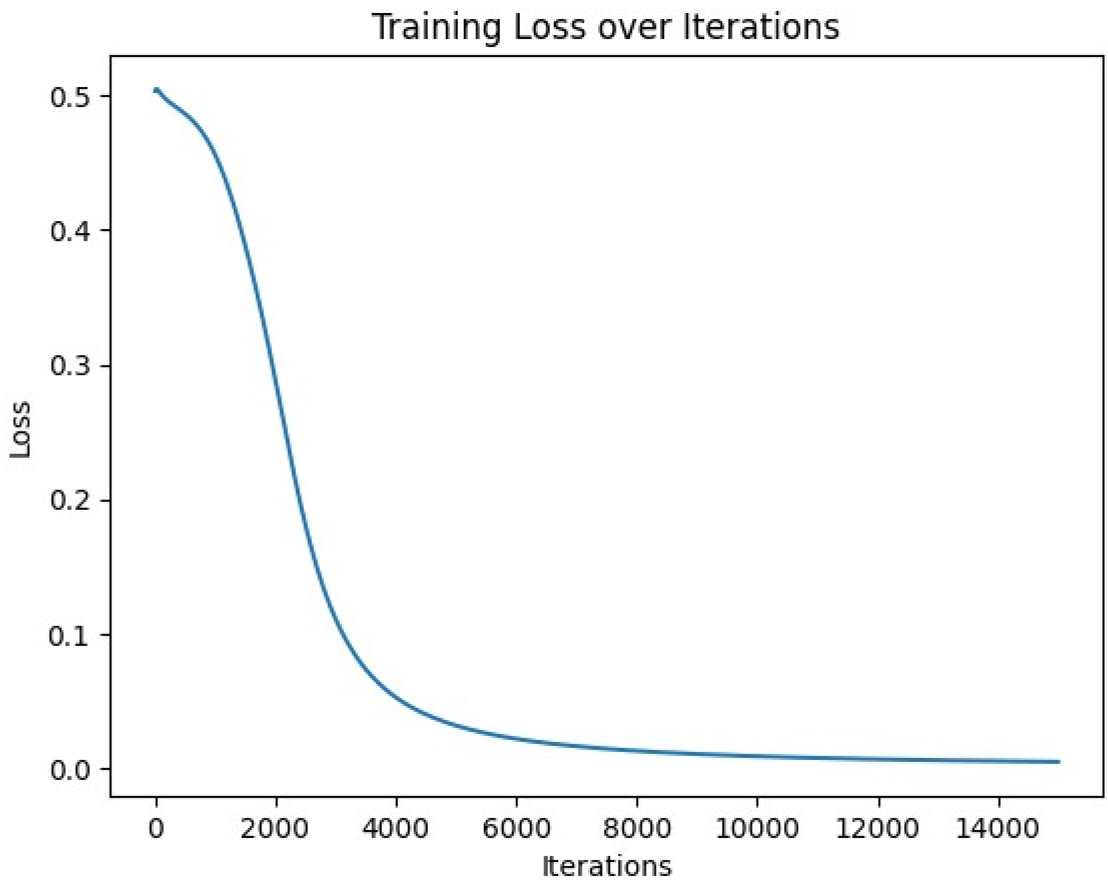
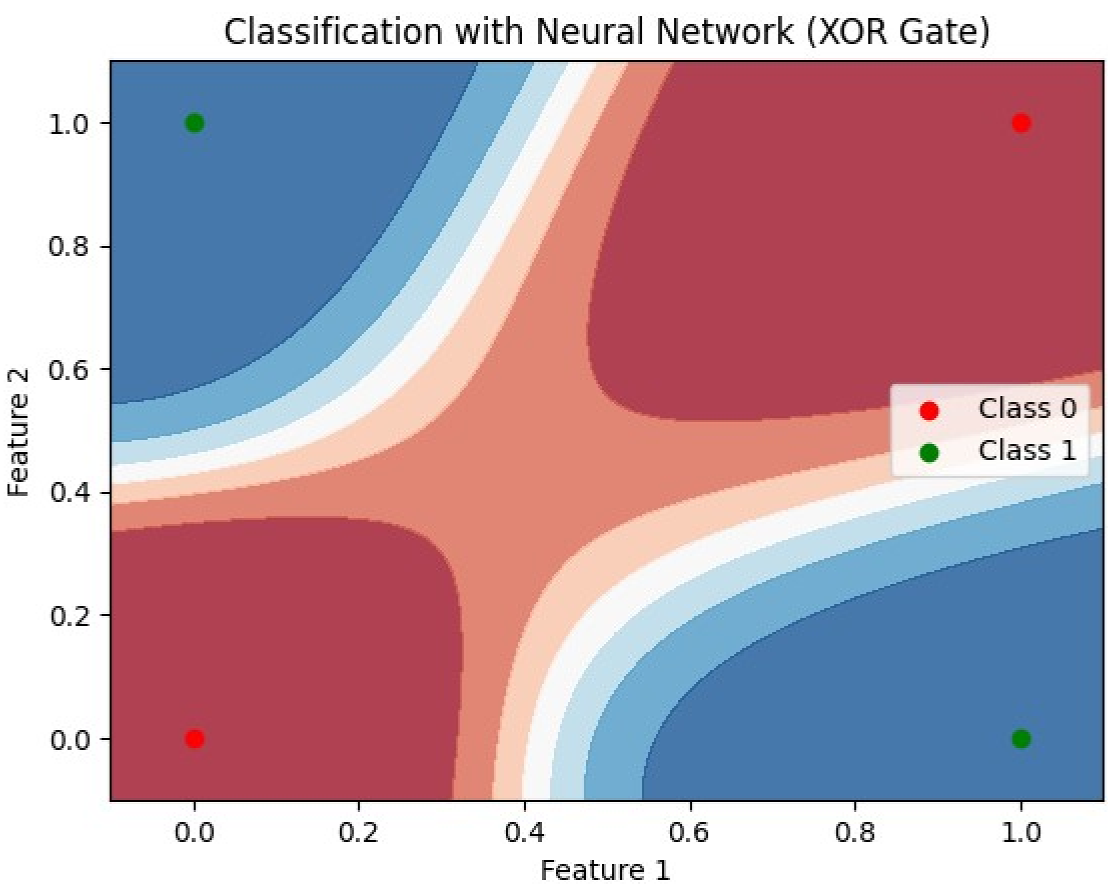
Sigmoid function

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

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

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

def forward(x, w1, w2, predict=False):
    a1 = np.dot(x, w1)
    z1 = sigmoid(a1)
    bias = np.ones((len(x), 1))
    z1 = np.concatenate((bias, z1), axis=1)
    a2 = np.dot(z1, w2)
    z2 = sigmoid(a2)
    if predict:
        return z2
    return a1, z1, a2, z2

def backprop(a2, X, z1, z2, y, w2, a1):
    delta2 = z2 - y
    Delta2 = np.dot(z1.T, delta2)
    delta1 = (delta2.dot(w2[1:, :].T)) * sigmoid_derivative(a1)
    Delta1 = np.dot(X.T, delta1)
    return delta2, delta1, Delta1, Delta2

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

Y = np.array([1, 1, 0, 0]).reshape(-1, 1)

w1 = np.random.randn(3, 5)
w2 = np.random.randn(6, 1)

lr = 0.1
costs = []
epochs = 15000
m = len(X)

for i in range(epochs):
    a1, z1, a2, z2 = forward(X, w1, w2)
    delta2, delta1, Delta1, Delta2 = backprop(a2, X, z1, z2, Y, w2, a1)
    w1 = w1 - lr * (1/m) * Delta1
    w2 = w2 - lr * (1/m) * Delta2
    c = np.mean(np.abs(delta2))
    costs.append(c)
```

```
predictions = forward(X, w1, w2, predict=True)
print("Predicted Output:")
for i in range(len(X)):
    print(f"Input: {X[i, 1:]}, Actual: {Y[i][0]}, Predicted Output: {predictions[i]
plt.plot(costs)
plt.title("Training Loss over Iterations")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()

x_min, x_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
y_min, y_max = X[:, 2].min() - 0.1, X[:, 2].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))

grid_data = np.c_[np.ones(xx.ravel().shape[0]), xx.ravel(), yy.ravel()]

Z = forward(grid_data, w1, w2, predict=True)
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.RdBu, alpha=0.8)
plt.scatter(X[Y.ravel() == 0, 1], X[Y.ravel() == 0, 2], color='red', label='Class 0
plt.scatter(X[Y.ravel() == 1, 1], X[Y.ravel() == 1, 2], color='green', label='Class

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Classification with Neural Network (XOR Gate)')
plt.legend()
plt.show()
```

```
Predicted Output:
Input: [1 0], Actual: 1, Predicted Output: 0.9945375912402531 ~~ 1
Input: [0 1], Actual: 1, Predicted Output: 0.9956548894241495 ~~ 1
Input: [0 0], Actual: 0, Predicted Output: 0.0023994825944017788 ~~ 0
Input: [1 1], Actual: 0, Predicted Output: 0.00720691140845269 ~~ 0
```


Training Loss over Iterations
0.5
0.4
0.3
0.2
0.1
0.0
Loss
0
2000
4000
6000
8000
10000
12000
14000
Iterations


Classification with Neural Network (XOR Gate)
1.0
0.8
0.6
0.4
0.2
0.0
Feature 2
Class 0
Class 1
0.0
0.2
0.4
0.6
0.8
1.0
Feature 1

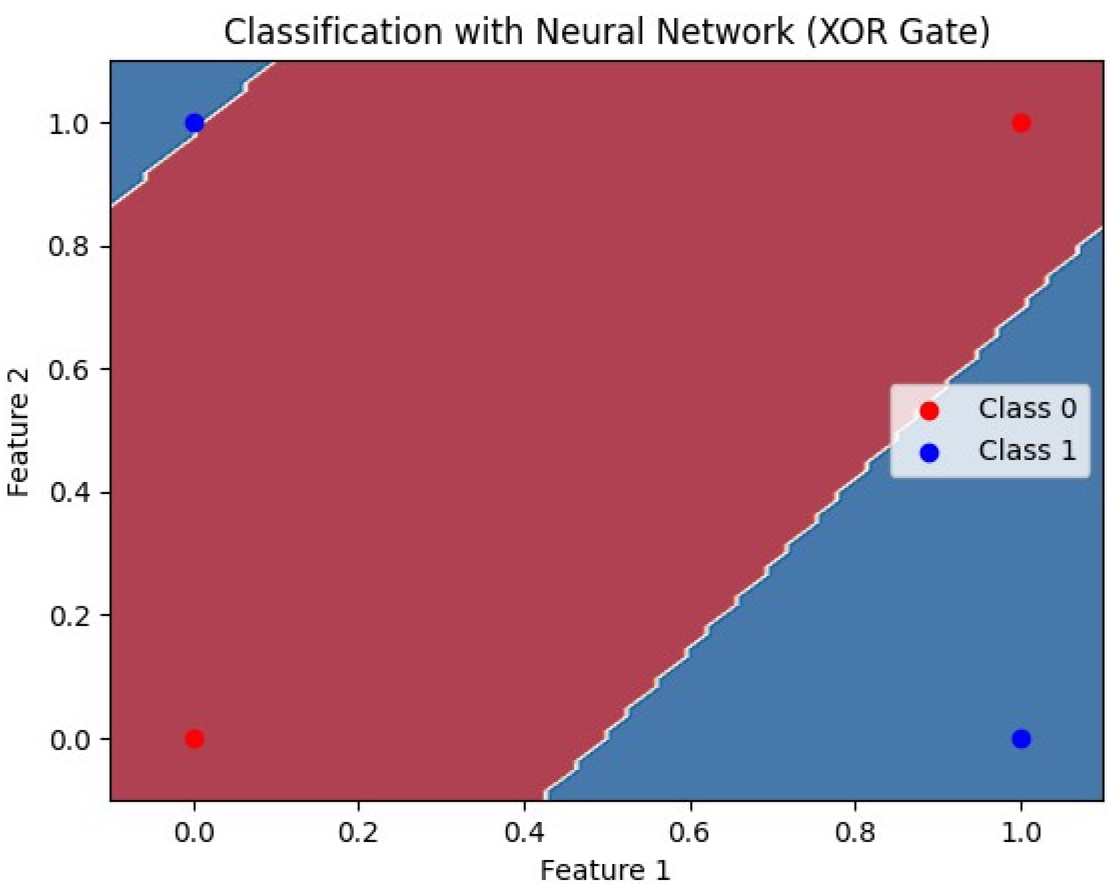
Unit Step Function

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def unit_step(x):
    return np.where(x > 0, 1, 0)

def forward(x, w1, w2, predict=False):
    a1 = np.dot(x, w1)
    z1 = unit_step(a1)
    bias = np.ones((len(x), 1))
    z1 = np.concatenate((bias, z1), axis=1)
    a2 = np.dot(z1, w2)
    z2 = unit_step(a2)
    if predict:
        return z2
    return a1, z1, a2, z2

def backprop(a2, X, z1, z2, y, w2, a1):
    delta2 = z2 - y
    Delta2 = np.dot(z1.T, delta2)
    delta1 = (delta2.dot(w2[1:, :].T)) * 1
    Delta1 = np.dot(X.T, delta1)
    return delta2, delta1, Delta1, Delta2

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

Y = np.array([1, 1, 0, 0]).reshape(-1, 1)

w1 = np.random.randn(3, 5)
w2 = np.random.randn(6, 1)

lr = 0.1
costs = []
epochs = 20000
m = len(X)

for i in range(epochs):
    a1, z1, a2, z2 = forward(X, w1, w2)
    delta2, delta1, Delta1, Delta2 = backprop(a2, X, z1, z2, Y, w2, a1)
    w1 = w1 - lr * (1/m) * Delta1
    w2 = w2 - lr * (1/m) * Delta2
    c = np.mean(np.abs(delta2))
    costs.append(c)

predictions = forward(X, w1, w2, predict=True)
print("Predicted Output:")
for i in range(len(X)):
    print(f"Input: {X[i, 1:]}, Actual: {Y[i][0]}, Predicted Output: {predictions[i]
```

```python
plt.plot(costs)
plt.title("Training Loss over Iterations")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()

x_min, x_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
y_min, y_max = X[:, 2].min() - 0.1, X[:, 2].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))

grid_data = np.c_[np.ones(xx.ravel().shape[0]), xx.ravel(), yy.ravel()]

Z = forward(grid_data, w1, w2, predict=True)
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.RdBu, alpha=0.8)
plt.scatter(X[Y.ravel() == 0, 1], X[Y.ravel() == 0, 2], color='red', label='Class 0
plt.scatter(X[Y.ravel() == 1, 1], X[Y.ravel() == 1, 2], color='blue', label='Class

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Classification with Neural Network (XOR Gate)')
plt.legend()
plt.show()
```

```
Predicted Output:
Input: [1 0], Actual: 1, Predicted Output: 1
Input: [0 1], Actual: 1, Predicted Output: 1
Input: [0 0], Actual: 0, Predicted Output: 0
Input: [1 1], Actual: 0, Predicted Output: 0
```


Training Loss over Iterations
Loss
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Iterations
0
2500
5000
7500
10000
12500
15000
17500
20000


Classification with Neural Network (XOR Gate)
Feature 2
Feature 1
0.0
0.2
0.4
0.6
0.8
1.0
Class 0
Class 1