认知科学与类脑计算

实验报告一

日期：2019/5/16

班级：16 人工智能

姓名：贾乘兴

学号：201600301304

一.需求分析：

Hopfield 神经网络模型是一种循环神经网络，从输出到输入有反馈连接。Hopfield 网络有离散型和连续型两种。反馈神经网络由于其输出端有反馈到其输入端；所以，Hopfield 网络在输入的激励下，会产生不断的状态变化。当有输入之后，可以求取出 Hopfield 的输出，这个输出反馈到输入从而产生新的输出，这个反馈过程一直进行下去。如果 Hopfield 网络是一个能收敛的稳定网络，则这个反馈与迭代的计算过程所产生的变化越来越小，一旦到达了稳定平衡状态；那么 Hopfield 网络就会输出一个稳定的恒值。对于一个 Hopfield 网络来说，关键是在于确定它在稳定条件下的权系数。本次实验的目的是加深对 Hopfield 模型的理解，能够使用 Hopfield 模型解决实际问题。根据 Hopfield 神经网络的相关知识，设计一个具有联想记忆功能的离散型 Hopfiled 神经网络。要求该网络可以正确识别 0-9 这 10 个数字，当数字被一定的噪声干扰后，仍具有较好的识别效果。

二.概要设计：

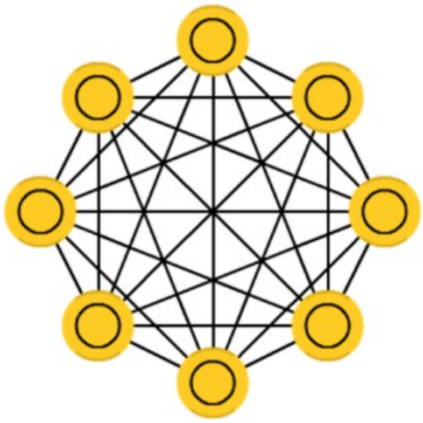
输入为手工设计的数字点阵，大小为 6*5。对于有像素的地方数值为 1，无像素的地方数值为 0。创建 Hopfield 网络，进行训练。

之后对输入数字点阵加入噪声，采用固定噪声和随机噪声两种方法。然后将数字点阵输入进创建并训练好的网络中进行测试。

三.详细设计：

1. 离散 Hopfield 网络设计

离散型的 Hopfield 神经网络，每个神经元节点是二值化 0、1 取值。



神经元节点之间连接的边有权重矩阵 W，且 W 矩阵是对角线为 0 的对称矩阵，神经元有当前状态和输出状态，我们分别用符号 u 和 v 表示，时刻 t，u 的更新公式如下：

$$u_j(t+1)=\sum_{i=1}^{n}W_{ij}v_j(t)-\theta_j$$

theta 为阈值，v 的更新为

$$v_j(t+1)=\mathrm{sgn}\left(u_j(t+1)\right)=\begin{cases}1, u_j\geq 0\\ 0, u_j<0\end{cases}$$

当下一次更新时 v 不再改变，即为网络的输出状态，表示如下

$$\lim_{t\to\infty} v(t) = v$$

对于权重，我们定义权重如下

$$W = X^T X - mI$$

其中 X 为样本矩阵，m 为样本个数，I 为各个样本的对角线的元素（保证对角线为 0），对于单个样本的更新，定义如下

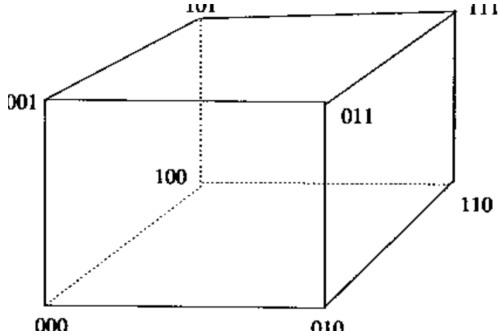
$$W = x^T x - I$$
$$x = [x_1, ..., x_n]$$

最终得到的矩阵为

$$W = \begin{bmatrix} 0 & x_1x_2 & \dots & x_1x_n \\ x_2x_1 & 0 & & \\ \dots & & \dots & \dots \\ x_nx_1 & & \dots & 0 \end{bmatrix}$$

多个矩阵的计算可由单个矩阵的叠加计算

神经网络中，每个节点存在 0、1 两个输出状态，故 n 个节点有 2^n 种状态，最终稳定的状态我们称为吸引子，即不断迭代后状态转移到该状态后稳定不变，在三维情况下可视化如下



我们的输入为一个顶角，根据输入状态，最终会趋于一个稳定的顶角，该顶角的状态我们称为吸引子

对于吸引子 x 满足的条件如下

$$x = f(Wx - \theta)$$

满足了该条件我们称为吸引子，我们引入能量函数，证明如下

由动力学定义能量函数为

$$E(t)=-\frac{1}{2}X^T(t)WX(t)+X^T(t)X$$

令时刻 t 到时刻 t+1 网络的能量变化为 delta E，状态的变化量为 delta x，则

$$\Delta E(t)=E(t+1)-E(t)$$
$$\Delta x(t)=x(t+1)-x(t)$$

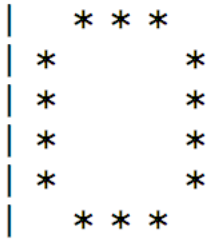
代入得

$$\Delta E(t)=-\Delta x^T(x)\left[Wx(t)-T\right]-\frac{1}{2}\Delta x^T(t)W\Delta x(t)$$

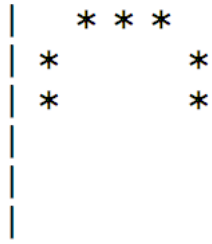
W 为对称矩阵，且对角线元素为 0，故可证，当 x 由 1 到 0 或由 0 到 1 时，deltaE 小于等于 0，当 delta x 为 0 时，delta E 也为 0。收敛性可证。

2. 固定噪声

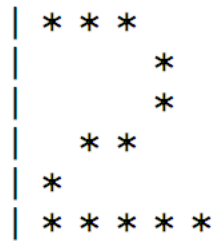
我们将图像截去上半部分或者下半部分，例如图像 0



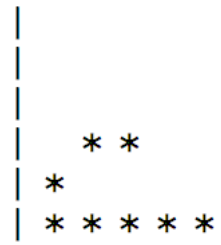
截去下半部为



图像 2



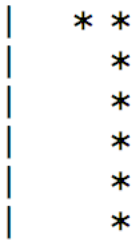
截去下半部分为



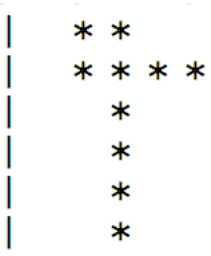
3. 随机噪声

我们设置概率为 10%，该概率表示，图像该点有 10%的概率变为另一数值。

例如数字 1 为

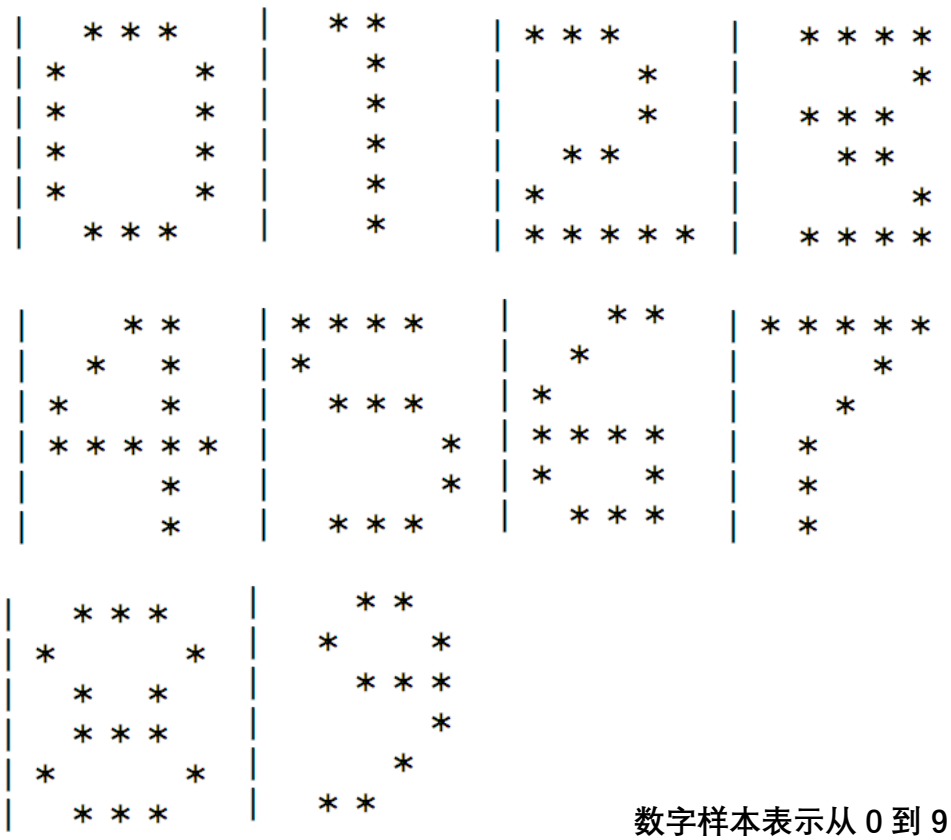


加入随机噪声后的 1 为



加入两种噪声以测试模型的稳定性

4. 手工设计数字样本（0-9）



数字样本表示从 0 到 9

四.调试分析：

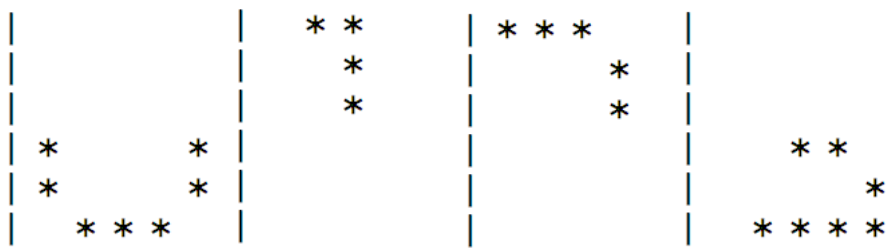
使用了两种方式实现模型，一种是利用 neupy 的 algorithm 模型，另一种方式是底层实现的方式，在实验中发现两种模型略有差异，但共同的问题是当样本增加时，难以保证效果，即吸引子的点与实际效果相比，较差。在样本数据只有三个的时候则还原效果较好。

可知随着需要的吸引子增加，模型的效果变差。

五.测试结果：

1.较大样本，0-9 手写数字

加入固定噪声后的数字如下

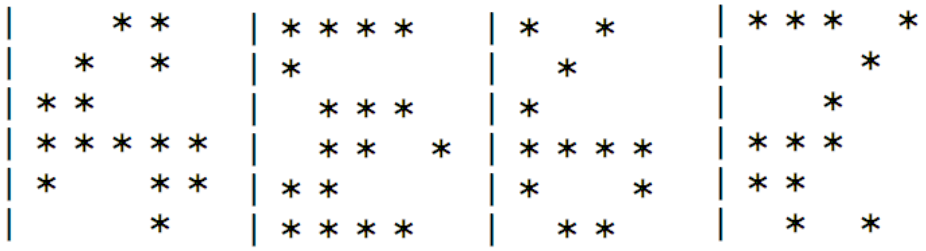


（等概率随机失去一半）

测试结果如下

（可见效果并没有非常好）

加入 10%随机噪声后数字

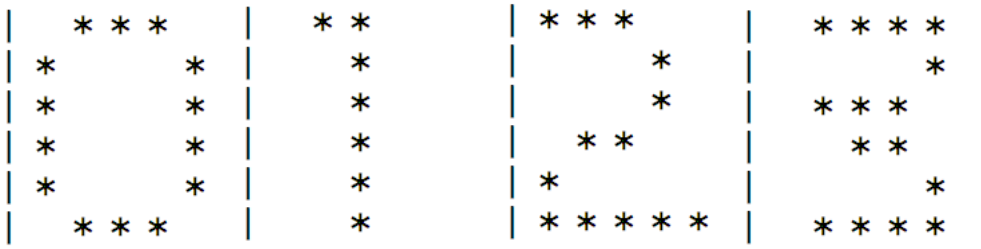


测试结果如下

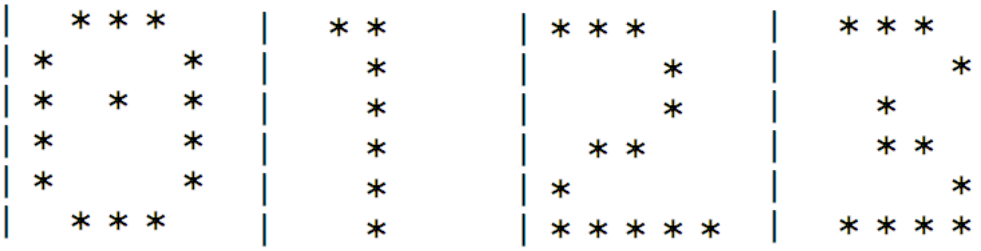
（效果好于固定噪声）

2.小样本测试，0-3 数字

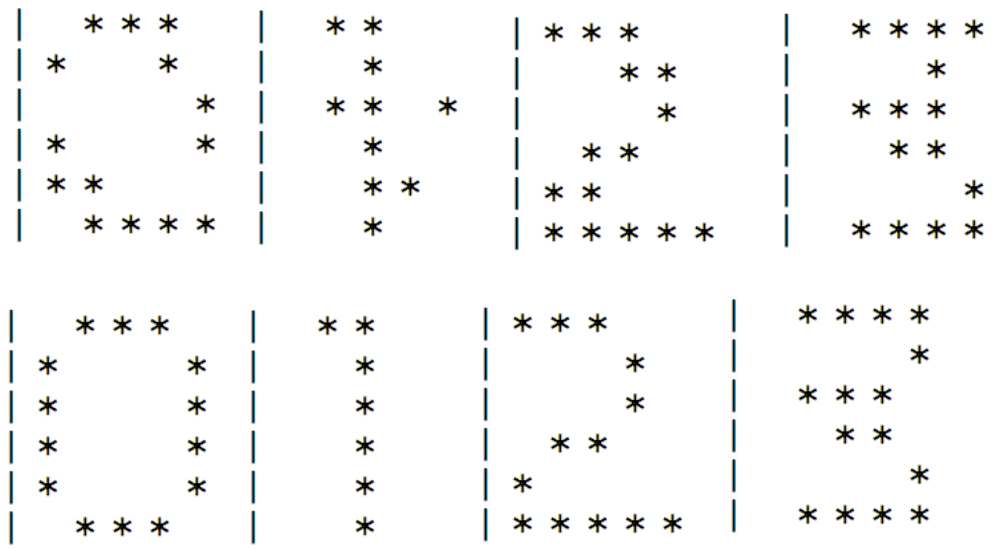
正常还原



固定噪声



随机噪声



可见小样本的情况下效果非常好

六.附录：

附录代码文件为 Hopfield.py 与 Hopfield1.py 两个文件

Hopfield.py

```
import numpy as np
from neupy import algorithms

def draw_bin_image(image_matrix):
    for row in image_matrix.tolist():
        print('| ' + ' '.join(' *'[val] for val in row))
    print('\n')

zero = np.matrix([
    0, 1, 1, 1, 0,
    1, 0, 0, 0, 1,
    1, 0, 0, 0, 1,
    1, 0, 0, 0, 1,
    1, 0, 0, 0, 1,
    0, 1, 1, 1, 0
])
```

```
one = np.matrix([
    0, 1, 1, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 1, 0, 0
 ])

two = np.matrix([
    1, 1, 1, 0, 0,
    0, 0, 0, 1, 0,
    0, 0, 0, 1, 0,
    0, 1, 1, 0, 0,
    1, 0, 0, 0, 0,
    1, 1, 1, 1, 1,
 ])

three = np.matrix([
    0, 1, 1, 1, 1,
    0, 0, 0, 0, 1,
    0, 1, 1, 1, 0,
    0, 0, 1, 1, 0,
    0, 0, 0, 0, 1,
    0, 1, 1, 1, 1
])

four = np.matrix([
    0, 0, 1, 1, 0,
    0, 1, 0, 1, 0,
    1, 0, 0, 1, 0,
    1, 1, 1, 1, 1,
    0, 0, 0, 1, 0,
    0, 0, 0, 1, 0
])

five = np.matrix([
    1, 1, 1, 1, 0,
    1, 0, 0, 0, 0,
    0, 1, 1, 1, 0,
    0, 0, 0, 0, 1,
    0, 0, 0, 0, 1,
    0, 1, 1, 1, 0
```

```
])

six = np.matrix([
    0, 0, 1, 1, 0,
    0, 1, 0, 0, 0,
    1, 0, 0, 0, 0,
    1, 1, 1, 1, 0,
    1, 0, 0, 1, 0,
    0, 1, 1, 1, 0
])

seven = np.matrix([
    1, 1, 1, 1, 1,
    0, 0, 0, 1, 0,
    0, 0, 1, 0, 0,
    0, 1, 0, 0, 0,
    0, 1, 0, 0, 0,
    0, 1, 0, 0, 0
])

eight = np.matrix([
    0, 1, 1, 1, 0,
    1, 0, 0, 0, 1,
    0, 1, 0, 1, 0,
    0, 1, 1, 1, 0,
    1, 0, 0, 0, 1,
    0, 1, 1, 1, 0
])

nine = np.matrix([
    0, 0, 1, 1, 0,
    0, 1, 0, 0, 1,
    0, 0, 1, 1, 1,
    0, 0, 0, 0, 1,
    0, 0, 0, 1, 0,
    0, 1, 1, 0, 0
])

def draw_all():
    draw_bin_image(nine.reshape(6, 5))
    draw_bin_image(eight.reshape(6, 5))
    draw_bin_image(seven.reshape(6, 5))
    draw_bin_image(six.reshape(6, 5))
    draw_bin_image(five.reshape(6, 5))
```

```
    draw_bin_image(four.reshape(6, 5))
    draw_bin_image(three.reshape(6, 5))
    draw_bin_image(two.reshape(6, 5))
    draw_bin_image(one.reshape(6, 5))
    draw_bin_image(zero.reshape(6, 5))

data1 = np.concatenate([zero, one, two, three, four, five, six, seven,
eight, nine], axis=0)
data2 = np.concatenate([zero, one, two, three], axis=0)
data = data2

dhnet = algorithms.DiscreteHopfieldNetwork(mode='sync',check_limit=False)
dhnet.train(data)

def half_pre(num):
    half_num = np.copy(num)
    prob = np.random.random(1)
    if prob>0.5:
        half_num[0][0:15] = 0
    else:
        half_num[0][15:30] = 0
    draw_bin_image(half_num.reshape(6, 5))
    result = dhnet.predict(half_num)
    draw_bin_image(result.reshape(6, 5))

def noise_pre(num):
    noise = np.random.random((1, 30))
    noise_num = np.copy(num)
    noise_num[noise <= 0.1] = - noise_num[noise <= 0.1] + 1
    draw_bin_image(noise_num.reshape(6, 5))
    result = dhnet.predict(noise_num)
    draw_bin_image(result.reshape(6, 5))

def normal_pre(num):
    draw_bin_image(num.reshape(6, 5))
    result = dhnet.predict(num)
    draw_bin_image(result.reshape(6, 5))

list1 = [zero, one, two, three, four, five, six, seven, eight, nine]
list2 = [zero, one, two, three]
list = list2

for num in list:
    normal_pre(num)
```

```
for num in list:
    half_pre(num)

for num in list:
    noise_pre(num)
```

Hopfield1.py

```
   import numpy as np
import random

# Data Type
uintType = np.uint8
floatType = np.float32

# Hopfield Class
class HOP(object):
    def __init__(self, N):
        self.N = N
        self.W = np.zeros((N, N), dtype = floatType)

    def kroneckerSquareProduct(self, factor):
        ksProduct = np.zeros((self.N, self.N), dtype = floatType)
        for i in range(0, self.N):
            ksProduct[i] = factor[i] * factor
        return ksProduct

    def trainOnce(self, inputArray):
        mean = float(inputArray.sum()) / inputArray.shape[0]
        self.W = self.W + self.kroneckerSquareProduct(inputArray - mean) /
(self.N * self.N) / mean / (1 - mean)
        index = range(0, self.N)
        self.W[index, index] = 0.

    def hopTrain(self, stableStateList):
        stableState = np.asarray(stableStateList, dtype = uintType)
        if np.amin(stableState) < 0 or np.amax(stableState) > 1:
            print ('Vector Range ERROR!')
            return
        # Train
        if len(stableState.shape) == 1 and stableState.shape[0] == self.N:
            print ('stableState count: 1')
```

```
            self.trainOnce(stableState)
        elif len(stableState.shape) == 2 and stableState.shape[1] == self.N:
            print ('stableState count: ' + str(stableState.shape[0]) )
            for i in range(0, stableState.shape[0]):
                self.trainOnce(stableState[i])
        else:
            print ('SS Dimension ERROR! Training Aborted.')
            return
        print ('Hopfield Training Complete.')

    # Run HOP to output
    def hopRun(self, inputList):
        inputArray = np.asarray(inputList, dtype = floatType)

        if len(inputArray.shape) != 1 or inputArray.shape[0] != self.N:
            print ('Input Dimension ERROR! Runing Aborted.')
            return

        # Run
        matrix = np.tile(inputArray, (self.N, 1))
        matrix = self.W * matrix
        ouputArray = matrix.sum(1)

        # Normalize
        m = float(np.amin(ouputArray))
        M = float(np.amax(ouputArray))
        ouputArray = (ouputArray - m) / (M - m)

        # Binary
        ouputArray[ouputArray < 0.5] = 0.
        ouputArray[ouputArray > 0] = 1.

        return np.asarray(ouputArray, dtype = uintType)

    # Reset HOP to initialized state
    def hopReset(self):
        self.W = np.zeros((self.N, self.N), dtype = floatType)

def printFormat(vector, NperGroup):
    string = ''
    for index in range(len(vector)):
        if index % NperGroup == 0:
            string += '\n'
            # ''' # \END/
```

```
        if str(vector[index]) == '0':
            string += ' '
        elif str(vector[index]) == '1':
            string += '*'
        else:
            string += str(vector[index])
    string += '\n'
    print (string)

# DEMO of Hopfield Net
def HOP_demo():
    zero = [0, 1, 1, 1, 0,
            1, 0, 0, 0, 1,
            1, 0, 0, 0, 1,
            1, 0, 0, 0, 1,
            1, 0, 0, 0, 1,
            0, 1, 1, 1, 0]
    one = [0, 1, 1, 0, 0,
           0, 0, 1, 0, 0,
           0, 0, 1, 0, 0,
           0, 0, 1, 0, 0,
           0, 0, 1, 0, 0,
           0, 0, 1, 0, 0]
    two = [1, 1, 1, 0, 0,
           0, 0, 0, 1, 0,
           0, 0, 0, 1, 0,
           0, 1, 1, 0, 0,
           1, 0, 0, 0, 0,
           1, 1, 1, 1, 1]
    three = [0, 1, 1, 1, 1,
             0, 0, 0, 0, 1,
             0, 1, 1, 1, 0,
             0, 0, 1, 1, 0,
             0, 0, 0, 0, 1,
             0, 1, 1, 1, 1]
    four = [0, 0, 1, 1, 0,
            0, 1, 0, 1, 0,
            1, 0, 0, 1, 0,
            1, 1, 1, 1, 1,
            0, 0, 0, 1, 0,
            0, 0, 0, 1, 0]
    five = [1, 1, 1, 1, 0,
            1, 0, 0, 0, 0,
            0, 1, 1, 1, 0,
```

```
           0, 0, 0, 1, 0,
           0, 0, 0, 1, 0,
           1, 1, 1, 1, 0]
    six = [0, 0, 1, 1, 0,
          0, 1, 0, 0, 0,
          1, 0, 0, 0, 0,
          1, 1, 1, 1, 0,
          1, 0, 0, 1, 0,
          0, 1, 1, 1, 0]
    seven = [1, 1, 1, 1, 1,
            0, 0, 0, 1, 0,
            0, 0, 1, 0, 0,
            0, 1, 0, 0, 0,
            0, 1, 0, 0, 0,
            0, 1, 0, 0, 0]
    eight = [0, 1, 1, 1, 0,
            1, 0, 0, 0, 1,
            0, 1, 1, 1, 0,
            0, 1, 1, 1, 0,
            1, 0, 0, 0, 1,
            0, 1, 1, 1, 0]
    nine = [0, 0, 1, 1, 0,
           0, 1, 0, 0, 1,
           0, 0, 1, 1, 1,
           0, 0, 0, 0, 1,
           0, 0, 0, 1, 0,
           0, 1, 1, 0, 0]

    hop = HOP(5 * 6)
    hop.hopTrain([zero, one, two, three, four, five, six, seven, eight,
nine])

    half_zero = [0, 1, 1, 1, 0,
                1, 0, 0, 0, 1,
                1, 0, 0, 0, 1,
                0, 0, 0, 0, 0,
                0, 0, 0, 0, 0,
                0, 0, 0, 0, 0]
    print ('Half-Zero:')
    printFormat(half_zero, 5)
    result = hop.hopRun(half_zero)
    print ('Recovered:')
    printFormat(result, 5)
```

```
half_two = [0, 0, 0, 0, 0,
           0, 0, 0, 0, 0,
           0, 0, 0, 0, 0,
           0, 1, 1, 0, 0,
           1, 0, 0, 0, 0,
           1, 1, 1, 1, 1]
print ('Half-Two:')
printFormat(half_two, 5)
result = hop.hopRun(half_two)
print ('Recovered:')
printFormat(result, 5)

half_two = [1, 1, 1, 0, 0,
           0, 0, 0, 1, 0,
           0, 0, 0, 1, 0,
           0, 0, 0, 0, 0,
           0, 0, 0, 0, 0,
           0, 0, 0, 0, 0]
print ('Another Half-Two:')
printFormat(half_two, 5)
result = hop.hopRun(half_two)
print ('Recovered:')
printFormat(result, 5)

half_eight = [0, 0, 0, 0, 0,
             0, 0, 0, 0, 0,
             0, 0, 0, 0, 0,
             0, 1, 1, 1, 0,
             1, 0, 0, 0, 1,
             0, 1, 1, 1, 0]
print('Another Half-Eight:')
printFormat(half_eight, 5)
result = hop.hopRun(half_eight)
print('Recovered:')
printFormat(result, 5)

half_seven = [1, 1, 1, 1, 1,
             0, 0, 0, 1, 0,
             0, 0, 1, 0, 0,
             0, 0, 0, 0, 0,
             0, 0, 0, 0, 0,
             0, 0, 0, 0, 0]
print('Another Half-Seven:')
printFormat(half_seven, 5)
```

```
    result = hop.hopRun(half_seven)
    print('Recovered:')
    printFormat(result, 5)


##########################
if __name__ == '__main__':
    HOP_demo()
```