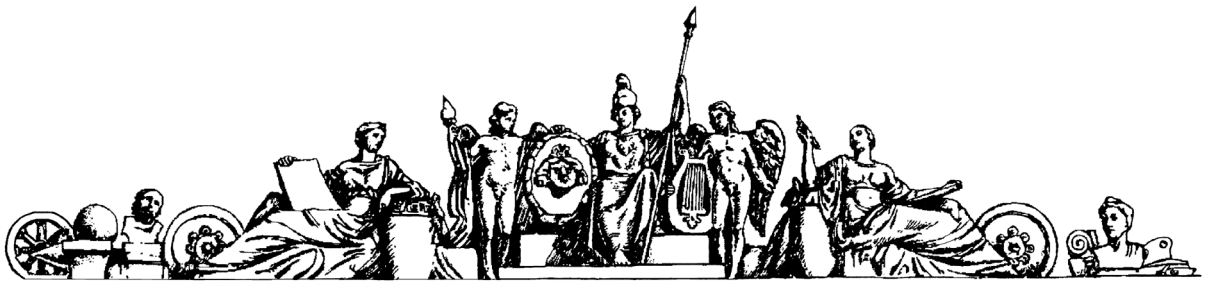

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н. Э. БАУМАНА

Факультет «Робототехника и комплексная автоматизация»
Кафедра «Системы автоматизированного проектирования»

Отчет по лабораторным работам по курсам: «Интеллектуальные подсистемы САПР», «Искусственные нейронные сети».

Студент _____ Лаур В. Ю.
группа РК6–103

Преподаватель _____ Федорук В. Г.

Москва 2012

Содержание

1 Лабораторная работа №2

1.1 Задание (вариант 33)

Для произвольно введеной последовательности целых чисел определить (и напечатать) максимально длинную подпоследовательность чисел, расположенных в возрастающем порядке.

1.2 Исходный код

```
maxSublist(L,MaxSub):-maxSublist(L,[],0,MaxSub,_).

maxSublist([],MaxSub,N,MaxSub,N):-!.
maxSublist(L,L1,N1,MaxSub,N):-ordPrefix(L,L2,1,N2,Rest),
  currMaxSublist(L1,L2,N1,N2,Sub,Len),
  maxSublist(Rest,Sub,Len,MaxSub,N).

ordPrefix([A,B|L],[A|L1],C,N,Rest):-A =< B, !, C1 is C+1,
  ordPrefix([B|L],L1,C1,N,Rest).
ordPrefix([A|L],[A],N,N,L).

currMaxSublist(_,L2,N1,N2,L2,N2):-N1 =< N2, !.
currMaxSublist(L1,_,N1,_,L1,N1).
```

2 Лабораторная работа №3

2.1 Краткое описание инстара Гроссберга

Структурная схема нейрона данного типа представлена на рис. 1.

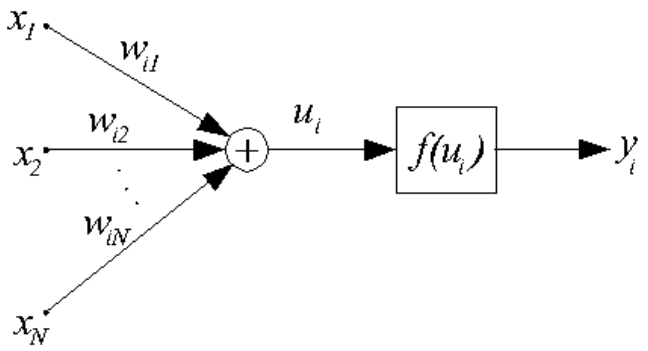

Рис. 1. Инстар Гроссберга

Особенности инстара, отличающие его от нейронов других типов:

1. функция активации $f(u_i)$ часто линейна, т.е. $y_i \sim u_i$;
2. входной вектор X нормализован таким образом, что его евклидова норма равна 1;
3. обучение инстара возможно как с учителем, так и без него.

Нормализация элементов вектора X производится по формуле:

$$x_j \leftarrow \frac{x_j}{\sqrt{x_1^2 + x_2^2 + \ldots + x_N^2}} \, .$$

Обучение инстара с учителем производится по правилу Гроссберга:

$$w_{ij}\left(t+1\right) = w_{ij}\left(t\right) + \eta d_i^k \left(x_j^k - w_{ij}\left(t\right)\right) \, , \qquad (1)$$

где η — коэффициент обучения, значение которого выбирается в диапазоне $(0, 1)$.

В качестве начальных обычно выбираются нулевые значения весовых коэффициентов. Необходимо обратить внимание на то, что на изменение значений весовых коэффициентов оказывают влияние только положительные примеры эталонных пар, для которых $d_i^k = 1$.

На процесс обучения инстара решающее влияние оказывает величина коэффициента обучения η. При $\eta = 1$ веса w_{ij} принимают значения

соответствующих входов x_j^k текущей эталонной пары за один цикл обучения (при этом происходит абсолютное «забывание» предыдущих значений $w_{ij}(t)$). При $\eta < 1$ в результате обучения коэффициенты w_{ij} принимают некоторые «усредненные» значения обучающих векторов X^k, $k = 1, 2, \ldots, p$.

Предположим, что i-ый инстар был обучен на единственной положительной эталонной паре $\langle X^1, 1\rangle$. При этом вектор входных весов инстара $W_i = [w_{i1}, w_{i2}, \ldots, w_{iN}]^T$ совпадет с обучившим вектором X^1. В режиме классификации на вход инстара подается вектор X^2, тогда на выходе вырабатывается сигнал

$$y_i = u_i = W^T \cdot X^2 = X^1 \cdot X^2 = \|X^1\|_2 \cdot \|X^2\|_2 \cdot \cos\left(\angle X^1 X^2\right) \ . \qquad (2)$$

Поскольку входные векторы X^1 и X^2 нормализованы (т. е. $\|X^1\|_2 = \|X^2\|_2 = 1$), то выходной сигнал инстара равен просто косинусу угла между векторами X^1 и X^2.

Функционирование инстара наглядно иллюстрируется графически (для двухмерного случая). В режиме обучения при предъявлении, например, трех положительных примеров, содержащих двухкомпонентные векторы X^1, X^2 и X^3, подбирается вектор входных весов W, представляющий собой «усреднение» этих входных векторов, как это показано на рис. 2.

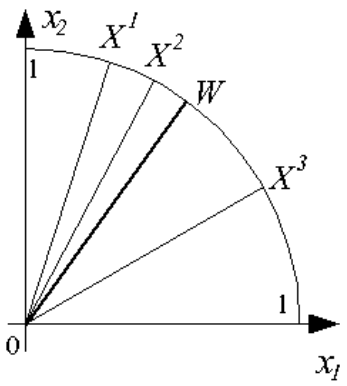

Рис. 2. Усреднение трех векторов

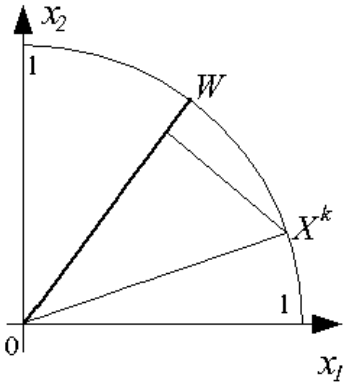

Рис. 3. Косинус

В режиме классификации при подаче на вход инстара очередного вектора X^k определяется степень его близости к «типичному» вектору W в виде косинуса угла между этими векторами, как это показано на рисунке 3.

Обучение инстара Гроссберга без учителя предполагает случайный выбор начальных значений входных весов w_{ij} и их нормализацию, подобную нормализации вектора входных сигналов X. Дальнейшее уточнение весов реализуется следующей формулой:

$$w_{ij}\,(t+1) = w_{ij}\,(t) + \eta y_i\,(t)\left(x_j^k - w_{ij}\,(t)\right) \ .$$

2.2 Реализация

В программе реализован инстар с трехкомпонентным входным вектором. В этом случае при нормализации все вектора (входные данные, веса синапсов) проецируются на единичную сферу.

При запуске программы в качестве аргумента командной строки передается имя файла, содержащего координаты обучающих данных (в формате x, y, z). Поскольку все точки нормируются, логично показать расположение точек в пространстве в сферических координатах (азимут φ и угол подъема θ). Это упрощает график, поскольку он становится двухмерным. На рисунке 4 показано расположение на поверхности сферы обучающих точек крестиками и положение весового вектора маленькой окружностью. Вертикальным линиям сетки на этом графике соответствуют меридианы, горизонтальным — параллели единичной сферы.

В процессе обучения нейрона можно наблюдать перемещение весового вектора в окне утилиты Gnuplot.

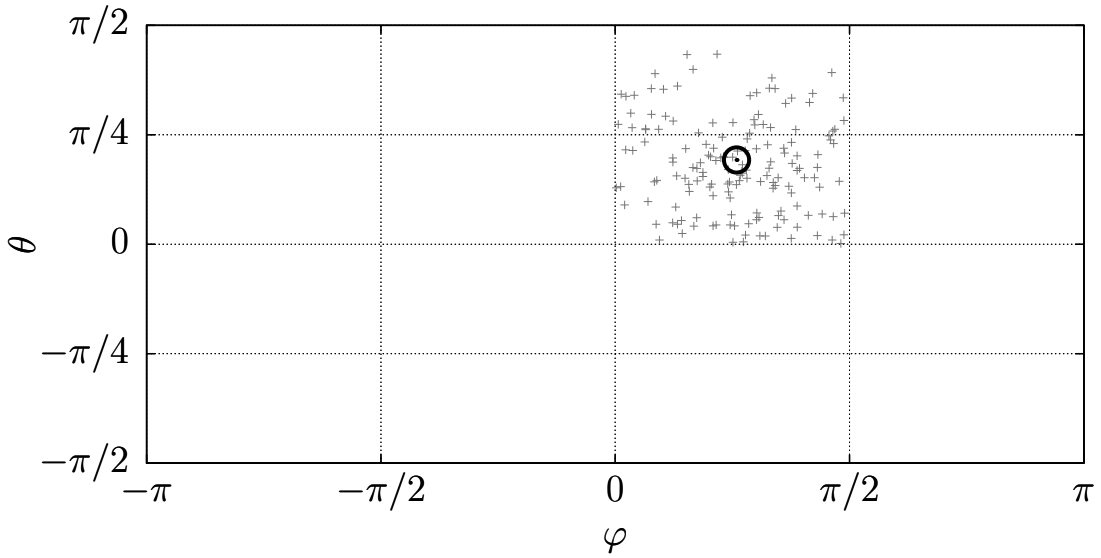

Рис. 4. Распределение обучающих данных в пространстве (φ, θ)

После обучения пользователю предлагается ввести координаты тестовой точки, которая в дальнейшем отображается на трехмерном графике в виде черной точки (рис 5). Для выхода из программы необходимо ввести три нулевых значения.

2.3 Обучение нейрона

Обучение нейрона происходит в режиме онлайн по формуле (1).

Для генерации обучающих точек была написана простенькая программа. Она генерирует случайным образом точки с неотрицательными компонентами x, y, z (рис. 4 и 5). и записывает их в файл, который в дальнейшем передается программе с нейроном.

Начальное положение вектора весовых коэффициентов задается случайным образом, но так, чтобы $\|W\| = 1$.

Выбор коэффициента обучения η предоставлен пользователю. Рекомендуемое значение — $0.001 \ldots 0.1$.

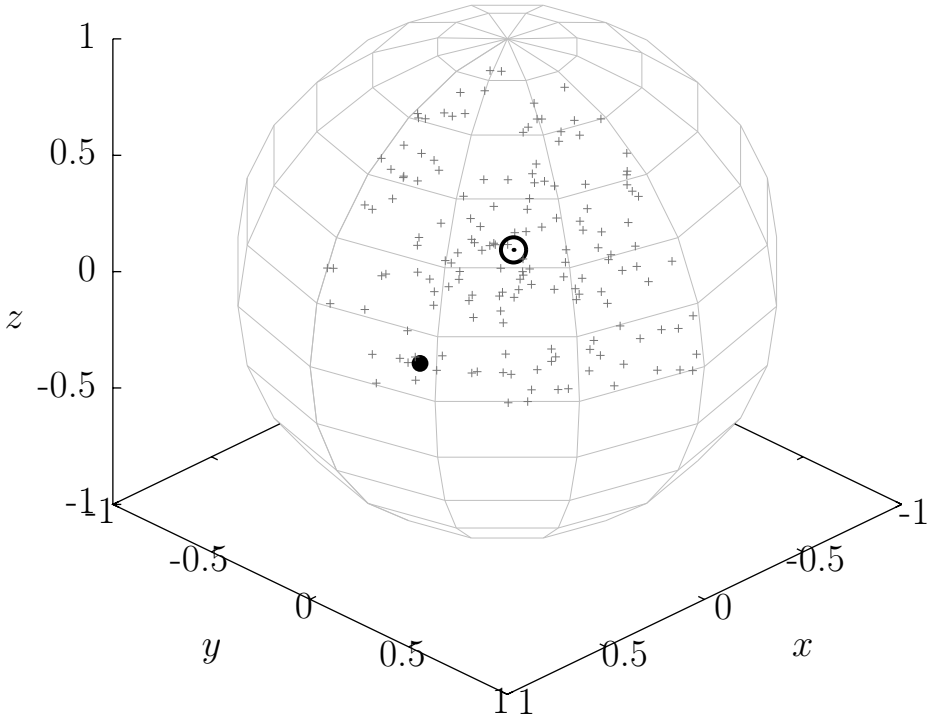

Рис. 5. Инстар в режиме классификации

2.4 Результаты обучения нейрона

На рис. 5 показано распределение обучающих данных в пространстве $[x, y, z]^T$ после нормирования, положение весового вектора (маленькой окружностью), положение анализируемого входного вектора (черная точка). Как и в двухмерном случае (рис. 2 и 3), выходное значение тестовой

точки определяется только косинусом угла $\angle W X^k$ (2). Численные значения результатов обучения сведены в таблицу 1.

Таблица 1. Результаты обучения

	x	y	z	φ	θ	out
W	0.565	0.598	0.568	0.8132	0.604	
X^k	1	1	1	0.785	0.615	0.999
	1	1	−1	0.785	−0.615	0.343
	3	−2	−1	−0.588	−0.271	−0.018

2.5 Листинг программы

2.5.1 Instar.cpp

```
#include "Instar.h"
const double PAUSE = 0.5; //пауза в гнуплоте

class Instar
{
public:
  Instar(double, FILE *, FILE *);
  void StudyOn(void);
  void PlotSphere(void);
  int Test(void);
protected:
  void GetStudyData(FILE *);
  void norm(point *);
  void PlotGreenPoint(int);
private:
  FILE * gpipe; //гнуплот
  double eta; //коэффициент обучения
  point * StudyMass; //массив обучающих точек
  point w; //веса синапсов
  int StudyMassSize; //количество обучающих точек в массиве
  point p; //это для последующего тестирования
};

//аргументы: коэфициент обучения, входной файл с данными
Instar::Instar(double a, FILE * studyfile, FILE * gplt)
{
  eta=a;  gpipe=gplt;
  srand(time(NULL));
  w.x=double(rand()/(RAND_MAX+1.0))*2-1;
  w.y=double(rand()/(RAND_MAX+1.0))*2-1;
  w.z=double(rand()/(RAND_MAX+1.0))*2-1;
  GetStudyData(studyfile); //получить обучающие данные
}

//записываем в массив обучающие выборки из файла
void Instar::GetStudyData(FILE * studyfile)
{
  register int i=0;
  char str[255];
```

```
  double outvalue;
  StudyMassSize=0;
  while(fgets(str, 255, studyfile)) //считаем количество строк входного файла
    StudyMassSize++;
  cout << StudyMassSize << " строк вх. файла" << endl;
  StudyMass = new point [StudyMassSize]; //выделяем память для данных
  //читаем входной файл:
  fseek(studyfile, 0, SEEK_SET);
  while(fgets(str, 255, studyfile))
  {
    sscanf(str, "%lf\t%lf\t%lf\t%lf\n", &StudyMass[i].x,
    &StudyMass[i].y, &StudyMass[i].z, &outvalue);
    norm(&StudyMass[i]);
    if(outvalue!=0) //игнорируем нулевые значения
      i++;
  }
  StudyMassSize=i; //пересчитываем без учета плохих точек
  fclose(studyfile);
}

//нормирует координаты точки, вычисляет сф. координаты
void Instar::norm(point * p)
{
  double TempNorm;
  if(p->x==0 && p->y==0 && p->z==0)   p->phi = p->theta =0;
  else
  {
    TempNorm=sqrt((p->x)*(p->x) + (p->y)*(p->y) + (p->z)*(p->z));
    p->phi=atan2(p->y, p->x);
    p->x=p->x/TempNorm;   p->y=p->y/TempNorm;
    p->z=p->z/TempNorm;   p->theta=asin(p->z);
  }
}

//обучение по правилу Гроссберга:
void Instar::StudyOn() //обучение онлайн
{
  register int i;
  for (i=0; i<StudyMassSize; i++)
  {
    w.x = w.x+eta*(StudyMass[i].x-w.x);
    w.y = w.y+eta*(StudyMass[i].y-w.y);
    w.z = w.z+eta*(StudyMass[i].z-w.z);
    w.phi=atan2(w.y, w.x);  w.theta=asin(w.z);
    fprintf(gpipe, "unset multiplot\n");
    PlotGreenPoint(i);  fprintf(gpipe, "pause %lf\n", PAUSE);
  }
  norm(&w);
}

void Instar::PlotGreenPoint(int PointNo)
{
  register int i;
  fprintf(gpipe, "set multiplot\n");
  fprintf(gpipe, "plot '-' u 1:2 w points lt rgb \"green\"\n");
  for (i = 0; i <= PointNo; i++)
    fprintf(gpipe, "%f\t%f\n", StudyMass[i].phi, StudyMass[i].theta);

  fprintf(gpipe, "end\n");
  fprintf(gpipe, "plot '-' u 1:2 w points lt rgb \"black\" lw 3\n");
```

```
  fprintf(gpipe, "%f\t%f\nend\n", w.phi, w.theta);
}

void Instar::PlotSphere()
{
  //подготовка файлов с точками:
  FILE * pointfile = fopen("points.txt", "w");
  FILE * weightfile = fopen("weights.txt", "w");
  if(pointfile==NULL || weightfile==NULL)   exit(-6);
  register int i;
  for (i=0; i<StudyMassSize; i++)
    fprintf(pointfile, "%f\t%f\n", StudyMass[i].phi, StudyMass[i].theta);
  fprintf(weightfile, "%f\t%f\n", w.phi, w.theta);
  fclose(pointfile); fclose(weightfile);
  sphereinit(gpipe);
  fprintf(gpipe, "splot cos(u)*cos(v), -cos(u)*sin(v), sin(u)");
  fprintf(gpipe, " with lines lc rgb \"cyan\",");
  fprintf(gpipe, "'points.txt' u 1:2 w points lc rgb \"green\",");
  fprintf(gpipe, "'weights.txt' u 1:2 w points lc rgb \"black\",");
  fprintf(gpipe, "'-' u 1:2 w points lc rgb \"blue\"\n");
  fprintf(gpipe, "%f\t%f\n", p.phi, p.theta);
  fprintf(gpipe, "end\n");
}

int Instar::Test()
{
  double outvalue;

  cout << "x="; cin >> p.x;
  cout << "y="; cin >> p.y;
  cout << "z="; cin >> p.z;
  if(p.x==0 && p.y==0 && p.z==0)     return 0;
  norm(&p);
  cout << "out=" << (outvalue=w.x*p.x +
    w.y*p.y + w.z*p.z) << endl;
  PlotSphere();
  return 1; //пусть возвращает ноль, когда надо закончить
}

//          НАЧАЛО ПРОГРАММЫ:
int main (int argc, char** argv)
{
  //обработка неправильного запуска программы:
  if(argc!=2)
  {
    HelpMessage();
    return -1;
  }
  FILE* studyfile = fopen(argv[1],"r");
  if(studyfile==NULL)    return -3;
  //готовим рисование картинки:
  FILE * gpipe=popen("gnuplot -persist -geometry 700x700","w");
  //FILE * gpipe=fopen("gplt.txt","w"); полезно для отладки
  if(!gpipe)     return -4;
  gnuplotinit(gpipe); //инициализации параметров картинки
  char ch;
  //создание и инициализация нейрона:
  Instar MyInstar(GetEta(), studyfile, gpipe);
  MyInstar.StudyOn();
  fflush(gpipe);
```

```
  cout << "Нейрон обучен на входном файле." << endl;
  cout << "Начать тестирование? (y/n)" << endl;
  cin >> ch;
  if(ch!='y' && ch!='Y')     return 0;

  cout << "Начинаем тестирование:" << endl;
  do
    fflush(gpipe);
  while(MyInstar.Test());
  fprintf(gpipe, "exit\n");
  pclose(gpipe);

  return 0;
}
```

2.5.2 Instar.h

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
using namespace std;

typedef struct point
{
  double x, y, z; //нормированные координаты точки
  double phi, theta; //в сферических координатах
} point;

//         ОПРЕДЕЛЕНИЕ ФУНКЦИЙ:
void HelpMessage(void)
{
  cout << "Передайте в качестве аргумента имя файла с обучающими "
     << "данными." << endl << "Формат входных данных:" << endl;
  cout << "XVALUE \\t YVALUE \\t DVALUE \\n\n" << endl;
}

double GetEta() //ввод параметра "eta"
{
  double eta;
  cout << "Введите число в интервале (0, 1)." << endl;
  cout << "Оно будет использовано как коэфф. обучения:" << endl;
  cin >> eta;
  if(eta<=0 || eta >=1)
  {
    cout << "Вы глупец." << endl;
    exit(-2);
  }
  return eta;
}
void gnuplotinit(FILE * gpipe)
{
  fprintf(gpipe, "set terminal X11\n");
  fprintf(gpipe, "set nokey\n");
  fprintf(gpipe, "set pointsize 1.5\n");
  fprintf(gpipe, "set xrange [-pi:pi]\n");
  fprintf(gpipe, "set yrange [-pi/2:pi/2]\n");
  fprintf(gpipe, "set multiplot\n");
```

```
  fprintf(gpipe, "\n");
}

void sphereinit(FILE * gpipe)
{
  fprintf(gpipe, "unset multiplot\n");
  fprintf(gpipe, "set dummy u,v\n");
  fprintf(gpipe, "set angles radian\n");
  fprintf(gpipe, "set parametric\n");
  fprintf(gpipe, "set samples 32, 32\n");
  fprintf(gpipe, "set isosamples 13, 13\n");
  fprintf(gpipe, "set mapping spherical\n");
  fprintf(gpipe, "set hidden3d offset 0 trianglepattern 3");
  fprintf(gpipe, " undefined 1 altdiagonal bentover\n");
  fprintf(gpipe, "set xrange [-1:1]\n");
  fprintf(gpipe, "set yrange [-1:1]\n");
  fprintf(gpipe, "set zrange [-1:1]\n");
  fprintf(gpipe, "set xlabel 'x'\n");
  fprintf(gpipe, "set ylabel 'y'\n");
  fprintf(gpipe, "set zlabel 'z'\n");
  fprintf(gpipe, "set ticslevel 0\n");
  fprintf(gpipe, "set urange [-pi/2 : pi/2] noreverse nowriteback\n");
  fprintf(gpipe, "set vrange [0.000 : pi] noreverse nowriteback\n");
}
```

2.5.3 generator.cpp

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
const int N = 300; //количество точек на выходе

int main(int argc, char const *argv[])
{
  if(argc!=2)   return -1;
  FILE* studyfile = fopen(argv[1],"w");
  if(studyfile==NULL)   return -3;
  srand(time(NULL));
  double x, y, z;
  register int i=0;
  while (i<N)
  {
    x=double(rand()/(RAND_MAX+1.0));
    y=double(rand()/(RAND_MAX+1.0));
    z=double(rand()/(RAND_MAX+1.0));
    fprintf(studyfile, "%lf\t%lf%lf\t%d\n", x, y, z, rand()%2);
    i++;
  }
  fclose(studyfile);
  return 0;
}
```

3 Лабораторная работа №4

3.1 Сети с самоорганизацией на основе конкуренции

Сетями с самоорганизацией называются сети, не требующие для своего обучения «учителя» и самостоятельно адаптирующие свои веса под обучающие данные. Такие сети строятся из нейронов типа WTA и подобных им. Как правило, это однослойные сети, в которых каждый нейрон получает все компоненты входного вектора X размерностью N. На рисунке 6 представлена структурная схема такой сети.

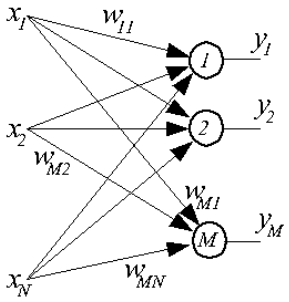

Рис. 6. Схема сети с самоорганизацией на основе конкуренции

Веса входных связей i-ого нейрона образуют вектор

$$W_i = \begin{bmatrix} w_{i1} & w_{i2} & \dots & w_{iN} \end{bmatrix}^T \ .$$

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.

Укрупненно процесс обучения сети выглядит следующим образом. На вход сети подается обучающий вектор X^k, для каждого нейрона определяется $d(X^k, W_i)$ — расстояние (в смысле выбранной метрики) между векторами X^k и W_i. Определяется нейрон–победитель, для которого это расстояние оказывается наименьшим. Вокруг нейрона–победителя образуется окрестность S_w^k из нейронов–соседей с известным «расстоянием» до победителя. Веса нейрона–победителя и веса его соседей из S_w^k уточняются, например, по правилу Кохонена

$$W_i^{k+1} = W_i^k + \eta_i^k \left(X^k - W_i^k \right) \ ,$$

где η_i^k — коэффициент обучения, значение которого уменьшается с увеличением расстояния от i-ого нейрона до победителя. Веса нейронов вне S_w^k не изменяются. Размер окрестности S_w^k и величина η_i^k с течением времени обучения уменьшаются.

3.2 Меры расстояний между векторами

В качестве меры измерения расстояния между векторами чаще всего используются:

- евклидова мера $\quad d(X, W_i) = \|X - W_i\| = \sqrt{\sum_{j=1}^{N} (x_j - w_{ij})^2}$;
- скалярное произведение

$$d(X, W_i) = 1 - X \cdot W_i = 1 - \|X\|_2 \cdot \|W_i\|_2 \cdot \cos(\angle X W_i) ;$$

- манхэттеновское расстояние $\quad d(X, W_i) = \sum_{j=1}^{N} |x_j - w_{ij}|$;
- m-норма $\quad d(X, W_i) = \max_j |x_j - w_{ij}|$.

3.3 Проблема мертвых нейронов

При «слепом» (как правило, случайном) выборе начальных значений весов часть нейронов может оказаться в области пространства, в которой отсутствуют обучающие данные или где их количество ничтожно мало. Такие нейроны имеют очень мало шансов на победу в конкурентной борьбе и адаптацию своих весов, вследствие чего они остаются мертвыми. В итоге уменьшается количество активных нейронов, участвующих в анализе входных данных, и, следовательно, увеличивается погрешность их интерпретации, называемая погрешностью квантования. Встает проблема активации всех нейронов сети на этапе обучения.

Такую активацию можно осуществить, базируясь на учете количества побед, одержанных каждым нейроном в ходе обучения. Существуют разные механизмы такого учета.

В одном из таких подходов каждому нейрону сети приписывается потенциал p_i, значение которого модифицируется после предъявления каждого обучающего вектора X^k по следующей формуле (в ней w — индекс нейрона-победителя):

$$\begin{aligned} p_i^{k+1} &= p_i^k + 1/M \quad \text{для} \quad i \neq w , \\ p_i^{k+1} &= p_i^k - p_{min} \quad \text{для} \quad i = w , \end{aligned} \tag{3}$$

где p_{min} — минимальный потенциал, разрешающий участие в конкурентной борьбе. Максимальное значение потенциала устанавливается равным 1. На практике хорошие результаты получены для $p_{min} = 0.75$.

В другом подходе для выявления победителя в конкурентной борьбе предлагается использовать не фактические значения расстоянии между векторами $d(X^k, W_i)$, а величины, промасштабированные количеством побед $N_i^w \cdot d(X^k, W_i)$, где N_i^w — количество побед, одержанных i-м нейроном к текущему моменту.

Как показали эксперименты, при использовании описанных выше механизмов двух–трех циклов обучения обычно достаточно для активации всех нейронов сети, поэтому в последующих циклах эти механизмы отключаются.

3.4 Алгоритмы обучения

Целью обучения сети с самоорганизацией на основе конкуренции является минимизация погрешности квантования

$$E_q = \frac{1}{p} \sum_{k=1}^{p} d\left(X^k, W_{w(k)}\right) ,$$

где p — количество обучающих векторов X^k, $W_{w(k)}$ — вектор весов нейрона–победителя при предъявлении вектора X^k.

Примеры результатов обучения, близких к оптимальным, представлены ниже на рисунках. Используются сети с 15 и 22 нейронами и двухкомпонентным входным вектором $X = [x_1, x_2]^T$. На левых рисунках представлено распределение данных в обучающих выборках, на правых — распределение весов нейронов обученной сети.

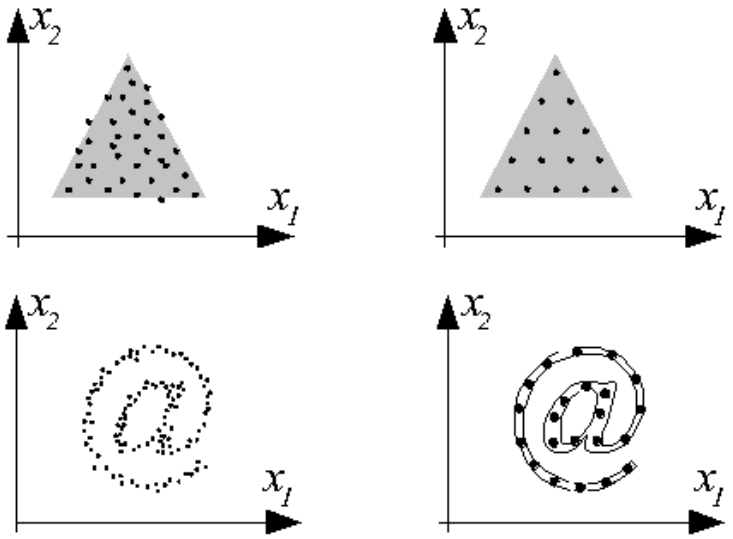

Рис. 7. Обучение сети

Для обучения сетей с самоорганизацией на основе конкуренции наибольшее распространение получили два алгоритма. Один из них описан ниже.

3.5 Алгоритм Кохонена

В нейронных сетях, предложенных Т. Кохоненом (1982 г.), соседство нейронов носит чисто 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).

Коррекция весов нейронов в ходе обучения выполняется по формуле

$$W_i^{k+1} = W_i^k + \eta^k G^k\left(i, X^k\right)\left(X^k - W_i^k\right) \ ,$$

где функция соседства $G^k(i, X^k)$ определяется, как правило, формулой Гаусса в виде

$$G^k(i, X^k) = \exp\left(-\frac{d^2\left(i, X^k\right)}{2(s^k)^2}\right) , \tag{4}$$

где $d(i, X^k)$ — расстояние от i-ого нейрона до нейрона–победителя с индексом w^k в k-ом цикле обучения. При этом $d(w^k, X^k) = 0$, $d(i, X^k) = 1$ для всех ближайших соседей w^k, $d(i, X^k) = 2$ для всех «внешних» ближайших соседей ближайших соседей нейрона победителя с индексом w^k и так далее.

Как обычно, коэффициент обучения η^k и параметр ширины функции Гаусса s^k уменьшаются в ходе обучения (с ростом k).

В результате обучения слоя Кохонена по такому алгоритму топологически соседние нейроны становятся типичными представителями кластеров обучающих данных, соседствующих в многомерном пространстве. В этом достоинство сетей Кохонена, называемых также *картами Кохонена*, — наглядность в представлении (путем одномерной или двумерной визуализации) многомерных данных.

3.6 Применение

Очевидным практическим приложением сетей с самоорганизацией является сжатие (с потерями) данных, в частности, покадровое сжатие изображений. Но мы воспользуемся ей для других целей.

Важным свойством сетей с самоорганизацией на основе конкуренции является способность к кластеризации данных и их распознаванию. Это обеспечивает их широкое применение для решения задач диагностики, например, неисправностей оборудования.

3.7 Реализация

При запуске программы в качестве аргумента командной строки передается имя файла, содержащего координаты обучающих данных (в формате x, y). Нормализация обучающих векторов не производится. Все нейроны связаны топологией «линейка». Связи между соседями показаны на графиках отрезками прямых линий. Количество нейронов задается как глобальная константа в коде программы.

3.8 Обучение нейрона

Для генерации обучающих точек был написан специальный генератор, способный «рисовать» спираль (рис. 10), окружность (рис. 11) или фигуру Лиссажу в форме восьмерки (рис. 8, 9) любым заданным количеством точек.

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

На первом этапе параметр ширины функции Гаусса s^k (см. формулу (4)) и коэффициент обучения η^k выбираются больши́ми. Экспериментально подобранные диапазоны: $s^k = 10 \ldots 2$ и $\eta^k = 0.6 \ldots 0.15$. Победители штрафуются по формуле (3), однако основной причиной оживления нейронов являются высокие значения параметров s^k и η^k. По окончании первого этапа веса нейронов выстраиваются по кривой линии (рис. 8).

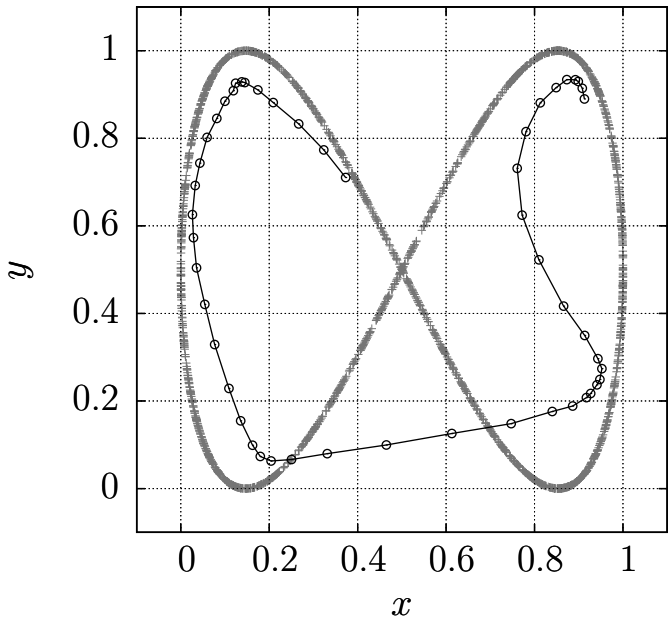

Рис. 8. Нейронная сеть после первого этапа обучения

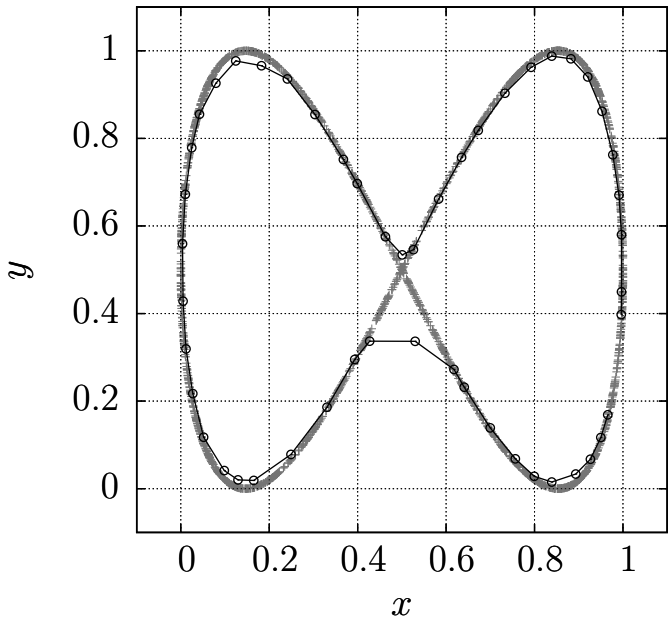

Рис. 9. Обученная нейронная сеть

На втором этапе победители не штрафуются. Параметр ширины функции Гаусса s^k убывает со временем в диапазоне $1.5 \ldots 0.01$; коэффициент обучения η^k — в диапазоне $0.5 \ldots 0.1$. Экспериментальное тестирование показало, что второй цикл обучения несколько уточняет позиции нейронов, дальнейшее же обучение бесполезно.

В процессе обучения можно наблюдать перемещение цепочки нейронов в окне утилиты Gnuplot.

3.9 Результаты обучения нейрона

На рис. 9–11 показано распределение обучающих данных в пространстве $[x, y]^T$ и положение весов нейронов. Несмотря на все приложенные усилия, некоторые нейроны остаются мертвыми. Это следствие правила соседства, предложенного Т. Кохоненом.

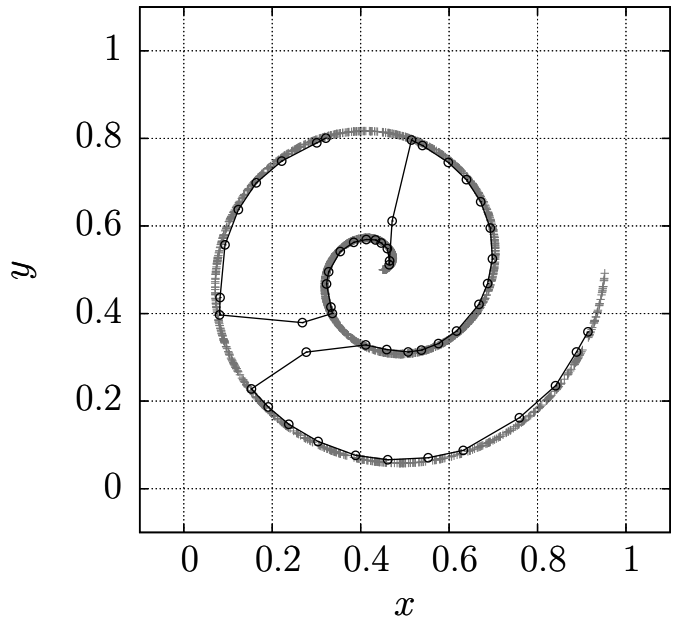

Рис. 10. Обученная нейронная сеть

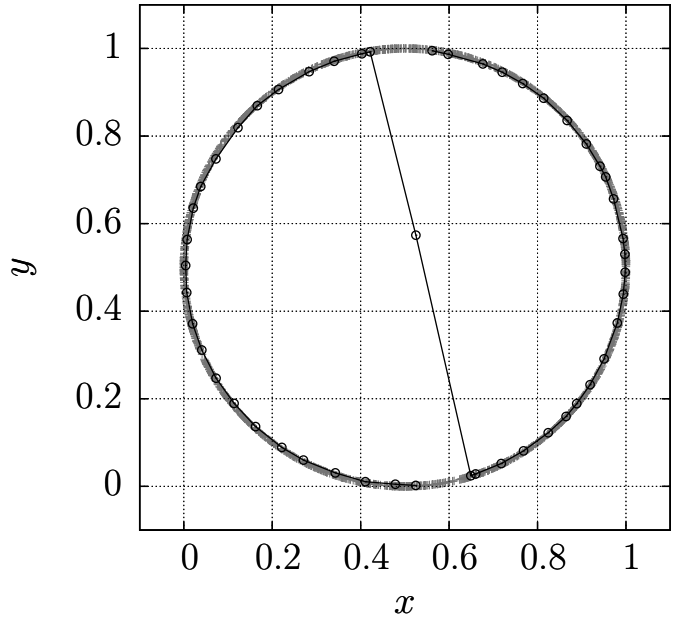

Рис. 11. Обученная нейронная сеть

3.10 Выводы

Алгоритм обучения Кохонена не очень хорош из-за характера соседства, не учитывающего реального геометрического расположения нейронов в пространстве. Вероятно, более сложная топология (квадратная или гексагональная сеть) хорошо себя покажет на полигональных объектах. На линейных объектах (рис. 9–11) же остаются мертвые нейроны.

3.11 Листинг программы

3.11.1 WTA.cpp

```
#include "WTA.h"

const int N = 2; //размерность входного вектора
const int M = 50; //количество нейронов
const int StudyCycles = 3; //количество циклов обучения
const double INF = 100000; //число, похожее на бесконечность

class NeuronNet
{
public:
  NeuronNet(FILE *, FILE *);
  void PlotStudyData();
  void Study();
  void PlotWeights();
  void PlotLabels(bool);
protected:
  void InitWeights();
  void GetStudyData(FILE *);
private:
  double w[M][N]; //веса нейронов
  FILE * gpipe; //гнуплот
  double eta; //коэффициент обучения
  double ** StudyMass; //массив обучающих точек
  int StudyMassSize; //количество обучающих точек в массиве
  double p[M]; //потенциалы для учета побед
};

NeuronNet::NeuronNet(FILE * studyfile, FILE * gplt) //инициализация весов
{
  gpipe=gplt;
  register int j;
  for(j=0; j<M; j++) //инициализируем потенциалы нейронов
    p[j]=0.5;
  InitWeights();
  GetStudyData(studyfile);
}

void NeuronNet::InitWeights()
{
  srand(time(NULL));
  register int i, j;
  for(i=0; i<M; i++)
    for(j=0; j<N; j++)
      w[i][j]=double(rand()/(RAND_MAX+1.0));
}

void NeuronNet::GetStudyData(FILE * studyfile)
{
  register int i;
  char str[255];
  StudyMassSize=0;
  while(fgets(str, 255, studyfile)) //считаем количество строк входного файла:
    StudyMassSize++;
  StudyMass = new double * [StudyMassSize]; //выделяем память для данных:
    for (i=0; i<StudyMassSize; i++)
      StudyMass[i]=new double [N];
```

```
  fseek(studyfile, 0, SEEK_SET);  //читаем входной файл:
  for(i=0; fgets(str, 255, studyfile); i++)
    sscanf(str, "%lf\t%lf\n", &StudyMass[i][0], &StudyMass[i][1]);
  fclose(studyfile);
  cout << StudyMassSize << " строк во вх. файле" << endl;
}

void NeuronNet::PlotStudyData()
{
  register int i;
  fprintf(gpipe, "plot '-' u 1:2 w points lt rgb \"green\" lw 2\n");
  for (i = 0; i < StudyMassSize; i++)
    fprintf(gpipe, "%f\t%f\n", StudyMass[i][0], StudyMass[i][1]);
  fprintf(gpipe, "end\n");
}

void NeuronNet::Study()
{
  const double p_min=0.75;
  const int FirstRuns = 2*M; //сколько проходов обучения штрафуем

  double eta_min, eta_max, eta_k; //коэффициент обучения
  double s_min, s_max, s_k; //параметр ширины функции Гаусса для соседства;
  int str, CurCycle=0;
  register int i, j;
  double dXW, mindXW; //расстояние d(X, W)
  int WinnerIndex=0; //номер нейрона - победителя
  //сначала штрафуем победителей:
  eta_max = 0.6;  eta_min = 0.15;
  s_max = 10; s_min = 2;
  for(str=0; str<StudyMassSize && str<FirstRuns; str++)
  {
    s_k = s_max - (s_max-s_min)/(FirstRuns-1)*CurCycle; //уменьшается со временем
    eta_k = eta_max - (eta_max-eta_min)/(FirstRuns-1)*CurCycle;
    //считаем квадраты расстояний для всех нейронов и находим победителя:
    for(j=0, mindXW=INF; j<M; j++)
    {
      for(i=0, dXW=0; i<N; i++)
        dXW += (StudyMass[str][i]-w[j][i])*(StudyMass[str][i]-w[j][i]);
      if(mindXW>dXW && p[j]>p_min)
      {
        WinnerIndex=j;
        mindXW=dXW;
      }
    }
    for(j=0; j<M; j++) //учитываем победы:
    {
      if(j==WinnerIndex)
      {
        p[WinnerIndex]=0;
        if(p[WinnerIndex]<0)
          p[WinnerIndex]-=p_min;
        continue;
      }
      p[j]+=1.0/M;
    }
    for(j=0; j<M; j++) //переучиваем соседей и победителя:
      for(i=0; i<N; i++)
        w[j][i] += eta_k*exp(-(WinnerIndex-j)*(WinnerIndex-j)/2/s_k)*
        (StudyMass[str][i]-w[j][i]);
```

```
    fprintf(gpipe, "unset multiplot\nset multiplot\n");
    fprintf(gpipe, "plot '-' w points lt rgb \"green\" lw 2\n");
    fprintf(gpipe, "%f\t%f\n", StudyMass[str][0], StudyMass[str][1]);
    fprintf(gpipe, "end\n");
    PlotWeights();
    fflush(gpipe);
  }
  cout << "Теперь не штрафуем победителей" << endl;
  eta_max = 0.5;  eta_min = 0.1;
  s_max = 1.5;  s_min = 0.01;
  for(CurCycle=0; CurCycle<StudyCycles; CurCycle++)
  {
    //уменьшается со временем по линейному закону:
    s_k = s_max - (s_max-s_min)/(StudyCycles-1)*CurCycle;
    eta_k = eta_max - (eta_max-eta_min)/(StudyCycles-1)*CurCycle;
    for(str=0; str<StudyMassSize; str++) //обучаем на входном файле:
    {
      //считаем квадраты расстояний для всех нейронов и находим победителя:
      for(j=0, mindXW=INF; j<M; j++)
      {
        for(i=0, dXW=0; i<N; i++)
          dXW += (StudyMass[str][i]-w[j][i])*(StudyMass[str][i]-w[j][i]);
        if(mindXW>dXW)
        {
          WinnerIndex=j;
          mindXW=dXW;
        }
      }
      //переучиваем соседей и победителя:
      for(j=0; j<M; j++)
      {
        for(i=0; i<N; i++)
          w[j][i] += eta_k*exp(-(WinnerIndex-j)*(WinnerIndex-j)/2/s_k)*
          (StudyMass[str][i]-w[j][i]);
      }
      /*PlotLabels(false);
      fprintf(gpipe, "unset multiplot\nset multiplot\n");
      fprintf(gpipe, "plot '-' w points lt rgb \"green\" lw 2\n");
      fprintf(gpipe, "%f\t%f\n", StudyMass[str][0], StudyMass[str][1]);
      fprintf(gpipe, "end\n");
      PlotWeights();
      fflush(gpipe);*/
    }
    //все обучено
    PlotLabels(false);
    fprintf(gpipe, "unset multiplot\nset multiplot\n");
    PlotWeights();
    PlotStudyData();
    PlotWeights();
    fflush(gpipe);
  }
  cout << "Сеть обучена" << endl;
}

void NeuronNet::PlotWeights()
{
  register int i;
  fprintf(gpipe, "plot '-' u 1:2 w linespoints lt rgb \"blue\" lw 2 pt 4\n");
  for(i=0; i<M; i++)
    fprintf(gpipe, "%f\t%f\n", w[i][0], w[i][1]);
```

```
  fprintf(gpipe, "end\n");
}

void NeuronNet::PlotLabels(bool PlotP)
{
  register int i;
  for(i=0; i<M; i++)
    if(PlotP)
      fprintf(gpipe, "set label %d \"%d_%2.2lf\" at %lf,%lf \n",
      i+1, i, p[i], w[i][0]+0.01, w[i][1]+0.02);
    else
      fprintf(gpipe, "set label %d \"%d\" at %lf,%lf \n",
      i+1, i, w[i][0]+0.01, w[i][1]+0.02);
}

//          НАЧАЛО ПРОГРАММЫ:
int main(int argc, char const *argv[])
{
  if(argc!=2) //обработка неправильного запуска программы:
  {
    HelpMessage();
    return -1;
  }
  FILE* studyfile = fopen(argv[1],"r");
  if(studyfile==NULL)    return -3;
  hellomessage();
  FILE * gpipe=popen("gnuplot -persist -geometry 700x700","w");
  //FILE * gpipe=fopen("gplt.txt","w"); полезно// для отладки
  //FILE * gpipe=stderr; полезно// для отладки
  if(!gpipe)
    return -4;
  gnuplotinit(gpipe); //инициализации параметров картинки
  NeuronNet wta(studyfile, gpipe);
  cout << "Теперь начинаем обучение" << endl;
  wta.Study();
  fclose(gpipe);
  return 0;
}
```

3.11.2 WTA.h

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
using namespace std;

//          ОПРЕДЕЛЕНИЕ ФУНКЦИЙ:
void hellomessage(void)
{
  cout << "Классификация данных по алгоритму Кохонена" << endl;
}

void HelpMessage(void)
{
  cout << "Неправильный запуск" << endl;
  cout << "Передайте в качестве аргумента имя файла с обучающими "
```

```
    << "данными." << endl << "Формат входных данных:" << endl;
  cout << "XVALUE \\t YVALUE \\n\n" << endl;
}

void gnuplotinit(FILE * gpipe)
{
  fprintf(gpipe, "set terminal X11\n");
  fprintf(gpipe, "set nokey\n");
  fprintf(gpipe, "set pointsize 1.5\n");
  fprintf(gpipe, "set xrange [-0.1:1.1]\n");
  fprintf(gpipe, "set yrange [-0.1:1.1]\n");
  fprintf(gpipe, "set multiplot\n");
  fprintf(gpipe, "\n");
}
```

3.11.3 generator.cpp

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.141592654
using namespace std;
const int N = 2000; //количество точек на выходе

void HelpMessage(void)
{
  cout << "Неправильный запуск" << endl;
  cout << "Передайте в качестве аргумента имя файла, в который"
    << "я запишу массив точек" << endl;
}

int main(int argc, char const *argv[])
{
  //обработка неправильного запуска программы:
  if(argc!=2)
  {
    HelpMessage();
    return -1;
  }
  FILE* studyfile = fopen(argv[1],"w");
  if(studyfile==NULL)   return -3;
  srand(time(NULL));
  double x, y, t;
  register int i=0;

  //спираль Архимеда
  /*while (i<N)
  {
    t=double(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;
    fprintf(studyfile, "%lf\t%lf\n", x, y);
    i++;
  }*/
  //две точки
  /*while (i<N)
  {
    x=double(rand()%2);
    y=1;
```

```
    fprintf(studyfile, "%lf\t%lf\n", x, y);
    i++;
  }*/
  //окружность
  /*while (i<N)
  {
    t=double(rand()/(RAND_MAX+1.0))*2*PI;
    y=sin(t)*0.5+0.5; x=cos(t)*0.5+0.5;
    fprintf(studyfile, "%lf\t%lf\n", x, y);
    i++;
  }*/
  //восьмерка Лиссажу
  while (i<N)
  {
    t=double(rand()/(RAND_MAX+1.0))*2*PI;
    y=sin(2*t)*0.5+0.5; x=sin(t)*0.5+0.5;
    fprintf(studyfile, "%lf\t%lf\n", x, y);
    i++;
  }
  fclose(studyfile);
  return 0;
}
```