Assignment 7

In [2]:
```
import numpy as np

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

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

# Define XOR Function Dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Define Neural Network Architecture
input_layer_size = 2
hidden_layer_size = 2
output_layer_size = 1

# Initialize Weights and Biases
weights1 = np.random.uniform(size=(input_layer_size, hidden_layer_size))
bias1 = np.random.uniform(size=(1, hidden_layer_size))
weights2 = np.random.uniform(size=(hidden_layer_size, output_layer_size))
bias2 = np.random.uniform(size=(1, output_layer_size))

# Define Learning Rate and Number of Epochs
learning_rate = 0.1
epochs = 10000

# Train Neural Network with Back Propagation Algorithm
for epoch in range(epochs):
    # Forward Propagation
    hidden_layer_output = sigmoid(np.dot(X, weights1) + bias1)
    output_layer_output = sigmoid(np.dot(hidden_layer_output, weights2) + bias2)

    # Back Propagation
    error = y - output_layer_output
    output_layer_delta = error * sigmoid_derivative(output_layer_output)
    hidden_layer_error = output_layer_delta.dot(weights2.T)
    hidden_layer_delta = hidden_layer_error * sigmoid_derivative(hidden_layer_out

    # Update Weights and Biases
    weights2 += hidden_layer_output.T.dot(output_layer_delta) * learning_rate
    bias2 += np.sum(output_layer_delta, axis=0, keepdims=True) * learning_rate
    weights1 += X.T.dot(hidden_layer_delta) * learning_rate
    bias1 += np.sum(hidden_layer_delta, axis=0, keepdims=True) * learning_rate

# Predict XOR Function
predictions = sigmoid(np.dot(sigmoid(np.dot(X, weights1) + bias1), weights2) + bi
print(predictions.round())
```

```
[[0.]
 [1.]
 [1.]
 [0.]]
```