In [18]:

```python
#program3
#1bm22ai037
#How does a feed forward neural network with multi layer perceptron architecture sol
import numpy as np
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
input_layer_neurons = 2
hidden_layer_neurons = 2
output_neurons = 1
learning_rate = 0.1
epochs = 10000
np.random.seed(42)
weights_input_hidden = np.random.uniform(size=(input_layer_neurons, hidden_layer_neu
weights_hidden_output = np.random.uniform(size=(hidden_layer_neurons, output_neurons
for epoch in range(epochs):
    hidden_layer_input = np.dot(X, weights_input_hidden)
    hidden_layer_output = sigmoid(hidden_layer_input)
    final_input = np.dot(hidden_layer_output, weights_hidden_output)
    final_output = sigmoid(final_input)
    error = y - final_output
    d_final_output = error * sigmoid_derivative(final_output)
    error_hidden_layer = d_final_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)
    weights_hidden_output += hidden_layer_output.T.dot(d_final_output) * learning_ra
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate
hidden_layer_output = sigmoid(np.dot(X, weights_input_hidden))
final_output = sigmoid(np.dot(hidden_layer_output, weights_hidden_output))
predictions = np.round(final_output).astype(int)
print("Predictions:")
for i in range(len(X)):
    print(f"Input: {X[i]} => Predicted Output: {predictions[i][0]}, Actual Output: {
```

```
Predictions:
Input: [0 0] => Predicted Output: 0, Actual Output: 0
Input: [0 1] => Predicted Output: 1, Actual Output: 1
Input: [1 0] => Predicted Output: 1, Actual Output: 1
Input: [1 1] => Predicted Output: 0, Actual Output: 0
```

In [17]:

```
#how does a feedforward neural network with a multilayer perceptron architecture sol
#do non-linear activation functions play in enabling the network to learn non-linear
import numpy as np
class NeuralNetwork:
    def __init__(self):
        self.W1 = np.random.rand(2, 2)
        self.b1 = np.random.rand(2)
        self.W2 = np.random.rand(2, 1)
        self.b2 = np.random.rand(1)
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
    def sigmoid_derivative(self, x):
        return x * (1 - x)
    def forward(self, X):
        self.z1 = np.dot(X, self.W1) + self.b1
        self.a1 = self.sigmoid(self.z1)
        self.z2 = np.dot(self.a1, self.W2) + self.b2
        self.output = self.sigmoid(self.z2)
        return self.output
    def backward(self, X, y, output):
        self.error = y - output
        self.d_output = self.error * self.sigmoid_derivative(output)
        self.error_hidden = self.d_output.dot(self.W2.T)
        self.d_hidden = self.error_hidden * self.sigmoid_derivative(self.a1)
        self.W2 += self.a1.T.dot(self.d_output)
        self.b2 += np.sum(self.d_output, axis=0)
        self.W1 += X.T.dot(self.d_hidden)
        self.b1 += np.sum(self.d_hidden, axis=0)
    def train(self, X, y, epochs):
        for _ in range(epochs):
            output = self.forward(X)
            self.backward(X, y, output)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
nn = NeuralNetwork()
nn.train(X, y, 10000)
output = nn.forward(X)
print(output)
```

```
[[0.01303231]
 [0.98885694]
 [0.98885053]
 [0.01146717]]
```

In [ ]: