Experiment 1

```python
import numpy as np
# Sigmoid function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
# XOR input and output
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[0],
              [1],
              [1],
              [0]])
# Initialize weights and biases randomly
np.random.seed(42)
inputLayerNeurons = 2
hiddenLayerNeurons = 2
outputNeurons = 1

# Weights and bias initialization
hidden_weights = np.random.uniform(size=(inputLayerNeurons, hiddenLayerNeurons))
hidden_bias = np.random.uniform(size=(1, hiddenLayerNeurons))
output_weights = np.random.uniform(size=(hiddenLayerNeurons, outputNeurons))
output_bias = np.random.uniform(size=(1, outputNeurons))
# Learning rate
lr = 0.1
epochs = 10000
# Training the neural network
for _ in range(epochs):
    # Forward Propagation
    hidden_layer_input = np.dot(X, hidden_weights) + hidden_bias
    hidden_layer_output = sigmoid(hidden_layer_input)
    final_input = np.dot(hidden_layer_output, output_weights) + output_bias
    final_output = sigmoid(final_input)

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

    error_hidden = d_output.dot(output_weights.T)
    d_hidden = error_hidden * sigmoid_derivative(hidden_layer_output)
    # Update weights and biases
    output_weights += hidden_layer_output.T.dot(d_output) * lr
    output_bias += np.sum(d_output, axis=0, keepdims=True) * lr
    hidden_weights += X.T.dot(d_hidden) * lr
    hidden_bias += np.sum(d_hidden, axis=0, keepdims=True) * lr
# Final Output
print("Final Output after training:")
print(np.round(final_output, 3))
```

```
Final Output after training:
[[0.053]
[0.952]
[0.952]
[0.052]]
```