## 一个简单的网络：

## from numpy import exp, array, random, dot

## class NeuralNetwork():

## def \_\_init\_\_(self):

## # seed生成的随机数

## # 确保每次程序运行时它会生成相同的数字

## random.seed(1)

## # 把随机权重值分配给3\*1的矩阵

## # 权重值的范围是- 1至1，平均值为0

## self.synaptic\_weights = 2 \* random.random((3, 1)) - 1

## # Sigmoid函数

## # 把输入权重值转化为0到1的概率

## def \_\_sigmoid(self, x):

## return 1 / (1 + exp(-x))

## # Sigmoid function的导函数

## def \_\_sigmoid\_derivative(self, x):

## return x \* (1 - x)

## # 通过试错训练神经网络

## # 每次调整权重值

## def train(self, training\_set\_inputs, training\_set\_outputs, number\_of\_training\_iterations):

## for iteration in xrange(number\_of\_training\_iterations):

## # 通过神经网络传递训练值

## output = self.think(training\_set\_inputs)

## # 计算误差

## error = training\_set\_outputs - output

## # 输入值转置矩阵乘以误差得出调整值，然后乘以sigmoid的梯度值

## # 偏差越大的权重值调整越多

## # 输入值为0不会导致权重的更改

## adjustment = dot(training\_set\_inputs.T, error \* self.\_\_sigmoid\_derivative(output))

## # 调整权重

## self.synaptic\_weights += adjustment

## def think(self, inputs):

## return self.\_\_sigmoid(dot(inputs, self.synaptic\_weights))

## if \_\_name\_\_ == "\_\_main\_\_":

## #神经网络初始化

## neural\_network = NeuralNetwork()

## print "Random starting synaptic weights: "

## print neural\_network.synaptic\_weights

## # 训练数据集. 4个二进制数字的例子

## # 每个例子有3个输入值和一个输出值

## training\_set\_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])

## training\_set\_outputs = array([[0, 1, 1, 0]]).T

## # 用数据集训练神经网络

## # 训练的迭代次数为10,000

## neural\_network.train(training\_set\_inputs, training\_set\_outputs, 10000)

## print "New synaptic weights after training: "

## print neural\_network.synaptic\_weights

## #用新的输入值预测输出值

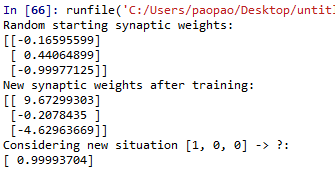
## print "Considering new situation [1, 0, 0] -> ?: "

## print neural\_network.think(array([1, 0, 0]))

## 学习这个代码时也出了很多问题，真的改了好久，以后编写千万要记住：空格！！！不然就会出现

或者输出错误。

这是一个单层前馈神经网络，通过训练简单的四组数据就行推测另一组数据得出的结果，并且运用梯度下降，最后得到输出：



结果特别接近1。

2017.7.12