人工神经网络：Hopfield神经网络的实现

# 1实验目的

1. 了解Hopfield神经网络的结构；
2. 了解Hopfield神经网络连接权值的设置方式；
3. 了解Hopfield神经网络的稳定状态。
4. 了解离散型、连续性、随机性Hopfield神经网络。

# 2实验要求

本次试验后，要求学生能：

1. 理解Hopfield神经网络的工作原理；
2. 能够设计离散型Hopfield神经网络结构及其连接权值；
3. 用python编程实现离散型Hopfield神经网络，并用于图形识别问题。

# 3实验原理

## 3.1实验介绍

本实验使用离散型Hopfield神经网络识别符号“+”和符号“×”。

在5\*5的格子内，用“0”和“1”描绘出“+”和“×”的形象作为训练和测试数据，训练Hopfield神经网络，并进行测试。

例如， [1 0 0 0 1]

[0 1 0 1 0]

[0 0 1 0 0]

[0 1 0 1 0]

[1 0 0 0 1]代表符号“×”。

## 3.2 离散型Hopfield神经网络

### 3.2.1网络结构

Hopfield神经网络（HNN）是全互联反馈神经网络，它的每一个神经元都和其他神经元相连接。

具有N个神经元的离散型Hopfield神经网络，可以由一个N×N阶矩阵**w**和一个N为列向量**θ**唯一确定。

其中，**w**是连接权值矩阵，wij表示第i个神经元和第j个神经元之间的连接强度，且wij = wji，wii = 0。**θ**是阈值向量θi表示第i个神经元的阈值。

若用vi(k)表示k时刻第i个神经元的状态，则神经元i随时间变化的规律为：

公式1

### 3.2.2稳定性

Hopfield神经网络中神经元相互作用，不断演化。如果从某一时刻开始，网络中的神经元状态都不再改变，则称该网络是稳定的。稳定性是Hopfield神经网络最重要的特性。

Hopfield神经网络是一个高维非线性动力学系统，可能有若干个稳态。从任意初始状态开始运动，总能到达某一个稳定状态。

引入了能量函数E，在网络运动过程中E总是在不断减小，最后趋于平衡，E定义为：

公式2

Hopfield等人已经证明出：只要连接权值矩阵是对角非负的对称矩阵，该网络就具有串行（不同时刻，神经元一个一个地异步更新）稳定性；若该矩阵为非负定矩阵，则该矩阵具有并行（不同时刻，神经元同时更新）稳定性。

### 3.2.1在联想问题中的应用

Hopfield神经网络总能运动到一个稳定状态，所以只要调整权值矩阵，使待记忆的样本成为网络的一个稳定状态，即可实现记忆和联想。可以用Hebb学习规则来设计Hopfield的连接权值矩阵，从而达到这一目的。

公式3

Hebb学习规则实际上就是人为地将要记忆的图形构建成一个局部最小值，输入数据会收敛向它最靠近的局部最小值，从而实现联想回忆的功能。

# 4实验步骤

## 4.1安装依赖

本实验需要用到python的扩展程序库Numpy进行数值计算，如果没有安装过这个库，使用国内的清华pip源安装，命令如下：

pip3 install numpy scipy matplotlib -i <https://pypi.tuna.tsinghua.edu.cn/simple>

导入需要的库。

import numpy as np

import copy

## 4.2确定网络模型

为了识别在5\*5的格子里用“0”、“1”描绘出的符号“+”和符号“×”，建立离散型Hopfield神经网络模型进行联想识别，模型由25\*25阶连接权值矩阵weight和25个神经元组成，阈值设置为0。

识别任务分为学习训练和联想回忆两个步骤。

## 4.3学习训练

学习阶段就是对给定的样本，调整其连接权值，使之成为网络的一个稳定状态。本实验使用的是Hebb规则设计Hopfield神经网络的连接权值，即，

按照上述公式设计的连接权值满足对称条件，可以保证神经网络的串行稳定性。

可调用numpy的np.dot(A,B)用矩阵乘法来计算连接权值，先将神经元节点矩阵乘2减1，再将它乘上自己的转置就得到了连接权值由第k个样本得到的增加量。

# 学习：计算连接权值矩阵

def get\_weight(case):

weight = np.zeros((25, 25))

case\_copy = copy.deepcopy(case)

for c in case\_copy:

c = 2 \* c - 1

weight += np.dot(np.atleast\_2d(c).T, np.atleast\_2d(c))

return weight

## 4.4联想回忆

联想阶段是指在已经调解好连接权值不变得情况下，给出部分或是收到干扰的信息，按照动力学规则改变神经元状态，使神经网络最终变到某个稳定状态

变化公式如下（U是1×25的神经元矩阵，W是25×25的连接权值矩阵，ui是U的第i个元素）：

更新时需要用到符号函数来计算新神经元的节点值。

def sign(a):

if a >= 0:

return 1

if a < 0:

return 0

采用异步更新策略，简单地按照神经元序号异步更新，第i个神经元新的值为神经元矩阵和weight矩阵第i列（由于对称，也是第i行）的乘积。直到更新后的神经元矩阵和更新前一样，说明已经到达了稳定状态。

# 联想：矩阵变换直到稳定状态

def recognize(case, weight):

case\_copy = copy.deepcopy(case)

# 由于权值矩阵为对称矩阵，必定收敛到稳定态

while 1:

case\_before = copy.deepcopy(case\_copy)

# 异步更新

for index in range(case\_copy.shape[0]):

case\_copy[index] = sign(np.dot(np.atleast\_2d(case\_copy),

np.atleast\_2d(weight[index]).T)[0][0])

if case\_before.any() == case\_copy.any():

return case\_copy

## 4.5训练和测试

（1）数据

用最标准的“+”和“×”作为训练数据，然后对标准的图形减少或添加若干点位，作为测试数据。

训练和测试数据如下：

train1 = np.array([0, 0, 1, 0, 0, # +

0, 0, 1, 0, 0,

1, 1, 1, 1, 1,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0,

])

train2 = np.array([1, 0, 0, 0, 1, # x

0, 1, 0, 1, 0,

0, 0, 1, 0, 0,

0, 1, 0, 1, 0,

1, 0, 0, 0, 1,

])

test1 = np.array([1, 0, 1, 0, 1, # +

0, 0, 1, 0, 0,

1, 1, 1, 1, 1,

0, 0, 1, 0, 0,

1, 0, 1, 0, 1,

0, 0, 1, 0, 0, # +

0, 0, 0, 0, 0,

1, 1, 1, 1, 1,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0, # +

0, 0, 1, 0, 0,

1, 1, 1, 1, 1,

0, 0, 0, 0, 0,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0, # +

0, 1, 1, 0, 0,

1, 1, 1, 1, 1,

0, 0, 1, 0, 0,

0, 0, 1, 0, 0,

])

test2 = np.array([1, 0, 1, 0, 1, # x

0, 1, 0, 1, 0,

1, 0, 1, 0, 1,

0, 1, 0, 1, 0,

1, 0, 1, 0, 1,

0, 0, 0, 0, 0, # x

0, 1, 0, 1, 0,

0, 0, 1, 0, 0,

0, 1, 0, 1, 0,

1, 0, 0, 0, 1,

1, 0, 0, 0, 1, # x

0, 1, 0, 1, 0,

0, 0, 1, 0, 0,

0, 1, 0, 1, 0,

0, 0, 0, 0, 0,

1, 0, 0, 0, 1, # x

0, 1, 1, 1, 0,

0, 0, 1, 0, 0,

0, 1, 0, 1, 0,

1, 0, 1, 0, 1,

])

train1 = train1.reshape((1, 25))

train2 = train2.reshape((1, 25))

test1 = test1.reshape((4, 25))

test2 = test2.reshape((4, 25))

（2）定义函数print\_picture（）把联想回忆的结果打印成图像，方便观察。

def print\_picture(picture):

i = 0

while i < len(picture):

print(picture [i: i + 5])

i += 5

（3）学习和联想

weight1 = get\_weight(train1)

weight2 = get\_weight(train2)

weight = weight1 + weight2

for i in range(len(weight)):

for j in range(len(weight[i])):

if i == j:

weight[i][j] = 0

print("训练+")

for i in range(len(train1)):

print("+稳定矩阵：%d" % (i + 1))

print\_picture(recognize(train1[i], weight))

print("测试+")

for i in range(len(test1)):

print("由第%d个图像联想或回忆出：" % (i + 1))

print\_picture(recognize(test1[i], weight))

print("训练x")

for i in range(len(train2)):

print("x稳定矩阵：%d" % (i + 1))

print\_picture(recognize(train2[i], weight))

print("测试x")

for i in range(len(test2)):

print("由第%d个图像联想或回忆出：" % (i + 1))

print\_picture(recognize(test2[i], weight))

# 5实验结果

## 5.1学习训练

将标准的“+”、“×”图形作为训练样本训练Hopfield神经网络。

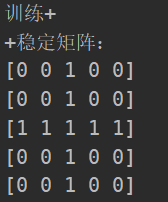


图1 “+”的训练样本

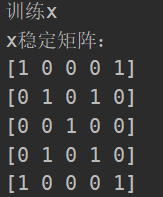


图2 “×”的训练样本

## 5.2联想回忆

（1）对含有噪声的“+”图像进行联想回忆

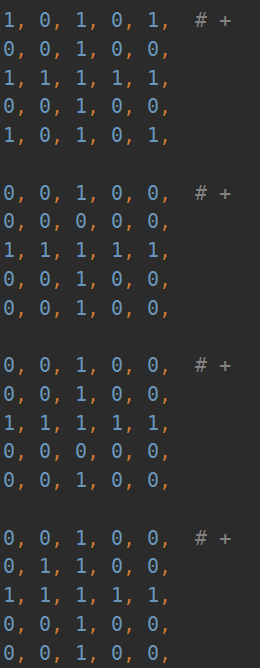
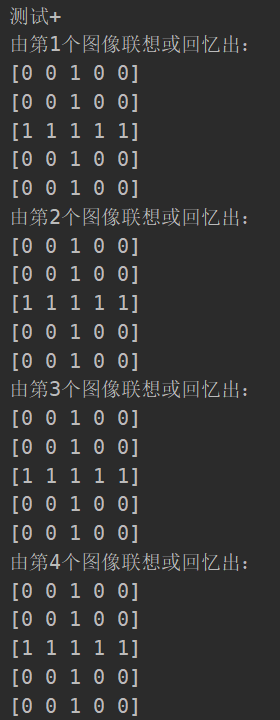
 

图3-1 “+”的测试样本 图3-2 “+”的联想或回忆结果

（2）对含有噪音的“×”符号进行联想回忆

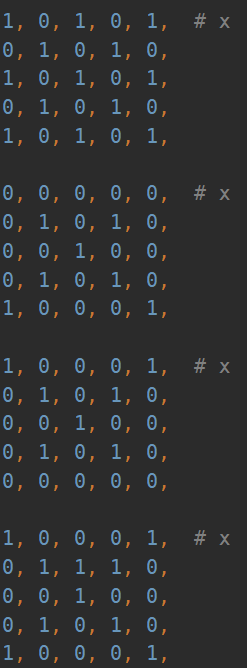
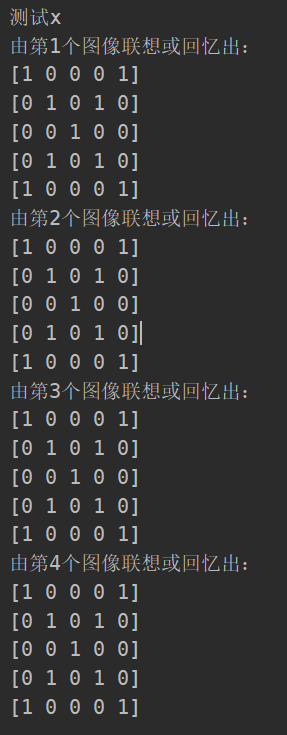
 

图4-1 “×”的测试样本 图4-2 “×”的联想或回忆结果

## 5.3实验小结

可看出，训练出的Hopfield神经网络识别结果准确，可以尝试让网络记忆更多的样本模式进行识别。

但是样本多且彼此接近时，容易引起混淆，当想保存在网络中的样本两两正交时，Hopfield神经网络可以有最大的记忆容量。