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	№3. Искусственный нейрон
Вариант	5-7. Нейроны типа WTA, набор данных №7

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 нейрона 5

Алгоритм обучения нейрона 6

Программная реализация 7

Процесс обучения и тестирования 9

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 искусственного трехвходового нейрона указанного преподавателем типа и обеспечивающую его обучение для решения задачи классификации.
3. Отладить модель нейрона и процедуру его обучения на произвольных двумерных данных. Рекомендуется, в тех ситуациях, когда возможно, использовать режим обучения "оффлайн".
4. Обучить разработанный нейрон на предложенном преподавателем варианте двухмерных данных (рис. 1) и проверить его работу на ряде контрольных точек.

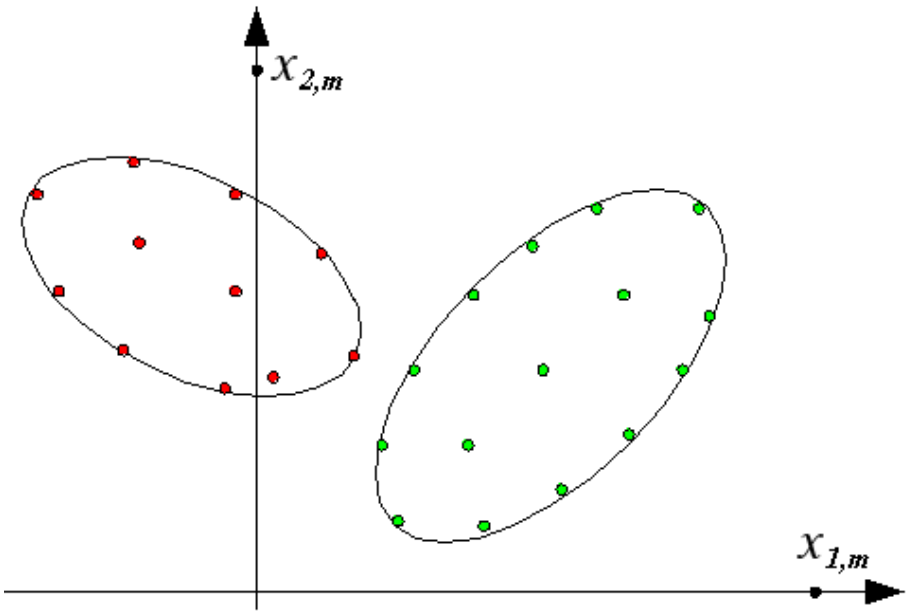

Рис. 1. Обучающие данные. Вариант №7

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы – создание программы, реализующей искусственный нейрон; разработка процедуры обучения нейрона; использование полученных результатов для решения тестовых задач классификации и аппроксимации.

Содержание отчета

В отчете представлены пункты:

1. Описание реализованной модели нейрона и процедуры его 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 нейрона

В работе был реализован искусственный нейрон типа WTA (Winner Takes All). Нейроны этого типа используются группами, в которых конкурируют между собой. Его структурная схема представлена на рис. 2.

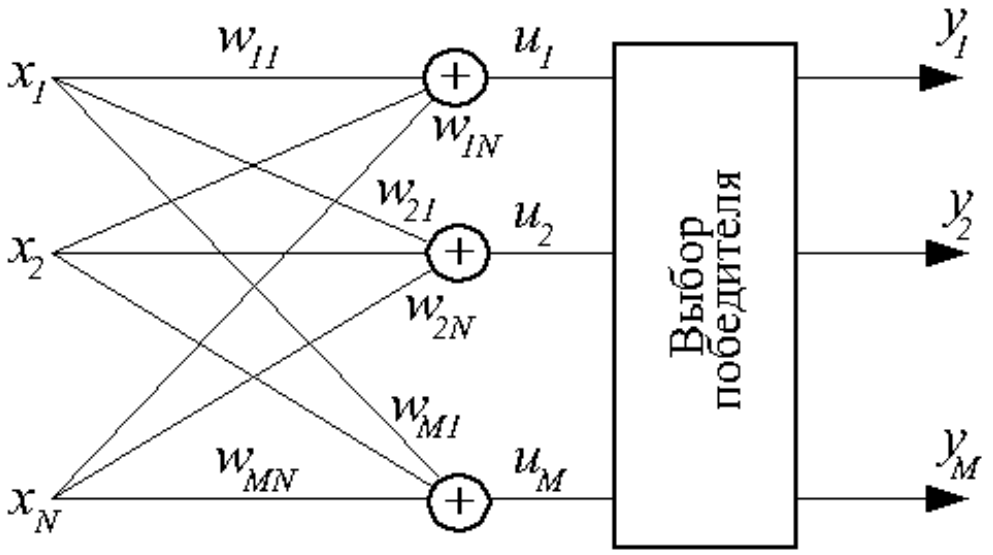

Рис. 2. Структурная модель слоя нейронов WTA. Обозначения: x_j, j = 1, 2, …, N - входные сигналы; w_{ij}, j = 0, 1, …, N - веса входных сигналов; u_i - взвешенная сумма входных сигналов; y_i, i = 1, 2, …, M - выходные сигналы

Каждый нейрон получает один и тот же вектор входных сигналов. Затем каждый нейрон рассчитывает выходной сигнал своего сумматора:

$$u_i = \sum_{j=0}^{N} w_{ij} \cdot x_j$$

Далее выбирается нейрон-победитель с наибольшим значением u_i, выходной сигнал y_i нейрона-победителя получает значение 1, а выходные сигналы остальных нейронов – 0.

Так как согласно принципу работы нейронов WTA нет смысла использовать один нейрон, то дополнительно к классу нейрона в программе реализован класс нейронного сети (состоящей из одного слоя нейронов).

Алгоритм обучения нейрона

Обучение нейрона производится без учителя. Начальные значения весовых коэффициентов всех нейронов выбираются случайным образом с нормализацией относительно 1.

При предъявлении каждого обучающего вектора X^k определяется нейрон-победитель, что дает ему право уточнить свои весовые коэффициенты по упрощенному правилу Гроссберга:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \cdot (x_j^k - w_{ij}(t))$$

где t – номер цикла обучения.

Проигравшие нейроны оставляют свои весовые коэффициенты неизменными.

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 нейрона, его 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 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. 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.

Класс нейронной сети содержит в себе массив нейронов, поле с файлом вывода данных, а также значение коэффициента η. Методы нейронной сети включают (приложение А, листинг А.3):

1. конструктор;
2. деструктор;
3. определение победителя;
4. обучение сети;
5. вывод весовых коэффициентов нейронов.

Процесс обучения и тестирования

Обучение производилось на точках, представленных на рис. 1. Координаты для точек были найдены посредством наложения сетки с шагом 0.1 (рис. 3).

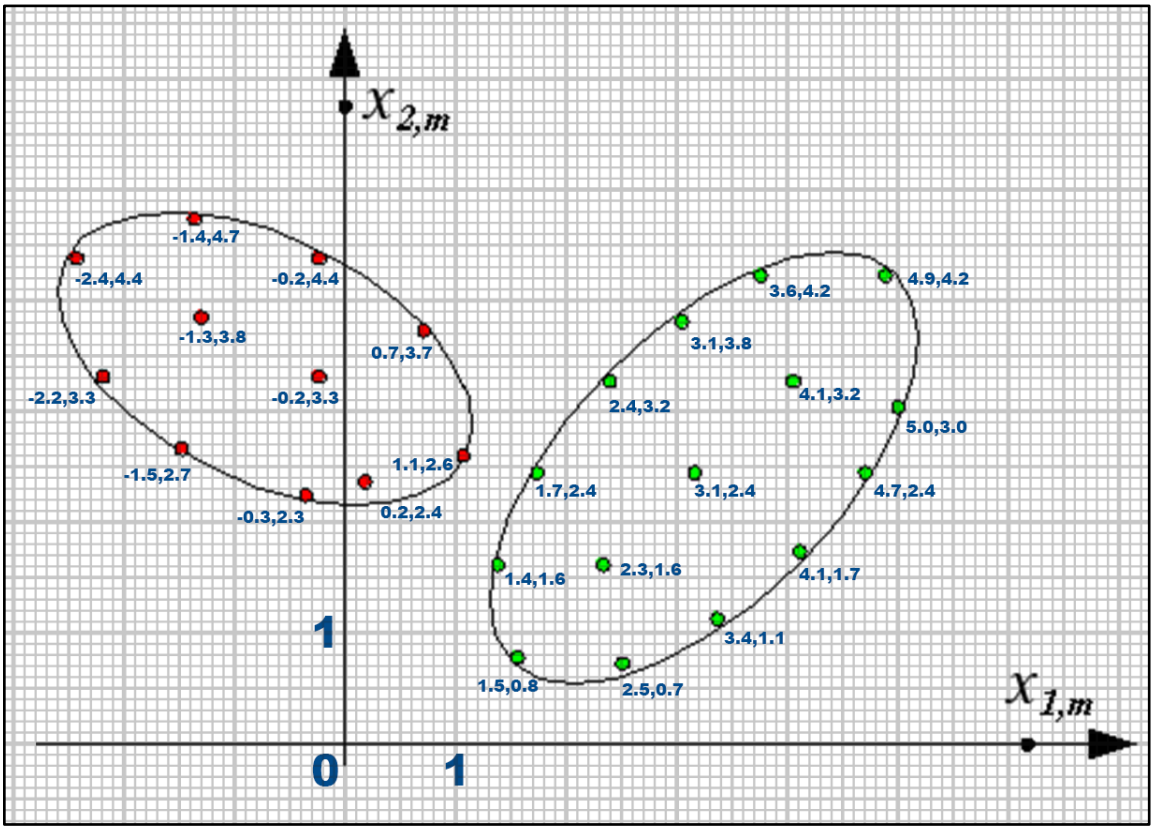

Рис. 3. Получение координат для обучающей выборки

Далее полученные данные были представлены в текстовой форме. В результате был получен следующий файл исходных данных *learn_data.dat* (листинг 2), где первая колонка – это x_1, а вторая – x_2.

Листинг 2. Файл обучающих данных

-1.4 4.7	-0.3 2.3	1.7 2.4	1.5 0.8
-2.4 4.4	0.2 2.4	3.1 2.4	2.5 0.7
-0.2 4.4	1.1 2.6	4.7 2.4	3.4 1.1
-1.3 3.8	-0.2 3.3	5.0 3.0	3.6 4.2

0.7 3.7 -2.2 3.3 -1.5 2.7	1.4 1.6 2.3 1.6 4.1 1.7	4.9 4.2 4.1 3.2	3.1 3.8 2.4 3.2

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;

После считывания данных они случайным образом копируются в новый вектор X_learn2, благодаря этому обеспечивается перемешивание входных данных, что в свою очередь упрощает процесс обучения.

Далее в функции обучения каждый входной вектор X_learn2 отдельно нормализуется и подается на вход нейронам.

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 хранятся нормализованные значения вектора X с принадлежностью данного вектора к определенному классу. 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, цвет кружка отражает принадлежность вектора к конкретному классу. Результаты работы программы представлены на изображениях 4-8:

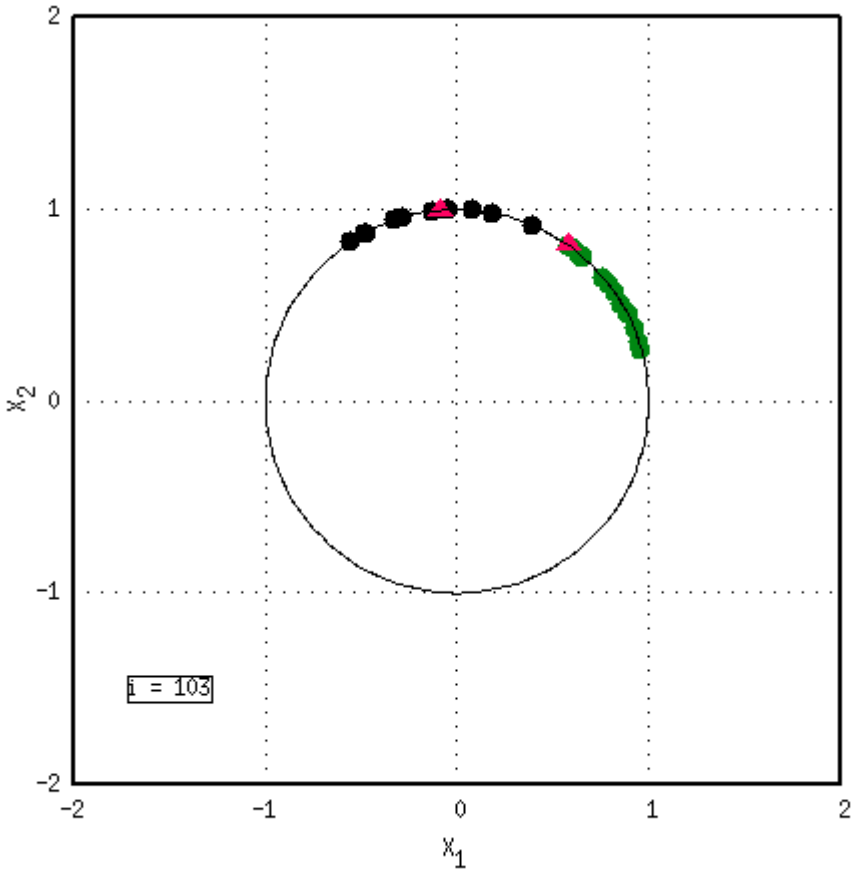

Рис. 4. Результаты работы двух нейронов

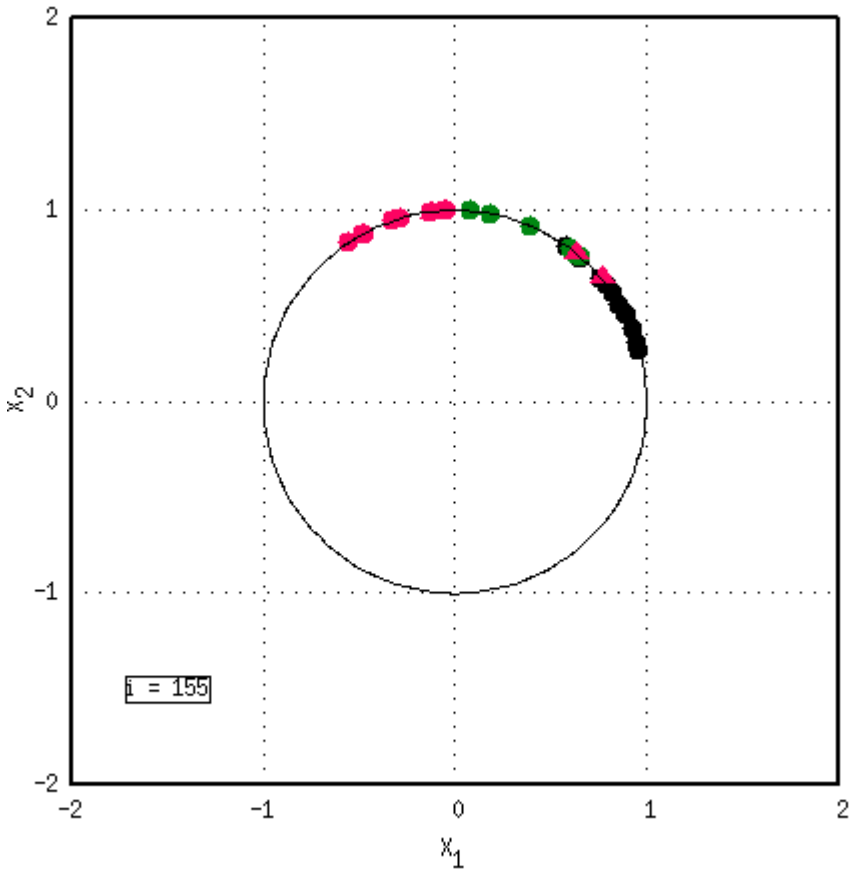

Рис. 5. Результаты работы трех нейронов

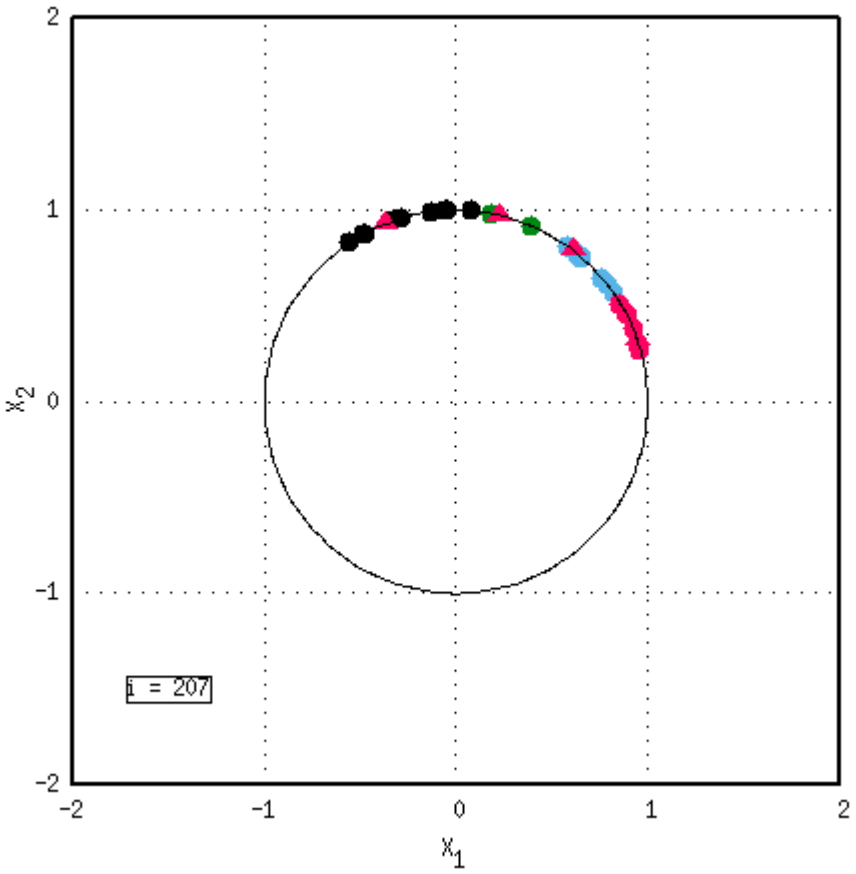

Рис. 6. Результаты работы четырех нейронов

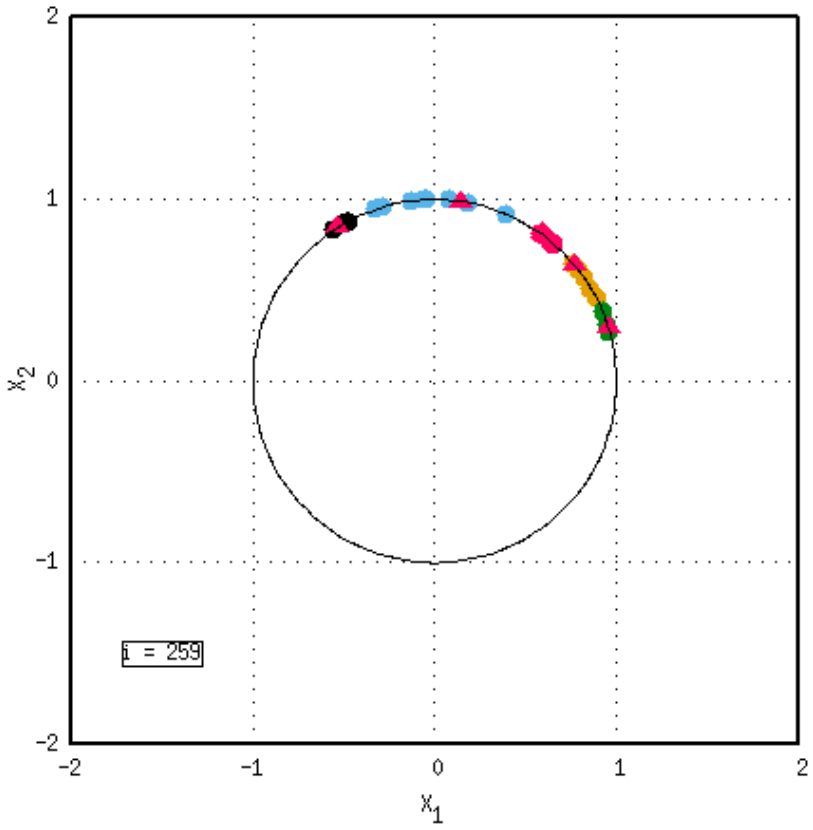

Рис. 7. Результаты работы пяти нейронов

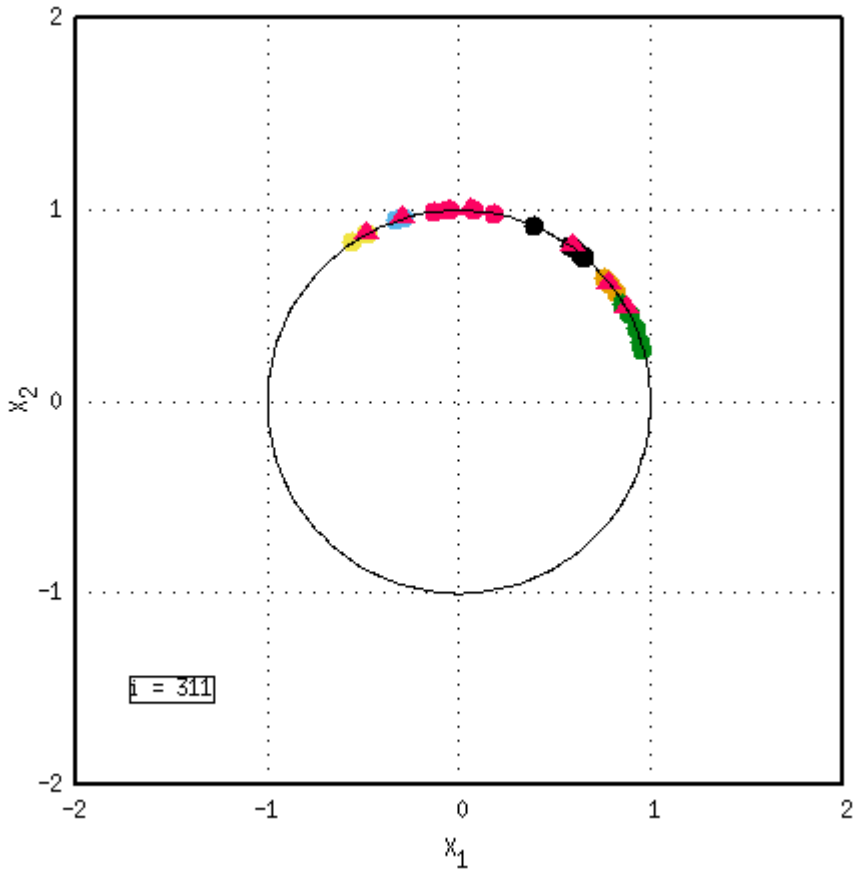

Рис. 8. Результаты работы шести нейронов

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 N1 "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 wins_count;
    int countdown;
 public:
    Neuron_WTA(int _input_num);
    void add_win();
    void reduce_countdown();
    bool win_check(int _max_wins, int _neuron_num);
    int get_wins_count();
    int get_countdown();
    double output_signal(std::vector<double>& _X);
    void change_weight(std::vector<double>& _X, double n);
    int get_input_num();
    void print_weights(std::ofstream& _file);
    void set_rand_W();
    std::vector<double> vector_normalize(std::vector<double>& _vector);
};
```

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_weight(std::vector<double>& X, double n) {
    for(int i = 0; i < input_num; i++) {
        W[i] = W[i] + n * (X[i] - W[i]);
    }
}

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

```
}

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


void Neuron_WTA::set_rand_W() {
    for(int i = 0; i < input_num; i++) {
        W[i] = rand() % 10;
        W[i] -= 5;
        W[i] /= 10;
    }
    W = vector_normalize(W);
}

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

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

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

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

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

// 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;
}
```

Листинг А.3. Файл *neuron_network.hpp*

```
#pragma once
#include <vector>
#include <iostream>
#include <fstream>
#include <ctime>
#include "neuron_wta.hpp"

class Network_WTA {
 private:
    std::vector<Neuron_WTA> Neurons;
    std::ofstream output_file;
    unsigned int neuron_num;
    double n;
 public:
    Network_WTA(unsigned int _neuron_num);
    ~Network_WTA();
    void 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);
};
```

Листинг А.4. Файл *neuron_network.cpp*

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

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

    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;
}

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

// Определение нейрона-победителя и переопределение его весов
void Network_WTA::define_winner(std::vector<double>& X, int iteration) {
    int max_wins = neuron_num;
    double max_output_signal = -10000.0;
    double temp = 0.0;
    int winner_index = 0;
    if (iteration < neuron_num) {
```

```
        for(int i = 0; i < Neurons.size(); i++) {
            if (Neurons[i].win_check(max_wins, neuron_num)) {
                temp = Neurons[i].output_signal(X);
                if(temp > max_output_signal) {
                    max_output_signal = temp;
                    winner_index = i;
                }
            } else {
                Neurons[i].reduce_countdown();
            }
        }
    } else {
        for(int i = 0; i < Neurons.size(); i++) {
            temp = Neurons[i].output_signal(X);
            if(temp > max_output_signal) {
                max output signal = temp;
                winner_index = i;
            }
        }
    }
    output_file << X[0] << " " << X[1] << " " << winner_index << " " <<
std::endl;

    Neurons[winner_index].add_win();
    Neurons[winner_index].change_weight(X, n);
    if (n > 0.1) {
        n -= 0.001;
    }
}

void Network_WTA::network_learn(std::vector<std::vector <double> >& X_learn,
std::ofstream& weights_file, int age) {
    std::vector<double> X;

    for(int j = 0; j < X_learn.size(); j++) {
        X = Neurons[j].vector_normalize(X_learn[j]);
        print_neurons_weights(weights_file);
        define_winner(X, age);
        weights_file << std::endl;
    }
}

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;
    }


    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;
    }

    Network_WTA N(num_of_neurons);

    int ages = num_of_neurons * 2;
    for (int i = 0; i < ages; i++) {
        // чтение из файла обучающей выборки
        ifstream learn_data_file;
        learn_data_file.open("learn_data.dat");
        if (!learn_data_file) {
            cout << "ERROR: can't open file 'learn_data.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();

        vector< vector<double> > X_learn2;
        while (X_learn.size() != 0) {
            int random_vector_num = rand() % X_learn.size();
            X_learn2.push_back(X_learn[random_vector_num]);
            X_learn.erase(X_learn.begin() + random_vector_num);
        }
        N.network_learn(X_learn2, 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[-2:2]
set xrange[-2:2]
set xlabel 'X_1'
set ylabel 'X_2'

set grid
set macro

set style increment user

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

```
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

    set object 1 circle front at 0.0,0.0 size 1.0 fillcolor rgb "black" lw 1

    plot 'test.dat' every 1:1:0:0:i:0 u 1:2:3 w p lc var, 'weights.dat' every
:::i::i w p ls 2
    pause 0.001
}
pause mouse
```

Листинг А.7. Файл *compile.sh*

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