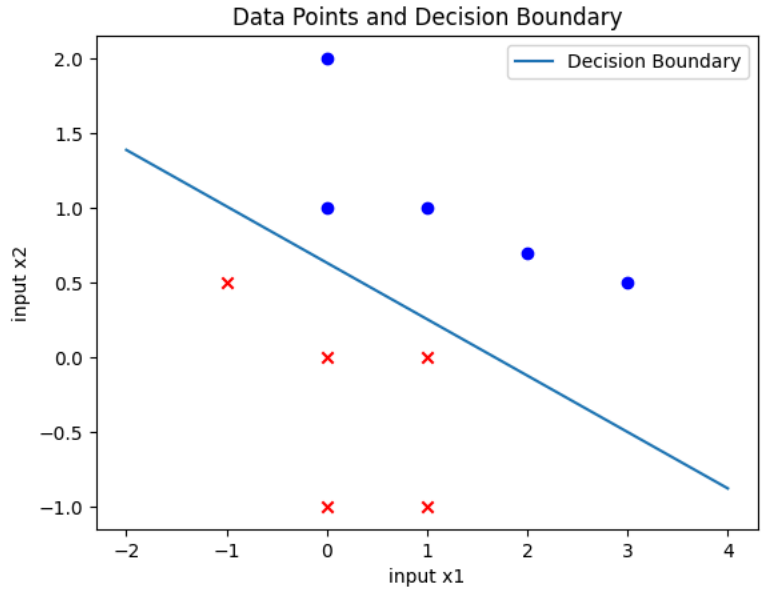
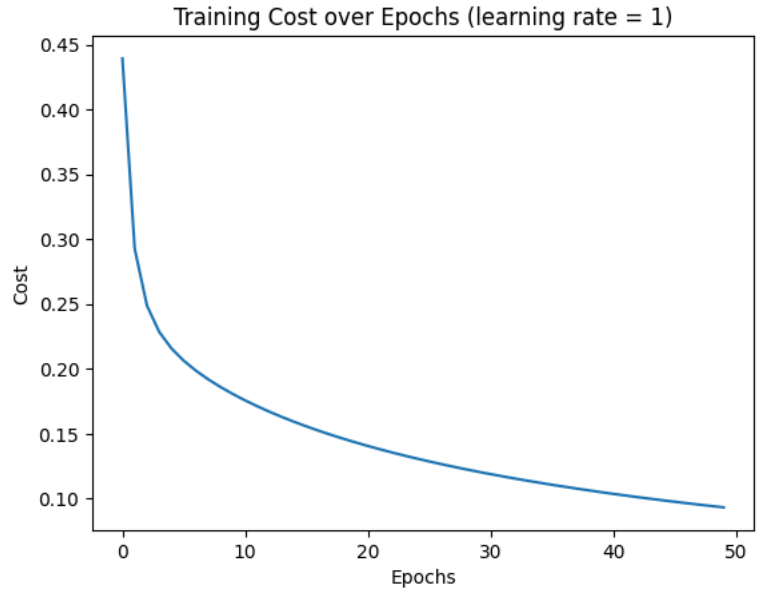
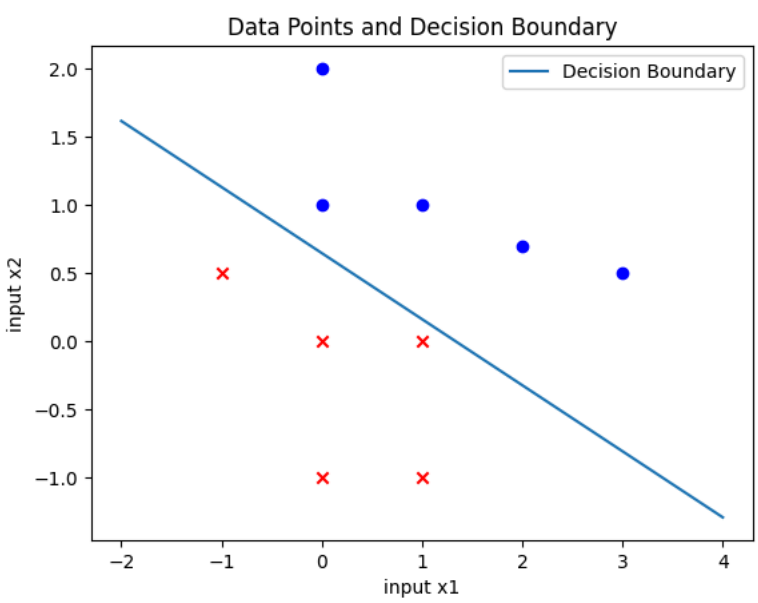
```
import numpy as np
import matplotlib.pyplot as plt


class NeuralNetwork:
    def __init__(self, learning_rate):
        self.weights = np.random.randn(3, 1)  # Initialize a 3x1 weight vector with random values
        self.learning_rate = learning_rate
        self.history = {'weights': [], 'cost': []}  # Dictionary to store weights and cost over epochs

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

    def forward_propagation(self, inputs):
        z = np.dot(inputs, self.weights)  # Linear combination
        return self.sigmoid(z)  # Apply sigmoid function

    def train(self, inputs_train, labels_train, epochs):
        m = len(labels_train)
        for epoch in range(epochs):
            predictions = self.forward_propagation(inputs_train)
            cost = -1/m * np.sum(labels_train * np.log(predictions) + (1 - labels_train) * np.log(1 - predictions))
            self.history['cost'].append(cost)

            gradient = np.dot(inputs_train.T, (predictions - labels_train)) / m
            self.weights -= self.learning_rate * gradient
            self.history['weights'].append(self.weights.copy())
        return self.history


# Define the dataset
inputs = np.array([
    [1, 1],
    [0, 1],
    [1, 0],
    [-1, 0.5],
    [3, 0.5],
    [2, 0.7],
    [0, -1],
    [1, -1],
    [0, 2],
    [0, 0]
])

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


# Add a bias term to the inputs
inputs_with_bias = np.hstack((np.ones((inputs.shape[0], 1)), inputs))
```

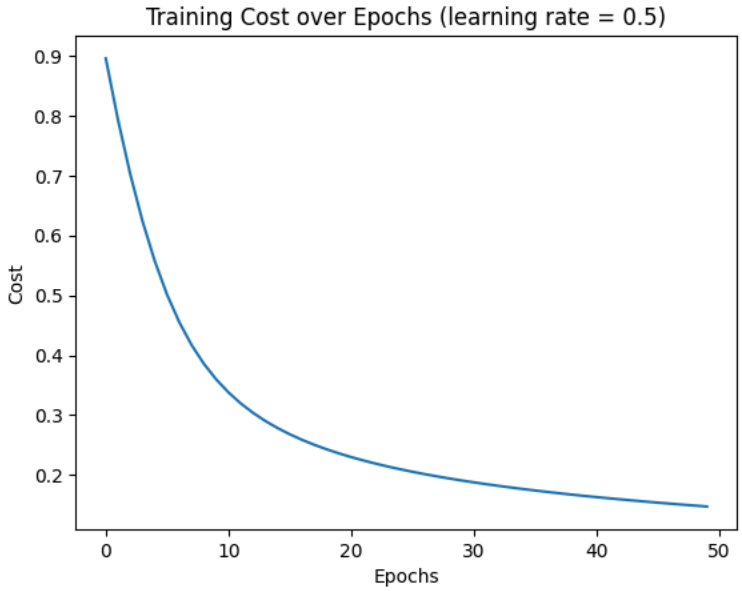
```
# Function to plot the data points and the decision boundary
def plot_decision_boundary(weights, inputs, labels):
    for i in range(len(labels)):
        if labels[i] == 1:
            plt.scatter(inputs[i, 1], inputs[i, 2], color='red', marker='x')
        else:
            plt.scatter(inputs[i, 1], inputs[i, 2], color='blue', marker='o')

    x_values = np.linspace(-2, 4, 100)
    y_values = -(weights[0] + weights[1] * x_values) / weights[2]
    plt.plot(x_values, y_values, label='Decision Boundary')
    plt.xlabel('input x1')
    plt.ylabel('input x2')
    plt.legend()
    plt.title('Data Points and Decision Boundary')
    plt.show()


# Train the neural network with learning rate = 1
nn1 = NeuralNetwork(1)
history1 = nn1.train(inputs_with_bias, labels, 50)
plot_decision_boundary(nn1.weights, inputs_with_bias, labels)
plt.plot(history1['cost'])
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Training Cost over Epochs (learning rate = 1)')
plt.show()
```



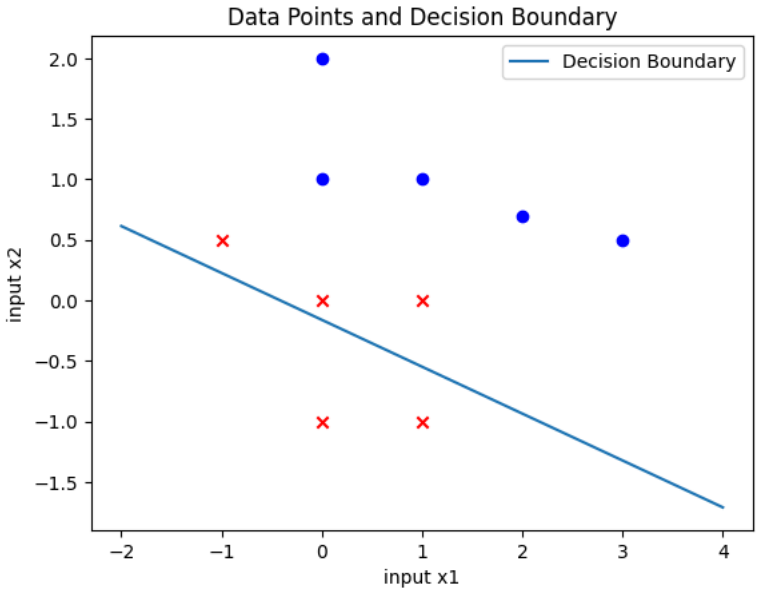


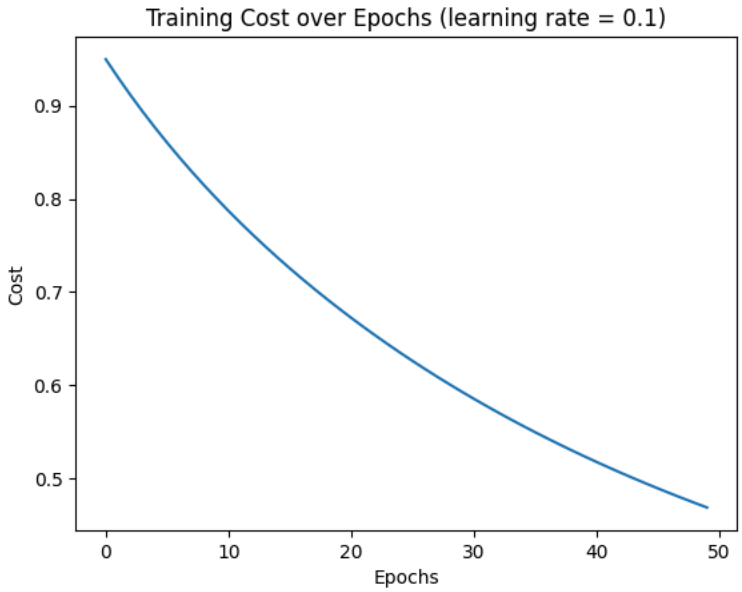
```
# Train the neural network with learning rate = 0.5
nn2 = NeuralNetwork(0.5)
history2 = nn2.train(inputs_with_bias, labels, 50)
plot_decision_boundary(nn2.weights, inputs_with_bias, labels)
plt.plot(history2['cost'])
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Training Cost over Epochs (learning rate = 0.5)')
plt.show()
```



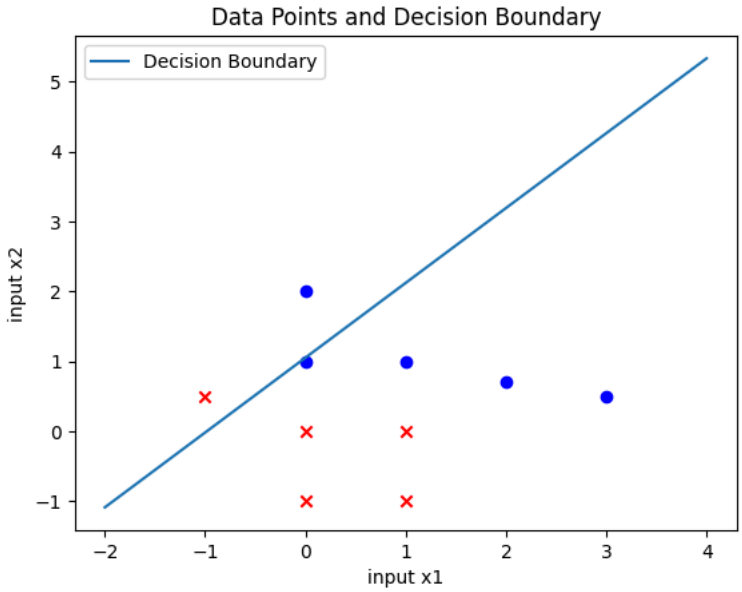


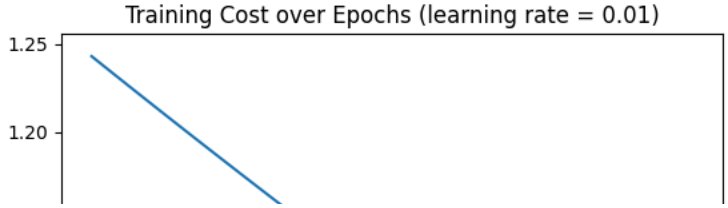

```
# Train the neural network with learning rate = 0.1
nn3 = NeuralNetwork(0.1)
history3 = nn3.train(inputs_with_bias, labels, 50)
plot_decision_boundary(nn3.weights, inputs_with_bias, labels)
plt.plot(history3['cost'])
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Training Cost over Epochs (learning rate = 0.1)')
plt.show()
```






```
# Train the neural network with learning rate = 0.01
nn4 = NeuralNetwork(0.01)
history4 = nn4.train(inputs_with_bias, labels, 50)
plot_decision_boundary(nn4.weights, inputs_with_bias, labels)
plt.plot(history4['cost'])
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Training Cost over Epochs (learning rate = 0.01)')
plt.show()
```






```
#Analysis of learning curves
print("Learning Curve Analysis:")
print("- Learning rate 1: Rapid initial decrease in cost, then leveling off.")
print("- Learning rate 0.5: Rapid decrease, more gradual than 1.")
print("- Learning rate 0.1: Gradual and steady decrease.")
print("- Learning rate 0.01: Very slow decrease.")
print("Optimal learning rate: 0.1, as it balances quick convergence and stability.")
```

```
Learning Curve Analysis:
- Learning rate 1: Rapid initial decrease in cost, then leveling off.
- Learning rate 0.5: Rapid decrease, more gradual than 1.
- Learning rate 0.1: Gradual and steady decrease.
- Learning rate 0.01: Very slow decrease.
Optimal learning rate: 0.1, as it balances quick convergence and stability.
```

Epochs

https://colab.research.google.com/drive/1ObYGbBQdjRYLmLR0n81L-If3B1PM1rYE?usp=sharing

Start coding or generate with AI.