Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Информационная безопасность (ИУ8)

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Лабораторная работа №7 на тему:

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

Вариант 8

Выполнил: Песоцкий А. А.,
студент группы ИУ8-61

Проверила: Коннова Н.С.,
доцент каф. ИУ8

Москва 2020 г.

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 $I \times J$. Произвести векторизацию матриц. Провести настройку весов РНС Хопфилда (рис. 1) согласно правилу ассоциативного обучения (Хебба). Задать функцию активации и реализовать алгоритм функционирования РНС Хопфилда в асинхронном режиме. Протестировать РНС на запомненных эталонных образах. Проверить функционирование РНС Хопфилда на искаженных паттернах (изменены порядка 10 % пикселей).

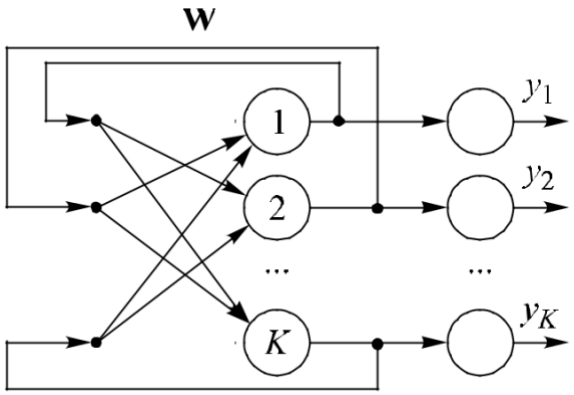


Рис 1. Рекуррентная нейронная сеть Хопфилд

РНС Хопфилда является *автоассоциативной* памятью, которая в ответ на входное воздействие-сигнал

$$\mathbf{x} = (x_1, x_2, ..., x_K),\ x_k \in \{-1, 1\},\quad k = 1, 2, ..., K,$$

формирует отклик структурно соответствующий прототипу.

$$\mathbf{y} = (y_1, y_2, ..., y_K),\ y_k \in \{-1, 1\},\quad k = 1, 2, ..., K,$$

В асинхронном режиме каждая эпоха с номером n = 1,2,… включает в себя следующие вычисления:

$$\mathrm{net}_k^{(n)} = \sum_{j=1}^{k-1} w_{jk} y_j^{(n)} + \sum_{j=k+1}^{K} w_{jk} y_j^{(n-1)},$$

$$y_k^{(n)} = f\left(\mathrm{net}_k^{(n)}\right),\ k = 1, 2, ..., K.$$

Здесь функция активации каждого нейрона

$$f\left(\text{net}_k^{(n)}\right)=\begin{cases}1, & \text{net}_k^{(n)}>0,\\ f\left(\text{net}_k^{(n-1)}\right), & \text{net}_k^{(n)}=0,\\ -1, & \text{net}_k^{(n)}<0.\end{cases}$$

Для начала работы РНС Хопфилда необходимы начальные условия:

$$y_k^{(0)}=x_k,\quad k=1,2,\ldots,K,$$

а также вычислить компоненты матрицы весов:

$$w_{jk}=\begin{cases}\sum_{l=1}^{L} x_j^{(l)}x_k^{(l)}, & j\neq k,\\ 0, & j=k,\end{cases}$$

где L – «емкость» ассоциативной памяти (количество запоминаемых образов); l – номер запоминаемых образов-паттернов.

Ход работы

Требуется запомнить три образа, представляющих собой графическое изображение символов «J», «K», «L». Закодируем их в виде биполярных матриц-паттернов размерности 5 × 3 (рис. 2). Символ «-1» будем выводить как « » (пробел) для большей читабельности. Векторизуем матрицы по столбцам для получения рабочих векторов длины 15:

$\mathbf{x}^{(1)}$ =[-1, -1, -1, 1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1]

$\mathbf{x}^{(2)}$ =[1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1]

$\mathbf{x}^{(3)}$ =[1, 1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1]

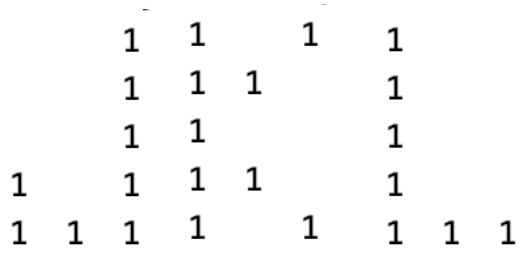


Рисунок 2. Биполярный код образов

Настроим веса РНС Хопфилда:

```
[[ 0.  3.  3.  1.  1. -1.  1. -1.  1. -1. -1. -3. -3. -3.  1.]
 [ 3.  0.  3.  1.  1. -1.  1. -1.  1. -1. -1. -3. -3. -3.  1.]
 [ 3.  3.  0.  1.  1. -1.  1. -1.  1. -1. -1. -3. -3. -3.  1.]
 [ 1.  1.  1.  0.  3. -3. -1. -3. -1.  1.  1. -1. -1. -1.  3.]
 [ 1.  1.  1.  3.  0. -3. -1. -3. -1.  1.  1. -1. -1. -1.  3.]
 [-1. -1. -1. -3. -3.  0.  1.  3.  1. -1. -1.  1.  1.  1. -3.]
 [ 1.  1.  1. -1. -1.  1.  0.  1.  3. -3.  1. -1. -1. -1. -1.]
 [-1. -1. -1. -3. -3.  3.  1.  0.  1. -1. -1.  1.  1.  1. -3.]
 [ 1.  1.  1. -1. -1.  1.  3.  1.  0. -3.  1. -1. -1. -1. -1.]
 [-1. -1. -1.  1.  1. -1. -3. -1. -3.  0. -1.  1.  1.  1.  1.]
 [-1. -1. -1.  1.  1. -1.  1. -1.  1. -1.  0.  1.  1.  1.  1.]
 [-3. -3. -3. -1. -1.  1. -1.  1. -1.  1.  1.  0.  3.  3. -1.]
 [-3. -3. -3. -1. -1.  1. -1.  1. -1.  1.  1.  3.  0.  3. -1.]
 [-3. -3. -3. -1. -1.  1. -1.  1. -1.  1.  1.  3.  3.  0. -1.]
 [ 1.  1.  1.  3.  3. -3. -1. -3. -1.  1.  1. -1. -1. -1.  0.]]
```

Рисунок 3. Матрица весов

1. **Протестируем РНС на эталонных паттернах:**

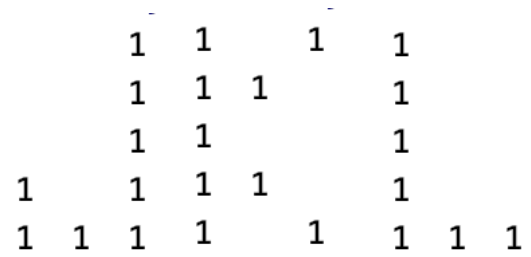
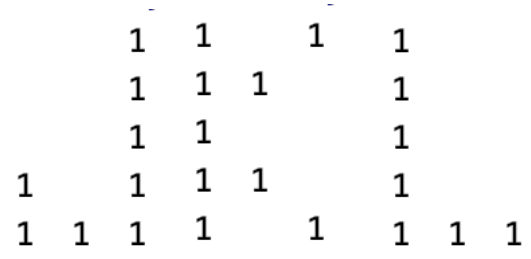


Рисунок 4. Эталоны

Результат работы программы:



2. Протестируем РНС на искаженных паттернах:

$\mathbf{x}^{(1)}$ =[-1, -1, -1, 1, 1, -1, -1, -1, -1, 1, -1, -1, 1, 1, 1]

$\mathbf{x}^{(2)}$ =[-1, -1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1]

$\mathbf{x}^{(3)}$ =[1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]

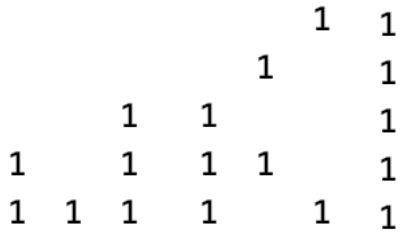


Рисунок 5. Искаженные паттерны

Результат работы программы (восстановление):

$\mathbf{x}^{(1)}$ =[-1, -1, -1, 1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1]

$\mathbf{x}^{(2)}$ =[1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1]

$\mathbf{x}^{(3)}$ =[1, 1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1]

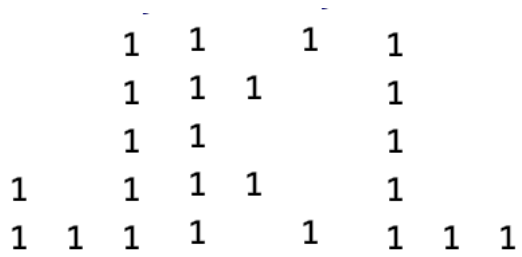


Рисунок 6. Восстановленные паттерны

Выводы

В ходе выполнения работы была изучена рекуррентная нейронная сеть Хопфилда. Были исследованы процедуры обучения и функционирования рекуррентной нейронной сети (РНС) Хопфилда в качестве устройства автоассоциативной памяти.

Принципиальное отличие РНС Хопфилда от других алгоритмов состоит в том, что все коэффициенты матрицы рассчитываются за один цикл, поэтому сразу после НС готова к работе.

Приложение А.

Файл 'hopfield.py'.

```
from computation import *

vectorized_pattern_1 = [-1, -1, -1, 1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1]
vectorized_pattern_2 = [1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1]
vectorized_pattern_3 = [1, 1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1]

'''
Функция восстановления эталона
:param vector_x: входной вектор
:param vector_target: эталон
:param matrix_w: матрица весов обратных связей
:param return: найденный вектор
'''
def recover_RNN(vector_x, vector_target, matrix_w):
    vector_y = vector_x
    vector_prev = list()

    temp = list()
    epoch = 0
    while vector_y != temp:
        temp = vector_target
        if epoch == 0:
            vector_prev = vector_y
        epoch += 1

        y_current = list()
        for k in range(len(vector_x)):
            sum_1 = 0
            for j in range(k - 1):
                sum_1 += matrix_w[j][k] * vector_y[j]
            sum_2 = 0
            for f in range(k + 1, len(vector_x)):
                sum_2 += matrix_w[f][k] * vector_prev[f]
            net = sum_1 + sum_2

            y = function(vector_prev[k], net)
            y_current.append(y)

        vector_prev = vector_y
```

```
        vector_y = y_current

    return vector_y


if __name__ == "__main__":
    patterns = [vectorized_pattern_1, vectorized_pattern_2, vectorized_pattern_3]
    matrix = get_matrix(patterns)
    test = [1, 1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, -1, 1]
    print_pattern(test)
    Y = recover_RNN(test, vectorized_pattern_3, matrix)
    print(Y)
```

Файл ‘computation.py’.

```
import numpy


'''
Вспомогательная функция печати символа
:param s: единичная клетка матрицы
:param newline: флаг переноса на новую строку
'''
def visual_print(s, newline):
    if s == 1:
        if newline == 'n':
            print("%3s" % s, end='')
        else:
            print("%3s" % s)
    else:
        if newline == 'n':
            print("%3s" % ' ', end='')
        else:
            print("%3s" % ' ')



'''
Функция печати паттерна
:param pattern: паттерн
'''
def print_pattern(pattern):
    for i in range(5):
        index = i
```

```
        for j in range(2):
            visual_print(pattern[index], 'n')
            index += 5
        visual_print(pattern[index], 'y')


'''
Функция получения матрицы весов
:param patterns: паттерны
:param return: найденный вектор
'''
def get_matrix(patterns):
    matrix = numpy.zeros((len(patterns[0]), len(patterns[0])))
    for i in range(len(patterns[0])):
        for j in range(len(patterns[0])):
            matrix[i][j] = get_weight(patterns, i, j)
    return matrix


'''
Функция подсчитывает f(net)
:param y_prev: значение с предыдущей эпохи
:param net: net
:param return: значение f(net)
'''
def function(y_prev, net):
    if net > 0:
        return 1
    elif net == 0:
        return y_prev
    else:
        return -1


'''
Функция полуения веса
:param patterns: векторизированные паттерны
:param i: итератор i
:param j: итератор j
:param return: вес
'''
def get_weight(patterns, i, j):
```

```
result = 0
if i != j:
    for s in patterns:
        result += s[i] * s[j]
return result
```