In [1]:
```python
import numpy as np
```

In [2]:
```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

In [3]:
```python
def sigmoid_derivative(x):
    return x * (1 - x)
```

In [4]:
```python
def binary_cross_entropy(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred + 1e-7) + (1 - y_true) * np.log(1 - y_pred + 1e-7))
```

In [7]:
```python
   def train_neural_network(X, y, epochs=10000, lr=0.1):
    input_dim = X.shape[1]
    weights = np.random.uniform(size=(input_dim, 1))
    bias = np.random.uniform(size=(1,))

    for epoch in range(epochs):
        linear_output = np.dot(X, weights) + bias
        predictions = sigmoid(linear_output)

        loss = binary_cross_entropy(y, predictions)

        error = predictions - y
        d_pred = error * sigmoid_derivative(predictions)

        weights -= lr * np.dot(X.T, d_pred)
        bias -= lr * np.sum(d_pred)

        if epoch % 1000 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}")

    return weights, bias
```

In [8]:
```python
def predict(X, weights, bias):
    return sigmoid(np.dot(X, weights) + bias) >= 0.5
```

In [9]:
```python
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

logic_gates = {
    "AND": np.array([[0], [0], [0], [1]]),
    "OR": np.array([[0], [1], [1], [1]]),
    "NAND": np.array([[1], [1], [1], [0]]),
    "NOR": np.array([[1], [0], [0], [0]]),
    "XOR": np.array([[0], [1], [1], [0]])
}

for gate_name, y in logic_gates.items():
    print(f"\nTraining for {gate_name} gate:")
    weights, bias = train_neural_network(X, y, epochs=10000, lr=0.1)
    predictions = predict(X, weights, bias).astype(int)
    print(f"Predictions for {gate_name} gate:\n{predictions.reshape(-1)}")
```

```
Training for AND gate:
Epoch 0, Loss: 0.8243
Epoch 1000, Loss: 0.1756
Epoch 2000, Loss: 0.1176
Epoch 3000, Loss: 0.0924
Epoch 4000, Loss: 0.0779
Epoch 5000, Loss: 0.0683
Epoch 6000, Loss: 0.0614
Epoch 7000, Loss: 0.0562
Epoch 8000, Loss: 0.0520
Epoch 9000, Loss: 0.0486
Predictions for AND gate:
[0 0 0 1]

Training for OR gate:
Epoch 0, Loss: 0.4667
Epoch 1000, Loss: 0.1245
Epoch 2000, Loss: 0.0812
Epoch 3000, Loss: 0.0635
Epoch 4000, Loss: 0.0535
Epoch 5000, Loss: 0.0470
Epoch 6000, Loss: 0.0423
Epoch 7000, Loss: 0.0387
Epoch 8000, Loss: 0.0359
Epoch 9000, Loss: 0.0336
Predictions for OR gate:
[0 1 1 1]

Training for NAND gate:
Epoch 0, Loss: 0.6903
Epoch 1000, Loss: 0.1800
Epoch 2000, Loss: 0.1191
Epoch 3000, Loss: 0.0932
Epoch 4000, Loss: 0.0784
Epoch 5000, Loss: 0.0686
Epoch 6000, Loss: 0.0617
Epoch 7000, Loss: 0.0564
Epoch 8000, Loss: 0.0522
Epoch 9000, Loss: 0.0487
Predictions for NAND gate:
[1 1 1 0]

Training for NOR gate:
Epoch 0, Loss: 1.1931
Epoch 1000, Loss: 0.1304
Epoch 2000, Loss: 0.0830
Epoch 3000, Loss: 0.0644
Epoch 4000, Loss: 0.0541
Epoch 5000, Loss: 0.0474
Epoch 6000, Loss: 0.0426
Epoch 7000, Loss: 0.0389
Epoch 8000, Loss: 0.0361
Epoch 9000, Loss: 0.0337
Predictions for NOR gate:
[1 0 0 0]

Training for XOR gate:
Epoch 0, Loss: 0.7645
Epoch 1000, Loss: 0.6931
Epoch 2000, Loss: 0.6931
Epoch 3000, Loss: 0.6931
Epoch 4000, Loss: 0.6931
Epoch 5000, Loss: 0.6931
Epoch 6000, Loss: 0.6931
Epoch 7000, Loss: 0.6931
Epoch 8000, Loss: 0.6931
Epoch 9000, Loss: 0.6931
Predictions for XOR gate:
[1 1 1 1]
```

In [ ]: