4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Program:

```
import random
import math
# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
# Derivative of sigmoid
def sigmoid_derivative(x):
    return x * (1 - x)
# Initialize weights randomly
def initialize_network():
    network = []
    hidden_layer = [{'weights':[random.random() for i in range(3)]} for i in range(2)]
    output_layer = [{'weights':[random.random() for i in range(3)]}]
    network.append(hidden_layer)
    network.append(output_layer)
    return network
# Forward propagation
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = neuron['weights'][-1]  # bias
            for i in range(len(neuron['weights'])-1):
                activation += neuron['weights'][i] * inputs[i]
            neuron['output'] = sigmoid(activation)
```

```
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs
# Backward propagation
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = []
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i+1]:
                    error += neuron['weights'][j] * neuron['delta']
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * sigmoid_derivative(neuron['output'])
# Update weights
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1] if i == 0 else [neuron['output'] for neuron in network[i-1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']
```

```
# Train network
def train_network(network, train, l_rate, n_epoch):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row[:-1])
            expected = [row[-1]]
            sum_error += (expected[0] - outputs[0])**2
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        if epoch % 1000 == 0:
            print("Epoch:", epoch, "Error:", sum_error)
# Predict
def predict(network, row):
    outputs = forward_propagate(network, row)
    return round(outputs[0])
# XOR dataset
dataset = [
    [0,0,0],
    [0,1,1],
    [1,0,1],
    [1,1,0]
]
# Initialize and train
network = initialize_network()
train_network(network, dataset, 0.5, 5000)
# Testing
print("\nTesting Network:")
for row in dataset:
```

```
    prediction = predict(network, row[:-1])
    print("Input:", row[:-1], "Expected:", row[-1], "Predicted:", prediction)
```

Output:

```
========== RESTART: C:/Users/saimo/OneDrive/Desktop/ML Programs/4.py ==========
Epoch: 0 Error: 1.3872385275537455
Epoch: 1000 Error: 0.06926015350983503
Epoch: 2000 Error: 0.011024343839106322
Epoch: 3000 Error: 0.0056350201704432855
Epoch: 4000 Error: 0.0037313190071551075

Testing Network:
Input: [0, 0] Expected: 0 Predicted: 0
Input: [0, 1] Expected: 1 Predicted: 1
Input: [1, 0] Expected: 1 Predicted: 1
Input: [1, 1] Expected: 0 Predicted: 0
```