实验四 Hopfield神经网络

姓名：唐川淇 学号：1131190111

# 问题重述

设计一个Hopfield网络，使其具有联想记忆功能，能正确识别阿拉伯数字，当数字被噪声污染后仍可以正确地识别。

# 问题分析

阿拉伯数字从0-8共有10个，所以假设神经网络共有是个稳态构成，每个稳态由一个10x10的矩阵构成，该矩阵用于模拟阿拉伯数字点阵，即每个数字都化成矩阵形式，有图形的部分用1来表示，没有图形的部分用-1表示。例如将阿拉伯数字1化为10x10的矩阵后为如下格式：



该矩阵由-1和1构成，其中1的部分为黑色构成了阿拉伯数字1的形状。以此类推将10个阿拉伯数字都建立成矩阵的形式，并将这10个矩阵作为训练集训练网络，由此可以得到一个记录了十个阿拉伯数字的Hopfield网络。为了测试网络识别效果，对阿拉伯数字矩阵添加噪音，使用网络对添加噪声的数字进行识别，并对其准确性进行研究。

# 模型的建立与求解

## 设计数字点阵

设计的十个阿拉伯数字点阵如下：



1 2



3 4



5 6



7 8



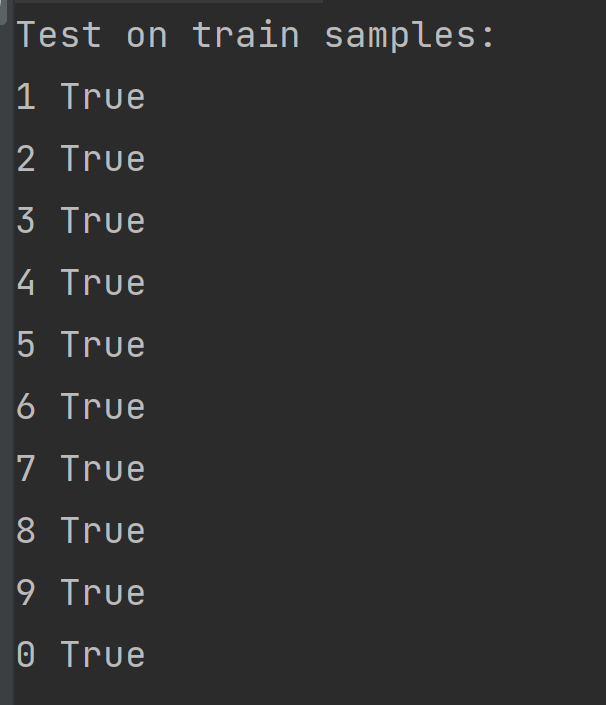
9 0

## 创建Hopfield网络

在实际建立网络时发现，由于数字与数字之间的差距较小，特别项数字6与数字8，数字5与数字6，这些数字之间的形状极为相似，使用一个Hopfield网络识别10个数字十分困难，因此我将10个数字分为4组，用4个Hopfield网络来识别10个阿拉伯数字，具体的分组情况如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 组别 | 元素1 | 元素2 | 元素3 |
| 1 | 1 | 2 | 3 |
| 2 | 4 | 5 | 6 |
| 3 | 7 | 8 |  |
| 4 | 9 | 0 |  |

首先按照分组将10个阿拉伯数字的矩阵输入到程序中，按照4个类别将数字分组。输入数据之后将数据矩阵化，然后训练4个Hopfield网络。训练好4个神经网络之后为了检验神经网络是否能识别10个阿拉伯数字，将设定好的10个阿拉伯数字矩阵带回到训练好的神经网络中看神经网络是否能准确的识别这10个阿拉伯数字，最终得到的结果如下：



发现，训练好的神经网络可以准确地识别10个阿拉伯数字，至此神经网络建立完毕。

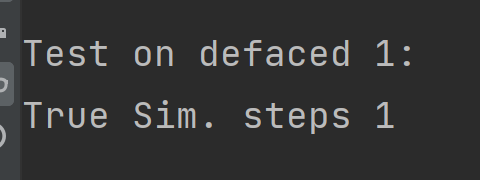
## 设计受到污染的数字点阵与识别

为了测试神经网络模型的鲁棒性，需要对阿拉伯数字矩阵添加噪音，然后研究神经网络是否能准确的识别带有噪声的矩阵。

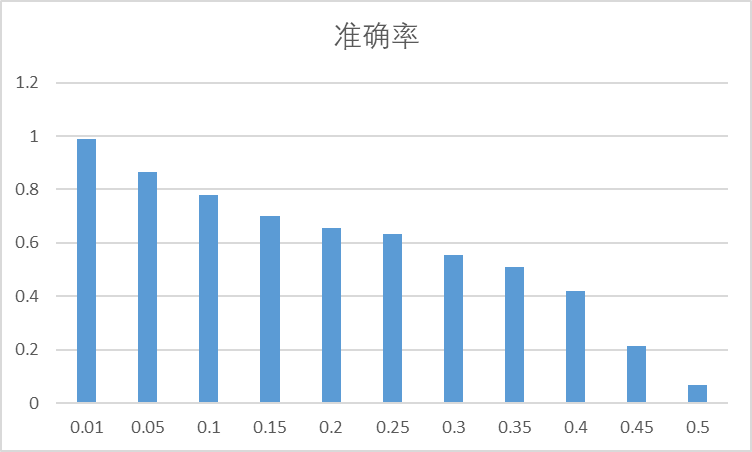
首先先随机改变矩阵中的一个数值，将10x10矩阵中的任意一个数值乘-1，得到新的矩阵用神经网络识别，以阿拉伯数字1为例，添加一个噪声点后的数字矩阵如下所示：



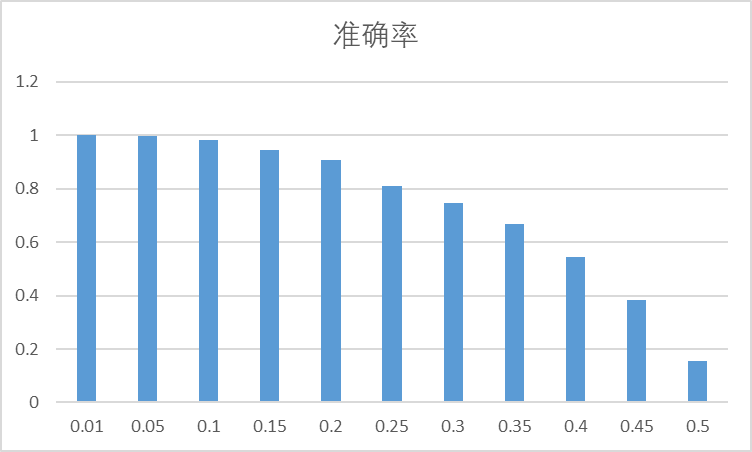
将带有噪声的数字矩阵带入到第一个神经网络，得到如下结果：



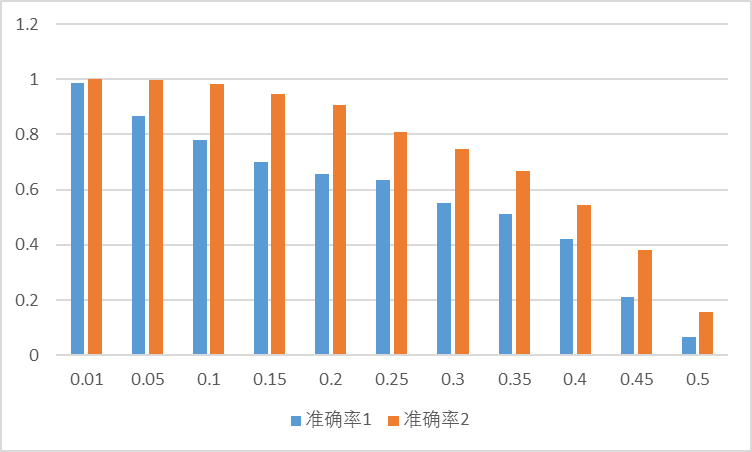
对阿拉伯数字一的识别成功。再随机更改矩阵中的一个数值，重复3次，发现神经网路都能正确的识别数字。为了研究神经网络识别的准确度随着噪声变化的变化，使用随机的方法随机添加噪音，逐渐增加噪音画出图像如下：



发现在上图中识别的准确性呈现先快再慢再快的变化趋势，呈现s型，当添加的噪音大于50%时，识别的准确率已经不到5%。整体来看，当添加的噪音少于10%时整体模型都有较好的识别效果。以上是对一个包含三个状态的神经网络的准确性进行的研究，对于包含两种状态的神经网路的变化趋势也需进行研究。因此对数字0添加噪音研究第四个神经网络的鲁棒性，依然根据上方法，不断增加矩阵的噪音，查看神经网络识别矩阵的准确性的变化趋势：



发现整体的下降速率逐渐增快，在噪音量小于20%时整体的神经网络都有较好的识别准确率，对比两个神经网络的识别准确性，画出如下图像：



整体来看，包含记录状态少的神经网络确实具有更高的识别准确率。由此可以得出结论，在保持其他参数一定的情况下，Hopfield神经网络的识别准确度和记录的状态成反比，也就是说一个Hopfield神经网络记录的状态越多，那么它对一个带有噪声的矩阵的识别正确率也越低。直观来看这个结论也符合实际，随着记录的状态的增加，带有大噪声的数据也更容易被划分到错误的类别。因此在设计Hopfield网络时需要考虑这两点因素设计更适合的神经网络。

附录

|  |
| --- |
| 附程序 |
| 介绍：Python |
| import numpy as np  import neurolab as nl  import random  #  target1 = [[-1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],  [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],  [-1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1]  ]  target2 = [[-1, 1, 1, -1, -1, -1, 1, 1, -1, -1,  -1, 1, 1, -1, -1, -1, 1, 1, -1, -1,  -1, 1, 1, -1, -1, -1, 1, 1, -1, -1,  -1, 1, 1, -1, -1, -1, 1, 1, -1, -1,  -1, 1, 1, -1, -1, -1, 1, 1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, 1, 1, -1,  -1, 1, 1, 1, 1, 1, 1, 1, 1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1],  [  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, -1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],  [  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, -1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, -1, -1, -1,  -1, 1, 1, -1, -1, -1, -1, -1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, -1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, -1, -1, -1,  -1, 1, 1, -1, -1, 1, 1, -1, -1, -1,  -1, 1, 1, -1, -1, 1, 1, -1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, -1, -1, -1,  -1, 1, 1, 1, 1, 1, 1, -1, -1, -1]  ]  target3 = [[  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, 1, 1, 1, -1, -1,  -1, -1, -1, -1, 1, 1, 1, -1, -1, -1,  -1, -1, -1, -1, 1, 1, -1, -1, -1, -1,  -1, -1, -1, -1, 1, 1, -1, -1, -1, -1,  -1, -1, -1, -1, 1, 1, -1, -1, -1, -1,  -1, -1, -1, -1, 1, 1, -1, -1, -1, -1],  [  -1, -1, 1, 1, 1, 1, 1, 1, 1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, 1, -1,  -1, -1, 1, 1, -1, -1, -1, 1, 1, -1,  -1, -1, 1, 1, -1, -1, -1, 1, 1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, 1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, 1, -1,  -1, -1, 1, 1, -1, -1, -1, 1, 1, -1,  -1, -1, 1, 1, -1, -1, -1, 1, 1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, 1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, 1, -1  ]  ]  target4 = [[  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, -1, -1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],  [  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1  ]  ]  chars1 = ['1', '2', '3']  chars2 = ['4', '5', '6']  chars3 = ['7', '8']  chars4 = ['9', '0']  target1 = np.asfarray(target1)  target2 = np.asfarray(target2)  target3 = np.asfarray(target3)  target4 = np.asfarray(target4)  # Create and train network  net1 = nl.net.newhop(target1)  net2 = nl.net.newhop(target2)  net3 = nl.net.newhop(target3)  net4 = nl.net.newhop(target4)  output1 = net1.sim(target1)  output2 = net2.sim(target2)  output3 = net3.sim(target3)  output4 = net4.sim(target4)  print("Test on train samples:")  for i in range(len(target1)):  print(chars1[i], (output1[i] == target1[i]).all())  for i in range(len(target2)):  print(chars2[i], (output2[i] == target2[i]).all())  for i in range(len(target3)):  print(chars3[i], (output3[i] == target3[i]).all())  for i in range(len(target4)):  print(chars4[i], (output4[i] == target4[i]).all())  cnt=0  for j in range(1000):  print("\nTest on defaced 0:")  test =np.asfarray([ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, -1, -1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1,  -1, -1, 1, 1, 1, 1, 1, 1, -1, -1  ])  for i in range(100):  aa=random.random()  if aa < 0.5:  test[i] = test[i]\*-1  out = net4.sim([test])  print((out[0] == target4[1]).all(), 'Sim. steps', len(net4.layers[0].outs))  if (out[0] == target4[1]).all():  cnt=cnt+1  print(cnt/1000) |