构造简单的神经网络

构造三层的神经网络

设定激活函数为sigmoid函数

设定X输入数据，尺寸为 5 x 3

设定y输出标签，尺寸为 5 x 1

三层神经网络，中间只有一层，有4个神经元。

设定第一层为L0， 中间层为L1， 输出层为L2

L0输入层的尺寸为 5 x 3

L0与L1之间的参数为w0，w0的尺寸为 3 x 4

L1中间层的尺寸为 5 x 4

L1与L2之间的参数为w1，w1的尺寸为 4 x 1

L2输出层的尺寸为 5 x 1

因此，可以得到3层神经网络的结构如下：

L0 -> w0 ->sigmoid -> l1 ->w1 -> sigmoid -> l2

前向传播从L0的x输入数据开始，

l0 = x;

l1 = sigmoid(w0x)

l2 = sigmoid(w1l1)

反向传播要做的是不断优化参数w，通过计算各个样本对于误差的影响程度

L2是最终输出，要与设定好的标签值y进行比较，误差值l2\_error要越小越好。

L2\_error = y – l2;

反向传播将误差l2\_error首先从l2传到sigmoid函数，计算sigmoid函数的反求导操作，并且要乘上l2\_error，因为l2\_error决定了反向传播是否有必要。当l2\_error很小时，说明结果与标签匹配的很好，那么反向传播就没有什么必要；当l2\_error很大时，说明误差很大，需要大幅度地改进参数w，因此用l2\_error来决定反向传播的力度。

L2\_delta = le\_error \* 反求导sigmoid;

L2\_delta 求的是前面的样本对于误差有多大的影响。

之后的就是按照这种步骤，不断往前传。

最后更新网路的参数w

反向传播首先更新w1参数，因为l2是通过w1 \* l2得到的，那么要求w1对于l2有多大影响，就对w1求导，得到l1，并乘以上一层传下来的l2\_delta，l2\_delta求的是前面的样本对于误差有多大影响。

W1 += l1..T.dot(l2\_delta);

同理，对于w0，也是通过求导与乘以上层传下来的误差影响值

W0 += l0.T.dot(l1\_delta)

最终w0, w1不断更新，使得误差值越来越小。

代码：  
import numpy as np  
  
  
def sigmoid(x, deriv=False):  
 *"""激活函数与反求导"""* if(deriv==True):  
 return x\*(1-x)  
  
 return 1 / (1 + np.exp(-x))  
  
x = np.array([  
 [0,0,1],  
 [0,1,1],  
 [1,0,1],  
 [1,0,1],  
 [1,0,1]  
])  
***#print(x.shape) 5 x 3***y = np.array([  
 [0],[1],[0],[1],[0]  
])  
***#print(y.shape) 5 x 1  
  
#随机数种子***np.random.seed(1)  
  
***#随机生成W参数***w0 = 2 \* np.random.random((3,4)) -1  
w1 = 2 \* np.random.random((4,1)) -1  
***#print(w0.shape) 3 x 4  
#print(w1.shape) 4 x 1***for i in range(600000):  
 ***#求出三个层的数据*** l0 = x  
 l1 = sigmoid(np.dot(l0, w0))  
 l2 = sigmoid(np.dot(l1, w1))  
  
 ***#l2层误差*** l2\_error = y - l2  
  
 ***#每隔10000次输出误差*** if i % 10000 == 0:  
 print("Error" + str(np.mean(np.abs(l2\_error))))  
  
 ***#反向传播*** l2\_delta = l2\_error \* sigmoid(l2, deriv=True)  
  
 l1\_error = l2\_delta.dot(w1.T)  
  
 l1\_delta = l1\_error \* sigmoid(l1, deriv=True)  
  
 ***#更新w 参数*** w1 += l1.T.dot(l2\_delta)  
 w0 += l0.T.dot(l1\_delta)