



**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» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

**по лабораторной работе № 5
по дисциплине «Интеллектуальные технологии информационной безопасности»**

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

Вариант 1

**Выполнил: Антипов И.С.,
студент группы ИУ8-63**

**Проверил: Волосова Н.К.,
преподаватель каф. ИУ8**

**г. Москва,
2021 г.**

1. Цель работы

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

2. Условие

Задано 3 образа для запоминания: 0, 1, 8

Представим образы в виде биполярных матриц-паттернов размерности 5x3:

-1	1	-1	-1	1	-1	1	1	1
1	-1	1	1	1	-1	1	-1	1
1	-1	1	-1	1	-1	1	1	1
1	-1	1	-1	1	-1	1	-1	1
-1	1	-1	1	1	1	1	1	1

Режим работы РНС Хопфилда: синхронный

3. Теоретическая часть

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

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

формирует отклик

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

структурно соответствующий прототипу.

Функция активации:

$$f(net_k^n) = \begin{cases} 1, & net_k^n > 0 \\ f(net_k^{n-1}), & net_k^n = 0 \\ -1, & net_k^n < 0 \end{cases}$$

В синхронном режиме каждая эпоха с номером $n=1,2,\dots$ включает в себя следующие вычисления:

$$net_k^n = \sum_{\substack{j=1 \\ (j \neq k)}}^K w_{jk} y_j^{n-1}, \quad y_k^n = f(net_k^n), \quad k = 1, 2, \dots, K$$

Для начала работы РНС Хопфилда необходимо задать начальные условия:

$$y_k^0 = x_k, \quad k = 1, 2, \dots, K$$

а также вычислить компоненты матрицы весов:

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

4. Ход работы

Вначале необходимо представить образы в виде вектора, который составлен на основе биполярных матриц-паттернов.

$$0 = [-1, 1, -1, 1, -1, 1, 1, -1, 1, 1, -1, 1, -1, 1, -1]$$

$$1 = [-1, 1, -1, 1, 1, -1, -1, 1, -1, -1, 1, -1, 1, 1, 1]$$

$$8 = [1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1]$$

Следующий шаг – настройка веса РНС Хопфилда

$$W = \left(\sum_{l=1}^3 X^{lT} x^l \right)'$$

Полученная матрица весов представлена в таблице 1.

Таблица 1. Матрица весов

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

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

Теперь поочередно искажим все образы, а потом восстановим их с помощью нейронной сети. Результаты восстановления представлены на рисунках 1-3

#				#				#						
#	#			#	#			#	#			#	#	
	#				#				#				#	
	#													
#	#	#						#	#	#		#	#	#

Рисунок 1 – Искажение 1 (1- оригинал, 2 – искаженный, 3 – восстановленный)

#				#				#						
#		#		#		#		#		#		#		#
#		#		#		#		#		#		#		#
	#				#				#				#	

Рисунок 2 – Искажение 1 (1- оригинал, 2 – искаженный, 3 – восстановленный)

#	#	#		#	#	#		#	#	#		#	#	#
#		#		#	#	#		#		#		#		#
#	#	#		#	#	#		#	#	#		#	#	#
#		#		#	#	#		#	#	#		#		#
#	#	#		#	#	#		#	#	#		#	#	#

Рисунок 3 – Искажение 1 (1- оригинал, 2 – искаженный, 3 – восстановленный)

Код программы приведен в Приложении А.

5. Выводы

В ходе выполнения лабораторной работы было произведено обучение рекуррентной нейронной сети (РНС) Хопфилда в качестве устройства автоассоциативной памяти.

Все испорченный образцы были успешно восстановлены, однако из-за схожести образцов 0 и 8 не все испорченные образцы от этих оригиналов восстанавливаются корректно

Приложение А. Исходный код программы

Файл main.py

```
import tabulate as tab

class NeuralNetwork:
    def __init__(self, matrix_size, original):
        self.__matrix_size = matrix_size
        self.__matrix = [[0] * (matrix_size) for _ in range(matrix_size)]
        self.__K = matrix_size
        self.__original = original
        self.count_matrix()

    def count_f_net(self, net, pred_f_net):
        if net > 0:
            return 1
        elif net < 0:
            return -1
        else:
            return pred_f_net

    def count_net(self, y, k):
        net = 0
        for j in range(self.__K):
            if j != k:
                net += self.__matrix[j][k] * y[j]
        return net

    def count_weight(self, j, k):
        sum = 0
        for i in range(len(self.__original)):
            sum += self.__original[i][j] * self.__original[i][k]
        return sum

    def count_matrix(self):
        for j in range(self.__matrix_size):
            for k in range(self.__matrix_size):
                if j == k:
                    self.__matrix[j][k] = 0
                else:
                    self.__matrix[j][k] = self.count_weight(j, k)

    def has_change(self, pred_result, result):
        for i in range(self.__matrix_size):
            if result[i] != pred_result[i]:
                return False
        return True

    def print_matrix(self):
        print(tab.tabulate(self.__matrix))

    def recognize(self, sample):
        size = len(sample)
        pred_result = [0] * size
        result = sample
        has_change = False
```

```

        while not has_change:
            for k in range(size):
                result[k] = self.count_f_net(self.count_net(result, k),
pred_result[k])

            has_change = self.has_change(pred_result, result)
            pred_result = result

        return result

def print_pattern(sample, raw, col):
    result = ''
    for i in range(col):
        for j in range(raw):
            if sample[raw * i + j] == -1:
                result += "  "
            else:
                result += " # "
        result += "\n"
    print(result)

def start():

    sample_zero = [-1, 1, -1, 1, -1, 1, 1, -1, 1, 1, -1, 1, -1, 1, -1]
    sample_one = [-1, 1, -1, 1, 1, -1, -1, 1, -1, -1, 1, -1, 1, 1, 1]
    sample_eight = [1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1]

    broken_sample_zero = [-1, 1, -1, 1, -1, 1, 1, -1, 1, -1, -1, 1, -1, 1,
-1]
    broken_sample_one = [-1, 1, -1, 1, 1, -1, -1, 1, -1, -1, -1, -1, 1, 1,
1]
    broken_sample_eight = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

    x = [sample_zero, sample_one, sample_eight]

    nw = NeuralNetwork(3 * 5, x)

    print("-" * 10, "Matrix", "-" * 10)
    nw.print_matrix()

    print("-" * 10, "Sample one", "-" * 10)
    print_pattern(sample_one, 3, 5)
    print_pattern(broken_sample_one, 3, 5)
    restored_one = nw.recognize(broken_sample_one)
    print_pattern(restored_one, 3, 5)

    print("-" * 10, "Sample zero", "-" * 10)
    print_pattern(sample_zero, 3, 5)
    print_pattern(broken_sample_zero, 3, 5)
    restored_one = nw.recognize(broken_sample_zero)
    print_pattern(restored_one, 3, 5)

    print("-" * 10, "Sample eight", "-" * 10)
    print_pattern(sample_eight, 3, 5)
    print_pattern(broken_sample_eight, 3, 5)
    restored_eight = nw.recognize(broken_sample_eight)
    print_pattern(restored_eight, 3, 5)

if __name__ == '__main__':
    start()

```