***Министерство образования Республики Беларусь***

***Учреждение Образования***

***«Брестский Государственный Технический Университет»***

***Кафедра ИИТ***

**Лабораторная работа № 6**

**По дисциплине ИМОД за VI семестр**

**Тема: «Нейронная сеть Хопфилда. Ассоциативная память»**

**Выполнил:**

Студент 4-го курса

Группы АС-59

Абоимов И.В.

**Проверил:**

Савицкий Ю. В.

Брест 2023

Цель работы: Изучить обучение и функционирование нейронной сети Хопфилда при решении задач распознавания образов.

**Ход работы**

**Данные:**

Данные представляют собой последовательность из {0, 1}:

+-+-+-+

|1|1|1|

+-+-+-+

|1|0|0|

+-+-+-+

|1|1|1| = 111 100 111 001 111 - формат данных представляет число 5.

+-+-+-+

|0|0|1|

+-+-+-+

|1|1|1|

+-+-+-+

Однако мы будем их преобразовывать в последовательность из {-1, 1} следующей функцией:

double\* to\_bipolar\_binary(int arr[], size\_t size) {

double\* data = new double[size];

for (size\_t i = 0; i < size; i++) {

if (arr[i] == 0)

data[i] = -1;

else

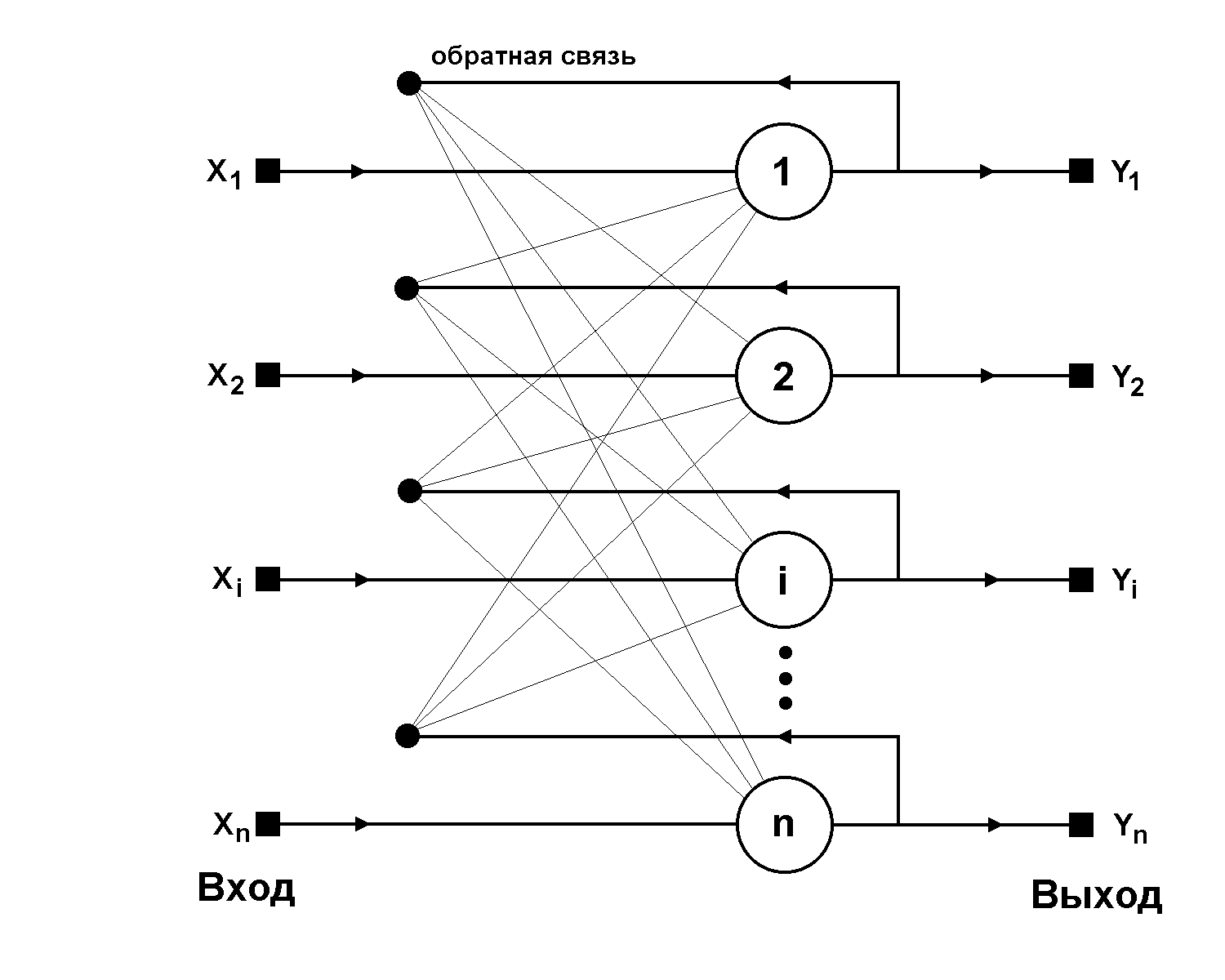
data[i] = 1;

}

return data;

}

**Архитектура сети:**



**Способности сети:**

Размерность входного вектора in = 15.

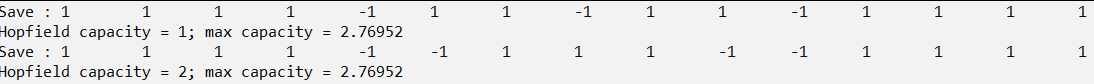
В таком случае максимальная ёмкость для запоминания сети:

Таким образом, мы можем сохранить всего два образа.

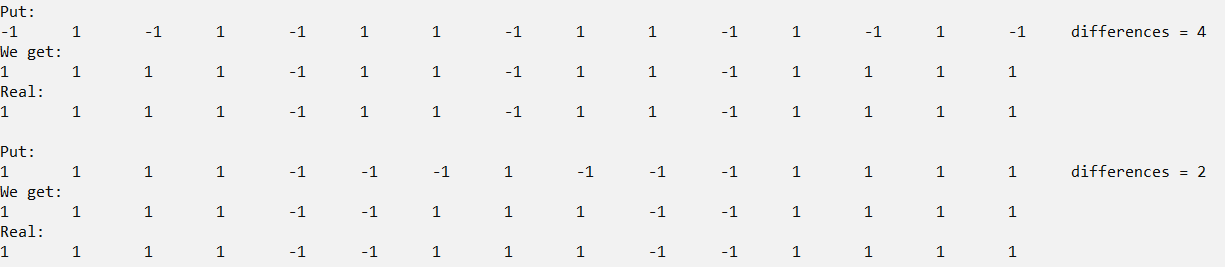
Также размерность матрицы весовых коэффициентов равна [in x in].

**Работа сети:**

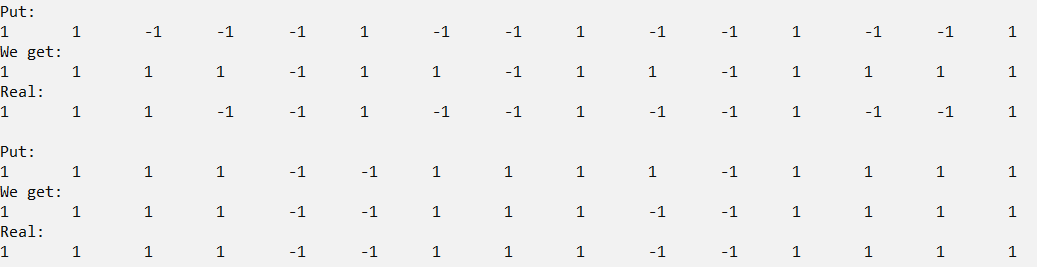
И так, сохраним числа 0 и 5:



А теперь подадим зашумлённые 0 и 5:



Как видно, сеть выдала необходимы результаты. Хорошо, теперь подадим числа зашумлённую 7 и чистую 6:

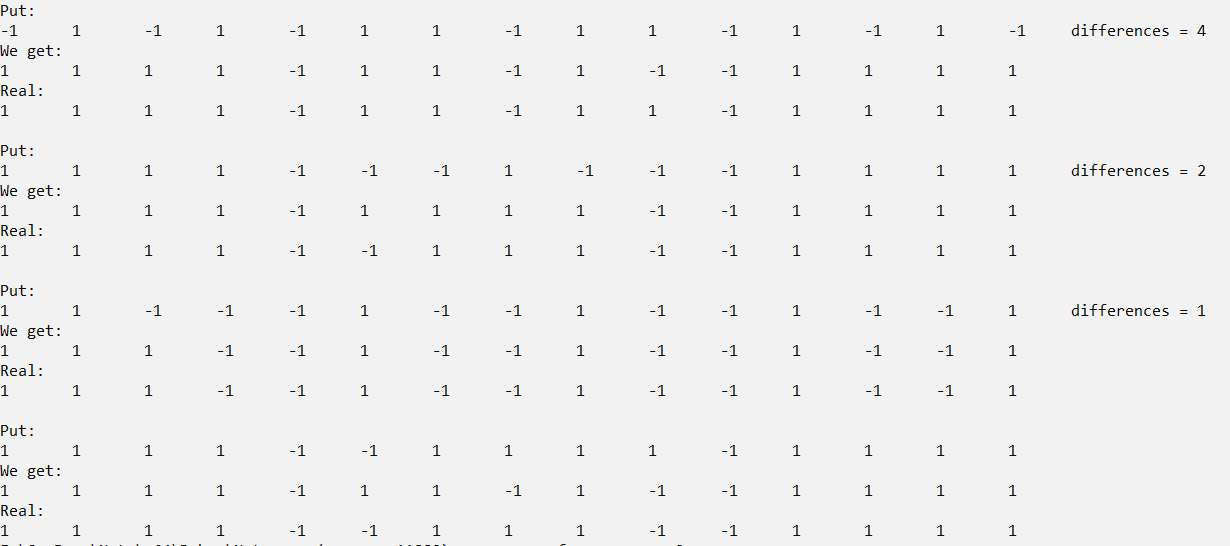


Как мы видим, сеть сказала, что 7 это 0, а 6 это 5.

Теперь давайте сохраним 7:



Подадим зашумлённые 0, 5, 7 и чистую 6.



Как видим, сеть справилась только с 7, а в остальных случаях мы получили химеры. Расстояние Химминга при этом равно 1 для ответов-химер по сравнению необходимыми ответами.

**Исследования сети:**

Отлично, а теперь давайте проведём исследования. Переберём все возможные комбинации 1 и -1 и рассчитаем энергию для каждой из них. Далее построим график для выявления слабостей сети.

Но прежде нам понадобятся функции перевода из двоичной системы счисления в десятичную:

size\_t bipolar\_binary\_sequence\_to\_number(double\* seq, size\_t size) {

size\_t number = 0;

for (size\_t i = 0, j = size - 1; i < size; i++, j--) {

if (seq[i] == 1)

number += pow(2, j);

}

return number;

}

А также функция перевода из десятичной системы счисления в двоичную:

double\* number\_to\_bipolar\_binary\_sequence(size\_t number, size\_t size) {

double\* bipol\_bin\_sequence = new double[size];

for (int i = size - 1; i >= 0; i--) {

if (number % 2 == 1)

bipol\_bin\_sequence[i] = 1;

else

bipol\_bin\_sequence[i] = -1;

number = number / 2;

}

return bipol\_bin\_sequence;

}

А теперь организуем перебор и сохраним результаты в csv-файл для дальнейшей визуализации результатов:

void count\_all\_hopfiled\_enegies(Hopfield& hopfiled) {

double\* temp;

size\_t size = hopfiled.get\_input\_range();

std::ofstream dump\_file;

dump\_file.open("Energy.csv");

for (size\_t i = 0; i < pow(2, size); i++) {

temp = number\_to\_bipolar\_binary\_sequence(i, size);

//write\_array\_inline(temp, size);

double energy = hopfiled.count\_energy(temp);

//std::cout << "\t energy = " << energy <<"\n";

dump\_file << i << ";" << energy << "\n";

delete[] temp;

}

dump\_file.close();

}

Также нам понадобиться функция, которая инвертирует биполярную бинарную последовательность, т.е. заменяет 1 на -1, а -1 на 1:

double\* invert\_bipolar\_binary\_sequence(double\* seq, size\_t size) {

double\* invert\_seq = new double[size];

for (size\_t i = 0; i < size; i++) {

if (seq[i] == -1)

invert\_seq[i] = 1;

else if (seq[i] == 1)

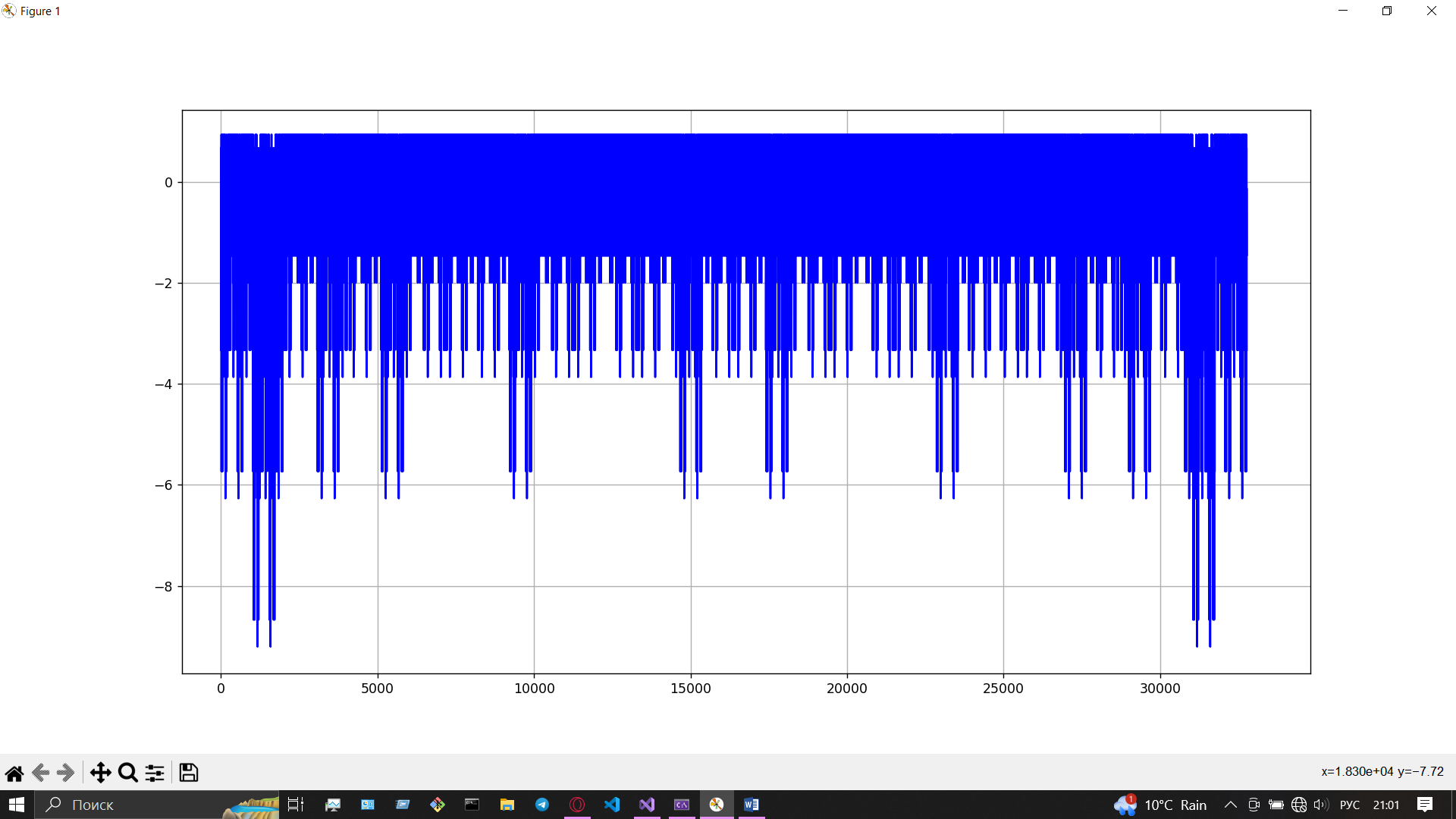
invert\_seq[i] = -1;

}

return invert\_seq;

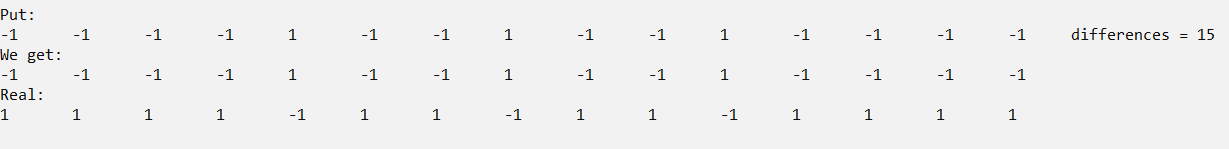
}

И так, приступим к исследованию. Переберём каждую возможную комбинацию из 1 и -1. Таких комбинаций 215 = 32768. Также нам пригодится знание, что последовательность нуля представляет собой число 31599 в десятичной системе счисления, а последовательность 5 = 31183. Энергию считаем по формуле:

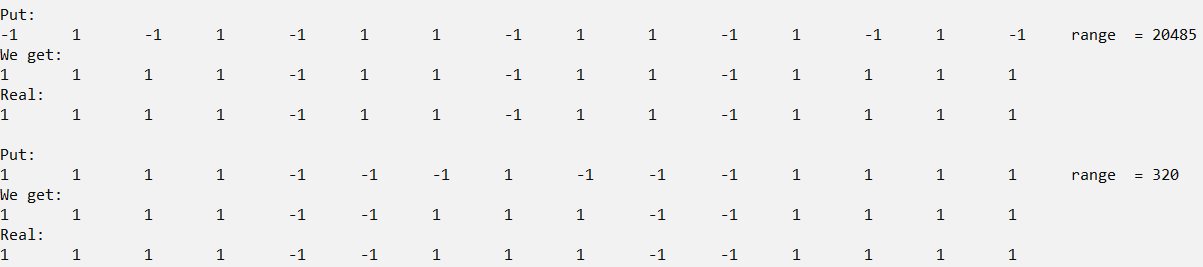


Что показывает нам данный график?

Как мы видим, действительно в районе от 31000 до 32000 находятся минимумы энергии, которые представлены нашими сохранёнными последовательностями 0 и 5. Но что мы видим? Такие же минимумы наблюдаются и в диапазоне значений от 1100 до 1600. Удалось выяснить эти числа 1584 и 1168. При этом число 1168 в биполярном бинарном представлении представляет собой инвертированную последовательность нуля. Подадим инвертированный ноль:

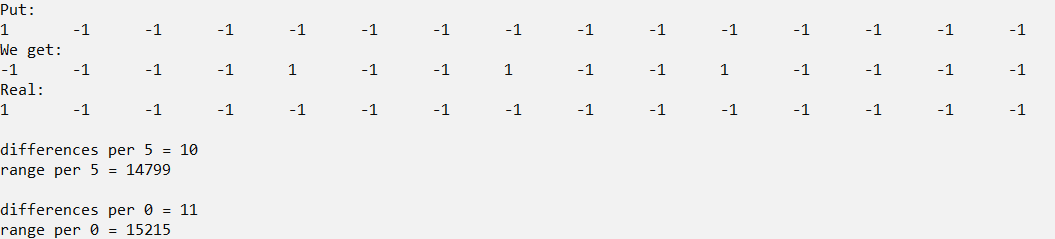


И в действительности, ответ сети — это инвертированный ноль. Возможно стоит оценивать эффективность сети не расстояние Хемминга, а разностью представления последовательностей в десятичной системе счисления, тогда:



Как мы видим расстояние от нормального нуля до искажённого равно 20485, и оно не идентифицируется как инвертированный ноль. Инвертированный ноль представляет число 11114. Это число всё же ближе к инвертированному нулю, чем к обычному, а значит расстояние Хемминга остаётся актуальным для определения качества сети.

Также давайте подадим сети медианное значение, т.е. число , представленное в биполярном бинарном виде:



Ответ сети – инвертированный ноль. И правда, расстояние Хемминга (Hem) у него самое наименьшее из всех опорных наборов, а именно:

Что ж, на этом можно окончить исследования и сделать вывод, что критическое расстояние Хемминга для определения правильного образа равно:

**Код сети:**

#pragma once

#include "DataSets.h"

#include <ctime>

size\_t count\_differences(double\* fir\_arr, double\* sec\_arr, size\_t size) {

size\_t differences = 0;

for (size\_t i = 0; i < size; i++)

if (fir\_arr[i] != sec\_arr[i])

differences++;

return differences;

}

class Hopfield {

double memory\_range;

double memory\_save;

double predict\_iterator = 0;

double\*\* W;

double activate(double x) {

if (x > 0)

return 1;

else if (x < 0)

return -1;

else

return x;

}

public:

double count\_energy(double\* input\_data) {

double enegry = 0;

for (size\_t i = 0; i < this->memory\_range; i++)

for (size\_t j = 0; j < this->memory\_range; j++)

enegry += W[i][j] \* input\_data[i] \* input\_data[j];

enegry \*= -0.5;

return enegry;

}

double get\_capacity() {

return this->memory\_save;

}

double get\_input\_range() {

return this->memory\_range;

}

double get\_max\_capasity() {

return this->memory\_range / (2.0 \* std::log(this->memory\_range));

}

void save(double\* data) {

double\*\* temp\_W = new double\* [this->memory\_range];

for (size\_t i = 0; i < this->memory\_range; i++)

temp\_W[i] = new double[this->memory\_range] {0};

for (size\_t i = 0; i < this->memory\_range; i++)

for (size\_t j = 0; j < this->memory\_range; j++)

temp\_W[i][j] = data[i] \* data[j];

for (size\_t i = 0; i < this->memory\_range; i++)

temp\_W[i][i] = 0;

for (size\_t i = 0; i < this->memory\_range; i++)

for (size\_t j = 0; j < this->memory\_range; j++)

W[i][j] += temp\_W[i][j] / this->memory\_range;

this->memory\_save += 1;

for (size\_t i = 0; i < memory\_range; i++)

delete[] temp\_W[i];

delete[] temp\_W;

}

double\* predict(double\* data, bool is\_auto = false) {

double\* answer = new double[this->memory\_range] {0};

for (size\_t i = 0; i < this->memory\_range; i++) {

for (size\_t j = 0; j < this->memory\_range; j++)

answer[i] += W[i][j] \* data[j];

answer[i] = this->activate(answer[i]);

if (answer[i] == 0)

answer[i] = data[i];

}

if (is\_auto)

if (count\_differences(data, answer, this->memory\_range))

return this->predict(answer);

else

return answer;

else

return answer;

}

Hopfield(size\_t memory\_range) {

this->memory\_range = memory\_range;

W = new double\* [this->memory\_range];

for (size\_t i = 0; i < this->memory\_range; i++)

W[i] = new double[this->memory\_range] {0};

this->memory\_save = 0;

}

~Hopfield() {

for (size\_t i = 0; i < memory\_range; i++)

delete[] W[i];

delete[] W;

}

};