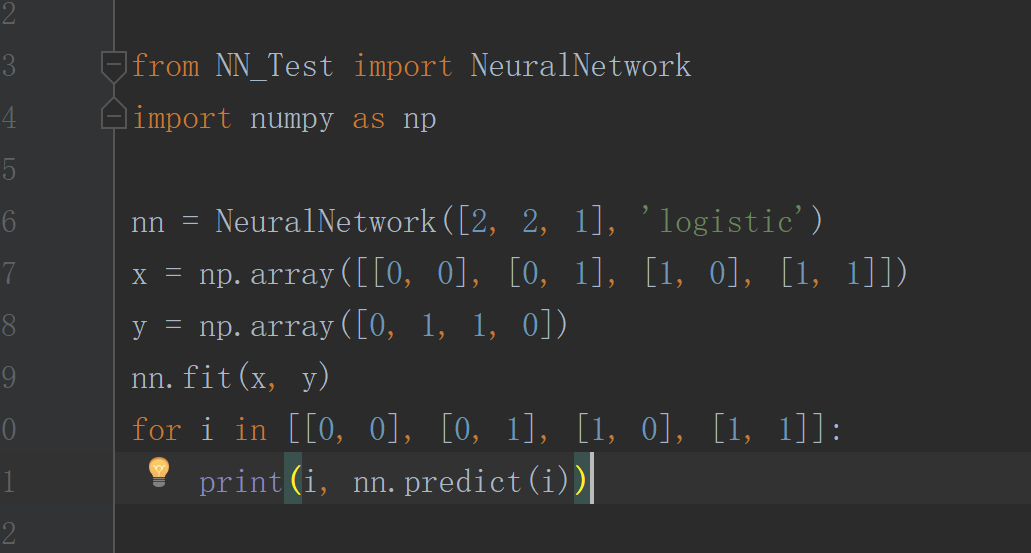
# 编程作业十

SA18006165李德运

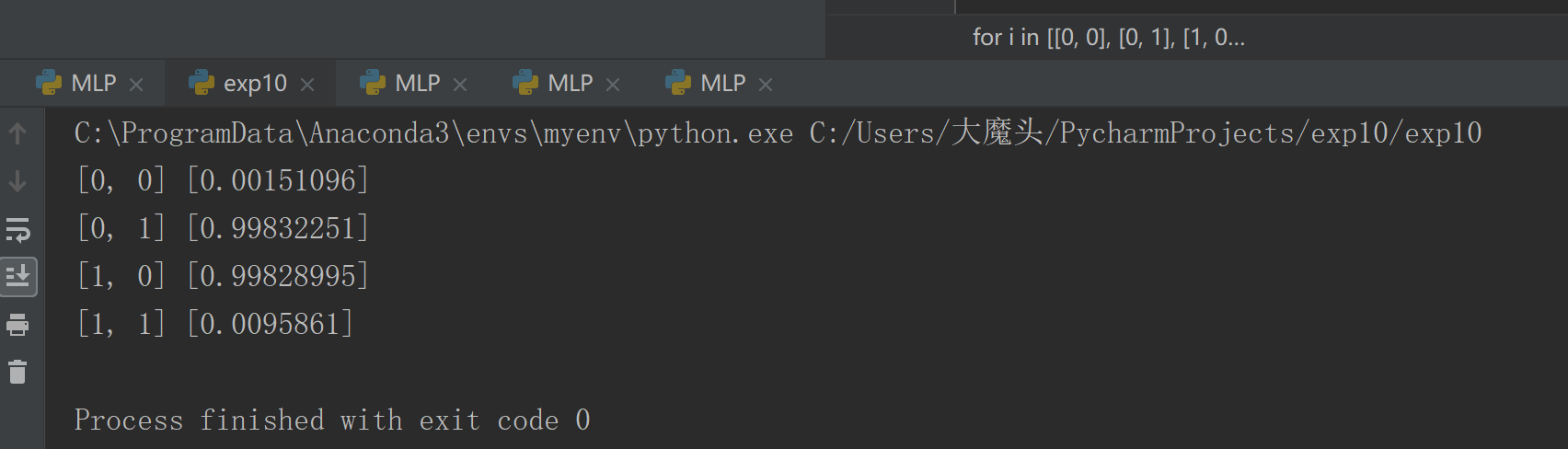
## 实验结果

### Sigmod函数

学习率 = 0.8，更新次数1e^10，



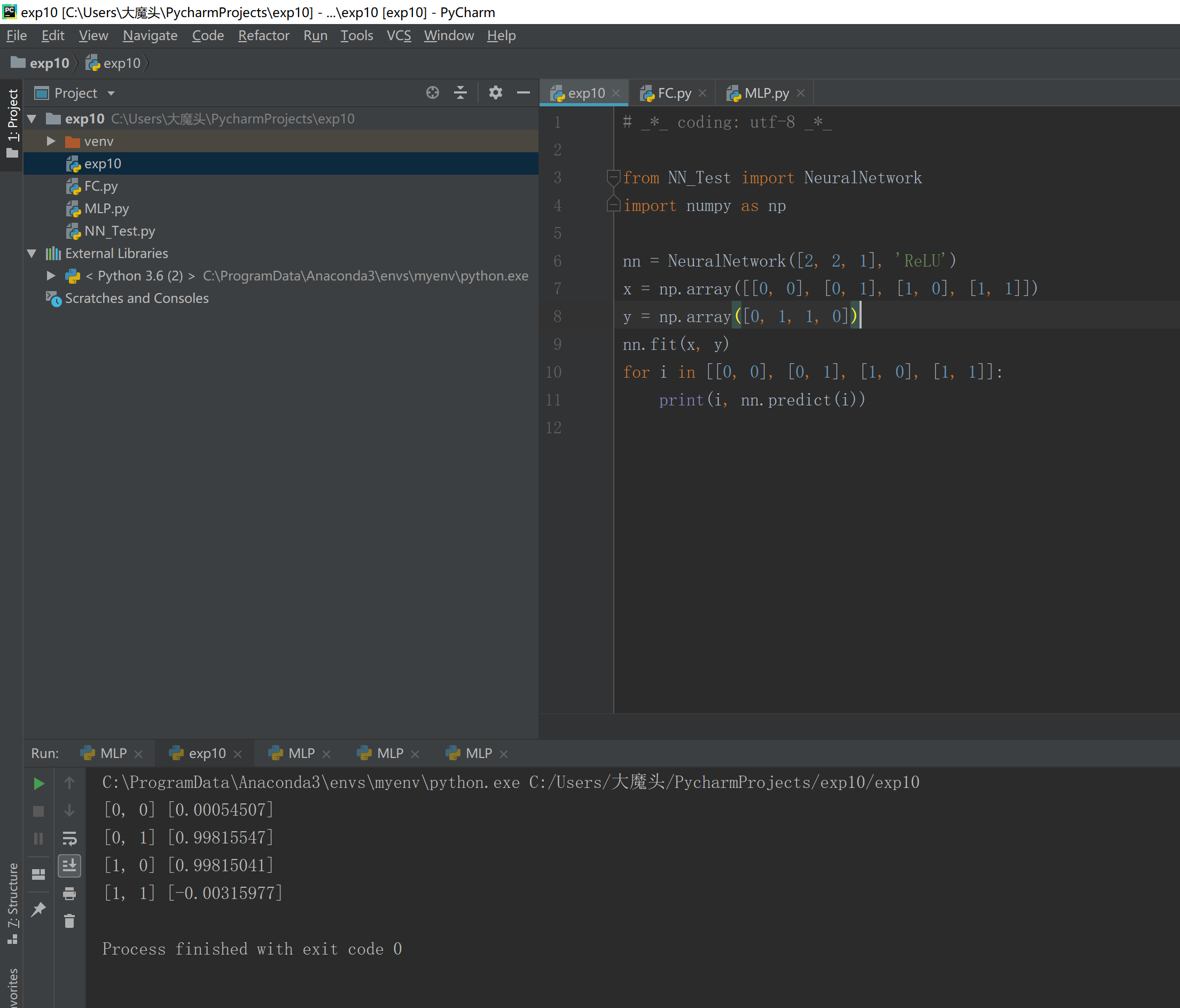
实验结果：



### RELU激活函数

学习率 0.7，更新次数1e^10

结果：



## 实验小结

实现了BP算法，对线性不可分的疑惑进行学习。实验中，通过不断的修改参数，熟悉每一个参数为何这么设定，对算法有更深刻的了解。

## 附录代码

# \_\*\_ coding: utf-8 \_\*\_  
  
import numpy as np  
  
  
def relu(x):  
 return np.tanh(x)  
  
  
def relu\_derivative(x):  
 return 1 - np.tanh(x) \* np.tanh(x)  
  
  
# sigmod函数  
def logistic(x):  
 return 1 / (1 + np.exp(-x))  
  
  
# sigmod函数的导数  
def logistic\_derivative(x):  
 return logistic(x) \* (1 - logistic(x))  
  
  
class NeuralNetwork:  
 def \_\_init\_\_(self, layers, activation='tanh'):  
 if activation == 'logistic':  
 self.activation = logistic  
 self.activation\_deriv = logistic\_derivative  
 elif activation == 'tanh':  
 self.activation = tanh  
 self.activation\_deriv = tanh\_derivative  
  
 # 随机产生权重值  
 self.weights = []  
 for i in range(1, len(layers) - 1): # 不算输入层，循环  
 self.weights.append((2 \* np.random.random((layers[i - 1] + 1, layers[i] + 1)) - 1) \* 0.25)  
 self.weights.append((2 \* np.random.random((layers[i] + 1, layers[i + 1])) - 1) \* 0.25)  
 # print self.weights  
  
 def fit(self, x, y, learning\_rate=0.2, epochs=10000):  
 x = np.atleast\_2d(x)  
 temp = np.ones([x.shape[0], x.shape[1] + 1])  
 temp[:, 0:-1] = x  
 x = temp  
 y = np.array(y)  
  
 for k in range(epochs): # 循环epochs次  
 i = np.random.randint(x.shape[0]) # 随机产生一个数，对应行号，即数据集编号  
 a = [x[i]] # 抽出这行的数据集  
  
 # 迭代将输出数据更新在a的最后一行  
 for l in range(len(self.weights)):  
 a.append(self.activation(np.dot(a[l], self.weights[l])))  
  
 # 减去最后更新的数据，得到误差  
 error = y[i] - a[-1]  
 deltas = [error \* self.activation\_deriv(a[-1])]  
  
 # 求梯度  
 for l in range(len(a) - 2, 0, -1):  
 deltas.append(deltas[-1].dot(self.weights[l].T) \* self.activation\_deriv(a[l]))  
  
 # 反向排序  
 deltas.reverse()  
  
 # 梯度下降法更新权值  
 for i in range(len(self.weights)):  
 layer = np.atleast\_2d(a[i])  
 delta = np.atleast\_2d(deltas[i])  
 self.weights[i] += learning\_rate \* layer.T.dot(delta)  
  
 def predict(self, x):  
 x = np.array(x)  
 temp = np.ones(x.shape[0] + 1)  
 temp[0:-1] = x  
 a = temp  
 for l in range(0, len(self.weights)):  
 a = self.activation(np.dot(a, self.weights[l]))  
 return a

# \_\*\_ coding: utf-8 \_\*\_  
  
from NN\_Test import NeuralNetwork  
import numpy as np  
  
nn = NeuralNetwork([2, 2, 1], 'ReLU')  
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  
y = np.array([0, 1, 1, 0])  
nn.fit(x, y)  
for i in [[0, 0], [0, 1], [1, 0], [1, 1]]:  
 print(i, nn.predict(i))