```
import numpy as np
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
def sigmoid_derivative(z):
    return sigmoid(z) * (1 - sigmoid(z))
def forward_propagation(X, weights, biases):
    activations = [X]  # Store activations for each layer
    zs = []  # Store weighted sums for each layer
    for w, b in zip(weights, biases):
        z = np.dot(w, activations[-1]) + b  # Weighted sum
        a = sigmoid(z)  # Apply activation function
        zs.append(z)
        activations.append(a)
    return activations, zs
def back_propagation(y, activations, zs, weights):
    m = y.shape[1]  # Number of training samples
    L = len(weights)  # Number of layers
    d_weights = [np.zeros(w.shape) for w in weights]  # Gradients for weights
    d_biases = [np.zeros(b.shape) for b in biases]  # Gradients for biases
    dz = activations[-1] - y  # Error term
    d_weights[-1] = np.dot(dz, activations[-2].T) / m
    d_biases[-1] = np.sum(dz, axis=1, keepdims=True) / m
    for l in range(L - 2, -1, -1):
        dz = np.dot(weights[l + 1].T, dz) * sigmoid_derivative(zs[l])
        d_weights[l] = np.dot(dz, activations[l].T) / m
        d_biases[l] = np.sum(dz, axis=1, keepdims=True) / m
    return d_weights, d_biases
X = np.array([[0, 0, 1, 1],  # Inputs
              [0, 1, 0, 1]])  # Two features
y = np.array([[0, 1, 1, 0]])  # XOR Output
np.random.seed(42)
```

```
layer_sizes = [2, 3, 1]  # Input Layer (2), Hidden Layer (3), Output Layer (1)
weights = [np.random.randn(layer_sizes[i+1], layer_sizes[i]) for i in range(len(layer_sizes)-1)]
biases = [np.random.randn(layer_sizes[i+1], 1) for i in range(len(layer_sizes)-1)]
epochs = 10000
learning_rate = 0.1
for epoch in range(epochs):
    activations, zs = forward_propagation(X, weights, biases)
    d_weights, d_biases = back_propagation(y, activations, zs, weights)
    weights = [w - learning_rate * dw for w, dw in zip(weights, d_weights)]
    biases = [b - learning_rate * db for b, db in zip(biases, d_biases)]
    if epoch % 1000 == 0:
        loss = np.mean((activations[-1] - y) ** 2)
        print(f"Epoch {epoch}, Loss: {loss:.4f}")
activations, _ = forward_propagation(X, weights, biases)
print("Final Output After Training:")
print(activations[-1])
```

Outputs:

```
Epoch 0, Loss: 0.3596
Epoch 1000, Loss: 0.2370
Epoch 2000, Loss: 0.1761
Epoch 3000, Loss: 0.0798
Epoch 4000, Loss: 0.0135
Epoch 5000, Loss: 0.0036
Epoch 6000, Loss: 0.0015
Epoch 7000, Loss: 0.0008
Epoch 8000, Loss: 0.0005
Epoch 9000, Loss: 0.0003
Final Output
After Training:
[[0.01660678 0.98552904 0.98623307 0.0159944 ]]
```