```
Name: Aarya Admane
Rollno:22630
Div:B Batch:B2


import numpy as np

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

# Derivative of sigmoid for backpropagation
def sigmoid_derivative(x):
    return x * (1 - x)

# Mean Squared Error Loss function
def mse_loss(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)


# Initialize neural network parameters
np.random.seed(42)
input_size = 2   # Number of input neurons
hidden_size = 3  # Number of neurons in hidden layer
output_size = 1  # Number of output neurons


# Initialize weights and biases randomly
W1 = np.random.randn(input_size, hidden_size)  # Input to hidden layer
weights
b1 = np.random.randn(1, hidden_size)           # Bias for hidden layer
W2 = np.random.randn(hidden_size, output_size) # Hidden to output
weights
b2 = np.random.randn(1, output_size)           # Bias for output layer


# Training data (XOR problem)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs
y = np.array([[0], [1], [1], [0]])              # Expected output


# Training parameters
epochs = 10000  # Number of training iterations
learning_rate = 0.1  # Learning rate


# Training loop
for epoch in range(epochs):
    # Forward Propagation
    z1 = np.dot(X, W1) + b1   # Linear transformation (Input → Hidden)
    a1 = sigmoid(z1)          # Activation function (Hidden layer)
    z2 = np.dot(a1, W2) + b2  # Linear transformation (Hidden →
```

```
Output)
    a2 = sigmoid(z2)          # Activation function (Output layer)

  # Compute loss
    loss = mse_loss(y, a2)

  # Backpropagation (Calculating Gradients)

d_a2 = (a2 - y)  # Loss gradient
d_z2 = d_a2 * sigmoid_derivative(a2)  # Backprop through activation
d_W2 = np.dot(a1.T, d_z2)  # Weight gradient (hidden → output)
d_b2 = np.sum(d_z2, axis=0, keepdims=True)  # Bias gradient (output)

d_a1 = np.dot(d_z2, W2.T)  # Backprop to hidden layer
d_z1 = d_a1 * sigmoid_derivative(a1)  # Backprop through activation
d_W1 = np.dot(X.T, d_z1)  # Weight gradient (input → hidden)
d_b1 = np.sum(d_z1, axis=0, keepdims=True)  # Bias gradient (hidden)


    # Update weights and biases
    W2 -= learning_rate * d_W2
    b2 -= learning_rate * d_b2
    W1 -= learning_rate * d_W1
    b1 -= learning_rate * d_b1

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


# Final Predictions
print("\nFinal Predictions:")
print(sigmoid(np.dot(sigmoid(np.dot(X, W1) + b1), W2) + b2))


Final Predictions:
[[0.54553169]
 [0.57471011]
 [0.53845304]
 [0.56280605]]
```