```python
import numpy as np

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

# Derivative of Sigmoid
def sigmoid_derivative(x):
    return x * (1 - x)

# Input dataset (XOR problem)
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

# Output dataset
y = np.array([[0], [1], [1], [0]])

# Seed for reproducibility
np.random.seed(42)

# Initialize weights and biases
input_neurons = 2
hidden_neurons = 2
output_neurons = 1

# Random weight initialization
w1 = np.random.uniform(size=(input_neurons, hidden_neurons))
b1 = np.random.uniform(size=(1, hidden_neurons))
w2 = np.random.uniform(size=(hidden_neurons, output_neurons))
b2 = np.random.uniform(size=(1, output_neurons))

# Learning rate
lr = 0.5

# Training process
for epoch in range(10000):
    # Forward Propagation
    hidden_input = np.dot(X, w1) + b1
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, w2) + b2
    final_output = sigmoid(final_input)
```

```python
    # Backpropagation
    error = y - final_output
    d_output = error * sigmoid_derivative(final_output)

    error_hidden = d_output.dot(w2.T)
    d_hidden = error_hidden * sigmoid_derivative(hidden_output)

    # Update weights and biases
    w2 += hidden_output.T.dot(d_output) * lr
    b2 += np.sum(d_output, axis=0, keepdims=True) * lr
    w1 += X.T.dot(d_hidden) * lr
    b1 += np.sum(d_hidden, axis=0, keepdims=True) * lr

# Final output after training
print("Final output after training:")
print(final_output)
```

```
Final output after training:
[[0.01890475]
 [0.98371361]
 [0.98369334]
 [0.01686123]]
```