С.Короткий

Нейронные сети Хопфилда и Хэмминга

Среди различных конфигураций искуственных нейронных сетей (НС) встречаются такие, при классификации которых по принципу обучения, строго говоря, не подходят ни обучение с учителем [1], ни обучение без учителя [2]. В таких сетях весовые коэффициенты синапсов рассчитываются только однажды перед началом функционирования сети на основе информации об обрабатываемых данных, и все обучение сети сводится именно к этому расчету. С одной стороны, предъявление априорной информации можно расценивать, как помощь учителя, но с другой – сеть фактически просто запоминает образцы до того, как на ее вход поступают реальные данные, и не может изменять свое поведение, поэтому говорить о звене обратной связи с "миром" (учителем) не приходится. Из сетей с подобной логикой работы наиболее известны сеть Хопфилда и сеть Хэмминга, которые обычно используются для организации ассоциативной памяти. Далее речь пойдет именно о них.

Структурная схема сети Хопфилда приведена на рис.1. Она состоит из единственного слоя нейронов, число которых является одновременно числом входов и выходов сети. Каждый нейрон связан синапсами со всеми остальными нейронами, а также имеет один входной синапс, через который осуществляется ввод сигнала. Выходные сигналы, как обычно, образуются на аксонах.

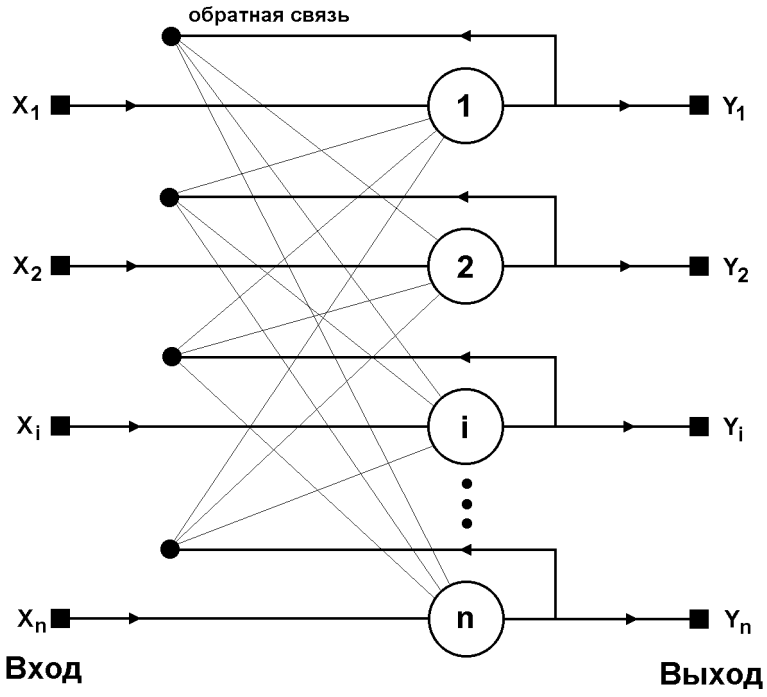

Рис.1 Структурная схема сети Хопфилда

Задача, решаемая данной сетью в качестве ассоциативной памяти, как правило, формулируется следующим образом. Известен некоторый набор двоичных сигналов (изображений, звуковых оцифровок, прочих данных, описывающих некие объекты или

характеристики процессов), которые считаются образцовыми. Сеть должна уметь из произвольного неидеального сигнала, поданного на ее вход, выделить ("вспомнить" по частичной информации) соответствующий образец (если такой есть) или "дать заключение" о том, что входные данные не соответствуют ни одному из образцов. В общем случае, любой сигнал может быть описан вектором $\mathbf{X} = \{ x_i: i=0...n-1\}$, n – число нейронов в сети и размерность входных и выходных векторов. Каждый элемент x_i равен либо +1, либо -1. Обозначим вектор, описывающий k-ый образец, через $\mathbf{X^k}$, а его компоненты, соответственно, – x_i^k, k=0...m-1, m – число образцов. Когда сеть распозна́ет (или "вспомнит") какой-либо образец на основе предъявленных ей данных, ее выходы будут содержать именно его, то есть $\mathbf{Y} = \mathbf{X^k}$, где $\mathbf{Y}$ – вектор выходных значений сети: $\mathbf{Y} = \{ y_i: i=0,...n-1\}$. В противном случае, выходной вектор не совпадет ни с одним образцовым.

Если, например, сигналы представляют собой некие изображения, то, отобразив в графическом виде данные с выхода сети, можно будет увидеть картинку, полностью совпадающую с одной из образцовых (в случае успеха) или же "вольную импровизацию" сети (в случае неудачи).

На стадии инициализации сети весовые коэффициенты синапсов устанавливаются следующим образом [3][4]:

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, i \neq j \\ 0, \qquad i = j \end{cases}$$

(1)

Здесь i и j – индексы, соответственно, предсинаптического и постсинаптического нейронов; x_i^k, x_j^k – i-ый и j-ый элементы вектора k-ого образца.

Алгоритм функционирования сети следующий (p – номер итерации):

1. На входы сети подается неизвестный сигнал. Фактически его ввод осуществляется непосредственной установкой значений аксонов:

$$y_i(0) = x_i, \quad i = 0...n-1,$$

(2)

поэтому обозначение на схеме сети входных синапсов в явном виде носит чисто условный характер. Ноль в скобке справа от y_i означает нулевую итерацию в цикле работы сети.

2. Рассчитывается новое состояние нейронов

$$s_j(p+1) = \sum_{i=0}^{n-1} w_{ij} y_i(p)$$

, $j=0...n-1$

(3)

и новые значения аксонов

$$y_j(p+1) = f\left[s_j(p+1)\right]$$

(4)

где f – активационная функция в виде скачка, приведенная на рис.2а.

3. Проверка, изменились ли выходные значения аксонов за последнюю итерацию. Если да – переход к пункту 2, иначе (если выходы застабилизировались) – конец. При этом выходной вектор представляет собой образец, наилучшим образом сочетающийся с входными данными.

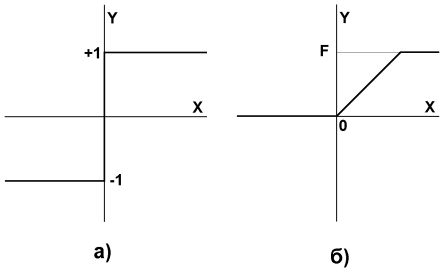

Рис.2 Активационные функции

Как говорилось выше, иногда сеть не может провести распознавание и выдает на выходе несуществующий образ. Это связано с проблемой ограниченности возможностей сети. Для сети Хопфилда число запоминаемых образов m не должно превышать величины, примерно равной 0.15•n. Кроме того, если два образа А и Б сильно похожи, они, возможно, будут вызывать у сети перекрестные ассоциации, то есть предъявление на входы сети вектора А приведет к появлению на ее выходах вектора Б и наоборот.

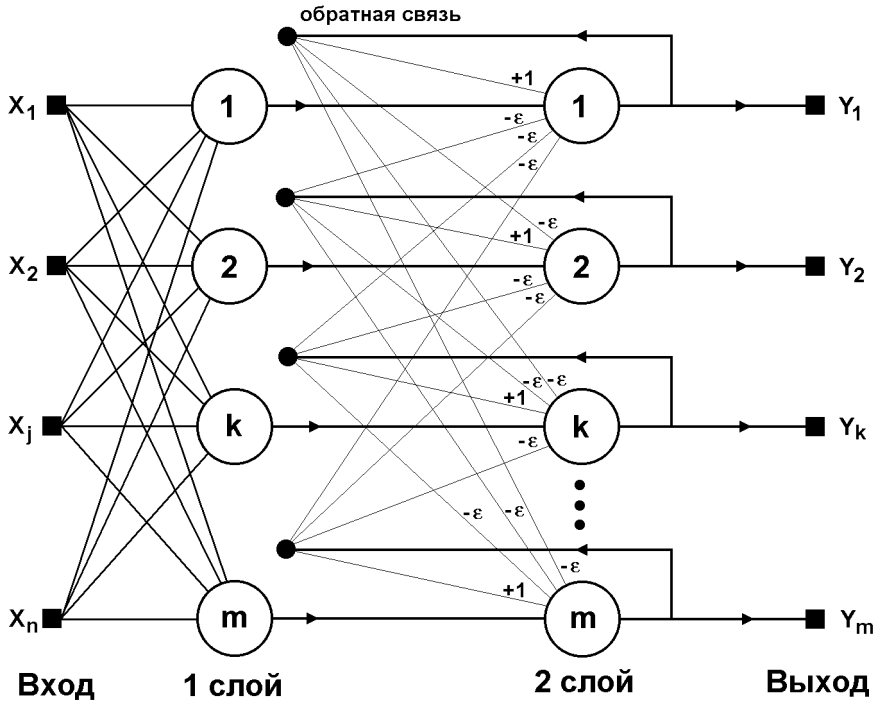

Рис.3 Структурная схема сети Хэмминга

Когда нет необходимости, чтобы сеть в явном виде выдавала образец, то есть достаточно, скажем, получать номер образца, ассоциативную память успешно реализует сеть Хэмминга. Данная сеть характеризуется, по сравнению с сетью Хопфилда, меньшими затратами на память и объемом вычислений, что становится очевидным из ее структуры (рис. 3).

Сеть состоит из двух слоев. Первый и второй слои имеют по m нейронов, где m – число образцов. Нейроны первого слоя имеют по n синапсов, соединенных со входами сети (образующими фиктивный нулевой слой). Нейроны второго слоя связаны между собой ингибиторными (отрицательными обратными) синаптическими связями. Единственный синапс с положительной обратной связью для каждого нейрона соединен с его же аксоном.

Идея работы сети состоит в нахождении расстояния Хэмминга от тестируемого образа до всех образцов. Расстоянием Хэмминга называется число отличающихся битов в двух бинарных векторах. Сеть должна выбрать образец с минимальным расстоянием Хэмминга до неизвестного входного сигнала, в результате чего будет активизирован только один выход сети, соответствующий этому образцу.

На стадии инициализации весовым коэффициентам первого слоя и порогу активационной функции присваиваются следующие значения:

$$w_{ik} = \frac{x_i^k}{2}$$

, $i=0...n-1$, $k=0...m-1$

(5)

$$T_k = n / 2, \quad k = 0...m-1$$

(6)

Здесь x_i^k – i-ый элемент k-ого образца.

Весовые коэффициенты тормозящих синапсов во втором слое берут равными некоторой величине $0 < \varepsilon < 1/m$. Синапс нейрона, связанный с его же аксоном имеет вес +1.

Алгоритм функционирования сети Хэмминга следующий:

1. На входы сети подается неизвестный вектор $\mathbf{X} = \{x_i : i=0...n-1\}$, исходя из которого рассчитываются состояния нейронов первого слоя (верхний индекс в скобках указывает номер слоя):

$$y_j^{(1)} = s_j^{(1)} = \sum_{i=0}^{n-1} w_{ij} x_i + T_j$$

, $j=0...m-1$

(7)

После этого полученными значениями инициализируются значения аксонов второго слоя:

$$y_j^{(2)} = y_j^{(1)}, \quad j = 0...m-1$$

(8)

2. Вычислить новые состояния нейронов второго слоя:

$$s_j^{(2)}(p+1) = y_j(p) - \varepsilon \sum_{k=0}^{m-1} y_k^{(2)}(p), k \neq j, j = 0...m-1$$

(9)

и значения их аксонов:

$$y_j^{(2)}(p+1) = f\left[s_j^{(2)}(p+1)\right],\ j = 0...m-1$$

(10)

Активационная функция f имеет вид порога (рис. 2б), причем величина F должна быть достаточно большой, чтобы любые возможные значения аргумента не приводили к насыщению.

3. Проверить, изменились ли выходы нейронов второго слоя за последнюю итерацию. Если да – перейди к шагу 2. Иначе – конец.

Из оценки алгоритма видно, что роль первого слоя весьма условна: воспользовавшись один раз на шаге 1 значениями его весовых коэффициентов, сеть больше не обращается к нему, поэтому первый слой может быть вообще исключен из сети (заменен на матрицу весовых коэффициентов), что и было сделано в ее конкретной реализации, описанной ниже.

Программная модель сети Хэмминга строится на основе набора специальных классов NeuronHN, LayerHN и NetHN – производных от классов, рассмотренных в предыдущих статьях цикла [1][2]. Описания классов приведены в листинге 1. Релизации всех функций находятся в файле NEURO_HN (листинг 2). Классы NeuronHN и LayerHN наследуют большинство методов от базовых классов.

В классе NetHN определены следующие элементы:

Nin и Nout – соответственно размерность входного вектора с данными и число образцов;

dx и dy – размеры входного образа по двум координатам (для случая трехмерных образов необходимо добавить переменную dz), dx*dy должно быть равно Nin, эти переменные используются функцией загрузки данных из файла LoadNextPattern;

DX и DY – размеры выходного слоя (влияют только на отображение выходого слоя с помощью функции Show); обе пары размеров устанавливаются функцией SetDxDy;

Class – массив с данными об образцах, заполняется функцией SetClasses, эта функция выполняет общую инициализацию сети, сводящуюся к запоминанию образцовых данных.

Метод Initialize проводит дополнительную инициализацию на уровне тестируемых данных (шаг 1 алгоритма). Метод Cycle реализует шаг 2, а метод IsConverged проверят, застабилизировались ли состояния нейронов (шаг 3).

Из глобальных функций – SetSigmoidAlfaHN позволяет установить параметр F активационной функции, а SetLimitHN задает коэффициент, лежащий в пределах от нуля до единицы и определяющий долю величины 1/m, образующую ε.

На листинге 3 приведена тестовая программа для проверки сети. Здесь конструируется сеть со вторым слоем из пяти нейронов, выполняющая распознавание пяти входных образов, которые представляют собой схематичные изображения букв размером 5 на 6 точек (см.рис.4а). Обучение сети фактически сводится к загрузке и запоминанию идеальных изображений, записанных в файле "charh.img", приведенном на листинге 4. Затем на ее вход поочередно

подаются зашумленные на 8/30 образы (см.рис.4б) из файла "charhh.img" с листинга 5, которые она успешно различает.

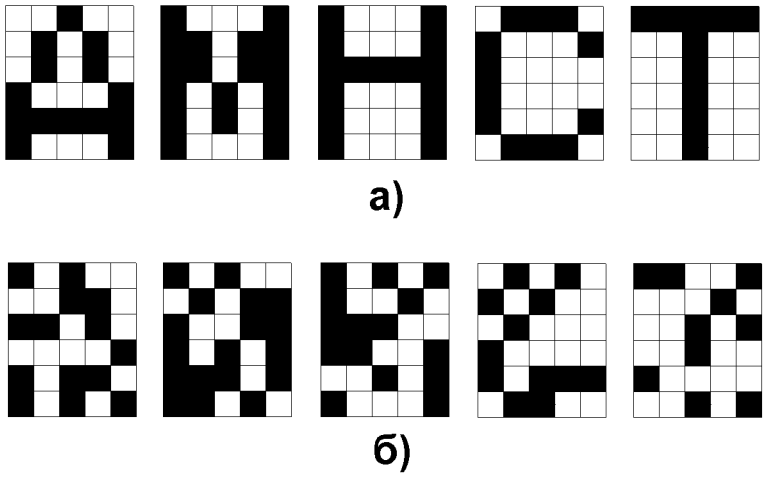

Рис. 4 Образцовые и тестовые образы

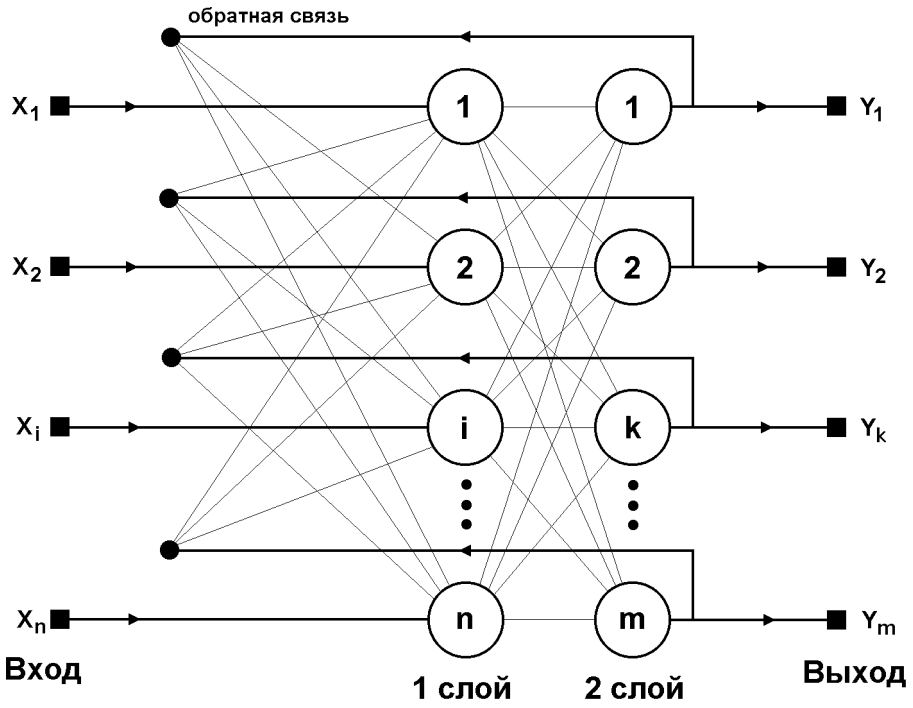

Рис.5 Структурная схема ДАП

В проект кроме файлов NEURO_HN и NEUROHAM входят также SUBFUN и NEURO_FF, описанные в [1]. Программа тестировалась в среде Borland C++ 3.1.

Предложенные классы позволяют моделировать и более крупные сети Хэмминга. Увеличение числа и сложности распознаваемых образов ограничивается фактически только объемом ОЗУ. Следует отметить, что обучение сети Хэмминга представляет самый простой алгоритм из всех рассмотренных до настоящего времени алгоритмов в этом цикле статей.

Обсуждение сетей, реализующих ассоциативную память, было бы неполным без хотя бы краткого упоминания о двунаправленной ассоциативной памяти (ДАП). Она является логичным

развитием парадигмы сети Хопфилда, к которой для этого достаточно добавить второй слой. Структура ДАП представлена на рис.5. Сеть способна запоминать пары ассоциированных друг с другом образов. Пусть пары образов записываются в виде векторов $\mathbf{X^k} = \{x_i^k: i=0...n-1\}$ и $\mathbf{Y^k} = \{y_j^k: j=0...m-1\}$, k=0...r-1, где r – число пар. Подача на вход первого слоя некоторого вектора $\mathbf{P} = \{p_i: i=0...n-1\}$ вызывает образование на входе второго слоя некоего другого вектора $\mathbf{Q} = \{q_i: j=0...m1\}$, который затем снова поступает на вход первого слоя. При каждом таком цикле вектора на выходах обоих слоев приближаются к паре образцовых векторов, первый из которых – **X** – наиболее походит на **P**, который был подан на вход сети в самом начале, а второй – **Y** – ассоциирован с ним. Ассоциации между векторами кодируются в весовой матрице $\mathbf{W^{(1)}}$ первого слоя. Весовая матрица второго слоя $\mathbf{W^{(2)}}$ равна транспонированной первой $(\mathbf{W^{(1)}})^T$. Процесс обучения, также как и в случае сети Хопфилда, заключается в предварительном расчете элементов матрицы **W** (и соответственно $\mathbf{W^T}$) по формуле:

$$w_{ij} = \sum_k x_i y_j,\ i = 0...n-1,\ j = 0...m-1$$

(11)

Эта формула является развернутой записью матричного уравнения

$$\mathbf{W} = \sum_k \mathbf{X^T Y}$$

(12)

для частного случая, когда образы записаны в виде векторов, при этом произведение двух матриц размером соответственно [n*1] и [1*m] приводит к (11).

В заключении можно сделать следующее обобщение. Сети Хопфилда, Хэмминга и ДАП позволяют просто и эффективно разрешить задачу воссоздания образов по неполной и искаженной информации. Невысокая емкость сетей (число запоминаемых образов) объясняется тем, что, сети не просто запоминают образы, а позволяют проводить их обощение, например, с помощью сети Хэмминга возможна классификация по критерию максимального правдоподобия [3]. Вместе с тем, легкость построения программных и аппаратных моделей делают эти сети привлекательными для многих применений.

Литература

1. С. Короткий, Нейронные сети: алгоритм обратного распространения.
2. С. Короткий, Нейронные сети: обучение без учителя.
3. Artificial Neural Networks: Concepts and Theory, IEEE Computer Society Press, 1992.
4. Ф.Уоссермен, Нейрокомпьютерная техника, М.,Мир, 1992.

Листинг 1

```
// FILE neuro_hn.h
#include "neuro.h"

// Hamming Net
class LayerHN;
class NetHN;

class NeuronHN: public Neuron
{
 friend LayerHN;
 friend NetHN;
public:
 virtual float Sigmoid(void);
};

class LayerHN: public LayerFF
{
                                        friend NetHN;
 NeuronHN _FAR *neurons;
public:
 LayerHN(unsigned nRang);
 ~LayerHN();
 void PrintSynapses(int,int){};
 void PrintAxons(int x, int y){};
};

class NetHN: public SomeNet
{
 LayerHN _FAR *layers;
 int Nin, Nout;
 int dx, dy, DX, DY;
 float _FAR * Class;  // [Nout]x[Nin] {+1;-1}
 unsigned char *name; // сети можно дать имя
public:
 NetHN(int N, int M)
 {
  layers = new LayerHN(M); Nin=N; Nout=M; name=NULL;
 };
 ~NetHN()
 {
  if(layers) delete layers; Nin=0; Nout=0;
layers=NULL;
 };
 LayerHN _FAR *GetLayer(void){return layers;};
 void SetClasses(float _FAR * ps) {Class=ps;};
 void Initialize(float _FAR *In);
 void Cycle(void);
 int IsConverged(void);
 int  LoadNextPattern(float _FAR *In);
 void SetDxDy(int x, int y, int _dx, int _dy)
 {if(x*y==Nin) {dx=x; dy=y;} DX=_dx; DY=_dy;};
 void SetName(unsigned char *s) {name=s;};
 void Show(void);
 void PrintAxons(int x, int y, int direction);
};

float SetSigmoidAlfaHN(float Al);
float SetLimitHN(float Al);
```

Листинг 2

```
// FILE neuro_hn.cpp FOR neuro_hn.prj
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "neuro_hn.h"

static int SigmoidType=THRESHOLD;
static float SigmoidAlfa=1.; // величина порога
static float Limit=0.9;      // eps=Limit*(1/Nout)

float SetSigmoidAlfaHN(float Al)
{
float a;
 a=SigmoidAlfa;
 SigmoidAlfa=fabs(Al);
 if(SigmoidAlfa<0.01) SigmoidAlfa=0.01;
 return a;
}

float SetLimitHN(float Al)
{
float a;
 a=Limit;
 Limit=fabs(Al);
 if(Limit>=1.) Limit=0.98;
 return a;
}

float NeuronHN::Sigmoid(void)
{
 switch(SigmoidType)
 {
  case THRESHOLD:
            if(state>SigmoidAlfa) return
SigmoidAlfa;
            else if(state<0) return 0;
            else return state;
  default:     return state;
 }
}

LayerHN::LayerHN(unsigned nRang)
{
 status=ERROR;
 if(nRang==0) return;
 neurons=new NeuronHN[nRang];
 if(neurons==NULL) return;
 rang=nRang;
 status=OK;
}

LayerHN::~LayerHN(void)
{
 if(neurons) delete [] neurons;
 neurons=NULL;
}

void NetHN::Initialize(float _FAR *In)
{
float sum;
 for(unsigned i=0;i<Nout;i++) // по классам
 {
  sum=0.;
  // расчет (7) с подстановкой (5) и (6)
  for(unsigned j=0;j<Nin;j++)
   sum+=Class[i*Nin+j] * In[j]; // число
совпадений...
                               // минус число
ошибок
//  C1=(Nin-sum)/2   - число ошибок
//  C2=Nin-C1;       - число совпадений
  sum=(Nin+sum)/2; // sum = C2(C1) - число
совпадений
  layers->neurons[i].state=sum;
  layers->neurons[i].axon=
  layers->neurons[i].Sigmoid();
 }
}

void NetHN::Cycle(void)
{
float sum;
 for(unsigned i=0;i<Nout;i++)
 {
  sum=0.;
  for(unsigned j=0;j<Nout;j++)
   if(i!=j) sum+=layers->neurons[j].axon;
  sum*=((1./Nout)*Limit);
  layers->neurons[i].state=
  layers->neurons[i].axon-sum;
  layers->neurons[i].state=  // рассчитываем
значения
  layers->neurons[i].Sigmoid(); // аксонов, но...
 }
 for(i=0;i<Nout;i++)
 layers->neurons[i].axon=
 layers->neurons[i].state; // ...обновляем их здесь
}

int NetHN::IsConverged(void)
{
int sum=0;
 for(unsigned i=0;i<Nout;i++)
 {
  if(layers->neurons[i].axon>0.) sum++;
 }
 if(sum==1) return 1;
 else return 0;
}

int NetHN::LoadNextPattern(float _FAR *IN)
{
unsigned char buf[256];
unsigned char *s, *ps;
int i;
 if(pf==NULL) return 1;

 if(imgfile) // данные расположены двумерно
 {
  for(i=0;i<dy;i++)
  {
   if(fgets(buf,256,pf)==NULL) return 2;
   for(int j=0;j<dx;j++)
   {
    if(buf[j]=='x'||buf[j]=='1') IN[i*dx+j]=1.;
    else IN[i*dx+j]=-1.;
   }
  }
  if(fgets(buf,256,pf)==NULL) return 2;
  return 0;
 }

 // данные в виде строки: 1 символ - 1 элемент
 if(fgets(buf,250,pf)==NULL) return 2;
 for(i=0;i<Nin;i++)
 {
  if(buf[i]=='0') IN[i]=-1.;
  else            IN[i]=1.;
 }
 return 0;
```

```
}

void NetHN::Show(void)
{
unsigned char sym[5]={ GRAFCHAR_EMPTYBLACK,
GRAFCHAR_DARKGRAY, GRAFCHAR_MIDDLEGRAY,
GRAFCHAR_LIGHTGRAY, GRAFCHAR_SOLIDWHITE };
int i,j,k;
float fmax=0.0;
 if(name) out_str(0,0,name,3);
 out_char(0,1,GRAFCHAR_UPPERLEFTCORNER,15);
 for(i=0;i<2*DX;i++)
  out_char(1+i,1,GRAFCHAR_HORIZONTALLINE,15);
 out_char(1+i,1,GRAFCHAR_UPPERRIGHTCORNER,15);

 for(j=0;j<DY;j++)
  for(i=0;i<DX;i++)
   if(layers->neurons[j*DX+i].axon>fmax)
    fmax=layers->neurons[j*DX+i].axon;

 for(j=0;j<DY;j++)
 {
  out_char(0,2+j,GRAFCHAR_VERTICALLINE,15);
  for(i=0;i<2*DX;i++)
  {
   if(fmax)
   {
    k=(int)(((layers->neurons[j*DX+i/2].axon)
      /fmax)*5.);
   }
   else k=0;
   if(k<0) k=0;
   if(k>=5) k=4;
   out_char(1+i, 2+j, sym[k], 15);
  }
  out_char(1+i, 2+j,GRAFCHAR_VERTICALLINE,15);
 }

 out_char(0,j+2,GRAFCHAR_BOTTOMLEFTCORNER,15);
 for(i=0;i<2*DX;i++)
  out_char(i+1,j+2,GRAFCHAR_HORIZONTALLINE,15);
 out_char(1+i,j+2,GRAFCHAR_BOTTOMRIGHTCORNER,15);
}

void NetHN::PrintAxons(int x, int y, int direction)
{
unsigned char buf[20];
 for(unsigned i=0;i<Nout;i++)
 {
  sprintf(buf,"%+7.2f ",layers->neurons[i].axon);
  out_str(x+8*i*direction,y+i*(!direction),buf,11);
 }
}
```

Листинг 3

```
// FILE neuroham.cpp FOR neuro_hn.prj
#include <conio.h>
#include <bios.h>
#include "neuro_hn.h"

#define INS 30  // число элементов во входных данных
#define OUTS 5  // число выходов (образцов)
#define TEST 5  // число тестовых (зашумленных)
образов

main()
{
int i,j,k=13;
unsigned char buf[20];
float _FAR *In;
float _FAR * cl;
 In = new float [INS]; // массив для ввода данных
 cl = new float [OUTS*INS]; // хранилище образцов
 NetHN Hn(INS,OUTS);   // создание сети

 SetLimitHN(0.5);
 SetSigmoidAlfaHN(INS); // установка размера порога

 Hn.SetDxDy(5,6,OUTS,1); // входные вектора - [5*6]
 Hn.OpenPatternFile("charh.img");
 for(i=0;i<OUTS;i++)   // загрузка образцов
 {
  Hn.LoadNextPattern(&cl[i*INS]);
 }
 Hn.SetClasses(cl);    // инициализация весов
 Hn.ClosePatternFile();

 ClearScreen();
 Hn.SetName("Hamming");
 Hn.OpenPatternFile("charhh.img");
 for(i=0;i<TEST;i++)   // цикл по тестируемым
образам
 {
  sprintf(buf,"pattern %d    ",i);
  out_str(0,10,buf,15);
  Hn.LoadNextPattern(In); // загрузка
  Hn.Initialize(In);      // инициализация входов
сети
  for(j=0;;j++)
  {
   sprintf(buf,"cycle %d     ",j);
   out_str(0,11,buf,15);
   Hn.Cycle();
   Hn.Show();
   Hn.PrintAxons(30,0,VERTICAL);
   if(kbhit() || k==13) k=getch();
   if(k==27) break; // ESC - безусловный выход
   // нажатие ENTER приведет к пошаговому просмотру,
   // любая другая клавиша задает непрерывное
   // выполнение итераций цикла вплоть до момента...
   if(Hn.IsConverged())
   {               // ...когда сеть
застабилизируется
    out_str(0,24,"Converged",15);
    k=getch();
    out_str(0,24,"         ",0);
    break;
   }
  }
 }
 end:
 Hn.ClosePatternFile();

 delete cl;
 delete In;
 return 0;
}
```

Листинг 4

Файл charh.img

```
..x..
.x.x.
.x.x.
x...x
xxxxx
x...x
A........
x...x
xx.xx
xx.xx
x.x.x
x.x.x
x...x
.M.......
x...x
x...x
xxxxx
x...x
x...x
x...x
..H......
.xxx.
x...x
x....
x....
x...x
.xxx.
......C...
xxxxx
..x..
..x..
..x..
..x..
..x..
.......T..
```

Листинг 5

Файл charhh.img

```
x.x..
..xx.
xx.x.
....x
x.xx.
x.x.x
A........
x.x..
.x.xx
x..xx
x.x.x
xxx.x
xx.x.
.M.......
x.x.x
x..x.
xxx..
xx..x
..x.x
x...x
..H......
.x.x.
x.x..
.x...
x....
x.xxx
.xx..
......C...
xx..x
...x.
```

```
..x.x
..x..
x....
..x.x
.......T..
```