In [18]:

```
import numpy as np
```

In [19]:

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

In [20]:

```
def initialization(input_feat, hidden_neurons, output_feat):
    W1 = np.random.randn(hidden_neurons, input_feat)
    W2 = np.random.randn(output_feat, hidden_neurons)
    b1 = np.zeros((hidden_neurons, 1))
    b2 = np.zeros((output_feat, 1))

    parameters = {"W1" : W1, "b1": b1,
                  "W2" : W2, "b2": b2}
    return parameters
```

In [21]:

```
def forward_propagation(X, Y, parameters):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = (Z1, A1, W1, b1, Z2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 - Y))
    cost = -np.sum(logprobs) / m
    return cost, cache, A2
```

In [22]:

```
def backward_propagation(X, Y, cache):
    m = X.shape[1]
    (Z1, A1, W1, b1, Z2, A2, W2, b2) = cache

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis = 1, keepdims = True)

    dA1 = np.dot(W2.T, dZ2)
    dZ1 = np.multiply(dA1, A1 * (1- A1))
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis = 1, keepdims = True) / m

    gradients = {"dZ2": dZ2, "dW2": dW2, "db2": db2,
                 "dZ1": dZ1, "dW1": dW1, "db1": db1}
    return gradients
```

In [23]:

```
def updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"] - learningRate * gradients["dW1"]
    parameters["W2"] = parameters["W2"] - learningRate * gradients["dW2"]
    parameters["b1"] = parameters["b1"] - learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"]
    return parameters
```

In [24]:

```
X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])
Y = np.array([[0, 1, 1, 0]])

hidden_neurons = 2
input_feat = X.shape[0]
output_feat = Y.shape[0]
parameters = initialization(input_feat, hidden_neurons, output_feat)
epoch = 100000
learningRate = 0.01
losses = np.zeros((epoch, 1))

for i in range(epoch):
    losses[i, 0], cache, A2 = forward_propagation(X, Y, parameters)
    gradients = backward_propagation(X, Y, cache)
    parameters = updateParameters(parameters, gradients, learningRate)
```

In [25]:

```
X = np.array([[1, 1, 0, 0], [0, 1, 0, 1]]) # XOR input
cost, _, A2 = forward_propagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0

print(prediction)
```

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