In [147]:

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

In [429]:

```
import random
import numpy as np
import pandas as pd
import scipy.special
import matplotlib.pyplot as plt
```

In [600]:

```
class neuralNetwork:
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # make nodes. input nodes, hidden nodes, output nodes
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # make weights. by normal distribution
        self.wih = np.random.normal(0.5, pow(self.hnodes, -0.5), (self.hnodes, self.inodes))
        self.who = np.random.normal(0.5, pow(self.onodes, -0.5), (self.onodes, self.hnodes))

        self.output_delta = np.zeros(outputnodes) + 1e-7
        self.hidden_delta = np.zeros(outputnodes) + 1e-7

        # set learning rate
        self.lr = learningrate

        self.sigmoid = lambda x: 1 / (1 + np.exp(-x))
        self.grad_sigmoid = lambda x: self.sigmoid(x) * (1.0 - self.sigmoid(x))

        self.getError = lambda y, t: t - y
        self.mse = lambda y, t: (1/2) * np.sum((y - t) ** 2)
        self.cross_entropy = lambda y, t, d: -np.sum(t * np.log(y + d)) / y.shape[0]

        pass

    def train(self, inputs_list, targets_list):
        inputs = np.array(inputs_list, ndmin = 2).T
        targets = np.array(targets_list, ndmin = 2).T

        hidden_inputs = np.dot(self.wih, inputs)
        hidden_outputs = self.sigmoid(hidden_inputs)

        final_inputs = np.dot(self.who, hidden_outputs)
        final_outputs = self.sigmoid(final_inputs)

        #output_errors = self.cross_entropy(final_outputs, targets, 1e-7)
        #hidden_errors = np.dot(self.who.T, output_errors)

        self.output_delta = np.dot(self.sigmoid(final_outputs) - targets, np.transpose(self.gra
        output_errors = np.dot(self.output_delta, np.transpose(self.sigmoid(hidden_outputs))

        delta = np.dot(self.output_delta, self.wih)
        self.hidden_delta = np.dot(delta, self.grad_sigmoid(hidden_outputs))
        hidden_errors = np.dot(self.hidden_delta, np.transpose(inputs))


        self.who = self.who - self.lr * output_errors
        self.wih = self.wih - self.lr * hidden_errors

        """t - y version
        # get error
        output_errors = self.getError(final_outputs, targets)
        hidden_errors = np.dot(self.who.T, output_errors)

        # update weight
        self.who += self.lr * np.dot((output_errors * final_outputs * (1.0 - final_outputs)), n
        self.wih += self.lr * np.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)),
        """

        pass

    def query(self, inputs_list):
        inputs = np.array(inputs_list, ndmin = 2).T

        hidden_inputs = np.dot(self.wih, inputs)
        hidden_outputs = self.sigmoid(hidden_inputs)

        final_inputs = np.dot(self.who, hidden_outputs)
        final_outputs = self.sigmoid(final_inputs)

        return final_outputs
```

```
  File "<ipython-input-600-cd00d3550712>", line 43
    delta = np.dot(self.output_delta, self.wih)
    ^
SyntaxError: invalid syntax
```

In [595]:

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

learning_rate = 0.5

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

In [596]:

```
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 [597]:

```
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 [598]:

```
epochs = 5000

for i in range(epochs):
    for record in training_data_list:
        all_values = record

        inputs = (np.asfarray(all_values[0:2]))

        targets = np.zeros(output_nodes) + 0.1

        targets[int(all_values[2])] = 0.9

        n.train(inputs, targets)
        pass
    if (i % 1000 == 0):
        print("----------------------------------------")
        print("epochs:", i)
        all_values
        inputs
        targets

        scorecard = []

        for record_ in test_data_list:
            all_values_ = record_
            correct_label_ = int(all_values_[2])
            inputs_ = (np.asfarray(all_values_[0:2]))
            outputs_ = n.query(inputs_)
            label_ = np.argmax(outputs_)

            outputs_

            print(correct_label_, "      correct label")
            print(label_, "       prediction₩n")

            plt.plot([correct_label_, label_], [0, 1])
            plt.show()

            if label_ == correct_label_:
                scorecard.append(1)
            else:
                scorecard.append(0)
                pass
            pass

        scorecard_array = np.asarray(scorecard)
        print("performance =", scorecard_array.sum() / scorecard_array.size, "₩n₩n₩n")
pass
```

```
---------------------------------------------------------------------------
----
ValueError                                Traceback (most recent call last)
<ipython-input-598-16dbb908d4e8> in <module>
     11         targets[int(all_values[2])] = 0.9
     12 
---> 13         n.train(inputs, targets)
     14         pass
     15     if (i % 1000 == 0):

<ipython-input-588-0e5b48c71836> in train(self, inputs_list, targets_lis
t)
     40         self.outptu_delta = np.dot(self.sigmoid(final_outputs) - targets
, np.transpose(self.grad_sigmoid(final_outputs)))
     41
---> 42         output_errors = np.dot(self.output_delta, np.transpose(self.si
gmoid(hidden_outputs)))
     43
     44         delta = np.dot(self.output_delta, self.wih)

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (2,) and (1,2) not aligned: 2 (dim 0) != 1 (dim 0)
```

In [599]:

```
scorecard = []

for record in test_data_list:
    all_values = record
    correct_label = int(all_values[2])
    inputs = (np.asfarray(all_values[0:2]))
    outputs = n.query(inputs)
    label = np.argmax(outputs)
    if label == correct_label:
        scorecard.append(1)
    else:
        scorecard.append(0)
        pass
    pass

scorecard_array = np.asarray(scorecard)
print("performance =", scorecard_array.sum() / scorecard_array.size)
```

```
performance = 0.5
```

In [ ]:

```

```

In [ ]:

```

```