|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **Московский государственный технический университет**  **имени Н.Э. Баумана** |

Факультет: Информатика и системы управления

Кафедра: Информационная безопасность

**Интеллектуальные технологии информационной безопасности**

**Лабораторная работа №5 на тему:**

«Исследование рекуррентной нейронной сети Хопфилда на примере задачи распознавания образа»

Вариант 4

Студент: Евула А. С.

Группа: ИУ8-63

Москва 2021

Цель работы:

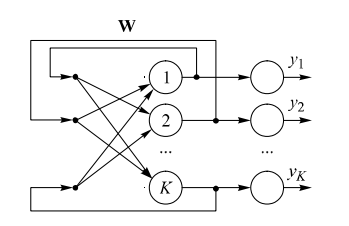
Исследовать процедуры обучения и функционирования рекуррентной нейронной сети (РНС) Хопфилда в качестве устройства автоассоциативной памяти.

Постановка задачи:

Закодировать запоминаемые образы в виде биполярных матриц-паттернов размерности IxJ. Произвести векторизацию матриц. Провести настройку весов РНС Хопфилда согласно правилу ассоциативного обучения. Задать функцию активации и реализовать алгоритм функционирования РНС Хопфилда в синхронном или асинхронном режиме. Протестировать РНС на запомненных эталонных образах. Проверить функционирование РНС Хопфилда на искаженных паттернах.

**Режим**: Асинхронный

**Запоминаемые образы**: 3, 4, 5.



Ход работы

Для лучшей работы НС была выбрана размерность образа равной 28 (7x4).

Были загружены образы “3”, “4”, “5”, произведена настройка весов.

Для проверки функционирования НС были поданы искаженные образы.

Степень искажения

По исходным данным, нейронная сеть распознает образ за ~2 эпохи.

Исходный код программы, ее вывод и веса НС данных представлены в Приложении.

Приложение

Исходный код программы (НС)

# Copyright 2021, Evula A. S., All rights reserved.

# IC8-63 BMSTU

import numpy as np

class NN:

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

self.I = 7 # высота образа

self.J = 4 # ширина

self.k = self.I \* self.J # кол-во пикселей

self.L = 3 # кол-во образов

self.x = [[]] \* self.L

self.x[0] = [1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1] # 3

self.x[1] = [1, 1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1] # 4

self.x[2] = [1, 1, 1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1, 1, 1] # 5

self.imgs = [3, 4, 5]

def train(self):

self.W = np.dot(np.transpose([self.x[0]]), [self.x[0]])

self.W += np.dot(np.transpose([self.x[1]]), [self.x[1]])

self.W += np.dot(np.transpose([self.x[2]]), [self.x[2]])

np.fill\_diagonal(self.W, 0)

np.savetxt('./Weights.txt', self.W, fmt='%2.0d')

def predict(self, X=0):

if X == 0:

X = self.x

for x in range(len(X)):

print('- '\*62)

self.epoch = 0

E = 1

prev = X[x]

while E > 0:

print(f'\tK: {self.epoch}', end=' ')

E, prev = self.tick(prev)

self.epoch += 1

index = -1

for i in range(len(self.x)):

if prev == self.x[i]:

index = i

print(f'\nrecognised image of \'{self.imgs[index]}\'')

print('\n\t origin | finite')

for i in range(self.I):

out = ''

for j in range(self.J):

out += '# ' if X[x][i + j\*self.I] == 1 else ' '

out += ' '

for j in range(self.J):

out += '# ' if self.x[index][i + j\*self.I] == 1 else ' '

print(f'\t{out}\t')

def tick(self, prev):

net = [0] \* self.k

y = [None] \* self.k

for k in range(len(prev)):

for j in range(len(prev)):

net[k] += self.W[j][k] \* prev[j]

y[k] = np.sign(net[k]) if net[k] else prev[k]

print(f'y(n): {y}')

E = [0 if y[k] == prev[k] else 1 for k in range(len(y))].count(1)

prev = y

return E, prev

def main():

nn = NN()

nn.train()

x = [[]] \* 3

x[0] = [-1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1] # 3

x[1] = [1, 1, 1, 1, 1, -1, 1, -1, -1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1, -1, 1] # 5

x[2] = [1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1] # 4

nn.predict(x)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Вывод программы

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

K: 0 y(n): [1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1]

K: 1 y(n): [1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1]

recognised image of '3'

origin | finite

# # # # # #

# #

# #

# # # # # # # #

# #

# # #

# # # # # #

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

K: 0 y(n): [1, 1, 1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1, 1, 1]

K: 1 y(n): [1, 1, 1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1, 1, 1]

recognised image of '5'

origin | finite

# # # # # # #

# #

# # #

# # # # # # # #

# # # #

#

# # # # # # #

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

K: 0 y(n): [1, 1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1]

K: 1 y(n): [1, 1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1]

recognised image of '4'

origin | finite

# # # #

# # # #

# # # # # # #

#

#

# #

# # # #

Веса нейронной сети

0 1 1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 3 1 1 3 3 3 3

1 0 3 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 1 1 1 1

1 3 0 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 1 1 1 1

1 -1 -1 0 -1 -1 3 3 -1 -3 3 -1 -1 3 3 -1 -3 3 -1 -1 3 1 -1 -1 1 1 1 1

-3 -1 -1 -1 0 3 -1 -1 3 1 -1 3 3 -1 -1 3 1 -1 3 3 -1 -3 -1 -1 -3 -3 -3 -3

-3 -1 -1 -1 3 0 -1 -1 3 1 -1 3 3 -1 -1 3 1 -1 3 3 -1 -3 -1 -1 -3 -3 -3 -3

1 -1 -1 3 -1 -1 0 3 -1 -3 3 -1 -1 3 3 -1 -3 3 -1 -1 3 1 -1 -1 1 1 1 1

1 -1 -1 3 -1 -1 3 0 -1 -3 3 -1 -1 3 3 -1 -3 3 -1 -1 3 1 -1 -1 1 1 1 1

-3 -1 -1 -1 3 3 -1 -1 0 1 -1 3 3 -1 -1 3 1 -1 3 3 -1 -3 -1 -1 -3 -3 -3 -3

-1 1 1 -3 1 1 -3 -3 1 0 -3 1 1 -3 -3 1 3 -3 1 1 -3 -1 1 1 -1 -1 -1 -1

1 -1 -1 3 -1 -1 3 3 -1 -3 0 -1 -1 3 3 -1 -3 3 -1 -1 3 1 -1 -1 1 1 1 1

-3 -1 -1 -1 3 3 -1 -1 3 1 -1 0 3 -1 -1 3 1 -1 3 3 -1 -3 -1 -1 -3 -3 -3 -3

-3 -1 -1 -1 3 3 -1 -1 3 1 -1 3 0 -1 -1 3 1 -1 3 3 -1 -3 -1 -1 -3 -3 -3 -3

1 -1 -1 3 -1 -1 3 3 -1 -3 3 -1 -1 0 3 -1 -3 3 -1 -1 3 1 -1 -1 1 1 1 1

1 -1 -1 3 -1 -1 3 3 -1 -3 3 -1 -1 3 0 -1 -3 3 -1 -1 3 1 -1 -1 1 1 1 1

-3 -1 -1 -1 3 3 -1 -1 3 1 -1 3 3 -1 -1 0 1 -1 3 3 -1 -3 -1 -1 -3 -3 -3 -3

-1 1 1 -3 1 1 -3 -3 1 3 -3 1 1 -3 -3 1 0 -3 1 1 -3 -1 1 1 -1 -1 -1 -1

1 -1 -1 3 -1 -1 3 3 -1 -3 3 -1 -1 3 3 -1 -3 0 -1 -1 3 1 -1 -1 1 1 1 1

-3 -1 -1 -1 3 3 -1 -1 3 1 -1 3 3 -1 -1 3 1 -1 0 3 -1 -3 -1 -1 -3 -3 -3 -3

-3 -1 -1 -1 3 3 -1 -1 3 1 -1 3 3 -1 -1 3 1 -1 3 0 -1 -3 -1 -1 -3 -3 -3 -3

1 -1 -1 3 -1 -1 3 3 -1 -3 3 -1 -1 3 3 -1 -3 3 -1 -1 0 1 -1 -1 1 1 1 1

3 1 1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 0 1 1 3 3 3 3

1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 0 3 1 1 1 1

1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 3 0 1 1 1 1

3 1 1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 3 1 1 0 3 3 3

3 1 1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 3 1 1 3 0 3 3

3 1 1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 3 1 1 3 3 0 3

3 1 1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 1 -3 -1 1 -3 -3 1 3 1 1 3 3 3 0