In [11]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [12]:

```
import random
import math

random.seed(1)
```

In [13]:

```
sigmoid = lambda x: 1.0 / (1.0 + math.exp(-x))

sigmoid_deriv = lambda x: sigmoid(x) * (1.0 - sigmoid(x))
```

In [14]:

```
def preprocess(data):
    data_list = list()
    target_list = list()

    for i in range(len(data)):
        data_list.append(data[i][:2])

    for i in range(len(data)):
        label = list()

        if (data[i][2] == 0):
            label.append(0)
            label.append(1)
        else:
            label.append(1)
            label.append(0)
        target_list.append(label)

    return data_list, target_list
```

In [15]:

```
class Node:
    def __init__(self, front_node) -> None:
        self.weight = list()
        self.errors = list()
        self.delta = list()
        self.output = 0.0

        for i in range(front_node + 1):
            self.weight.append(random.random())
            self.errors.append(0)
            self.delta.append(0)
```

In [16]:

```
class NN:
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate) -> None:
        self.hidden = list()
        self.out = list()
        self.lr = learningrate
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        for i in range(self.hnodes):
            self.hidden.append(Node(self.inodes))

        for i in range(self.onodes):
            self.out.append(Node(self.hnodes))

    def forward(self, data_list):
        for i in range(len(self.hidden)):
            for j in range(len(data_list)):
                self.hidden[i].output = self.hidden[i].output + self.hidden[i].weight[j] * data_list
            self.hidden[i].output = self.hidden[i].output + self.hidden[i].weight[-1]

        for i in range(len(self.out)):
            for j in range(len(self.out)):
                self.out[i].output = self.out[i].output + self.out[i].weight[j] * sigmoid(self.hidde
            self.out[i].output = self.out[i].output + self.out[i].weight[-1]

    def backward(self, data_list, target_list):
        for i in range(len(self.out)):
            for j in range(len(self.hidden)):
                self.out[i].delta[j] = (sigmoid(self.out[i].output) - target_list[i]) * sigmoid_deri
                self.out[i].errors[j] = self.out[i].delta[j] * sigmoid(self.hidden[j].output)
            self.out[i].delta[-1] = (sigmoid(self.out[i].output) - target_list[i]) * sigmoid_deriv(s
            self.out[i].errors[-1] = self.out[i].delta[-1] * 1

        for i in range(len(self.hidden)):
            for j in range(len(data_list)):
                delta = 0.0
                for k in range(len(self.out)):
                    delta = delta + self.out[k].delta[i] * self.out[k].weight[i]
                self.hidden[i].delta[-1] = delta * sigmoid_deriv(self.hidden[i].output)
                self.hidden[i].errors[-1] = self.hidden[i].delta[-1]

            delta = 0.0
            for k in range(len(self.out)):
                delta = delta + self.out[k].delta[i] * self.out[k].weight[i]
            self.hidden[i].delta[j] = delta * sigmoid_deriv(self.hidden[i].output)
            self.hidden[i].errors[j] = self.hidden[i].delta[j] * data_list[j]

    def update_w(self):
        for i in range(len(self.out)):
            for j in range(len(self.out[i].weight)):
                self.out[i].weight[j] = self.out[i].weight[j] - self.lr * self.out[i].errors[j]

        for i in range(len(self.hidden)):
            for j in range(len(self.hidden[i].weight)):
                self.hidden[i].weight[j] = self.hidden[i].weight[j] - self.lr * self.hidden[i].error

    def init_node(self):
        for i in range(len(self.hidden)):
```

```
            self.hidden[i].output = 0.0
            for j in range(len(self.hidden[i].weight)):
                self.hidden[i].errors[j] = 0.0
                self.hidden[i].delta[j] = 0.0

        for i in range(len(self.out)):
            self.out[i].output = 0.0
            for j in range(len(self.out[i].weight)):
                self.out[i].errors[j] = 0.0
                self.out[i].delta[j] = 0.0
```

In [17]:

```
input_nodes = 2
hidden_nodes = 2
output_nodes = 2

learning_rate = 0.5

n = NN(input_nodes, hidden_nodes, output_nodes, learning_rate)
```

In [18]:

```
training_data_list = [[3.5064385449265267, 2.34547092892632525, 0],
                      [4.384621956392097, 3.4530853889904205, 0],
                      [4.841442919897487, 4.02507852317520154, 0],
                      [3.5985868973088437, 4.1621314217538705, 0],
                      [2.887219775424049, 3.31523082529190005, 0],
                      [9.79822645535526, 1.1052409596099566, 1],
                      [7.8261241795117422, 0.6711054766067182, 1],
                      [2.5026163932400305, 5.800780055043912, 1],
                      [5.032436157202415, 8.650625621472184, 1],
                      [4.095084253434162, 7.69104329159447, 1]]
```

In [19]:

```
data_list, target_list = preprocess(training_data_list)
```

In [20]:

```
epoch = 10000

for i in range(epoch):
    MSE = 0.0
    for j in range(len(data_list)):
        n.forward(data_list[j])
        for k in range(len(n.out)):
            MSE = MSE + (target_list[j][k] - sigmoid(n.out[k].output)) ** 2

        n.backward(data_list[j], target_list[j])
        n.update_w()
        n.init_node()

    if (i % 1000 == 0):
        print("epoch:", i)
        print("error:", MSE/20)
```

```
epoch: 0
error: 0.32071244083404277
epoch: 1000
error: 0.0020196216872093546
epoch: 2000
error: 0.0007246857863503646
epoch: 3000
error: 0.0004341774015649732
epoch: 4000
error: 0.00030805909501916616
epoch: 5000
error: 0.00023799551671978723
epoch: 6000
error: 0.00019355194844099943
epoch: 7000
error: 0.00016290720122355917
epoch: 8000
error: 0.00014052727583017217
epoch: 9000
error: 0.00012348149470233502
```

In [21]:

```
test_data_list = [[3.5064385449265267, 2.34547092892632525, 0],
                  [4.384621956392097, 3.4530853889904205, 0],
                  [4.841442919897487, 4.02507852317520154, 0],
                  [3.5985868973088437, 4.1621314217538705, 0],
                  [2.887219775424049, 3.31523082529190005, 0],
                  [9.79822645535526, 1.1052409596099566, 1],
                  [7.8261241795117422, 0.6711054766067182, 1],
                  [2.5026163932400305, 5.800780055043912, 1],
                  [5.032436157202415, 8.650625621472184, 1],
                  [4.095084253434162, 7.69104329159447, 1]]
```

In [22]:

```
data_list, target_list = preprocess(test_data_list)
```

In [23]:

```
scorecard = []
sum = 0

for i in range(len(data_list)):
    correct_label = target_list[i]
    n.forward(data_list[i])

    if ((sigmoid(n.out[0].output) > 0.5) and (sigmoid(n.out[1].output) < 0.5)):
        label = [1, 0]
    else:
        label = [0, 1]

    if label == correct_label:
        scorecard.append(1)
    else:
        scorecard.append(0)
        pass
    pass
    print(correct_label)
    print(label)
    print()

    n.init_node()

for i in range(len(scorecard)):
    sum += scorecard[i]
print("performance:", sum / len(scorecard))
```

```
[0, 1]
[0, 1]

[0, 1]
[0, 1]

[0, 1]
[0, 1]

[0, 1]
[0, 1]

[0, 1]
[0, 1]

[1, 0]
[1, 0]

[1, 0]
[1, 0]

[1, 0]
[1, 0]

[1, 0]
[1, 0]

[1, 0]
```

```
[1, 0]

performance: 1.0
```