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 *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Введение в искусственный интеллект»

Студент	Кочетков Глеб
Группа	РК6-12М
Тип задания	лабораторная работа
Тема лабораторной работы	№4. Искусственная нейронная сеть
Вариант	№3. Сеть с самоорганизацией на основе конкуренции. Классификация данных по алгоритму Кохонена

Студент _____ **Кочетков Г.О.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Федорук В.Г.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2021 г.

Оглавление

Задание на лабораторную работу 3

Цель выполнения лабораторной работы 4

Содержание отчета 4

Описание реализованной модели нейросети 5

Алгоритм обучения нейросети 7

Программная реализация 8

Процесс обучения и тестирования 10

Источники 14

Приложение А. Текст программы 15

Задание на лабораторную работу

1. Лабораторная работа выполняется в среде ОС Linux с использованием компилятора gcc/g++ языка программирования C/C++. Для создания графических иллюстраций рекомендуется использовать утилиту gnuplot.
2. Разработать, используя язык C/C++, программу, моделирующую поведение искусственной нейронной сети указанного преподавателем типа и обеспечивающую ее обучение для решения задач сжатия данных с потерями, классификации и аппроксимации.
3. Отладить модель нейронной сети и процедуру ее обучения на произвольных данных.
4. Обучить разработанную нейросеть на предложенном преподавателем варианте двухмерных данных (рис. 1).

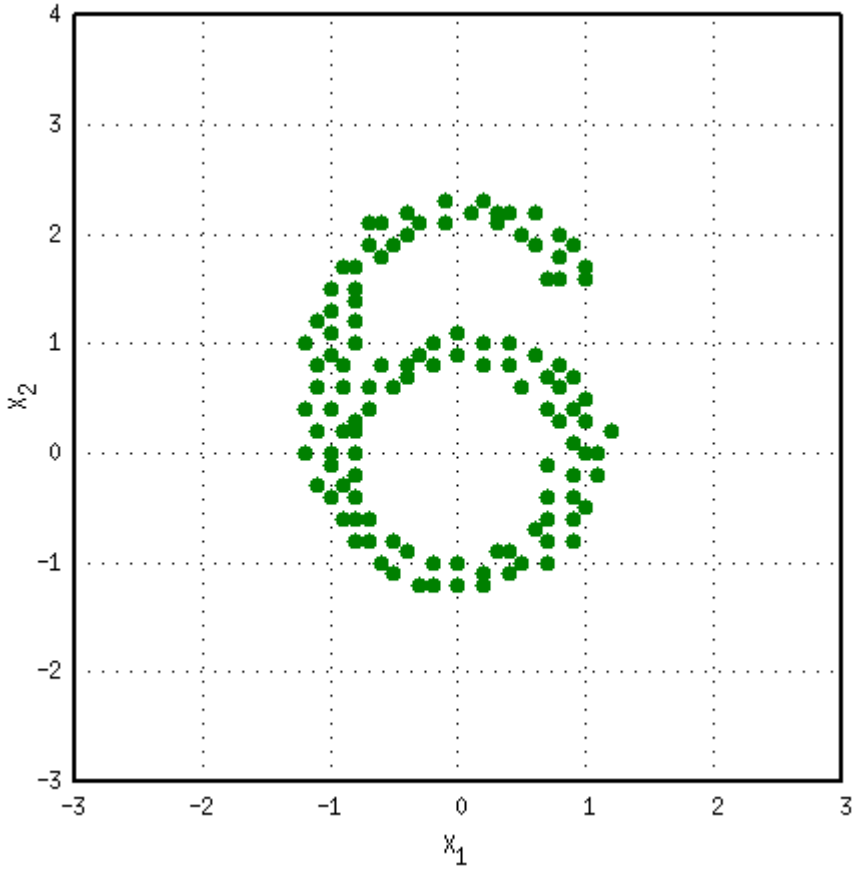

Рис. 1. Обучающие данные.

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы – создание программы, реализующей искусственную нейронную сеть; разработка процедуры обучения сети; использование полученных результатов для решения тестовых задач сжатия данных, классификации и аппроксимации.

Содержание отчета

В отчете представлены пункты:

1. Описание реализованной модели нейросети и процедуры её обучения.
2. Рисунок, иллюстрирующий распределение в пространстве $[x_1,x_2]^T$ обучающих данных.
3. Численные значения, характеризующие начальное состояние, ход обучения и его результат (например, начальные и итоговые значения входных весов нейрона, величина коэффициента обучения, количество циклов обучения и т.п.).
4. Графическое представление результатов обучения нейрона (например, график зависимости выходного сигнала нейрона от входных данных $[x_1,x_2]^T$).
5. Исходный код программы.

Описание реализованной модели нейросети

В работе была реализована искусственный нейронная сеть с самоорганизацией на основе конкуренции. Сеть строится на основе нейронов WTA, в данном случае это однослойная сеть, в которой каждый нейрон получает все компоненты входного вектора X размерностью N. Структурная схема сети представлена на рис. 2:

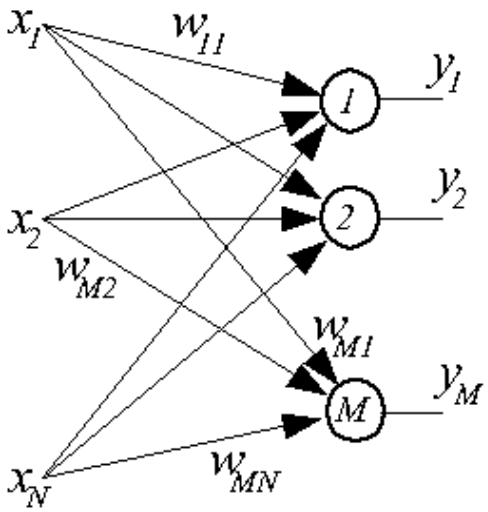

Рис. 2. Структурная схема сети. Обозначения: x_j, j = 1, 2, …, N - входные сигналы; w_{ij}, j = 0, 1, …, N - веса входных сигналов; u_i - взвешенная сумма входных сигналов; y_i, i = 1, 2, …, M - выходные сигналы

Кроме связей, явно представленных в схеме, на этапе обучения имеют место связи между нейронами, позволяющие судить о степени "соседства" нейронов друг с другом, при этом смысл понятия "соседство" может быть разным.

В нейронных сетях, предложенных Т.Кохоненом, соседство нейронов носит топологический характер. В простом случае нейроны слоя Кохонена образуют одномерную цепочку, при этом каждый нейрон имеет, в общем случае, двух ближайших соседей (слева и справа). В более сложном случае нейроны Кохонена образуют двумерную сетку с четырьмя соседями у каждого нейрона (слева, справа, сверху, снизу). В еще более сложном случае сетка гексагональна — у каждого нейрона шесть соседей на плоскости (по

циферблату часов — 2, 4, 6, 8, 10, 12 часов). В данной работе реализован случай, в котором нейроны образуют одномерную цепочку.

Алгоритм обучения нейросети

Подобные сети не требуют для обучения «учителя» и самостоятельно адаптируют свои веса под обучающие данные. Начальные значения весовых коэффициентов всех нейронов выбираются случайным образом.

Алгоритм обучения состоит из следующих шагов:

1. На вход сети подается обучающий вектор X^k.
2. Для каждого нейрона определяется $d(X^k, W_i)$ – расстояние между векторами X^k и W_i. В данной работе в качестве измерения расстояния используется эвклидова мера:

$$d(X^k, W_i) = ||X - W_i||_2 = \sqrt{\sum_{j=1}^{N} (x_j - w_{ij})^2}$$

3. Выбор нейрона-победителя с наименьшим расстоянием.
4. Вокруг победителя образуется окрестность S_w^k из нейронов-соседей с известным «расстоянием» до победителя.
5. Веса нейрона-победителя и веса его соседей из области S_w^k уточняются по алгоритму Кохонена:

   $W_i^{k+1} = W_i^k + \eta_i^k \cdot G^k(i, X^k) \cdot (X^k - W_i^k)$,

   где функция соседства $G^k(i, X^k)$ определяется в виде:

   $G^k(i, X^k) = \exp(\frac{d^2(i, X^k)}{2 \cdot (\sigma^k)^2})$,

   где $d(X^k, W_i)$ – расстояние от i-ого нейрона до нейрона победителя.
6. Уменьшение коэффициентов η_i^k и σ^k.

Однако при подобном алгоритме обучения могут образовываться «мертвые» нейроны, которые ни разу не победили в процессе обучения, чтобы этого избежать, используется модифицированное обучение на учете числа побед нейронов и штрафах для наиболее «зарвавшихся» нейронов – после достижения определенного количества побед самый активный нейрон выбывает из процесса обучения на определенное количество циклов.

Программная реализация

Программная реализация нейросети, её обучения и использования была выполнена на языке C++.

Реализация разбита на несколько файлов (листинг 1). Полное содержание каждого файла расположено в приложении А.

Листинг 1. Файлы исходных данных программы

```
.
├── compile.sh
├── main.cpp
├── neuron_wta.cpp
├── neuron_wta.hpp
├── neuron_network.cpp
├── neuron_network.hpp
└── plot_window.gnu
```

В файлах *neuron_wta.hpp* и *neuron_wta.cpp* находятся объявление и реализация класса нейрона WTA соответственно. В файле *main.cpp* находятся считывание и подготовка обучающих данных, обучение нейрона. Файл *plot_window.gnu* содержит скрипт для утилиты *gnuplot*, который создает изображение тестовой выборки, которую получает нейронная сеть. Также был написан shell-скрипт *compile.sh*, который позволяет правильно скомпилировать программу.

Основная часть программы содержится в реализации классов нейрона и нейронной сети (слоя). Класс нейрона хранит текущие веса своих входов и количество этих входов, счетчик побед нейрона, время «выключения» нейрона, дистанция до нейрона-победителя. Методы нейрона включают (приложение А, листинг А.1):

1. конструктор по количеству входов;
2. подсчет взвешенной суммы входных сигналов;

3. корректировка весовых коэффициентов;
4. установка случайных значений весовых коэффициентов (при помощи функции srand());
5. печать значений весовых коэффициентов;
6. нормализация вектора;
7. метод учета количества побед;
8. метод расчета дистанции до нейрона-победителя.

Класс нейронной сети содержит в себе массив нейронов, поле с файлом вывода данных, а также значение коэффициентов $\eta_i^k, \sigma^k, S_w^k$. Методы нейронной сети включают (приложение А, листинг А.3):

1. конструктор;
2. деструктор;
3. определение победителя;
4. обучение сети;
5. вывод весовых коэффициентов нейронов;
6. расчет дистанции до нейрона-победителя;
7. метод уменьшения коэффициентов $\eta_i^k, \sigma^k, S_w^k$.

Процесс обучения и тестирования

Обучение производилось на точках, представленных на рис. 1.

Данные представлены в текстовой форме в файле исходных данных *coordinates.dat* (листинг 2), где первая колонка – это x_1, а вторая – x_2.

Листинг 2. Часть данных из файла

-1.0 1.1	0.8 2.0	-1.1 0.6	0.8 1.6
0.7 0.7	-0.8 1.5	0.5 2.0	-0.8 0.0
-0.7 -0.8	-1.1 0.2	0.9 0.4	-0.7 0.6
-1.0 -0.1	-0.8 -0.4	0.0 0.9	0.0 -1.2
0.1 2.2	0.4 0.8	-0.9 -0.6	-0.5 -1.1
0.7 -0.6	-0.4 0.8	-0.4 -0.9	-0.8 -0.8
0.2 -1.1	1.0 0.0	-0.3 2.1	-1.0 -0.4
-0.2 -1.2	0.7 -1.0	0.9 -0.4	-0.7 -0.6
-0.6 2.1	-0.8 0.3	-0.7 1.9	-1.2 0.0

Данные в программе считываются с помощью *CSV*-парсера, реализованного в рамках предыдущих работ. После считывания из данных формируются следующий вектор, который передается в функцию обучения нейронной сети (приложение А, листинг А.4):

vector< vector<double> > X_learn - вектор входных векторов нейрона;

Из-за небольшого количества входных данных обучение производилось в несколько эпох. Количество эпох определяется динамически в зависимости от количества нейронов.

Результаты работы программы

Вывод программы записывается в файлы test.dat и weights.dat. В файле test.dat значения вектора X с принадлежностью данного вектора к определенному нейрону. В файле weights.dat хранятся значения весовых коэффициентов нейронов. Оба файла используются в скрипте plot_window.gnu (листинг А.6) для преобразования вывода программы в изображение при помощи программы gnuplot. Треугольниками обозначаются нейроны, кружками – векторы входных данных, помимо этого, соседние нейроны соединены линиями. Результаты работы программы представлены на изображениях 3-7:

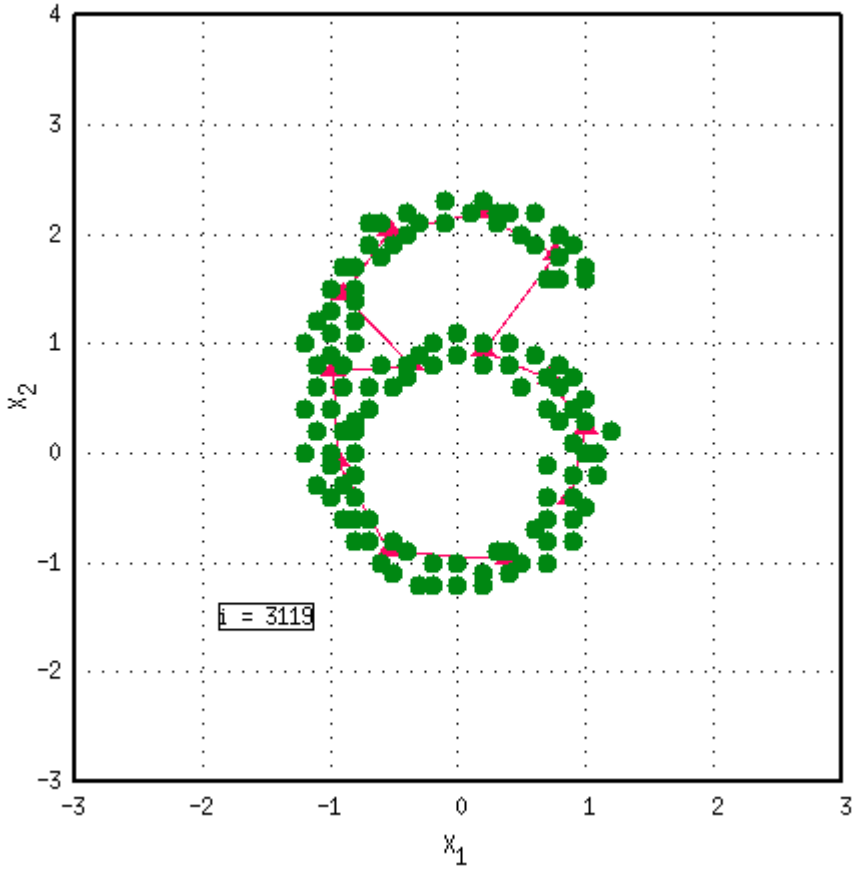

Рис. 3. Результаты работы 12-ти нейронов

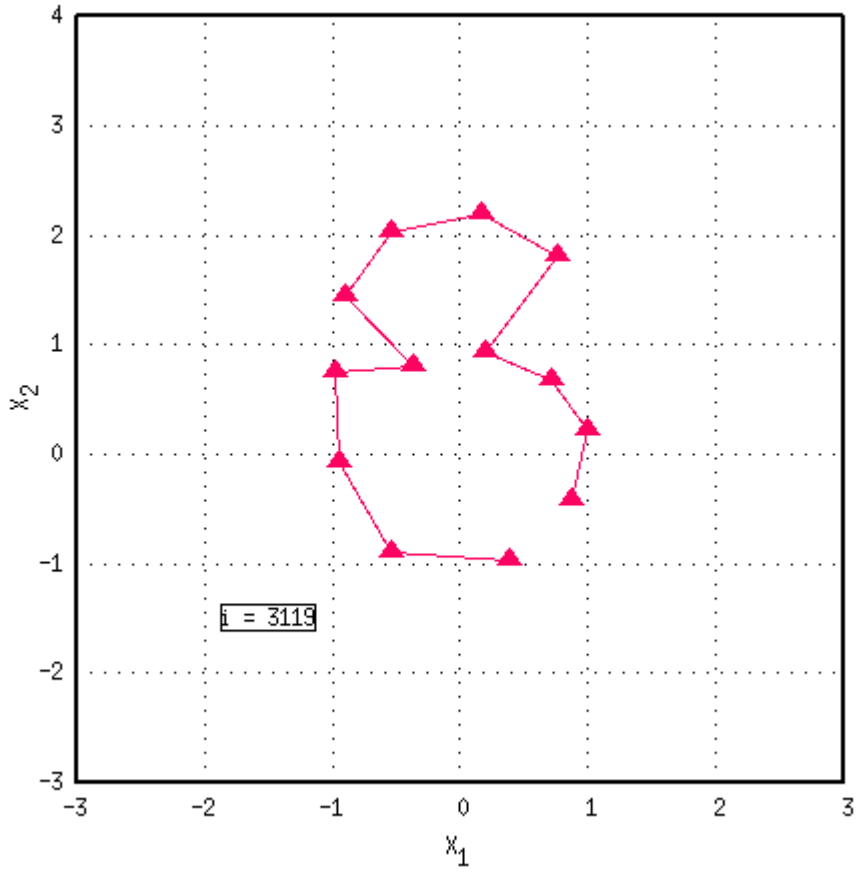

Рис. 4. Результаты работы 13-ти нейронов без отображения входных данных

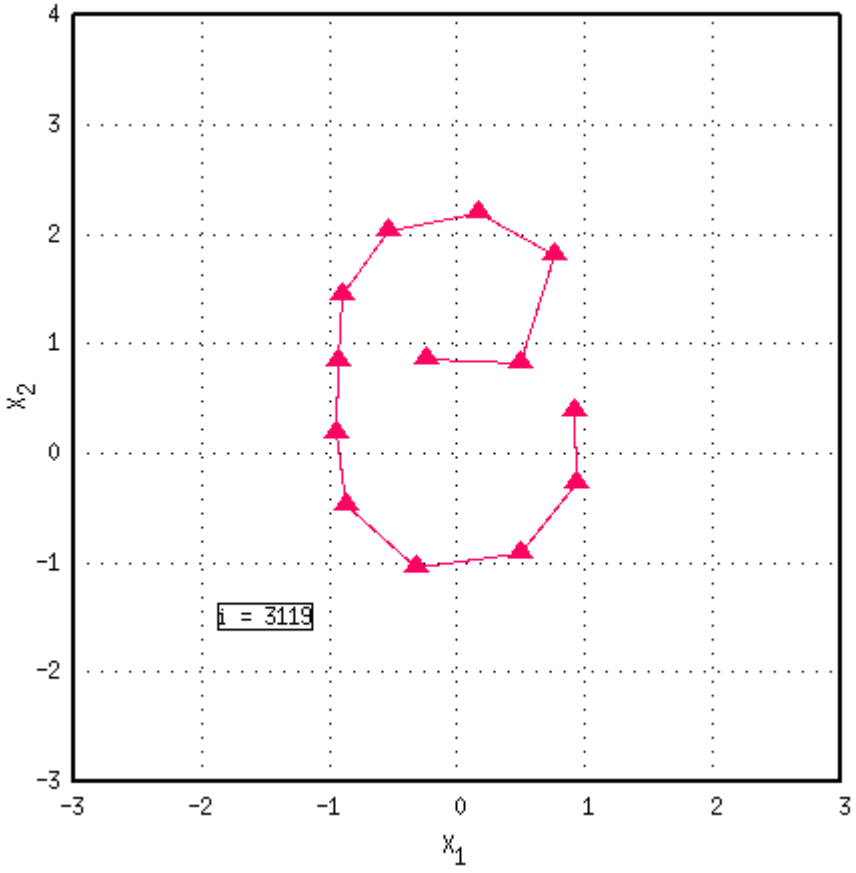

Рис. 5. Результаты работы 13-ти нейронов без отображения входных данных

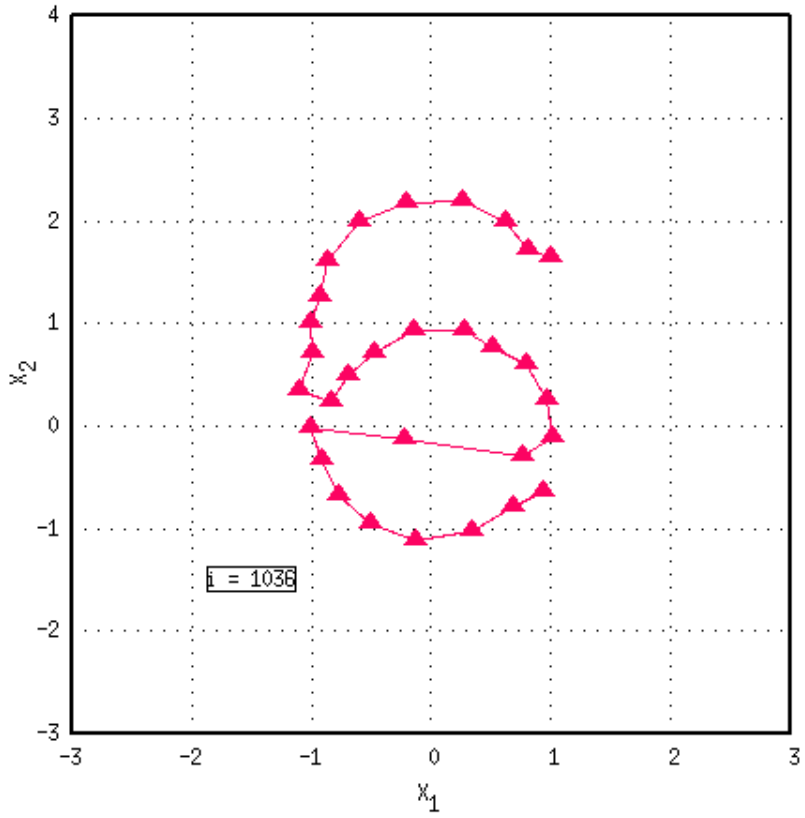

Рис. 6. Результаты работы 30-ти нейронов без отображения входных данных

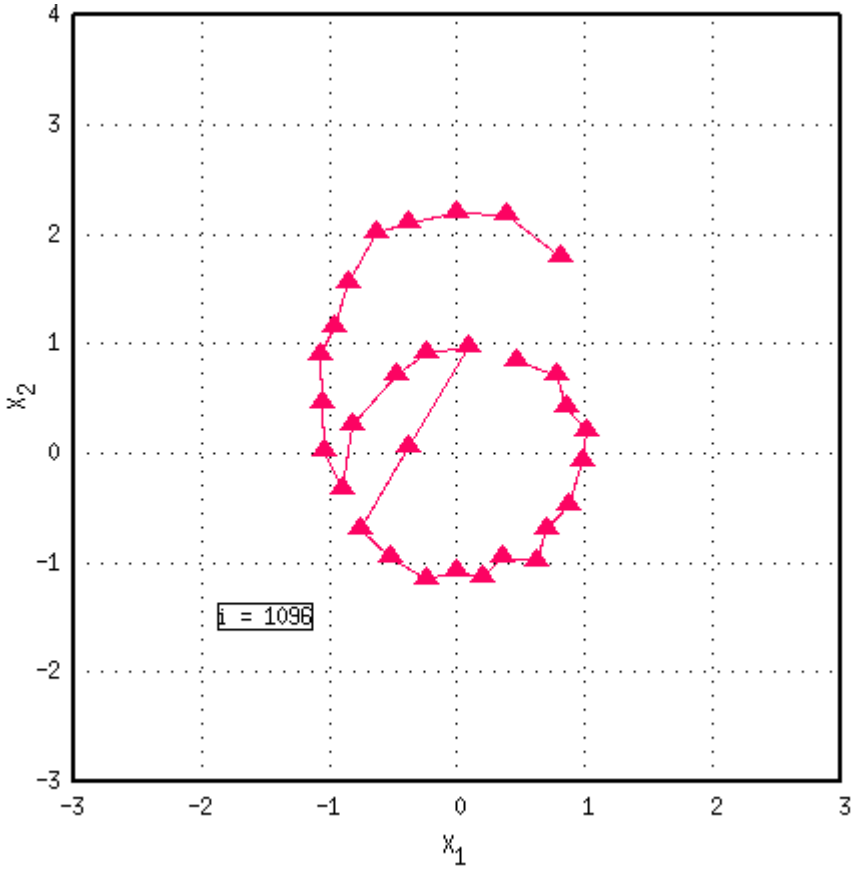

Рис. 7. Результаты работы 30-ти нейронов без отображения входных данных

Источники

1. Федорук, В. Г. Нейроны типа WTA / В. Г. Федорук. — Текст : электронный // БиГОР : [сайт]. — URL: http://bigor.bmstu.ru/?cnt/?doc=NN/014-neurons.mod/?cou=NN/base.cou.
2. Федорук, В. Г. Методические указания по проведению лабораторной работы №1 "Программирование искусственного нейрона" по курсу "Искусственные нейронные сети" / В. Г. Федорук. — Текст : электронный // fedoruk.comcor.ru : [сайт]. — URL: http://fedoruk.comcor.ru/AI_mag/NNlab/lab1.html.

Приложение А. Текст программы

Листинг А.1. Файл *neuron_wta.hpp*

```
#pragma once
#include <vector>
#include <random>
#include <cmath>
#include <iostream>
#include <fstream>

class Neuron_WTA {
 private:
    std::vector<double> W;
    int input_num;
    int distance_to_winner;
    int wins_count;
    int countdown;
 public:
    Neuron_WTA(int _input_num);
    void reduce_countdown();
    int get_countdown();
    int get_wins_count();
    bool win_check(int _max_wins, int neuron_num);
    double output_signal(std::vector<double>& _X);
    void change_weights(std::vector<double>& _X, double _n, double _sigma);
    int get_input_num();
    std::vector<double> get_weights();
    void print_weights();
    void set_rand_W();
    void add_win();
    double calculate_distance(std::vector<double>& _X);
    std::vector<double> vector_normalize(std::vector<double>& _vector);
    void set_distance_to_winner(int _distance);
    int get_distance();
    void print_weights(std::ofstream& file);
};
```

Листинг А.2. Файл *neuron_wta.cpp*

```
#include "neuron_wta.hpp"

Neuron_WTA::Neuron_WTA(int _input_num) {
    input_num = _input_num;
    wins_count = 0;
    countdown = 0;
    W.resize(input_num);
}

double Neuron_WTA::output_signal(std::vector<double>& X) {
    double sum = 0.0;
    for(int i = 0; i < input_num; i++) {
        sum += W[i] * X[i];
    }
    return sum;
}

void Neuron_WTA::change_weights(std::vector<double>& X, double n, double
sigma) {
```

```
    for(int i = 0; i < input_num; i++) {
        W[i] = W[i] + n * exp(-(distance_to_winner * distance_to_winner) / (2
* sigma * sigma)) * (X[i] - W[i]);
    }
}

int Neuron_WTA::get_input_num() {
    return input_num;
}

int Neuron_WTA::get_countdown() {
    return countdown;
}

int Neuron_WTA::get_wins_count() {
    return wins_count;
}

std::vector<double> Neuron_WTA::get_weights() {
    return W;
}

void Neuron_WTA::print_weights() {
    for(int i = 0; i < W.size(); i++) {
        std::cout << "W" << i+1 << " = " << W[i] << std::endl;
    }
}

void Neuron_WTA::set_distance_to_winner(int _distance) {
    distance_to_winner = _distance;
}

void Neuron_WTA::reduce_countdown() {
    countdown--;
}

bool Neuron_WTA::win_check(int max_wins, int neurons_num) {
    if ((wins_count < max_wins) && (countdown == 0)) {
        return true;
    }
    if (wins_count == max_wins) {
        countdown = max_wins * max_wins + neurons_num;
        wins_count = 0;
        return false;
    }
    if (countdown != 0) {
        return false;
    }
    return false;
}

int Neuron_WTA::get_distance() {
    return distance_to_winner;
}

void Neuron_WTA::add_win() {
    wins_count++;
}

void Neuron_WTA::set_rand_W() {
    double temp;
    for(int i = 0; i < input_num; i++) {
```

```
        W[i] = rand() % 3;
        temp = rand() % 100;
        temp /= 100;
        W[i] -= 1.5;
        W[i] += temp;
    }
    //W = vector_normalize(W);
}

// Нормализация вектора
std::vector<double> Neuron_WTA::vector_normalize(std::vector<double>& _vector)
{
    double sum = 0.0;
    std::vector<double> vector = _vector;
    for(int i = 0; i < vector.size(); i++) {
        sum += (vector[i] * vector[i]);
    }
    double root = sqrt(sum);
    for(int i = 0; i < vector.size(); i++) {
        vector[i] /= root;
    }
    return vector;
}

//Эвклидова мера
double Neuron_WTA::calculate_distance(std::vector<double>& _X) {
    double sum = 0.0;
    std::vector<double> X = _X;
    for (int i = 0; i < input_num; i++) {
        sum += ((X[i] - W[i]) * (X[i] - W[i]));
    }
    sum = sqrt(sum);
    return sum;
}

void Neuron_WTA::print_weights(std::ofstream& weights_file) {
        weights_file << W[0] << " " << W[1] << std::endl;
}
```

Листинг А.3. Файл *neuron_network.hpp*

```
#pragma once
#include <vector>
#include <iostream>
#include <fstream>
#include "neuron_wta.hpp"
#define CLASSES_NUM 10

class Network_WTA {
 private:
    std::vector<Neuron_WTA> Neurons;
    std::ofstream output_file;
    double n;
    double sigma;
    int Swk;
 public:
    Network_WTA(unsigned int _neuron_num);
    ~Network_WTA();
    int define_winner(std::vector<double>& _X, int iteration);
    void network_learn(std::vector<std::vector <double> >& X_learn,
```

```
std::ofstream& weights_file, int age);
    void print_neurons_weights(std::ofstream& _file);
    void calculate_distance_to_winner(std::vector<double>& _X, int
_winner_index);
    void check_koeffs();
};
```

Листинг А.4. Файл *neuron_network.cpp*

```
#include "neuron_network.hpp"
#define INPUT_NUMS 2
#define N 1.0
#define SIGMA 1.0
#define SWK 14

Network_WTA::Network_WTA(unsigned int _neuron_num) {
    // Файл для записи выходных данных
    output_file.open("test.dat");
    if (!output_file) {
        std::cout << "ERROR: can't open file 'test.dat'" << std::endl;
        return;
    }

    srand(time(NULL));
    for(int i = 0; i < _neuron_num; i++) {
        Neuron_WTA Ner(INPUT_NUMS);
        Ner.set_rand_W();
        Neurons.push_back(Ner);
    }

    n = N;
    sigma = SIGMA;
    Swk = Neurons.size();
}

Network_WTA::~Network_WTA() {
    output_file.close();
}

// Определение нейрона-победителя и переопределение его весов
int Network_WTA::define_winner(std::vector<double>& X, int iteration) {
    int max_wins = Neurons.size();
    double min_distance = 100000.0;
    double temp_distance = -10000.0;
    int winner_index = 0;
    if (iteration < 1) {
        for(int i = 0; i < Neurons.size(); i++) {
            if (Neurons[i].win_check(max_wins, Neurons.size())) {
                temp_distance = Neurons[i].calculate_distance(X);
                if(temp_distance < min_distance) {
                    min_distance = temp_distance;
                    winner_index = i;
                }
            } else {
                Neurons[i].reduce_countdown();
            }
        }
    } else {
        for(int i = 0; i < Neurons.size(); i++) {
            temp_distance = Neurons[i].calculate_distance(X);
```

```
            if(temp_distance < min_distance) {
                min_distance = temp_distance;
                winner_index = i;
            }
        }
    }
    output_file << X[0] << " " << X[1] << " " << winner_index << " " <<
std::endl;
    return winner_index;
}

void Network_WTA::calculate_distance_to_winner(std::vector<double>& X, int
winner_index) {
    int distance = 0;
    for (int i = 0; i < Neurons.size(); i++) {
        Neurons[i].set distance to winner(abs(winner index - i));
    }
}

void Network_WTA::check_koeffs() {
    if(n > 0.08) {
        n -= 0.0007;
    } else {
        n = 0.08;
    }
    if(sigma > 0.06) {
        sigma -= 0.0007;
    } else {
        sigma = 0.06;
    }
    if (Swk > 2) {
        Swk -= 1;
    } else {
        Swk = 2;
    }
}
void Network_WTA::network_learn(std::vector<std::vector <double> >& X_learn,
std::ofstream& weights_file, int age) {
    std::vector<double> X;
    int winner_index;

    for(int i = 0; i < X_learn.size(); i++) {
        //X = Neurons[i].vector_normalize(X_learn[i]);
        X = X_learn[i];
        winner_index = define_winner(X, age);
        Neurons[winner_index].add_win();
        calculate_distance_to_winner(X, winner_index);
        for(int j = 0; j < Neurons.size(); j++) {
            if(Neurons[j].get_distance() < Swk) {
                Neurons[j].change_weights(X, n, sigma);
            }
        }
        print_neurons_weights(weights_file);
        weights_file << std::endl;
        check_koeffs();
    }
}
void Network_WTA::print_neurons_weights(std::ofstream& weights_file) {
    for(int i = 0; i < Neurons.size(); i++) {
        Neurons[i].print_weights(weights_file);
    }
}
```

Листинг А.5. Файл *main.cpp*

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include "neuron_network.hpp"

using namespace std;

// Убирает пробелы с концов переданной строки
string trim_spaces(string line) {
    string trimmed;
    string space_like_characters = " \t\n";

    size_t first_non_space = line.find_first_not_of(space_like_characters);
    size_t last_non_space = line.find_last_not_of(space_like_characters);

    trimmed = line.substr(first_non_space, last_non_space - first_non_space +
1);

    return trimmed;
}

// Разделяет строку на массив строк по переданному разделителю, сокращая
пробелы на концах разделенных строк
vector<double> csv_parser(string line, string delimiter) {
    vector<double> splited_line;

    size_t prev_pos = 0, delimiter_pos = line.find(delimiter);
    string sub_line;
    while (delimiter_pos != string::npos) {
        sub_line = line.substr(prev_pos, delimiter_pos - prev_pos);
        splited_line.push_back(stod(trim_spaces(sub_line)));

        prev_pos = delimiter_pos + delimiter.size();
        delimiter_pos = line.find(delimiter, prev_pos);
    }
    splited_line.push_back(stod(trim_spaces(line.substr(prev_pos))));

    return splited_line;
}

int main(int argc, char** argv) {
    if (argc != 2) {
        cout << "ERROR: wrong num of arguments" << endl;
        return 0;
    }

    unsigned int num_of_neurons = atoi(argv[1]);
    if (num_of_neurons < 1) {
        cout << "ERROR: number of neurons must be > 0" << endl;
        return 0;
    }
    // чтение из файла обучающей выборки
    ifstream learn_data_file;
    learn_data_file.open("coordinates.dat");
    if (!learn_data_file) {
        cout << "ERROR: can't open file 'coordinates.dat'" << endl;
        return 1;
```

```
    }

    vector< vector<double> > X_learn;
    vector<double> x_local(2);
    string line;
    while(getline(learn_data_file, line)) {
        vector<double> values = csv_parser(line, " ");
        x_local[0] = values[0];
        x_local[1] = values[1];
        X_learn.push_back(x_local);
    }
    learn_data_file.close();

    Network_WTA N(num_of_neurons);

    std::ofstream weights_file;
    weights_file.open("weights.dat");
    if (!weights_file) {
        std::cout << "ERROR: can't open file 'weights.dat'" << std::endl;
        return 0;
    }

    int ages = num_of_neurons * 2;
    for (int i = 0; i < ages; i++) {
        N.network_learn(X_learn, weights_file, i);
    }
    weights_file.close();
    return 0;
}
```

Листинг А.6. Файл *plot_window.gnu*

```
reset
set term x11
set size square

set border linewidth 1.5
set pointsize 1.7

# стиль для тестовых данных с классом 1 (класс 0 отображается дефолтным
стилем)
set style line 1 lc rgb '#008712' pt 7   # circle green

# стиль для весов
set style line 2 lc rgb '#fc0362' pt 9   # triangle red

unset key

set tics scale 0.1
set xtics 1
set ytics 1
set yrange[-3:4]
set xrange[-3:3]
set xlabel 'X_1'
set ylabel 'X_2'

set grid
set macro

set style increment user
```

```
N = system("wc -l test.dat")
neurons = 13

do for [i=0:(N-1)] {
    # номер итерации обучения
    LABEL = "i = " . i
    set obj 10 rect at -1.5,-1.5 size char strlen(LABEL), char 1
    set obj 10 fillstyle empty border -1 front
    set label 10 at -1.5,-1.5 LABEL front center


    plot 'weights.dat' every :::i::i w l ls 2, 'weights.dat' every :::i::i w p
ls 2, 'coordinates.dat' u 1:2 w p ls 1

    pause 0.01
}
pause mouse
```

Листинг А.7. Файл *compile.sh*

```
c++ main.cpp neuron_wta.cpp neuron_network.cpp
```