|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 ЛАБОРАТОРНЫХ РАБОТ**

**ПО КУРСУ “ВВЕДЕНИЕ В ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ”**

Студент Мигранов Рустам Рамилевич

Группа РК6-11М

Тип задания Лабораторные работы

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мигранов Р.Р.**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Федорук В.Г.**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2019 г.*

**Оглавление**

[1. Лабораторная работа № 2 3](#_Toc28798878)

[1.1. Индивидуальное задание (вариант 58) 3](#_Toc28798879)

[1.2. Описание алгоритма 3](#_Toc28798880)

[1.3. Исходный код программы 3](#_Toc28798881)

[1.4. Пример работы программы 7](#_Toc28798882)

[2. Лабораторная работа № 3 8](#_Toc28798883)

[2.1. Краткое описание модели персептрона 8](#_Toc28798884)

[2.2. Обучение персептрона 9](#_Toc28798885)

[2.3. Индивидуальное задание (вариант 1-5) 9](#_Toc28798886)

[2.4. Результаты обучения нейрона 11](#_Toc28798887)

[2.5. Исходный код программы 11](#_Toc28798888)

[3. Лабораторная работа № 4 14](#_Toc28798889)

[3.1. Индивидуальное задание (вариант 4) 14](#_Toc28798890)

[3.2. Краткое описание ИНС с самоорганизацией на основе конкуренции 14](#_Toc28798891)

[3.3. Реализация 15](#_Toc28798892)

[3.4. Обучение ИНС 15](#_Toc28798893)

[3.5. Результаты обучения ИНС 16](#_Toc28798894)

# Лабораторная работа № 2

## Индивидуальное задание (вариант 58)

Ферзь находится на поле C3 шахматной доски. Необходимо найти последовательность из 15 ходов, завершающуюся в поле F6 и обеспечивающую прохождение всех полей шахматной доски. При этом ни одно поле нельзя проходить более одного раза.

## Описание алгоритма

Алгоритм основан на поиске в глубину. В самом начале ферзь делает шаг влево на максимальное количество клеток. Пройденные клетки запоминаются в массиве, чтобы в следующие ходы не проходить их ещё раз. Далее, алгоритм пытается сделать шаги в других направлениях, пока не достигнет конечной клетки. При этом на каждом шаге хранится количество сделанных ходов, чтобы в момент достижения конечной ячейки число шагов было равным 15.

## Исходный код программы

Листинг 1. Код разработанной программы на языке Prolog

|  |
| --- |
| *% --- Steps*  *% --- step left*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+7, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+6, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+5, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+4, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+3, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+2, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+1, NewY is Y, concat\_([NewX], [NewY], NewPosition).  *% --- step right*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-7, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-6, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-5, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-4, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-3, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-2, NewY is Y, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-1, NewY is Y, concat\_([NewX], [NewY], NewPosition).  *% --- step top*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y-7, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y-6, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y-5, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y-4, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y-3, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y-2, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y-1, concat\_([NewX], [NewY], NewPosition).  *% --- step bottom*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y+7, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y+6, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y+5, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y+4, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y+3, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y+2, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X, NewY is Y+1, concat\_([NewX], [NewY], NewPosition).  *% --- step top-left*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+7, NewY is Y-7, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+6, NewY is Y-6, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+5, NewY is Y-5, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+4, NewY is Y-4, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+3, NewY is Y-3, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+2, NewY is Y-2, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+1, NewY is Y-1, concat\_([NewX], [NewY], NewPosition).  *% --- step bottom-right*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-7, NewY is Y+7, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-6, NewY is Y+6, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-5, NewY is Y+5, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-4, NewY is Y+4, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-3, NewY is Y+3, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-2, NewY is Y+2, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-1, NewY is Y+1, concat\_([NewX], [NewY], NewPosition).  *% --- step top-right*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-7, NewY is Y-7, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-6, NewY is Y-6, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-5, NewY is Y-5, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-4, NewY is Y-4, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-3, NewY is Y-3, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-2, NewY is Y-2, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X-1, NewY is Y-1, concat\_([NewX], [NewY], NewPosition).  *% --- step bottom-left*  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+7, NewY is Y+7, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+6, NewY is Y+6, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+5, NewY is Y+5, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+4, NewY is Y+4, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+3, NewY is Y+3, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+2, NewY is Y+2, concat\_([NewX], [NewY], NewPosition).  step([X|YL], NewPosition):- devide\_H(YL, Y), NewX is X+1, NewY is Y+1, concat\_([NewX], [NewY], NewPosition).    check([X|YL]):- devide\_H(YL, Y), X >= 1, X =< 8, Y >= 1, Y =< 8.  next\_cell(From, To):- step(From, To), check(To).  path(From, To, Way, MoveCount, [H | T], OS):-  next\_cell(From, NewPos),  create\_path(From, NewPos, P),  copy(Way, WayCopy),  [HP | TP] = P,  check\_pos(TP, WayCopy),  NewPos = To,  get\_list\_size(Way, WaySize),  get\_list\_size(TP, PSize),  Sum is (WaySize + PSize),  Sum = 57,  MoveCount = 14,  H = From,  T = [NewPos | []].    path(From, To, Way, MoveCount, [H | T], OS):-  next\_cell(From, NewPos),  create\_path(From, NewPos, P),  copy(Way, WayCopy),  [HP | TP] = P,  check\_pos(TP, WayCopy),  NewPos \= To,  get\_list\_size(Way, WaySize),  get\_list\_size(TP, PSize),  Sum is (WaySize + PSize),  Sum < 57,  MoveCount < 14,  H = From,  concat\_(TP, Way, NewWay),  MC is MoveCount + 1,  path(NewPos, To, NewWay, MC, T, OS).    path\_call(From, To, Path):- path(From, To, [From], 0, Path, OS),  open('result.txt', append, OS),  write(OS, Path),  nl(OS),  false  ;  close(OS)  ).    path\_call\_all(From, To, Path):- findall(P, path(From, To, [From], 0, P, OS), Path),  open('result\_all.txt', append, OS),  (  write(OS, Path),  false  ;  close(OS)  ).    */\* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*  *% Help-functions*  *- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - \*/*  check\_pos2([], [], []):- !.  check\_pos2([], [H], Path):- !.  check\_pos2([], [\_ | T], Path):- [H1 | T1] = Path, [H2 | T2] = T, check\_pos2([H1 | T1], [H2 | T2], Path).  check\_pos2([H1 | T1], [H2 | T2], Path):- H1 \= H2, check\_pos2(T1, [H2 | T2], Path).  check\_pos2([H1 | T1], [H2 | T2], Path):- H1 = H2, !, fail.    *% Проверяет, нет ли элементов из 1 массива во втором массиве. Возвращает true, если их нет.*  *% Первый аргумент - потенциальный путь, второй - весь прошедший путь*  check\_pos([H1 | T1], [H2 | T2]):- check\_pos2([H1 | T1], [H2 | T2], [H1 | T1]).  check\_pos([H1 | T1], []):- true.    create\_path(From, NewPos, P):- devide\_C(From, X1, Y1),  devide\_C(NewPos, X2, Y2),  create\_path\_x(X1, X2, MasX),  create\_path\_x(Y1, Y2, MasY),  combine(MasX, MasY, P).    create\_path\_x(X1, X2, [H | []]):- X1 = X2, H = X2, !.  create\_path\_x(X1, X2, [X1 | T]):- X1 < X2, X is X1 + 1, create\_path\_x(X, X2, T), !.  create\_path\_x(X1, X2, [X1 | T]):- X1 > X2, X is X1 - 1, create\_path\_x(X, X2, T).    combine([], [], []):- !.  combine([H1 | []], [], []):- !.  combine([], [H2 | []], []):- !.  combine([H1|[]], [H2 | T2], [[H1, H2] | T3]):- combine([H1 | []], T2, T3), !.  combine([H1 | T1], [H2 | []], [[H1, H2] | T3]):- combine(T1, [H2 |[]], T3), !.  combine([H1 | T1], [H2 | T2], [[H1, H2] | T3]):- combine(T1, T2, T3).    devide\_H([Head|\_], Head):- !.  devide\_T([\_|Tail], Tail):- !.    devide\_C(C, X, Y):- devide\_H(C, X),  devide\_T(C, YM),  devide\_H(YM, Y).    concat\_([], List, List):- !.  concat\_([H|T], L2, [H|T3]):- concat\_(T, L2, T3).    compare\_coords(C1, C2):- C1 = C2.    push\_back([],X,[X]):- !.  push\_back([H|T],X,[H|Res]):- push\_back(T,X,Res).    get\_list\_size([], 0):- !.  get\_list\_size([\_|T], Length):- get\_list\_size(T, NewLength), Length is NewLength + 1.    copy(L,R) :- accCp(L,R).  accCp([],[]).  accCp([H|T1],[H|T2]) :- accCp(T1,T2). |

## 

## Пример работы программы

Решение задачи по заданному условию, в котором необходимо добраться до конечной точки за 15 ходов и пройти всю доску, занимает неопределенно большое количество времени. В целях ускорения процесса проверки программы количество ходов было уменьшено до 13, а количество пройденных клеток было принято взять равным 57.

Предикат, возвращающий решение задачи, имеет название path\_call. В качестве аргументов он принимает координаты начальной клетки, координаты конечной клетки и массив, в который запишутся координаты пройденных вершин.

Результат работы программы приведён в листинге 2.

Графическая интерпретация решения представлена на рисунке 1.

Листинг 2. Результат работы программы

|  |
| --- |
| path\_call([3,3], [6,6], Path).  Path = [[3,3],[8,3],[8,1],[1,1],[1,8],[8,8],[8,4],[2,4],[2,7],[7,7],[7,5],[3,5],[3,6],[4,6],[5,6],[6,6]] |

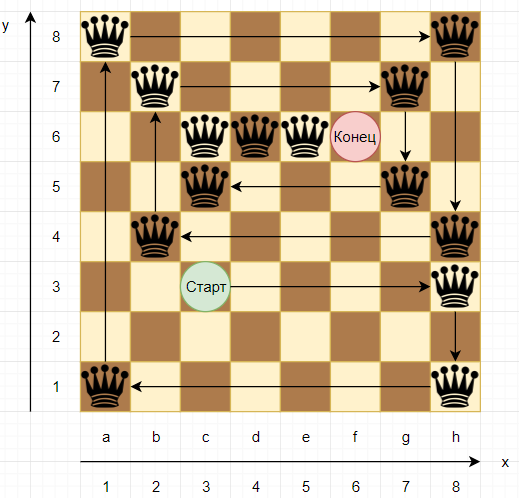


Рис. 1 Графическое решение задачи

# Лабораторная работа № 3

## Краткое описание модели персептрона

Данная модель искусственного нейрона предложена в 1943 г. и называется также моделью МакКаллока-Питтса. В модели нейрон считается бинарным элементом, его структурная схема представлена ниже.

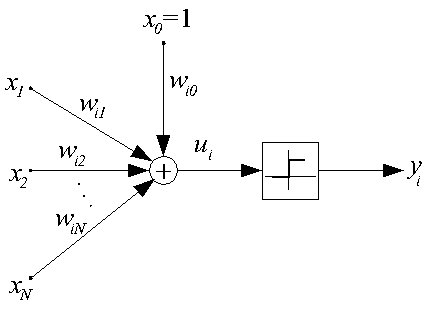


Рис 2. Модель персептрона.

Выходной сигнал нейрона может принимать только два значения {0, 1} по следующему правилу:

(1)

Обучение персептрона требует учителя, т.е. множества {*< ,>, ..., < ,>* } пар, где <вектор входных сигналов , ожидаемое значение выходного сигнала >.

Обучение (отыскание весовых коэффициентов *wij*) сводится к задаче минимизации целевой функции:

(2)

## Обучение персептрона

Для обучения персептрона используется следующий алгоритм:

1. Выбираются (случайно) начальные значения весов (*j* = 0, 1, 2, ..., N) нейрона.
2. Для каждой обучающей пары *< ,>*выполняется ряд циклов (их номера обозначим через *t*) уточнения значений входных весов по формуле:

, (3)

где - правила обучения Видроу-Хоффа:

(4)

## Индивидуальное задание (вариант 1-5)

Обучение персептрона для распределения обучающих данных представленных ниже.

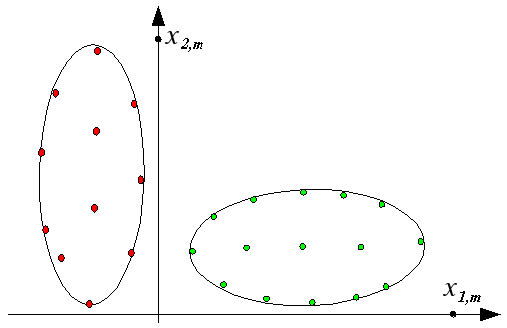


Рис. 3 Распределение обучающих данных.

Для решения данной задачи используется модель двухвходового персептрона с входом поляризации (рисунок 4).

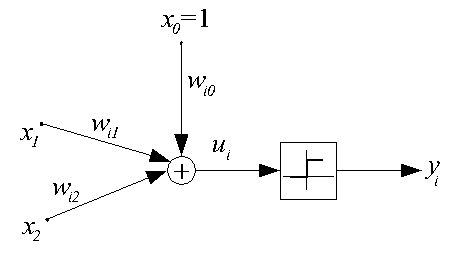


Рис 4. Двухвходовой персептрон с входом поляризации

Для такого нейрона уравнение плоскости имеет следующий вид:

(5)

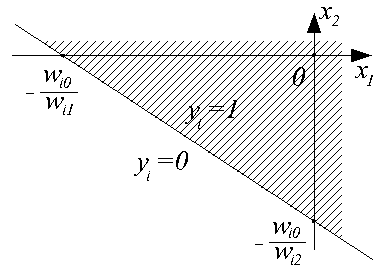
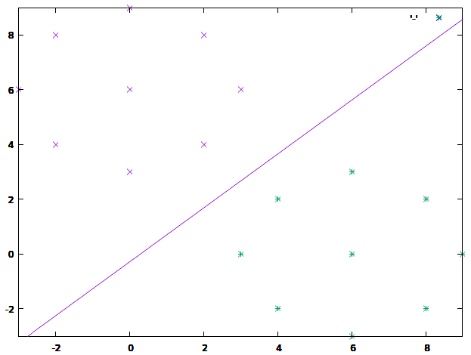
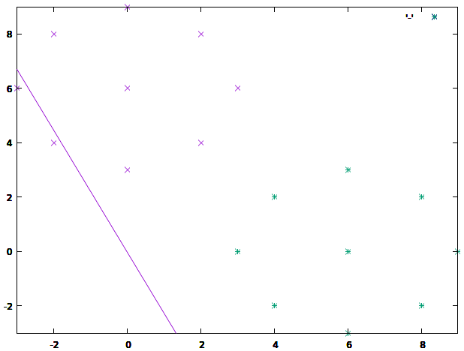
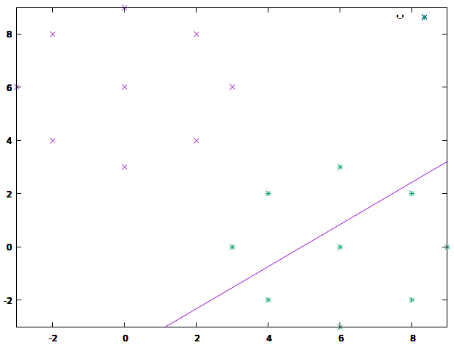


Рис 5. Плоскость, полученная с помощью уравнения (5).

Уравнение плоскости (5) пересекается с плоскостью   по линии, представленной на рисунке 5. Эта линии с помощью функции активации позволяет разбивать набор обучающих данных на два класса. Таким образом персептрон реализует задачу линейной классификации обучающих данных.

## Результаты обучения нейрона

Ниже представлены результаты обучения персептрона на заранее сгенерированном распределении обучающей выборки, состоящем из 18 точек. При этом для корректного обучения нейрона необходимо, чтобы данные из одного множества были равномерно распределены и не подавались на вход нейрону одной группой. Количество входных нейронов 3 (с учетом входа поляризации), начальные вектор весов выбирается случайным образом в диапазоне [0;1]. Количество эпох обучения 1.



а) 1 элемент б) 2 элемент в) 3 элемент

выборки выборки выборки

Рис.6 Обучение нейрона

Нейрон обучился уже после третьей обучающей пары, что видно по рисунку 6.

## Исходный код программы

Листинг 3. Функция main

|  |
| --- |
| #include <iostream>  #include <string>  #include <fstream>    #include "Perceptron.h"    std::string train\_data\_filename = "train\_data.tsv";    int main() {  int try\_limit;  int epoch\_count;    std::cout << "Введите число эпох" << std::endl;  std::cin >> epoch\_count;    std::cout << "Введите предельное количество циклов обучения" << std::endl;  std::cin >> try\_limit;    if (epoch\_count < 1 || try\_limit < 1) {  return -1;  }    std::ifstream fin(train\_data\_filename);  double x1, x2, d;  train\_pairs tps;  while (fin >> x1 >> x2 >> d) {  dvector xk = dvector(2);  xk[0] = x1;  xk[1] = x2;  train\_pair tp = train\_pair(xk, d);  tps.push\_back(tp);  }  fin.close();    FILE\* gnuplotPipe;  gnuplotPipe = popen("gnuplot -persistent", "w");  fprintf(gnuplotPipe, "set multiplot**\n**");  fprintf(gnuplotPipe, "set xrange [-3:9]**\n**");  fprintf(gnuplotPipe, "set yrange [-3:9]**\n**");    Perceptron perceptron(2);    perceptron.train(tps, epoch\_count, try\_limit, gnuplotPipe);    print\_dvector(perceptron.W);  } |

Листинг 4. Класс Perceptron

|  |
| --- |
| #include <vector>  #include <iostream>  #include <cstdlib>  #include <ctime> // содержит time()    typedef std::vector<double> dvector;  typedef std::pair<dvector, double> train\_pair;  typedef std::vector<train\_pair> train\_pairs;    float pause = 1.0;    class Perceptron {    public:  int N;  dvector W;    Perceptron(int \_N);    int activ\_func(double u);  double random\_for\_w();    void train(const train\_pairs &tp, int epoch, int try\_limit, FILE \*gnuplotPipe);  };    void print\_dvector(dvector &dv) {  for (int i = 0; i < dv.size(); i++) {  std::cout << dv[i] << " ";  }  std::cout << std::endl;  }    void plot\_dots(const train\_pairs &tps, FILE \*gnuplotPipe) {  for (int i = 0; i < tps.size(); i++) {  int color = 0;  int pt = 1;  if (tps[i].second == 1) {  color = 2;  pt = 3;  } else {  color = 1;  pt = 2;  }  fprintf(gnuplotPipe, "plot '-' w p lc %d pt %d lw 1**\n**", color, pt);  fprintf(gnuplotPipe, "%lf %lf**\n**", tps[i].first[0], tps[i].first[1]);  fprintf(gnuplotPipe, "e**\n**");  }  }    Perceptron::Perceptron(int \_N): N(\_N) {  W = dvector(N + 1);  srand(time(NULL));  for (int i = 0; i < W.size(); i++) {  // W[i] = 0;  W[i] = random\_for\_w();  }  W[0] = -0.94;  W[1] = 0.19;  W[2] = -0.24;  }    int Perceptron::activ\_func(double u) {  return (u >= 0);  }    double Perceptron::random\_for\_w() {  return (double)(rand() % 201 - 100) / 100;  }    void Perceptron::train(const train\_pairs &tps, int epoch\_count, int try\_limit, FILE \*gnuplotPipe) {  double k , b;    for (int epoch = 0; epoch < epoch\_count; epoch++) {  for (int tp\_number = 0; tp\_number < tps.size(); tp\_number++) {  const train\_pair &tp = tps[tp\_number];  std::cout << tp.first[0] << " " << tp.first[1] << std::endl;  std::cout << W[0] << " " << W[1] << " " << W[2] << std::endl;  int try\_count = 0;  k = - (W[1] / W[2]), b = - (W[0] / W[2]);    fprintf(gnuplotPipe, "unset multiplot**\n**set multiplot**\n**");  plot\_dots(tps, gnuplotPipe);  fprintf(gnuplotPipe, "plot %lf\*x+%lf notitle**\n**", k, b);  fprintf(gnuplotPipe, "pause %f**\n**", pause);  fflush(gnuplotPipe);    while (try\_count < try\_limit) {  double u = 0;  u += W[0];  for (int i = 1; i < tp.first.size() + 1; i++) {  u += W[i] \* tp.first[i - 1];  }  double y = activ\_func(u);  std::cout << "y = " << y << "; d = " << tp.second << "; u = " << u << std::endl;  if (y == tp.second) {  print\_dvector(W);  break;  }  W[0] += tp.second - y;  for (int i = 1; i < tp.first.size() + 1; i++) {  W[i] += tp.first[i - 1]\*(tp.second - y);  }  k = - (W[1] / W[2]), b = - (W[0] / W[2]);  try\_count++;  }  fprintf(gnuplotPipe, "unset multiplot**\n**set multiplot**\n**");  plot\_dots(tps, gnuplotPipe);  fprintf(gnuplotPipe, "plot %lf\*x+%lf notitle**\n**", k, b);  fprintf(gnuplotPipe, "pause %f**\n**", pause);  fflush(gnuplotPipe);  }  }  } |

# Лабораторная работа № 4

## Индивидуальное задание (вариант 4)

Разработать, используя язык C/C++, программу, моделирующую поведение искусственной нейронной сети с самоорганизацией на основе конкуренции и обеспечивающую ее обучение для решения задач классификации данных по алгоритму нейронного газа.

## Краткое описание ИНС с самоорганизацией на основе конкуренции

Данные сети не требуют для своего обучения “учителя” и поэтому самостоятельно адаптируют веса под обучающие данные. Такие сети строятся из нейронов типа WTA и подобных им. Как правило, это однослойные сети, в которых каждых нейрон получает все компоненты входного вектора *X* размерностью *N*. На рисунке 7 представлена структурная схема такой сети.

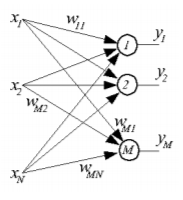


Рис. 7 Схема сети с самоорганизацией на основе конкуренции

Процесс обучения выглядит следующим образом. На вход сети подается обучающий вектор . Для каждого нейрона определяется расстояние между векторами и - . Определяется нейрон-победитель, для которого это расстояние оказывается наименьшим. Веса нейрона – победителя и веса его соседей уточняются по алгоритму нейронного газа:

, (6)

где – коэффициент обучения, значение которого уменьшается с увеличением расстояния от *i*-ого нейрона до победителя.

На каждой итерации цикла обучения все нейроны сортируются в последовательности возрастания расстояния

, (7)

где – номер *i*-ого нейрона в последовательности. Для нейрона-победителя .

Значение функции соседства *i*-ого нейрона определяется следующим выражением:

, (8)

где определяет уровень соседства и является величиной, уменьшающейся по ходу обучения. При стремящемся к 0, алгоритм превращается в алгоритм WTA.

## Реализация

При запуске программы в качестве первого аргумента командной строки передается имя файла, содержащего координаты обучающих данных (в формате *x*, *y*). Вторым аргументом является количество нейронов в сети. Третий аргумент – это количество эпох обучения. Нормализация обучающих данных в программе не производится. Данные в файле уже представлены в нормализованном виде.

## Обучение ИНС

Для генерации обучающих точек была написана специальная программа, способная генерировать координаты точек, раскиданных по Архимедовой спирали любым заданным количеством точек.

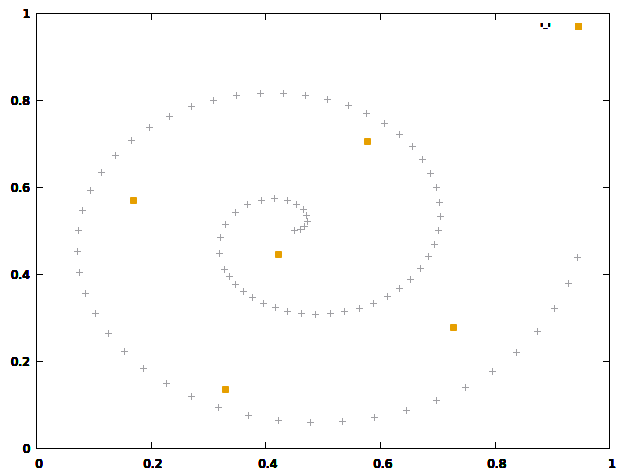
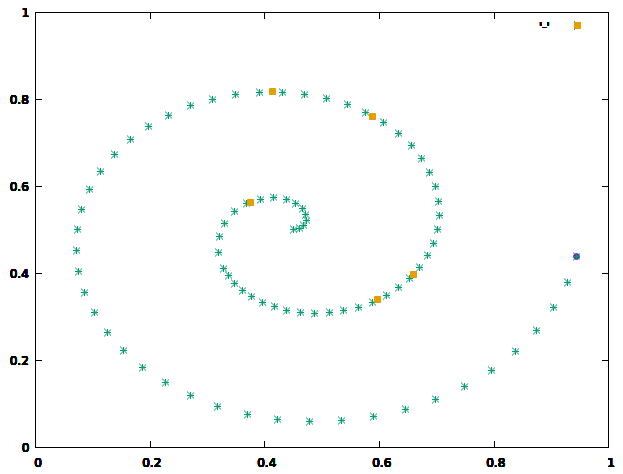
Процесс обучения состоит из двух этапов. Задача первого этапа —оживить все нейроны, хаотично «раскиданные» по пространству. Задача второго этапа — непосредственно обучить нейронную сеть. И на первом, и на втором этапе с ростом *k* параметры и убывают по линейному закону в заданном диапазоне.

На первом этапе параметр ширины функции Гаусса (см. формулу 8) и коэффициент обучения выбираются большими. Экспериментально подобранные диапазоны: = 1.0 … 0.5 и = 0.6 … 0.2.

На втором этапе параметр ширины функции Гаусса убывает со временем в диапазоне 0.1 … 0.01, коэффициент обучения – в диапазоне 0.1 … 0.01. Экспериментальное тестирование показало, что второй цикл обучения несколько уточняет позиции нейронов, дальнейшее же обучение бесполезно.

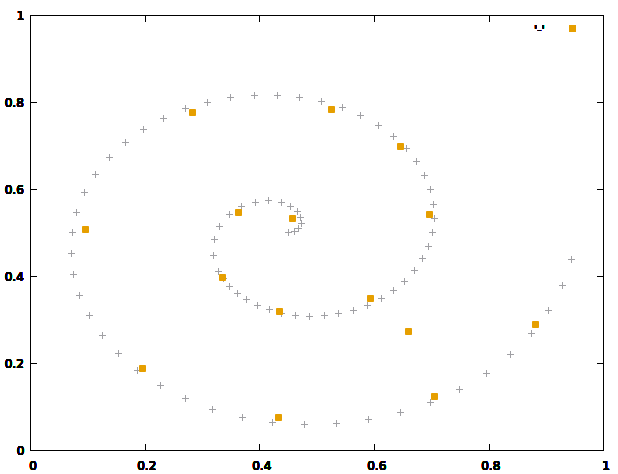
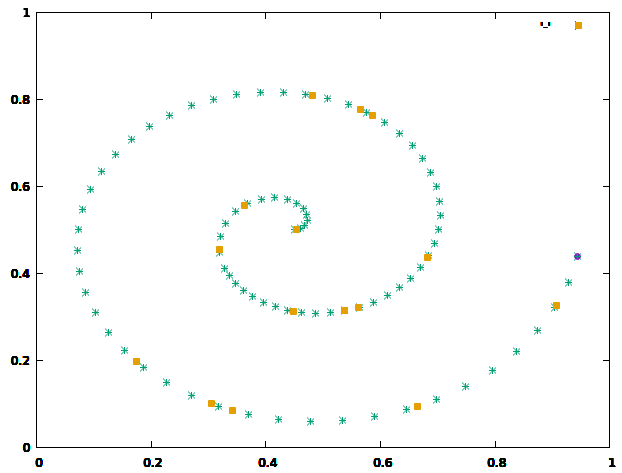
## Результаты обучения ИНС

На рис. 8-10 показано распределение обучающих данных в пространстве зеленым (до обработки этой точки) и серым (после обработки) цветом и положение весов нейронов оранжевым цветом. Количество обучающих данных равно 100. Некоторые нейроны остаются мертвыми. Особенно это проявляется при малом количестве нейронов. Это следствие правила соседства по алгоритму нейронного газа.



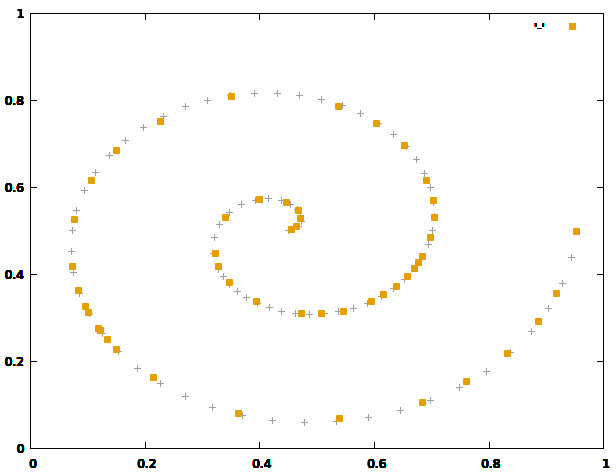
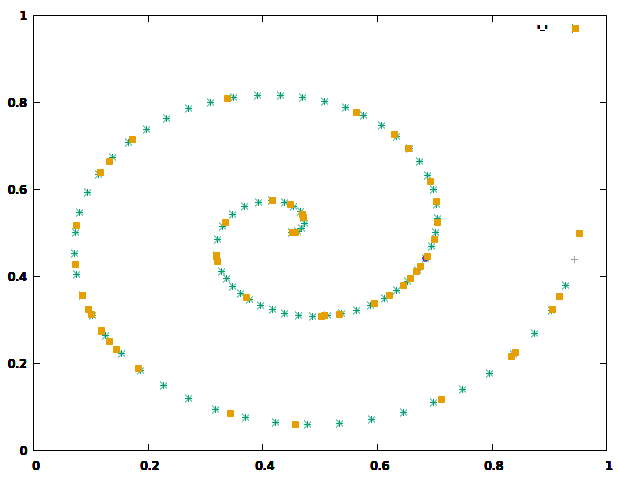
а) до обучения б) после обучения

Рис. 8 Обучающие данные и веса при 5 нейронах



а) до обучения б) после обучения

Рис. 9 Обучающие данные и веса при 15 нейронах



а) до обучения б) после обучения

Рис. 10 Обучающие данные и веса при 50 нейронах

Результат обучения нейронной сети сильно зависит от количества нейронов в сети. Например, по полученным результатам для 100 обучающих выборок равномерная классификация получается при 15 нейронах.

## Исходный код программы

Листинг 5. Файл main.cc

|  |
| --- |
| #include <stdio.h>  #include <iostream>  #include <fstream>  #include <cmath>  #include "NeuronNet.hh"  #define INPUT\_VECTOR\_SIZE 2  // 1 аргумент - файл с данными  // 2 аргумент - количество нейронов  // 3 аргумент - количество циклов обучения  int main(int argc, char \*\*argv) {  if (argc != 4) {  std::cerr << "Неправильный ввод.**\n**Введите данные следующим образом:**\n**"  "1 аргумент - файл с данными**\n**"  "2 аргумент - количество нейронов,**\n**"  "3 аргумент - количество циклов обучения.**\n**" << std::endl;  exit(EXIT\_FAILURE);  }  char \*filename = argv[1];  int neurons\_number = atoi(argv[2]);  int epoch\_num = atoi(argv[3]);    srand(time(NULL));  NeuronNet neuronNet(neurons\_number, INPUT\_VECTOR\_SIZE);    vector\_fvector train\_vectors;    std::ifstream fin(filename);  float x1, x2;  while (fin >> x1 >> x2) {  fvector train\_vector(INPUT\_VECTOR\_SIZE);  train\_vector[0] = x1;  train\_vector[1] = x2;  train\_vectors.push\_back(train\_vector);  }    FILE\* gnuplotPipe;  gnuplotPipe = popen("gnuplot -persistent", "w");  fprintf(gnuplotPipe, "set multiplot**\n**");  fprintf(gnuplotPipe, "set xrange [0:1]**\n**");  fprintf(gnuplotPipe, "set yrange [0:1]**\n**");    neuronNet.train(train\_vectors, epoch\_num, gnuplotPipe);    fclose(gnuplotPipe);  } |

Листинг 6. Файл Neuron.hh

|  |
| --- |
| #include <vector>  #include <ctime>  #include <cstdlib>    #define W\_RAND\_FROM 0  #define W\_RAND\_TO 1  #define W\_RAND\_STEP 0.001    #define PI\_MIN 0.75  #define PI\_MAX 1.0    #define PI 3.141592654    typedef std::vector<float> fvector;    class Neuron {  private:  float G(int m, float SigmaK);    public:  // int number; // номер нейрона в массиве  float eta; // коэффициент обучения  int WiSize; // размер вектора весов  fvector Wi; // вектор весов  float pi; // потенциал для штрафования    // Neuron(int \_number, float \_eta): number(\_number), eta(\_eta) {};  Neuron(int \_WiSize, float \_eta);  Neuron(){};    void changeW(int m, float SigmaK, const fvector & Xk);  void changeEta(float etaMin, float etaMax, int k, int Kmax);    void changePiWin();  void changePiLose(int M);    bool check\_participate();  };    float random\_value(float from, float to, float step) {  int numbers\_num = (to - from) / step;  return from + ((float) (rand() % numbers\_num) \* step);  }    void spiral\_evklida(float \*x, float \*y) {  float t = float (rand() / (RAND\_MAX + 1.0)) \* 4 \* PI;  \*y = 0.04 \* t \* sin(t) + 0.5;  \*x = 0.04 \* t \* cos(t) + 0.5 - 0.05;  }    Neuron::Neuron(int \_WiSize, float \_eta): WiSize(\_WiSize), eta(\_eta), pi(PI\_MIN) {  Wi = fvector(WiSize);  spiral\_evklida(&Wi[0], &Wi[1]);  // float Wfrom = W\_RAND\_FROM, Wto = W\_RAND\_TO, Wstep = W\_RAND\_STEP;  // for (int i = 0; i < WiSize; i++) {  // Wi[i] = random\_value(Wfrom, Wto, Wstep);  // }  }    void Neuron::changeW(int m, float SigmaK, const fvector & Xk) {  for (int j = 0; j < Wi.size(); j++) {  // printf("%f\n", eta \* G(m, SigmaK) \* (Xk[j] - Wi[j]));  Wi[j] = Wi[j] + eta \* G(m, SigmaK) \* (Xk[j] - Wi[j]);  }  }    void Neuron::changeEta(float etaMin, float etaMax, int k, int Kmax) {  float a = (etaMin / etaMax);  float b = (float)(k + 1) / (float) (Kmax);  eta = etaMax \* pow(a, b);  }    float Neuron::G(int m, float SigmaK) {  return exp((-1) \* (m / SigmaK));  }    void Neuron::changePiWin() {  pi -= PI\_MIN;  if (pi < 0.0) pi = 0.0;  }    void Neuron::changePiLose(int M) {  pi += 1.0 / M;  if (pi > PI\_MAX) pi = PI\_MAX;  }    bool Neuron::check\_participate() {  return pi >= PI\_MIN;  } |

Листинг 7. Файл NeuronNet.hh

|  |
| --- |
| #include "Neuron.hh"  #include <utility>  #include <algorithm>    #define SIGMA\_INIT 0.1  #define SIGMA\_MAX 1.0  #define SIGMA\_MIN 0.001    #define ETA\_INIT 0.01  #define ETA\_MAX 1.0  #define ETA\_MIN 0.001    float pause = 2;    typedef std::vector<Neuron> neuron\_vector;  typedef std::vector<fvector> vector\_fvector;    class NeuronNet {  private:  float calculate\_d(const fvector &, const fvector &);    public:  int neurons\_number; // количество нейронов  neuron\_vector neurons; // вектор нейронов  float sigmak;    NeuronNet(int \_neurons\_number, int input\_vector\_size);    void train(vector\_fvector train\_vectors, int epochNum, FILE \*gnuplotPipe);    vector\_fvector getW();  };    void print\_neuronNet(const NeuronNet & neuronNet) {  const neuron\_vector & nv = neuronNet.neurons;  for (int i = 0; i < 1; i++) {  std::cout << "Нейрон " << i << std::endl;  for (int j = 0; j < nv[0].WiSize; j++) {  printf("W%d %d=%f; pi=%f**\n**", i, j, nv[i].Wi[j],nv[i].pi);  }  printf("Sigma=%f; Eta =%f**\n**", neuronNet.sigmak, neuronNet.neurons[0].eta);  std::cout << std::endl;  }  }    void plot\_dots(FILE \*gnuplotPipe, const vector\_fvector & train\_vectors, const vector\_fvector & W, int k) {  fprintf(gnuplotPipe, "unset multiplot**\n**set multiplot**\n**");  int color, pt;  for (int i = 0; i < k; i++) {  color = 0;  pt = 1;  fprintf(gnuplotPipe, "plot '-' w p lc %d pt %d lw 1**\n**", color, pt);  fprintf(gnuplotPipe, "%lf %lf**\n**", train\_vectors[i][0], train\_vectors[i][1]);  fprintf(gnuplotPipe, "e**\n**");  }    if (k < train\_vectors.size()) {  color = 1;  pt = 7;  fprintf(gnuplotPipe, "plot '-' w p lc %d pt %d lw 1**\n**", color, pt);  fprintf(gnuplotPipe, "%lf %lf**\n**", train\_vectors[k][0], train\_vectors[k][1]);  fprintf(gnuplotPipe, "e**\n**");  }    for (int i = k; i < train\_vectors.size(); i++) {  color = 2;  pt = 3;  fprintf(gnuplotPipe, "plot '-' w p lc %d pt %d lw 1**\n**", color, pt);  fprintf(gnuplotPipe, "%lf %lf**\n**", train\_vectors[i][0], train\_vectors[i][1]);  fprintf(gnuplotPipe, "e**\n**");  }    for (int i = 0; i < W.size(); i++) {  color = 4;  pt = 5;  fprintf(gnuplotPipe, "plot '-' w p lc %d pt %d lw 1**\n**", color, pt);  fprintf(gnuplotPipe, "%lf %lf**\n**", W[i][0], W[i][1]);  fprintf(gnuplotPipe, "e**\n**");  }  fprintf(gnuplotPipe, "pause %f**\n**", pause);  fflush(gnuplotPipe);  }    NeuronNet::NeuronNet(int \_neurons\_number, int input\_vector\_size): neurons\_number(\_neurons\_number), sigmak(SIGMA\_INIT) {  neurons = neuron\_vector(neurons\_number);  for (int i = 0; i < neurons\_number; i++) {  neurons[i] = Neuron(input\_vector\_size, ETA\_INIT);  }  }    void NeuronNet::train(vector\_fvector train\_vectors, int epochNum, FILE \*gnuplotPipe) {  const int Kmax = train\_vectors.size();  for (int epoch = 0, kGlobal = 0; epoch < epochNum; epoch++) {  for (int k = 0; k < Kmax; k++, kGlobal++) {  if (k == 3) {  pause = 0.1;  }  plot\_dots(gnuplotPipe, train\_vectors, getW(), k);  fvector Xk = train\_vectors[k];  printf("x=%f**\t**y=%f**\n**", Xk[0], Xk[1]);  std::vector<std::pair<int, float>> neuronId\_d\_vector;  for (int i = 0; i < neurons.size(); i++) {  if (k < 3) {  if (neurons[i].check\_participate()) {  fvector Wi = neurons[i].Wi;  const float d = calculate\_d(Xk, Wi);  neuronId\_d\_vector.push\_back(std::pair<int, float>(i, d));  }  } else {  fvector Wi = neurons[i].Wi;  const float d = calculate\_d(Xk, Wi);  neuronId\_d\_vector.push\_back(std::pair<int, float>(i, d));  }  }  // Сортировка по d  std::sort(neuronId\_d\_vector.begin(), neuronId\_d\_vector.end(),  [](const std::pair<int, float>& p1, const std::pair<int, float>& p2) {  return (p1.second < p2.second);  });  for (int m = 0; m < neuronId\_d\_vector.size(); m++) {  int neuronId = neuronId\_d\_vector[m].first;  neurons[neuronId].changeW(m, sigmak, Xk);  neurons[neuronId].changeEta(ETA\_MIN, ETA\_MAX, kGlobal, Kmax \* (epochNum));  }  if (k < 3) {  for (int i = 0; i < neurons.size(); i++) {  if (i != neuronId\_d\_vector[0].first) {  neurons[i].changePiLose(neurons\_number);  } else {  neurons[i].changePiWin();  }  }  }  float a = (SIGMA\_MIN / SIGMA\_MAX);  float b = (float)((kGlobal + 1)) / (float) (Kmax \* (epochNum));  sigmak = SIGMA\_MAX \* pow(a, b);  printf("k = %d**\n**", kGlobal);  }  plot\_dots(gnuplotPipe, train\_vectors, getW(), Kmax);  }  }    float NeuronNet::calculate\_d(const fvector & Xk, const fvector & Wi) {  float d = 0.0;  for (int i = 0; i < Xk.size(); i++) {  float delta = pow((Xk[i] - Wi[i]), 2);  d += delta;  }  return sqrt(d);  }    vector\_fvector NeuronNet::getW() {  vector\_fvector W;  for (int i = 0; i < neurons\_number; i++) {  fvector Wi = neurons[i].Wi;  W.push\_back(Wi);  }  return W;  } |