北京师范大学2018-2019学年第一学期 “模式识别”

姓 名	学 号	日 期
刘源	201611210134	2018.11.28
仲兆威	201611210139	2018.11.28
梁崎霖	201611210201	2018.11.28

实验题目：

第 1 题

设计一个三维模式三类问题的BP网络，用下表中数据中的一部分训练BP网络，数据余下的一部分测试BP网络。

样本	ω_1			ω_2			ω_3		
	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3
1	1.58	2.32	−5.8	0.21	0.03	−2.21	−1.54	1.17	0.64
2	0.67	1.58	−4.78	0.37	0.28	−1.8	5.41	3.45	−1.33
3	1.04	1.01	−3.63	0.18	1.22	0.16	1.55	0.99	2.69
4	−1.49	2.18	−3.39	−0.24	0.93	−1.01	1.86	3.19	1.51
5	−0.41	1.21	−4.73	−1.18	0.39	−0.39	1.68	1.79	−0.87
6	1.39	3.16	2.87	0.74	0.96	−1.16	3.51	−0.22	−1.39
7	1.20	1.40	−1.89	−0.38	1.94	−0.48	1.40	−0.44	0.92
8	−0.92	1.44	−3.22	0.02	0.72	−0.17	0.44	0.83	1.97
9	0.45	1.33	−4.38	0.44	1.31	−0.14	0.25	0.68	−0.99
10	−0.76	0.84	−1.96	0.46	1.49	0.68	−0.66	−0.45	0.08

源代码

```
import math
import random

random.seed(0)


def rand(a, b):
    return (b - a) * random.random() + a
```

```
def make_matrix(m, n, fill=0.0):
    mat = []
    for i in range(m):
        mat.append([fill] * n)
    return mat


def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))


def sigmoid_derivative(x):
    return x * (1 - x)


class BPNeuralNetwork:
    def __init__(self):
        self.input_n = 0
        self.hidden_n = 0
        self.output_n = 0
        self.input_cells = []
        self.hidden_cells = []
        self.output_cells = []
        self.input_weights = []
        self.output_weights = []
        self.input_correction = []
        self.output_correction = []

    def setup(self, ni, nh, no):
        self.input_n = ni + 1
        self.hidden_n = nh
        self.output_n = no
        # init cells
        self.input_cells = [1.0] * self.input_n
        self.hidden_cells = [1.0] * self.hidden_n
        self.output_cells = [1.0] * self.output_n
        # init weights
        self.input_weights = make_matrix(self.input_n, self.hidden_n)
        self.output_weights = make_matrix(self.hidden_n, self.output_n)
        # random activate
        for i in range(self.input_n):
            for h in range(self.hidden_n):
                self.input_weights[i][h] = rand(-0.2, 0.2)
        for h in range(self.hidden_n):
            for o in range(self.output_n):
                self.output_weights[h][o] = rand(-2.0, 2.0)
        # init correction matrix
        self.input_correction = make_matrix(self.input_n, self.hidden_n)
        self.output_correction = make_matrix(self.hidden_n, self.output_n)

    def predict(self, inputs):
        # activate input layer
```

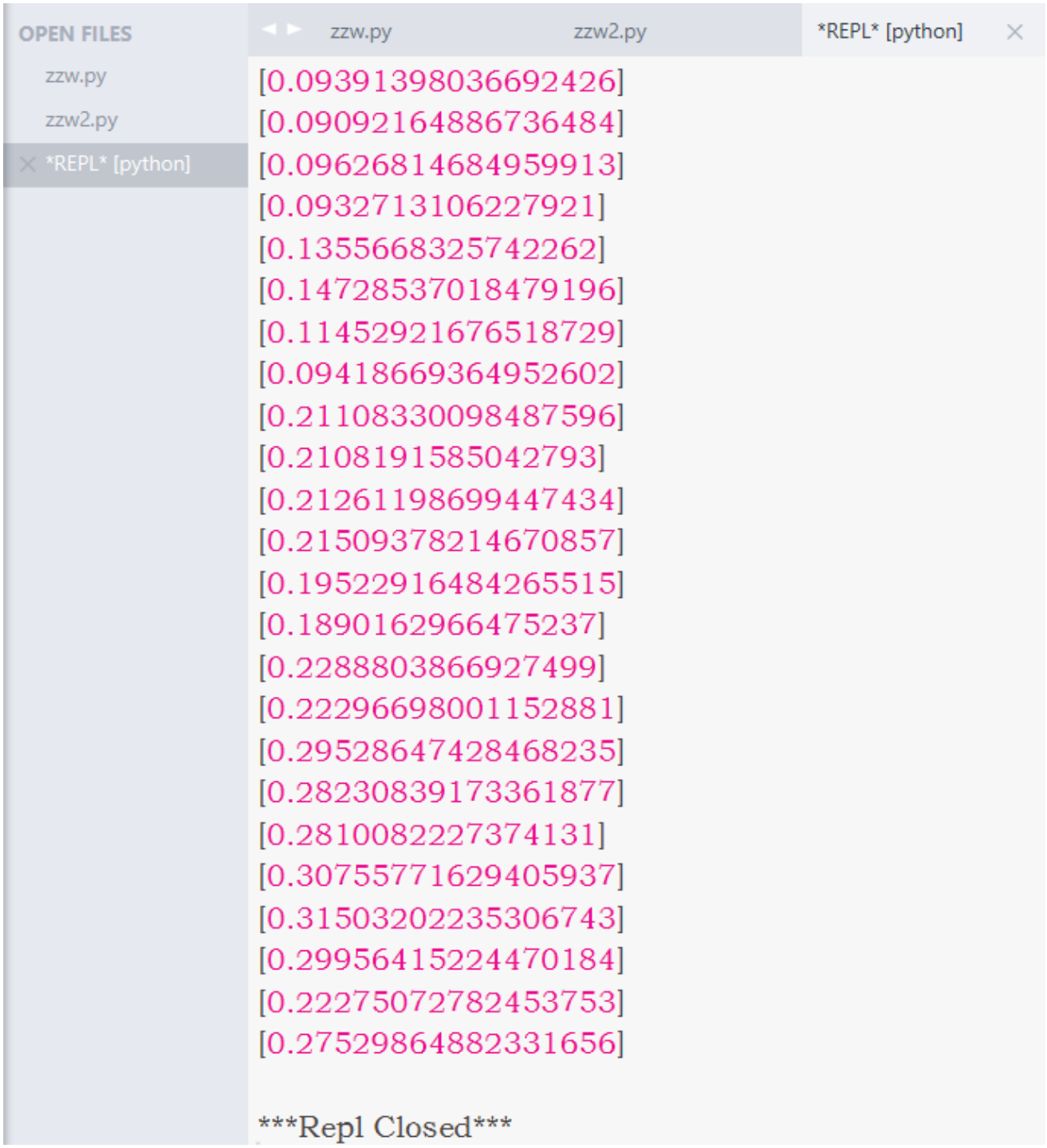
```
        for i in range(self.input_n - 1):
            self.input_cells[i] = inputs[i]
        # activate hidden layer
        for j in range(self.hidden_n):
            total = 0.0
            for i in range(self.input_n):
                total += self.input_cells[i] * self.input_weights[i][j]
            self.hidden_cells[j] = sigmoid(total)
        # activate output layer
        for k in range(self.output_n):
            total = 0.0
            for j in range(self.hidden_n):
                total += self.hidden_cells[j] * self.output_weights[j][k]
            self.output_cells[k] = sigmoid(total)
        return self.output_cells[:]

    def back_propagate(self, case, label, learn, correct):
        # feed forward
        self.predict(case)
        # get output layer error
        output_deltas = [0.0] * self.output_n
        for o in range(self.output_n):
            error = label[o] - self.output_cells[o]
            output_deltas[o] = sigmoid_derivative(self.output_cells[o]) * error
        # get hidden layer error
        hidden_deltas = [0.0] * self.hidden_n
        for h in range(self.hidden_n):
            error = 0.0
            for o in range(self.output_n):
                error += output_deltas[o] * self.output_weights[h][o]
            hidden_deltas[h] = sigmoid_derivative(self.hidden_cells[h]) * error
        # update output weights
        for h in range(self.hidden_n):
            for o in range(self.output_n):
                change = output_deltas[o] * self.hidden_cells[h]
                self.output_weights[h][o] += learn * change + correct *
self.output_correction[h][o]
                self.output_correction[h][o] = change
        # update input weights
        for i in range(self.input_n):
            for h in range(self.hidden_n):
                change = hidden_deltas[h] * self.input_cells[i]
                self.input_weights[i][h] += learn * change + correct *
self.input_correction[i][h]
                self.input_correction[i][h] = change
        # get global error
        error = 0.0
        for o in range(len(label)):
            error += 0.5 * (label[o] - self.output_cells[o]) ** 2
        return error

    def train(self, cases, labels, limit=10000, learn=0.05, correct=0.1):
        for j in range(limit):
```

```
            error = 0.0
            for i in range(len(cases)):
                label = labels[i]
                case = cases[i]
                error += self.back_propagate(case, label, learn, correct)

    def test(self):
        cases = [
            [1.58,2.32,-5.8],
            [0.67,1.58,-4.78],
            [-1.49,2.18,-3.39],
            [-0.41,1.21,-4.73],
            [1.39,3.16,2.87],
            [1.2,1.4,-1.89],
            [-0.92,1.44,-3.22],
            [0.45,1.33,-4.38],
            [0.37,0.28,-1.8],
            [0.18,1.22,0.16],
            [-0.24,0.93,-1.01],
            [-1.18,0.39,-0.39],
            [0.74,0.96,-1.16],
            [-0.38,1.94,-0.48],
            [0.02,0.72,-0.17],
            [0.44,1.31,-0.14],
            [5.41,3.45,-1.33],
            [1.55,0.99,2.69],
            [1.86,3.19,1.51],
            [3.51,-0.22,-1.39],
            [1.4,-0.44,0.92],
            [0.44,0.83,1.97],
            [0.25,0.68,-0.99],
            [-0.66,-0.45,0.08]
        ]
        labels = [[0.1],[0.1],[0.1],[0.1],[0.1],[0.1],[0.1],[0.1],[0.2],[0.2],[0.2],
[0.2],[0.2],[0.2],[0.2],[0.2],[0.3],[0.3],[0.3],[0.3],[0.3],[0.3],[0.3],[0.3]]
        test_t = [
            [1.04,1.01,-3.63],#3  0.1
            [-0.76,0.84,-1.96],#10   0.1
            [0.21,0.03,-2.21],#11   0.2
            [0.46,1.49,0.68],#20    0.2
            [-1.54,1.17,0.64],#21   0.3
            [1.68,1.79,-0.87]#25   0.3
            ]
        self.setup(3, 5, 1)
        self.train(cases, labels, 10000, 0.05, 0.1)
        #for case in test_t:
        for case in cases:
            print(self.predict(case))


if __name__ == '__main__':
    nn = BPNeuralNetwork()
    nn.test()
```

程序说明

```
# 计算(a,b)中的随机数
def rand(a, b)
# 设置m*n的矩阵
def make_matrix(m, n, fill = 0.0)
# 定义sigmoid函数
def sigmoid(x):
# 初始化输入层、隐藏层、输出层
def setup(self, ni, nh, no)
# 激活输入层、隐藏层、输出层
def predict(self, inputs)
```

运行结果



结果解释

算出来的结果如果在0.1左右，则证明是第一组的样本，如果在0.2左右，则证明是第二组的样本，以此类推。如果用测试集测试，正确率仅在50%左右，程序还有待改进。