Глава 4
ПАКЕТ NEURAL NETWORKS TOOLBOX

4.1. Назначение пакета Neural Networks Toolbox

Пакет Neural Networks Toolbox (нейронные сети) содержит средства для проектирования, моделирования, обучения и использования множества известных парадигм аппарата искусственных нейронных сетей (ИНС), от базовых моделей персептрона до самых современных ассоциативных и самоорганизующихся сетей. Пакет может быть использован для решения множества разнообразных задача, таких как обработка сигналов, нелинейное управление, финансовое моделирование и т. п.

Для каждого типа архитектуры и обучающего алгоритма ИНС имеются функции инициализации, обучения, адаптации, создания и моделирования, демонстрации и примеры применения.

4.2. Обзор функций пакета Neural Networks Toolbox

В состав пакета Neural Networks входят более 150 различных функций, образуя собой своеобразный макроязык программирования и позволяя пользователю создавать, обучать и использовать самые различные НС. С помощью команды

» help nnet

можно получить перечень входящих в пакет функций. Для получения справки по любой функции можно использовать команду

» help имя_функции

Данные функции по своему назначению делятся на ряд групп. Рассмотрим основные из них.

4.2.1. Функции активации (передаточные функции) и связанные с ними функции

compet(X) — функция конкуренции — в качестве аргумента использует матрицу **X**, столбцы которой ассоциируются с векторами входов. Возвращает разреженную матрицу с единичными элементами, индексы которых соответствуют индексам наибольших элементов каждого столбца.

Пример

```
» X = [0.9 -0.6;0.1 0.4;0.2 -0.5;0 0.5];
» compet(X)
ans =
   (1,1)     1
   (4,2)     1
```

В форме **compet(code)**, где переменная **code** может принимать значения **'deriv'** (имя производной функции), **'name'** (полное имя), **'output'** (диапазон выхода), **'active'** (возможный диапазон входов).

Примеры

```
» compet('deriv')
ans =
   ''
» compet('name')
ans =
Competitive
» compet('output')
ans =
     0     1

» compet('active')
ans =
   -Inf    Inf
```

Данная функция используется при создании НС со слоем «соревнующихся» нейронов (как, например, в сетях встречного распространения).

hardlim(X) — пороговая функция активации с порогом $\theta = 0$; аргумент имеет тот же смысл, что и для предыдущей команды. Возвращает матрицу, размер которой равен размеру матрицы **X**, а элементы имеют значения 0 или 1 — в зависимости от знака соответствующего элемента **X**.

Пример

```
» X = [0.9 -0.6;0.1 0.4;0.2 -0.5;0 0.5];
» hardlim(X)
ans =
   1     0
   1     1
   1     0
   1     1
```

В форме **hardlim(code)** возвращает информацию, аналогичную рассмотренной для команды **compet**.

hardlims(X) — знаковая или сигнатурная функция активации; действует так же, как функция **hardlim(X)**, но возвращает значения -1 или $+1$.

logsig(X) — сигмоидальная логистическая функция. Возвращает матрицу, элементы которой являются значениями логистической функции (см. табл. 2.1) от аргументов — элементов матрицы **X**.

poslin(X) — возвращает матрицу значений полулинейной функции (табл. 2.1).

purelin(X) — возвращает матрицу значений линейной функции активации (табл. 2.1).

radbas(X) — возвращает матрицу значений радиальной базисной функции (табл. 2.1).

satlin(X) — возвращает матрицу значений полулинейной функции с насыщением (табл. 2.1).

satlins(X) — возвращает матрицу значений линейной функции с насыщением (табл. 2.1).

softmax(X) — возвращает матрицу, элементы которой вычисляются по формуле

$$\frac{\exp(x_{ij})}{\sum\limits_{i=1}^{N}\exp(x_{ij})}$$

где N — число строк матрицы-аргумента **X**.

tansig(X) — возвращает матрицу значений сигмоидальной (гиперболический тангенс) функции (см. табл. 2.1).

tribas(X) — возвращает матрицу значений треугольной функции принадлежности (см. табл. 2.1).

dhardlim(X,Y) — производная пороговой функции активации. Аргументами являются матрица входов **X** и матрица выходов **Y**; матрицы имеют одинаковый размер. Возвращается матрица того же размера с нулевыми элементами.

dhardlms(X,Y) — производная знаковой функции активации (см. табл. 2.1). Возвращается матрица с нулевыми элементами.

dlogsig(X,Y) — производная сигмоидальной логистической функции. Возвращается матрица с элементами $y_{ij}(1-y_{ij})$.

Примеры

```
» X = [0.1; 0.8; -0.7];
» Y = logsig(X)
Y =
  0.5250
```

```
    0.6900
    0.3318
» dY_dX = dlogsig(X,Y)
dY_dX =
    0.2494
    0.2139
    0.2217
```

dposlin(X,Y) — производная полулинейной функции. Возвращается матрица с элементами, равными единице для соответственных положительных элементов матрицы-аргумента **Y** и равными нулю в противоположном случае.

dpurelin(X,Y) — производная линейной функции активации. Возвращается матрица с единичными элементами.

dradbas(X,Y) — производная радиальной базисной функции (см. табл. 2.1). Возвращается матрица с элементами $-2x_{ij}y_{ij}$.

dsatlin(X,Y) — возвращает матрицу значений производной полулинейной функции с насыщением (табл. 2.1). Элементы такой матрицы — единицы, если соответственные элементы матрицы **Y** принадлежат интервалу $(0, 1)$, и нули в противоположном случае.

dsatlins(X,Y) — возвращает матрицу значений производной линейной функции с насыщением (табл. 2.1). Элементы такой матрицы — единицы, если соответственные элементы матрицы **Y** принадлежат интервалу $(-1, 1)$, и нули в противоположном случае.

dtansig(X,Y) — возвращает матрицу значений производной сигмоидальной функции — гиперболического тангенса (табл. 2.1). Элементы этой матрицы определяются выражением $1 - y_{ij}^2$.

dtribas(X,Y) — возвращает матрицу значений производной треугольной функции активации (табл. 2.1). Элементы этой матрицы определяются выражением: 1, если $-1 < y_{ij} < 0$; -1, если $0 \leqslant y_{ij} < 1$; 0 в противоположном случае.

4.2.2. Функции обучения нейронных сетей. Позволяют устанавливать алгоритм и параметры обучения НС заданной конфигурации по желанию пользователя. В группу входит следующие функции.

[net,tr] = trainbfg(net,Pd,Tl,Ai,Q,TS,VV,TV) — функция обучения, реализующая разновидность квазиньютоновского алгоритма обратного распространения ошибки (BFGS). Аргументы функции:

net — имя обучаемой НС,

Pd — наименование массива задержанных входов обучающей выборки,

Tl — массив целевых значений выходов,

Ai — матрица начальных условий входных задержек,

Q — количество обучающих пар в одном цикле обучения (размер «пачки»),

TS — вектор временных интервалов,

VV — пустой ([]) массив или массив проверочных данных,

TV — пустой ([]) массив или массив тестовых данных.

Функция возвращает обученную нейронную сеть **net** и набор записей **tr** для каждого цикла обучения (**tr.epoch** — номер цикла, **tr.perf** — текущая ошибка обучения, **tr.vperf** — текущая ошибка для проверочной выборки, **tr.tperf** — текущая ошибка для тестовой выборки).

Процесс обучения происходит в соответствии со значениями следующих параметров (в скобках приведены значения по умолчанию):

net.trainParam.epochs (100) — заданное количество циклов обучения,

net.trainParam.show (25) — количество циклов для показа промежуточных результатов,

net.trainParam.goal (0) — целевая ошибка обучения,

net.trainParam.time (∞) — максимальное время обучения в секундах,

net.trainParam.min_grad (10^{-6}) — целевое значение градиента,

net.trainParam.max_fail (5) — максимально допустимая кратность превышения ошибки проверочной выборки по сравнению с достигнутым минимальным значением,

net.trainParam.searchFcn (**'srchcha'**) — имя используемого одномерного алгоритма оптимизации.

Структуры и размеры массивов:

Pd — No×Ni×TS— клеточный массив, каждый элемент которого P{i,j,ts} есть матрица Dij×Q,

TI — NI×TS — клеточный массив, каждый элемент которого P{i,ts} есть матрица Vi×Q,

Ai — NI×LD — клеточный массив, каждый элемент которого Ai{i,k} есть матрица Si×Q,

где

Ni = net.numInputs (количество входов сети),

NI = net.numLayers (количество ее слоев),

LD = net.numLayerDelays (количество слоев задержки),

Ri = net.inputsi.size (размер i-го входа),

Si = net.layersi.size (размер i-го слоя),

Vi = net.targetsi.size (размер целевого вектора),

Dij = Ri * length(net.inputWeightsi,j.delays).

Если массив **VV** — не пустой, то он должен иметь структуру, определяемую следующими компонентами:

VV.PD — задержанные значения входов проверочной выборки,

VV.TI — целевые значения,

VV.Ai — начальные входные условия,

VV.Q — количество проверочных пар в одном цикле обучения,

VV.TS — временные интервалы проверочной выборки.

Эти параметры используются для задания останова процесса обучения в случаях, когда ошибка для проверочной выборки не уменьшается или начинает возрастать.

Структуру, аналогичную структуре массива **VV**, имеет массив **TV**.

Рассматриваемая функция, заданная в форме **trainbfg(code)**, возвращает для значений аргумента, соответственно, **'pnames'** и **'pdefaults'**, имена параметров обучения и их значения по умолчанию.

Для использования функции в НС, определяемых пользователем, необходимо:

1) установить параметр **net.trainFcn = 'trainbfg'** (при этом параметры алгоритма будут заданы по умолчанию);

2) установить требуемые значения параметров (**net.trainParam**).

Процесс обучения останавливается в случае выполнения любого из следующих условий:

- превышено заданное количество циклов обучения (**net.trainParam.epochs**),
- превышено заданное время обучения (**net.trainParam.time**),
- ошибка обучения стала меньше заданной (**net.trainParam.goal**),
- градиент стал меньше заданного (**net.trainParam.min_grad**),
- возрастание ошибки проверочной выборки по сравнению с достигнутым минимальным превысило заданное значение (**net.trainParam.max_fail**).

Пример

```
» P = [0 1 2 3 4 5]; % Задание входного вектора
» T = [0 0 0 1 1 1]; % Задание целевого вектора
» % Создание и тестирование сети
» net = newff([0 5],[2 1],{'tansig','logsig'},'traincgf');
» a = sim(net,P)
a =
    0.0586    0.0772    0.0822    0.0870    0.1326    0.5901
» % Обучение с новыми параметрами и повторное тестирование
```

```
» net.trainParam.searchFcn = 'srchcha';
» net.trainParam.epochs = 50;
» net.trainParam.show = 10;
» net.trainParam.goal = 0.1;
» net = train(net,P,T);
TRAINCGF-srchcha, Epoch 0/50, MSE 0.295008/0.1,
Gradient 0.623241/1e-006
TRAINCGF-srchcha, Epoch 1/50, MSE 0.00824365/0.1,
Gradient 0.0173555/1e-006
TRAINCGF, Performance goal met.
» a = sim(net,P)
a =
  0.0706    0.1024    0.1474    0.9009    0.9647    0.9655
```

В данном примере созданная многослойная НС, обученная с установками по умолчанию, вначале показала плохой результат отображения обучающей выборки, но после изменения параметров и повторного обучения сети результат стал вполне приемлемым.

[net,tr] = trainbr(net,Pd,Tl,Ai,Q,TS,VV) — функция, реализующая так называемый Байесовский метод обучения, сущность которого заключается в подстройке весов и смещений сети на основе алгоритма Левенберга–Марквардта. Данный алгоритм минимизирует комбинацию квадратов ошибок и весов с выбором наилучшей такой (для получения наилучших обобщающих свойств сети). Эта процедура известна как Байесовская регуляризация, откуда следует название метода.

Аргументы, параметры, возвращаемые величины и использование — такие же, как у предыдущей функции. Сказанное остается в силе для всех остальных функций данной группы.

[net,tr] = traincgb(net,Pd,TI,Ai,Q,TS,VV) — функция обучения НС, реализующая разновидность алгоритма сопряженных градиентов (так называемый метод Powell–Beale).

[net,tr] = traincgf(net,Pd,TI,Ai,Q,TS,VV) — функция обучения НС, реализующая разновидность алгоритма обратного распространения ошибки в сочетании с методом оптимизации Флетчера–Поуэлла.

[net,tr] = traincgp(net,Pd,TI,Ai,Q,TS,VV) — то же, что в предыдущем случае, но с использованием метода Polak–Ribiere.

[net,tr] = traingd(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая «классический» алгоритм обратного распространения ошибки.

[net,tr] = traingda(net,Pd,TI,Ai,Q,TS,VV) — то же, что в предыдущем случае, но с адаптацией коэффициента скорости обучения.

[net,tr] = traingdm(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая модифицированный алгоритм обратного распространения ошибки с введенной «инерционностью» коррекции весов и смещений.

[net,tr] = traingdx(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая комбинированный алгоритм обучения, объединяющий особенности двух вышеприведенных.

[net,tr] = trainlm(net,Pd,TI,Ai,Q,TS,VV) — данная функция возвращает веса и смещения НС, используя алгоритм оптимизации Левенберга–Марквардта.

[net,tr] = trainoss(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая разновидность алгоритма обратного распространения ошибки с использованием метода секущих.

[net,tr] = trainrp(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая разновидность алгоритма обратного распространения ошибки, так называемый упругий алгоритм обратного распространения (resilient backpropagation algorithm, RPROP).

[net,tr] = trainscg(net,Pd,TI,Ai,Q,TS,VV) — данная функция возвращает веса и смещения НС, используя алгоритм масштабируемых сопряженных градиентов.

[net,tr] = trainwb(net,Pd,TI,Ai,Q,TS,VV) — данная функция корректирует веса и смещения сети в соответствии с заданной функцией обучения нейронов.

[net,tr] = trainwb1(net,Pd,TI,Ai,Q,TS,VV) — то же, что и предыдущая функция, но одновременно на вход сети предъявляется только один вектор входа.

[net,Ac,El] = adaptwb(net,Pd,TI,Ai,Q,TS) — функция адаптации весов и смещений НС. Используется совместно с функциями (см. ниже) **newp** и **newlin**. Возвращает массив выходов слоев **Ac** и массив ошибок слоев **EI**.

4.2.3. Функции настройки слоев нейронов. Функции данной группы являются вспомогательными при работе с некоторыми рассмотренными функциями обучения НС (например, **trainwb**, **trainwb1**, **adaptwb**), а также используются при настройках однослойных нейросетевых структур (персептронов, слоев Кохонена и т. п.).

[dB,LS] = learncon(B,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция настройки весов с введением «чувства справедливости» (см. выше). Аргументы:

B — S×1 вектор смещений,

P — 1×Q входной вектор,

Z — S×Q матрица взвешенных входов,

N — S×Q матрица входов,
A — S×Q матрица выходных векторов,
T — S×Q матрица целевых векторов слоя,
E — S×Q матрица ошибок,
gW — S×R градиент критерия эффективности по отношению к вектору весов,
gA — S×Q градиент критерия эффективности по отношению к вектору выхода,
D — S×S матрица расстояний между нейронами,
LP — параметр обучения, LP = [],
LS — состояние обучения, в начале — [].
Возвращаемые величины:
dB — S×1 вектор изменений весов (или смещений),
LS — новое состояние обучения.
Функция в форме **learncon(code)** возвращает следующую информацию:
при аргументе **'pnames'** — имена параметров обучения,
при **'pdefaults'** — значения параметров по умолчанию,
при **'needg'** — 1, если эта функция использует **gW** или **gA**.
Алгоритм выполнения функции сначала вычисляет «чувство справедливости» нейрона по выражению **c = (1-lr)*c+lr*a**, а уже затем корректирует вес в соответствии с формулой
b = exp(1-log(c))-b.

Пример

```
» a = rand(3,1);
» b = rand(3,1);
» lp.lr = 0.5; % Задание параметра обучения
» dW = learncon(b,[],[],[],a,[],[],[],[],[],lp,[])
dW =
  0.3449
  0.7657
  0.5405
```

learngd — функция коррекции весов и смещений, реализующая градиентный алгоритм оптимизации.
Запись:
[dW,LS] = learngd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,LS] = learngd(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learngd(code)

Описание. Аргументы функции: **W** — матрица весов или вектор смещения, остальные аргументы — как у предыдущей функции.
Возвращаемые параметры — как у предыдущей функции.

learngdm — функция практически аналогична предыдущей, но используемый алгоритм оптимизации — градиентный метод с инерционной составляющей.

[dW,LS] = learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция коррекции весов, использующая правило Хебба, в соответствии с которым веса корректируются по выражению **dw =lr*a*p′**.

[dW,LS] = learnhd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция реализует модификацию правила Хебба, при котором корректировка весов осуществляется по соотношению: **dw = lr*a*p′-dr*w**.

[dW,LS] = learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция подстройки весов «входной звезды» (нейрона слоя Гроссберга), реализующая выражение: **dw = lr*a*(p′-w)**.

[dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция подстройки весов слоя Кохонена, реализующая выражение **dw = = lr*(p′-w)**, если $\mathbf{a} \neq \mathbf{0}$, и **0** в противоположном случае.

[dW,LS] = learnlv1(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) и

[dW,LS] = learnlv2(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функции настройки сетей встречного распространения.

[dW,LS] = learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция настройки нейрона типа «выходная звезда», реализующая выражение: **dw = lr*(a-w)*p′**.

[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция, реализующая алгоритм обучения персептрона.

[dW,LS] = learnpn(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — то же, что и предыдущая функция, но с нормализацией входов. Более эффективна при больших изменениях входных сигналов.

[dW,LS] = learnsom(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция обучения самоорганизующихся карт.

[dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция обучения, реализующая так называемый алгоритм Видрова–Хоффа (Widrow–Hoff), основанный на соотношении

dw = lr*e*pn′ и известный также, как дельта-правило или правило наименьших квадратов.

4.2.4. Функции одномерной оптимизации. Функции данной группы можно рассматривать как вспомогательные для функций обучения нейронных сетей. Реализуют различные алгоритмы одномерного поиска.

srchbac — функция реализует так называемый алгоритм перебора с возвратами (backtracking).

srchbre — функция реализует комбинированный метод оптимизации, объединяющий метод золотого сечения и квадратичную интерполяцию.

srchcha — функция реализует разновидность метода оптимизации с применением кубической интерполяции.

srchgol — функция реализует метод золотого сечения.

srchhyb — функция реализует комбинированный метод оптимизации, объединяющий метод дихотомии и кубическую интерполяцию.

4.2.5. Функции инициализации слоев и смещений. Для многих нейронных сетей этапом, предваряющим процедуру их обучения, является этап инициализации (задания некоторых — обычно выбираемых случайным образом) весов и смещений сети. Такая инициализация выполняется с помощью функций данной группы.

initcon(S,PR) — функция, устанавливающая смещения нейронов в зависимости от среднего выхода нейрона. Аргументы: **s** — количество нейронов, **PR = [Pmin Pmax]** — матрица (с двумя столбцами) минимальных и максимальных значений входов, по умолчанию [1 1]. Возвращает вектор смещений. Используется совместно с командой **learncon**.

Пример

```
» b = initcon(3)
b =
  8.1548
  8.1548
  8.1548
```

initzero — функция задания нулевых начальных значений весам или смещениям. Аргументы те же, что и у предыдущей команды.

midpoint(S,PR) — функция инициализации, устанавливающая веса в соответствии со средними значениями входов.

randnc(S,R) — функция задания матрицы весов. Возвращает матрицу размера **S**×**R** со случайными элементами, нормализованную по столбцам (векторы-столбцы имеют единичную длину).

randnr(S,R) — то же, что предыдущая функция, но возвращает матрицу весов, нормализованную по строкам.

rands — функция инициализации весов/смещений заданием их случайных значений из диапазона $[-1, 1]$.

Запись:

W = rands(S,PR)

M = rands(S,R)

v = rands(S)

Описание. Аргументы те же, что и для функции **initcon**; значение R по умолчанию -1. Возвращается матрица соответствующего размера.

4.2.6. Функции создания нейронных сетей

network — функция создания нейронной сети пользователя.

Запись:

net = network

net =

network(numInputs,numLayers,biasConnect,inputConnect, layerConnect,outputConnect,targetConnect)

Описание. Функция возвращает созданную нейронную сеть с именем **net** и со следующими характеристиками (в скобках даны значения по умолчанию):

numInputs — количество входов (0).

numLayers — количество слоев (0),

biasConnect — булевский вектор с числом элементов, равным количеству слоев (нули),

inputConnect — булевская матрица с числом строк, равным количеству слоев, и числом столбцов, равным количеству входов (нули),

layerConnect — булевская матрица с числом строк и столбцов, равным количеству слоев (нули),

outputConnect — булевский вектор-строка с числом элементов, равным количеству слоев (нули).

targetConnect — вектор-строка, такая же, как предыдущая (нули).

net = newc(PR,S,KLR,CLR) — функция создания слоя Кохонена. Функция использует аргументы:

PR — R×2 матрицу минимальных и максимальных значений для R входных элементов,

S — число нейронов,

KLR — коэффициент обучения Кохонена (по умолчанию 0.01),

CLR — коэффициент «справедливости» (по умолчанию 0.001)

и возвращает слой Кохонена с заданным именем.

net = newcf(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) — функция создания разновидности многослойной НС с обратным распространением ошибки — так называемой каскадной НС. Такая сеть содержит скрытых NI слоев, использует входные функции типа **dotprod** и **netsum**, инициализация сети осуществляется функцией **initnw**.

Аргументы функции:

PR — R×2 матрица минимальных и максимальных значений R входных элементов,

Si — размер i-го скрытого слоя, для NI слоев,

TFi — функция активации нейронов i-го слоя, по умолчанию **'tansig'**.

BTF — функция обучения сети, по умолчанию **'traingd'**.

BLF — функция настройки весов и смещений, по умолчанию **'learngdm'**.

PF — функция ошибки, по умолчанию **'mse'**.

Пример

```
» P = [0 1 2 3 4 5 6 7 8 9 10];
» T = [0 1 2 3 4 3 2 1 2 3 4];
» net = newcf([0 10],[5 1],'tansig' 'purelin'); % Создание новой сети
» net.trainParam.epochs = 50; % Задание количества циклов обучения
» net = train(net,P,T); % Обучение НС
TRAINLM, Epoch 0/50, MSE 7.77493/0, Gradient 138.282/1e-010
TRAINLM, Epoch 25/50, MSE 4.01014e-010/0, Gradient 0.00028557/1e-010
TRAINLM, Epoch 50/50, MSE 1.13636e-011/0, Gradient 1.76513e-006/1e-010
TRAINLM, Maximum epoch reached, performance goal was not met.
» Y = sim(net,P); % Использование НС
» plot(P,T,P,Y,'okv') % Графическая иллюстрация работы сети
```

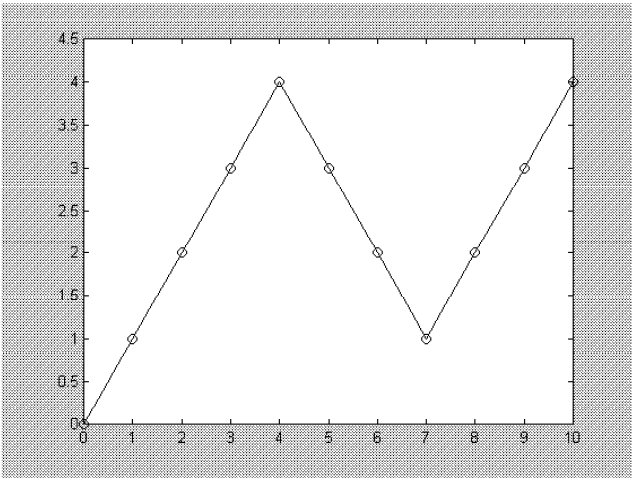

Рис. 4.1. Иллюстрация работы сети

На рис. 4.1 точками отображены элементы обучающей выборки, линией — выход сети.

net = newelm(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) – функция создания сети Элмана. Аргументы — такие же, как и у предыдущей функции.

net = newff(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) — функция создания «классической» многослойной НС с обучением по методу обратного распространения ошибки.

net = newfftd(PR,ID,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF, PF) — то же, что и предыдущая функция, но с наличием задержек по входам. Дополнительный аргумент **ID** — вектор входных задержек.

net = newgrnn(P,T,spread) — функция создания обобщенно-регрессионной сети. Аргументы:

P — R×Q матрица Q входных векторов,

T — S×Q матрица Q целевых векторов,

spread — отклонение (по умолчанию 1.0).

net = newhop(T) — функция создания сети Хопфилда. Использует только один аргумент.

T — R×Q матрица Q целевых векторов (значения элементов должны быть +1 или −1).

net = newlin(PR,S,ID,LR) — функция создания слоя линейных нейронов. Аргументы:

PR — R×2 матрица минимальных и максимальных значений для R входных элементов,

S — число элементов в выходном векторе,

ID — вектор входной задержки (по умолчанию [0]),

LR — коэффициент обучения (по умолчанию 0.01).

Возвращается новый линейный слой.

При записи в форме **net = newlin(PR,S,0,P)** используется аргумент

P — матрица входных векторов,

возвращается линейный слой с максимально возможным коэффициентом обучения при заданной **P**.

net = newlind(P,T) — функция проектирования нового линейного слоя. Данная функция по матрицам входных и выходных векторов методом наименьших квадратов определяет веса и смещения линейной НС.

net = newlvq(PR,S1,PC,LR,LF) — функция создания сети встречного распространения.

Аргументы:

PR — R×2 матрица минимальных и максимальных значений R входных элементов,

S1 — число скрытых нейронов,

PC — S2 элементов вектора, задающих доли принадлежности к различным классам,

LR — коэффициент обучения, по умолчанию 0.01,

LF — функция обучения, по умолчанию **'learnlv2'**.

net = newp(PR,S,TF,LF) — функция создания персептрона. Аргументы:

PR — R×2 матрица минимальных и максимальных значений R входных элементов,

S — число нейронов,

TF — функция активации, по умолчанию **'hardlim'**,

LF — функция обучения, по умолчанию **'learnp'**.

net = newpnn(P,T,spread) — функция создания вероятностной НС. Аргументы — как у функции **newgrnn**.

net = newrb(P,T,goal,spread) — функция создания сети с радиальными базисными элементами. Аргументы **P**, **T**, **spread** — такие же, как у функции **newgrnn**; аргумент **goal** — заданная среднеквадратичная ошибка.

net = newrbe(P,T,spread) — функция создания сети с радиальными базисными элементами с нулевой ошибкой на обучающей выборке.

net = newsom(PR,[D1,D2,...],TFCN,DFCN,OLR,OSTEPS,TLR, TND) — функция создания самообучающейся карты с аргументами:

PR — R×2 матрица минимальных и максимальных значений R входных элементов,

I — размеры i-го слоя, по умолчанию [5 8],

TFCN — топологическая функция, по умолчанию **'hextop'**,

DFCN — функция расстояния, по умолчанию **'linkdist'**,

OLR — коэффициент обучения фазы упорядочивания, по умолчанию 0.9,

OSTEPS — число шагов фазы упорядочивания, по умолчанию 1000,

TLR — коэффициент обучения фазы настройки, по умолчанию 0.02,

TND — расстояние для фазы настройки, по умолчанию 1.

4.2.7. Функции преобразования входов сети. Функции данной группы преобразуют значения входов с использованием операций умножения или суммирования.

netprod(Z1,Z2,...) — возвращает матрицу, элементы которой определяются как произведения элементов входных векторов и смещений. Аргументы **Z1,Z2,...** — матрицы, чьи столбцы ассоциированы с входами или смещениями.

Примеры

```
» z1 = [1 2 4;3 4 1];
» z2 = [-1 2 2; -5 -6 1];
» n = netprod(z1,z2)
n =
  -1     4     8
  -15    -24    1
» b = [0; -1];
» n = netprod(z1,z2,concur(b,3)) % Функция concur(b,3) создает 3 копии вектора смещения
n =
   0     0     0
   15    24    -1
```

netsum(Z1,Z2,...) — то же, что в предыдущем случае, но вместо умножения используется суммирование.

dnetprod(Z,N) — возвращает матрицу значений первой производной входов, преобразованных функцией **N = netprod(Z1,Z2,...)**.

Пример

```
» Z1 = [0; 1; -1];
» Z2 = [1; 0.5; 1.2];
» N = netprod(Z1,Z2)
N =
        0
   0.5000
  -1.2000
» dN_dZ2 = dnetprod(Z2,N)
dN_dZ2 =
   0
   1
  -1
```

dnetsum(Z,N) — то же, что и в предыдущем случае, но по отношению к функции **netsum(Z1,Z2,...)**.

4.2.8. Функции весов и расстояний

boxdist(pos) — функция определения box-расстояния между нейронами в слое. Имеет один аргумент **pos**-матрицу размера N×S, элементы которой определяют координаты нейронов, возвращает матрицу размера S×S расстояний. Расстояния (элементы возвращаемой матрицы) вычисляются по выражению:

$D_{ij} = max(abs(P_i - P_j))$, где P_i и P_j — векторы, содержащие координаты нейронов i и j.

Пример

```
» pos = rand(3,4) % Случайное размещение 4-х нейронов в 3-мерном пространстве (генерация случайной матрицы 3×4)
pos =
  0.8600   0.4966   0.6449   0.3420
  0.8537   0.8998   0.8180   0.2897
  0.5936   0.8216   0.6602   0.3412
» d = boxdist(pos)
d =
       0   0.3635   0.2151   0.5639
  0.3635        0   0.1614   0.6100
  0.2151   0.1614        0   0.5282
  0.5639   0.6100   0.5282        0
```

dist — функция вычисления евклидова расстояния.

Запись:

Z = dist(W,P) — возвращает матрицу, элементы которой являются евклидовыми расстояниями между строками (векторами) матрицы **W** и столбцами матрицы **P** (матрицы должны иметь соответствующие размеры).

D = dist(pos) — в такой форме функция аналогична функции **boxdist(pos)** за тем исключением, что возвращается матрица евклидовых расстояний.

negdist(W,P) — функция идентична предыдущей, но элементы возвращаемой матрицы являются евклидовыми расстояниями, взятыми со знаком минус.

mandist(W,P) — функция аналогична предыдущей, но элементы возвращаемой матрицы являются расстояниями по Манхэттену, которое для векторов **x** и **y** определяется соотношением: D = sum(abs(**x** − **y**)).

linkdist(pos) — функция определения линейного расстояния между нейронами в слое. Аналогична функции **boxdist**, отличаясь от последней алгоритмом определения расстояния:

$D_{ij} = 0$, если $i = j$;

$D_{ij} = 1$, если евклидово расстояние между P_i и P_j меньше или равно 1;

$Dij = 2$, если существует k такое, что $D_{ik} = D_{kj} = 1$;

$D_{ij} = 3$, если существуют k1 и k2 такие, что $D_{ik\mathbf{1}} = D_{k\mathbf{1}k\mathbf{2}} = D_{k\mathbf{2}j} = 1$;

$D_{ij} = N$, если существуют k1, k2, ..., kN такие, что $D_{ik\mathbf{1}} = D_{k\mathbf{1}k\mathbf{2}} = \ldots = D_{kNj} = 1$;

$D_{ij} = S$, если не выполнено ни одно из предыдущих условий.

dotprod(W,P) — функция придания входам **P** некоторых весов **W**. Возвращает матрицу **Z = W*P**.

normprod(W,P) — функция аналогична предыдущий, но каждый элемент возвращаемой матрицы дополнительно делится на сумму элементов соответствующего столбца-сомножителя матрицы **P**.

4.2.9. Функции размещения нейронов (топологические функции). Функции данной группы используются при создании самоорганизующихся карт.

gridtop(dim1,dim2,...,dimN) — функция размещения N слоев нейронов в узлах регулярной прямоугольной N-мерной решетки. **dim1,dim2,...,dimN** — число нейронов в слоях. Возвращает матрицу, содержащую N строк и (**dim1**×**dim2**×...×**dimN**) столбцов с координатами нейронов.

Пример

```
» pos = gridtop(2,3)
pos =
  0   1   0   1   0   1
  0   0   1   1   2   2
```

hextop(dim1,dim2,...,dimN) — функция аналогична предыдущей, но размещение нейронов производится в узлах гексагональной (шестиугольной) решетки.

Пример

```
» pos = hextop(8,5); plotsom(pos)
```

(см. рис. 4.2).

randtop(dim1,dim2,...,dimN) — аналогична функции **gridtop(dim1,dim2,...,dimN)**, но координаты нейронов выбираются случайным образом.

Пример

```
pos = randtop(16,12); plotsom(pos)
```

(см. рис. 4.3).

4.2.10. Функции использования нейронных сетей

[Y,Pf,Af] = sim(net,P,Pi,Ai) — функция, моделирующая работу нейронной сети. Аргументы: **net** — имя сети, **P** — ее входы, **Pi** — массив начальных условий входных задержек (по умолчанию они нулевые), **Ai** — массив начальных условий задержек слоя нейронов (по умолчанию они нулевые). Функция возвращает значения выходов **Y** и массивы конечных условий задержек.

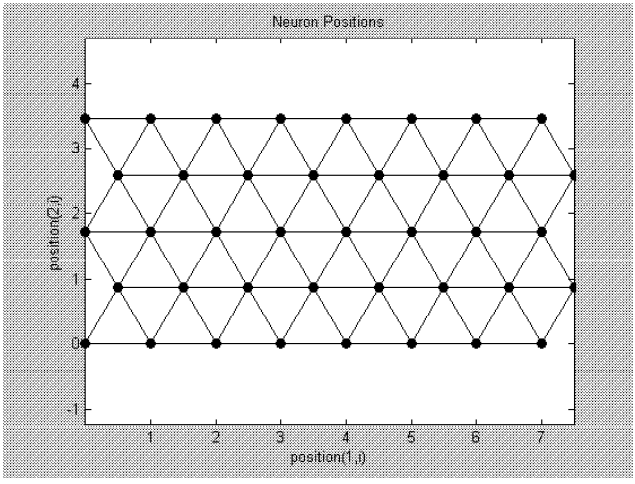

Рис. 4.2. Результат выполнения команды **hextop(8,5)**

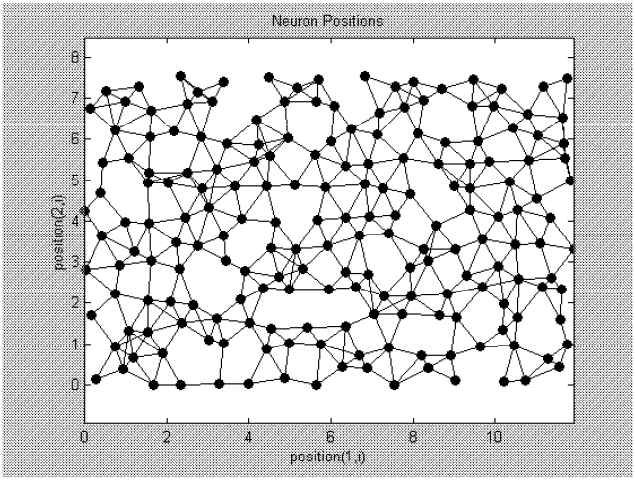

Рис. 4.3. Результат выполнения команды **randtop(16,12)**

Аргументы **Pi,Ai, Pf,Af** используются только в случаях, когда сеть имеет задержки по входам или по слоям нейронов.

Структура данных аргументов:

P — массив размера Ni×TS, каждый элемент которого P{i,ts} является матрицей размера Ri×Q.

Pi — массив размера Ni×ID, каждый элемент которого Pi{i,k} (*i*-й вход в момент ts = k − ID) является матрицей размера Ri×Q (по умолчанию — ноль).

Ai — массив размера Nl×LD, каждый элемент которого Ai{i,k} (выход *i*-го слоя в момент ts = k−LD) является матрицей размера Si×Q (по умолчанию — ноль).

Y — массив размера NO×TS, каждый элемент которого Y{i,ts} является матрицей размера Ui×Q.

Pf — массив размера Ni×ID, каждый элемент которого Pf{i,k} (*i*-й вход в момент ts = TS + k − ID) является матрицей размера Ri×Q.

Af — массив размера NI×LD, каждый элемент которого Af{i,k} (выход *i*-го слоя в момент ts = TS + k − LD) является матрицей размера Si×Q, при этом

Ni = net.numInputs — количество входов сети,

Nl = net.numLayers — количество ее слоев,

No = net.numOutputs — количество выходов сети,

ID = net.numInputDelays — входные задержки,

LD = net.numLayerDelays — задержки слоя,

TS = Number of time steps — число временных интервалов,

Q = Batch size — размер набора подаваемых векторов,

Ri = net.inputs{ i }.size — размер *i*-го вектора входа,

Si = net.layers{ i }.size — размер *i*-го слоя,

Ui = net.outputs{ i }.size — размер *i*-го вектора выхода.

net = init(net) функция инициализирует нейронную сеть с именем **net**, устанавливая веса и смещения сети в соответствии с установками **net.initFcn** и **net.initParam**.

[net,Y,E,Pf,Af] = adapt(net,P,T,Pi,Ai) — функция адаптации НС. Выполняет адаптацию сети в соответствии с установками **net.adaptFcn** и **net.adaptParam**. Здесь **E** — ошибки сети, **T** — целевые значения выходов (по умолчанию — ноль); остальные аргументы — как у команды **sim**.

[net,tr] = train(net,P,T,Pi,Ai) — функция осуществляет обучение НС в соответствии с установками **net.trainFcn** и **net.trainParam**. Здесь **tr** — информация о выполнении процесса обучения (количество циклов и соответствующая ошибка обучения).

disp(net) — функция возвращает развернутую информацию о структуре и свойствах НС.

Пример

```
» net = newp([-1 1; 0 2],3); % Создание НС типа персептрон
» disp(net)
  Neural Network object:
  architecture:
      numInputs: 1
      numLayers: 1
      biasConnect: [1]
      inputConnect: [1]
      layerConnect: [0]
      outputConnect: [1]
      targetConnect: [1]
      numOutputs: 1 (read-only)
      numTargets: 1 (read-only)
      numInputDelays: 0 (read-only)
      numLayerDelays: 0 (read-only)
  subobject structures:
      inputs: {1×1 cell} of inputs
      layers: {1×1 cell} of layers
      outputs: {1×1 cell} containing 1 output
      targets: {1×1 cell} containing 1 target
      biases: {1×1 cell} containing 1 bias
      inputWeights: {1×1 cell} containing 1 input weight
      layerWeights: {1×1 cell} containing no layer weights
  functions:
      adaptFcn: 'adaptwb'
      initFcn: 'initlay'
      performFcn: 'mae'
      trainFcn: 'trainwb'
  parameters:
adaptParam: .passes
      initParam: (none)
      performParam: (none)
      trainParam: .epochs, .goal, .max_fail, .show,
          .time
  weight and bias values:
      IW: {1×1 cell} containing 1 input weight matrix
      LW: {1×1 cell} containing no layer weight matrices
      b: {1×1 cell} containing 1 bias vector
  other:
      userdata: (user stuff)
```

display(net) — то же, что предыдущая команда, но дополнительно возвращает имя нейронной сети.

4.2.11. Графические функции

hintonw(W,maxw,minw) — функция возвращает так называемый хинтоновский график матрицы весов, при котором каждый весовой коэффициент отображается квадратом с площадью, пропорциональной величине данного коэффициента. Знак отображается цветом квадрата.

Аргументы:

W — матрица весов, **maxw** и **minw** — минимальное и максимальные значения ее коэффициентов (могут не задаваться).

Пример

```
» W = rands(2,3)
W =
  -0.8842    0.6263   -0.7222
  -0.2943   -0.9803   -0.5945
» hintonw(W)
```

(см. рис. 4.4).

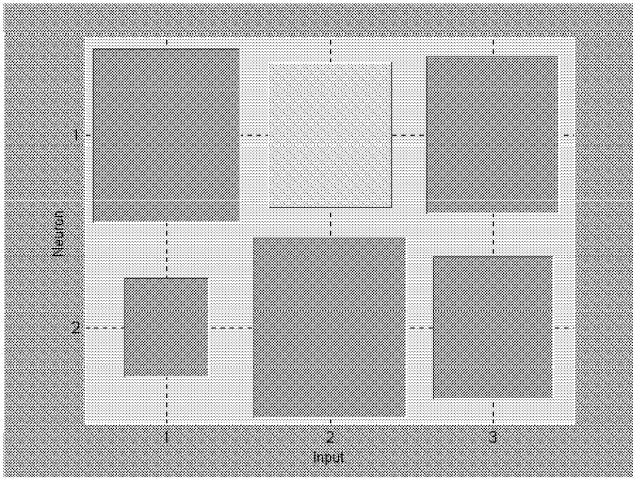

Рис. 4.4. Иллюстрация к выполнению функции **hintonw**

hintonwbm(W,b,maxw,minw) — то же, что и предыдущая функция, но на графике отображаются не только веса, но и смещения.

plotbr(tr,name,epoch) — функция возвращает графики изменения критерия качества НС в процессе обучения при использовании Байесовского метода (см. выше).

Аргументы:

tr — запись процесса обучения, **name** — имя НС, **epoch** — количество циклов обучения (по умолчанию — длина записи обучения).

Пример

```
» p = [-1:.05:1];
» t = sin(2*pi*p)+0.1*randn(size(p));
» net = newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
% Создание новой сети
» [net,tr] = train(net,p,t); % Обучение сети
TRAINBR, Epoch 0/100, SSE 228.933/0, SSW 21461.7,
Grad 2.33e+002/1.00e-010, #Par 6.10e+001/61
TRAINBR, Epoch 25/100, SSE 0.235423/0, SSW 211.044,
Grad 9.43e-002/1.00e-010, #Par 1.35e+001/61
TRAINBR, Epoch 50/100, SSE 0.240881/0, SSW 121.647,
Grad 1.87e-001/1.00e-010, #Par 1.23e+001/61
TRAINBR, Epoch 75/100, SSE 0.239867/0, SSW 116.884,
Grad 1.62e-002/1.00e-010, #Par 1.22e+001/61
TRAINBR, Epoch 100/100, SSE 0.239762/0, SSW 116.871,
Grad 9.60e-003/1.00e-010, #Par 1.22e+001/61
TRAINBR, Maximum epoch reached.
» plotbr(tr)
```

(см. рис. 4.5).

plotep(w,b,e) — функция отображает позиции весов и смещений на поверхности ошибки НС.

Аргументы:

w, b, e — соответственно, матрицы весов, смещений и ошибок. Возвращается вектор, используемый для продолжения графика, созданного функцией **plotes** (см. ниже).

plotes(wv,bv,e,v) — функция возвращает график поверхности ошибки одновходового нейрона.

Аргументы:

wv, bv — соответственно, наборы значений веса и смещения нейрона, **e** — матрица значений ошибки, **v** — опция вида изображения (по умолчанию, $[-37.5, 30]$). Использование функции иллюстрирует рис. 4.8 (см. ниже).

plotpc(W,b) — функция возвращает график линии решения для персептрона.

Аргументы:

W — матрица весов, **b** — вектор смещений. Используется совместно с функцией **plotpv** (см. ниже).

plotperf(tr,goal,name,epoch) — возвращает график изменения критерия качества НС в процессе обучения. Аргументы: **tr** — за-

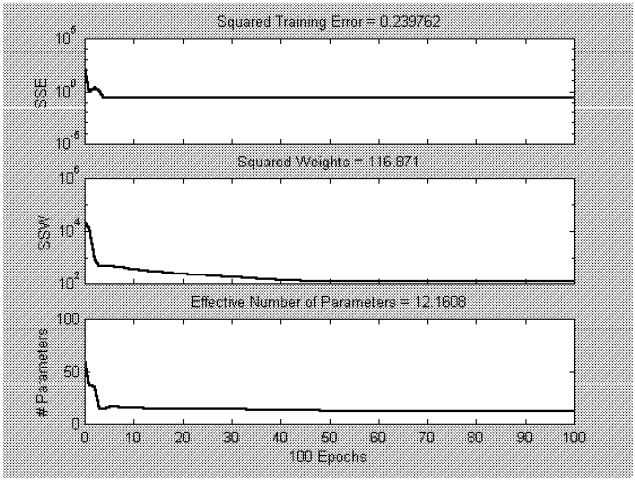

Рис. 4.5. Результат выполнения функции **plotbr(tr)**

пись процесса обучения, **goal** — целевое значение критерия, **name** — имя НС, **epoch** — количество циклов обучения.

plotpv(p,t) — функция возвращает графическое отображение входных **p** и целевых **t** векторов персептрона.

Пример

```
» p = [0 0 1 1; 0 1 0 1];
» t = [0 0 0 1];
» plotpv(p,t)
```

(см. рис. 4.6).

plotsom(pos) — функция возвращает графическое представление расположения нейронов в самоорганизующихся картах (см. **Функции размещения нейронов**, рис. 4.2, 4.3).

plotv(M,t) — функция графического изображения векторов.

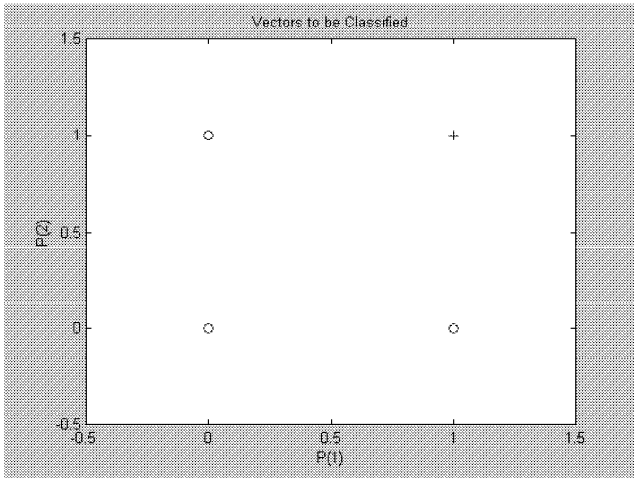

Рис. 4.6. Результат использования функции **plotpv(p,t)**

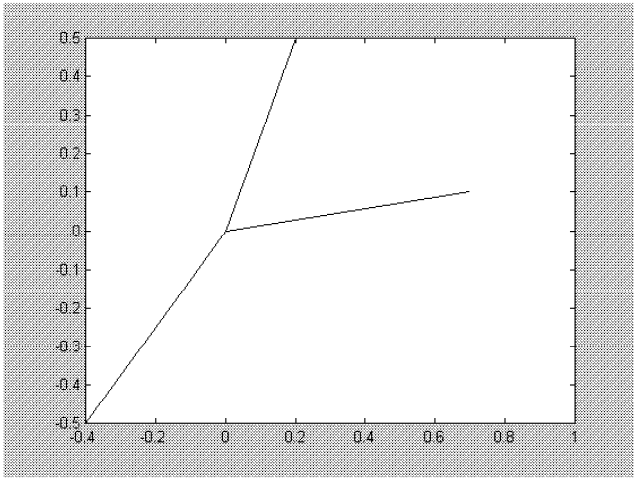

Рис. 4.7. Результат использования функции **plotv(M,t)**

Аргументы:

M — матрица с двумя строками, столбцы которой ассоциированы с векторами, **t** — опция, задающая тип линии.

Пример

```
» plotv([-.4 0.7 .2; -0.5 .1 0.5],'-')
```

(см. рис. 4.7).

plotvec(M,C,m) — функция графического изображения векторов разными цветами.

Аргументы:

M — матрица с двумя строками, столбцы которой ассоциированы с векторами, **C** — строка задания цветов, **m** — тип точек, указывающих концы векторов (по умолчанию +).

4.2.12. Прочие функции

errsurf(p,t,wv,bv,f) — функция, возвращающая матрицу значений поверхности ошибок нейрона с одним входом и одним выходом в зависимости от значений веса и смещения. Аргументы:

p — вектор значений входа,.

t — вектор значений выхода,.

wv — набор значений веса нейрона,.

bv — набор значений смещения.

f — название реализуемой функции активации (строка).

Размер возвращаемой матрицы = (количество значений **bv**)× ×(количество значений **wv**).

Пример

```
» p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1];
» t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
» wv = -1:.1:1; bv = -2.5:.25:2.5;
» es = errsurf(p,t,wv,bv,'logsig');
» plotes(wv,bv,es,[60 30])
```

(см. рис. 4.8)

maxlinlr(P) — возвращает максимальную величину коэффициента обучения для линейного слоя нейронов. Здесь **P** — матрица входов.

При записи в форме **maxlinlr(P,'bias')** функция возвращает максимальную величину коэффициента обучения для линейного слоя нейронов со смещением.

gensim(net,st) — функция генерирует нейросетевой блок Simulink (см. рис. 4.9) для последующего моделирования НС средствами этого пакета.

Пример

```
» net = newff([0 1],[5 1]); % Создание новой НС
» gensim(net)
```

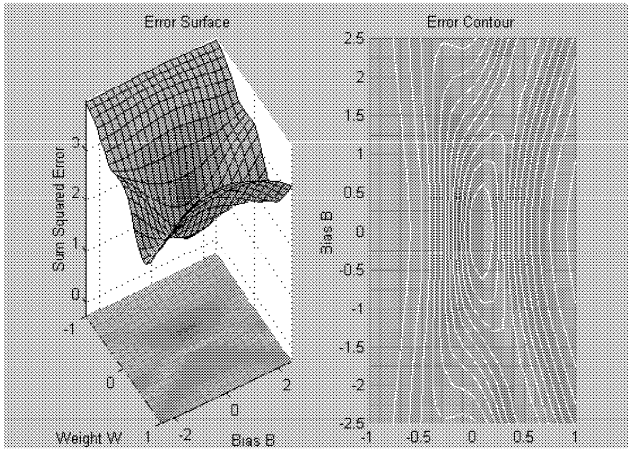

Рис. 4.8. Иллюстрация к примеру выполнения функции **errsurf**

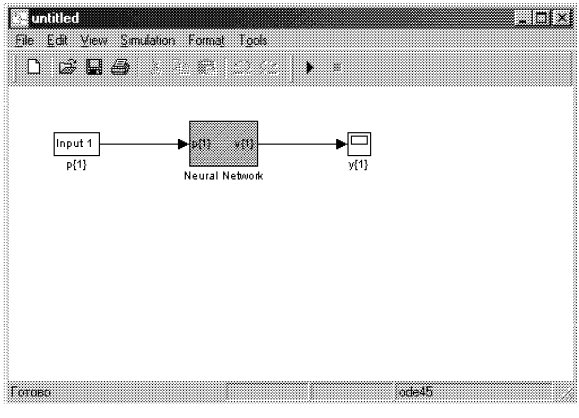

Рис. 4.9. Результат выполнения функции **gensim**

initlay(net) — функция инициализации слоев нейронной сети. В качестве аргумента использует имя (идентификатор) **net** НС. Возвращает нейронную сеть, слои нейронов в которой инициализированы в соответствии с функцией **net.layers{ i }.initFcn**. В форме **initlay(code)**, где строковая переменная code может принимать значения **'pnames'** или **'pdefaults'** функция возвращает информацию о именах или о значениях по умолчанию параметров инициализации.

initnw(net,i) — функция инициализации слоя i. Возвращает НС, веса и смещения в i-м слое которой обновлены в соответствии с алгоритмом инициализации Nguyen-Widrow (так, что зоны «влияния» каждого нейрона в слое распределены равномерно).

initwb(net,i) — почти то же, что в предыдущем случае, но веса и смещения i-го слоя инициализируются в соответствии с их собственными функциями инициализации.

ddotprod — функция определения производной от результата **Z** умножения матрицы весов **W** на матрицу входов **P**.

Запись:

dZ_dP = ddotprod('p',W,P,Z)
dZ_dW = ddotprod('w',W,P,Z)

Примеры

```
» W = [0 -1 0.2; -1.1 1 0];
» P = [0.1; 0.6; -0.2];
» Z = dotprod(W,P) % Вычисление произведения Z=W*P
Z =
  -0.6400
   0.4900
» dZ_dP = ddotprod('p',W,P,Z)
dZ_dP =
        0   -1.0000    0.2000
  -1.1000    1.0000         0
» dZ_dW = ddotprod('w',W,P,Z)
dZ_dW =
   0.1000
   0.6000
  -0.2000
```

4.3. Примеры создания и использования нейронных сетей

4.3.1. Нейронные сети для аппроксимации функций. Создадим обобщенно-регрессионную НС с именем **a** для аппроксимации функции вида

$y = x^2$ на отрезке $[-1,\ 1]$, используя следующие экспериментальные данные:

$x = [-1\ \ -0.8\ \ -0.5\ \ -0.2\ 0\ 0.1\ 0.3\ 0.6\ 0.9\ 1]$,

$y = [1\ 0.64\ 0.25\ 0.04\ 0\ 0.01\ 0.09\ 0.36\ 0.81\ 1]$.

Процедура создания и использования данной НС описывается следующим образом:

» P = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1]; % Задание входных значений

» T = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1]; % Задание выходных значений

» a = newgrnn(P,T,0.01); % Создание НС с отклонением 0.01

» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8]) % Опрос НС

Y =

0.8200 0.6400 0.0400 0.0900 0.8100

Как видно, точность аппроксимации в данном случае получилась не очень высокой.

Можно попытаться улучшить качество аппроксимации за счет подбора величины отклонения, но в условиях примера приемлемый результат легко достигается путем применения сети с радиальными базисными элементами:

» a = newrbe(P,T);

» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8]) % Опрос НС

Y =

0.8100 0.4900 0.0900 0.1600 0.6400

Созданную сеть можно сохранить для последующего использования набором в командной строке **save('a')**; при этом будет создан файл **a.mat**, т. е. файл с именем НС и расширением mat. В последующих сеансах работы сохраненную сеть можно загрузить, используя функцию **load('a')**. Естественно, допустимы все другие формы записи операторов **save** и **load**.

Рассмотрим теперь аналогичную задачу, но с использованием линейной НС.

Пусть экспериментальная информация задана значениями:

$x = [+1.0\ +1.5\ +3.0\ -1.2]$,

$y = [+0.5\ +1.1\ +3.0\ -1.0]$.

Процесс создания, обучения и использования линейной НС с именем b иллюстрируется приведенными функциями и рис. 4.10.

» P = [+1.0 +1.5 +3.0 -1.2];

» T = [+0.5 +1.1 +3.0 -1.0];

» maxlr = maxlinlr(P,'bias'); % Определение величины коэффициента обучения

» b = newlin([-2 2],1,[0],maxlr); % Создание линейной НС с именем b

» b.trainParam.epochs = 15; % Задание количества циклов обучения

» b = train(b,P,T); % Обучение НС

TRAINWB, Epoch 0/15, MSE 2.865/0.

TRAINWB, Epoch 15/15, MSE 0.0730734/0.

```
TRAINWB, Maximum epoch reached.
» p = -1.2;
» y = sim(b,p) % Опрос сети
y =
   -1.1803
```

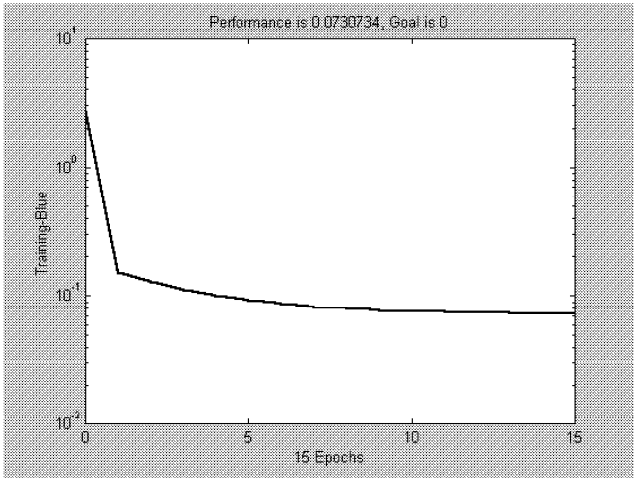

Рис. 4.10. Изменение ошибки сети в процессе ее обучения

4.3.2. Прогнозирование значений процесса. Рассмотрим теперь такой пример. Предположим, что имеется сигнал (функция времени), описываемый соотношением $x(t) = \sin(4\pi t)$, который подвергается дискретизации с интервалом 0.025 с.

Построим линейную нейронную сеть, позволяющую прогнозировать будущее значение подобного сигнала по 5 предыдущим. Решение данной задачи иллюстрируется ниже.

```
» t = 0:0.025:5; % Задание диапазона времени от 0 до 5 секунд
» x = sin(t*4*pi); % Предсказываемый сигнал
» Q = length(x);
» % Создание входных векторов
» P = zeros(5,Q); % Создание нулевой матрицы P
» P(1,2:Q) = x(1,1:(Q-1));
» P(2,3:Q) = x(1,1:(Q-2));
» P(3,4:Q) = x(1,1:(Q-3));
» P(4,5:Q) = x(1,1:(Q-4));
» P(5,6:Q) = x(1,1:(Q-5));
```

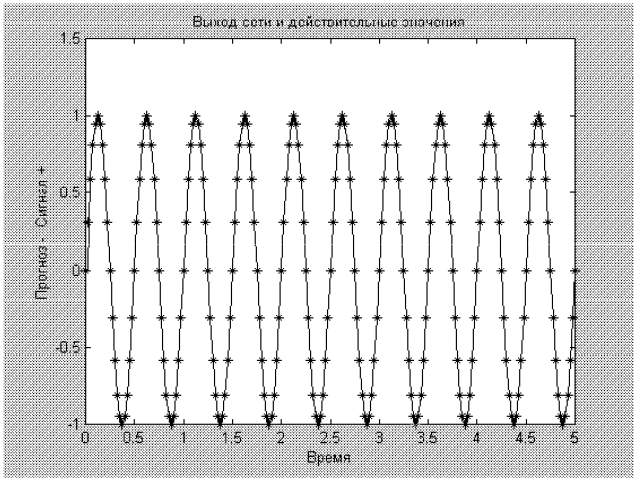

Рис. 4.11. Исходный сигнал и прогноз

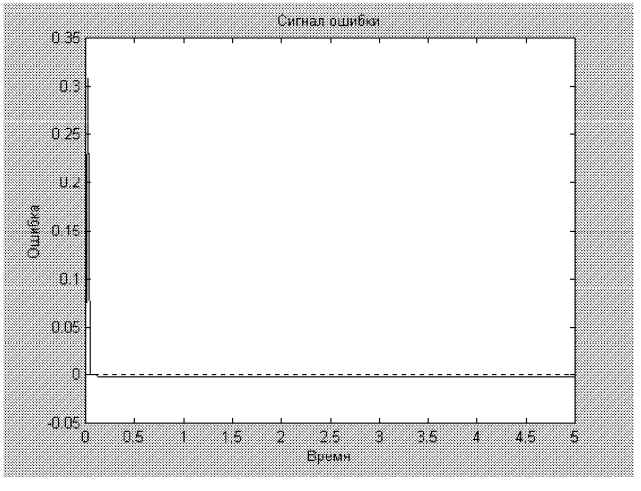

Рис. 4.12. Ошибка прогноза

```
» s = newlind(P,x); % Создание новой НС с именем s
» y = sim(s,P); % Расчет прогнозируемых значений
» % Создание графиков исходного сигнала и прогноза
» plot(t,y,t,x,'*')
» xlabel('Время');
» ylabel('Прогноз - Сигнал +');
» title('Выход сети и действительные значения');
» % Расчет и создание графика ошибки прогноза
» e = x-y;
» plot(t,e)
» hold on
» plot([min(t) max(t)],[0 0],':r')
» hold off
» xlabel('Время');
» ylabel('Ошибка');
» title('Сигнал ошибки');
```

В данном случае сеть создавалась с помощью функции newlind, при которой не требуется дополнительного обучения. Судя по графикам результатов, приведенных на рис. 4.11 и 4.12, точность прогноза с использованием линейной НС можно считать хорошей.

4.3.3. Использование слоя Кохонена. Рассмотрим задачу автоматического выявления (в режиме обучения без учителя) центров кластеров входов для двумерного случая с использованием слоя Кохонена (слоя «соревнующихся» нейронов). Решение данной задачи приведено ниже.

```
» X = [0 1; 0 1]; % Задание диапазонов возможного положения центров
кластеров
» % Задание параметров для моделирования исходных данных,
» % принадлежащих 8 классам (кластерам)
» clusters = 8;
» points = 10;
» std_dev = 0.05;
» P = nngenc(X,clusters,points,std_dev); % Моделирование
входных данных
» h = newc([0 1;0 1],8,.1); % Создание слоя Кохонена
» h.trainParam.epochs = 500; % Задание количества циклов
обучения
» h = init(h); % Инициализация сети
» h = train(h,P); % Обучение сети
» w = h.IW{1};
» % Вывод графика исходных данных и выявленных центров кластеров
» plot(P(1,:),P(2,:),'+r');
» hold on; plot(w(:,1),w(:,2),'ob');
» xlabel('p(1)');
```

```
» ylabel('p(2)');
» p = [0; 0.2]; % Задание нового входного вектора
» y = sim(h,p) % Опрос сети
» y =
   (3,1)     1
```

Работу обученной сети иллюстрирует рис. 4.13 и результат ее опроса (который выдается в форме разреженной матрицы). В усло-

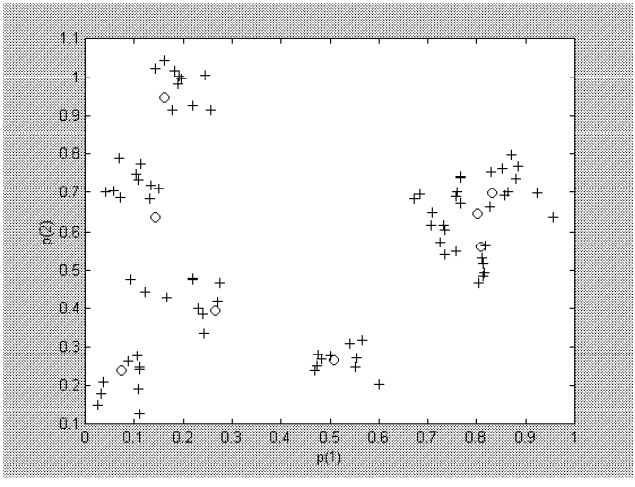

Рис. 4.13. Исходные данные и выявленные центры кластеров

виях примера предъявленный вектор отнесен к третьему классу (кластеру).

4.3.4. Сеть Хопфилда с двумя нейронами. Рассмотрим сеть Хопфилда, имеющую два нейрона и два устойчивых состояния, отображаемых векторами [1 −1] и [−1 1]. Представим эти векторы с помощью рис. 4.14, выводимого программой

```
» T = [+1 -1; -1 +1];
» plot(T(1,:),T(2,:),'r*')
» axis([-1.1 1.1 -1.1 1.1]);
» title('Пространство векторов НС Хопфилда');
» xlabel('a(1)');
» ylabel('a(2)');
```

Создадим сеть Хопфилда (с именем H) и проверим ее работу, подав на вход векторы, соответствующие устойчивым точкам. Если

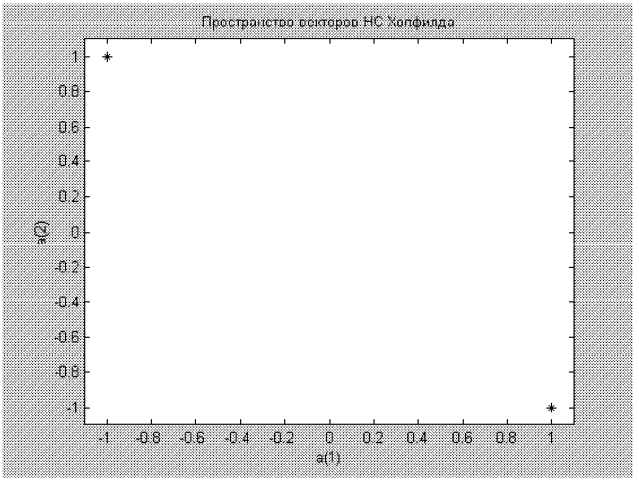

Рис. 4.14. Устойчивые точки сети Хопфилда

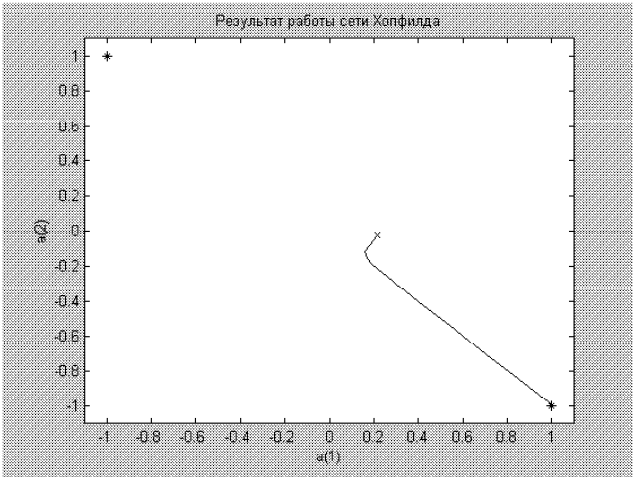

Рис. 4.15. Результат работы сети Хопфилда

сеть работает правильно, она должна выработать эти же векторы без каких-либо изменений.

```
» H = newhop(T); % Создание НС Хопфилда
» [Y,Pf,Af] = sim(H,2,[],T);Y % Опрос сети Хопфилда
Y =
   1    -1
  -1     1
```

Как видно из результата опроса, сеть работает правильно. Подадим теперь на ее вход произвольный вектор.

```
» a = {rands(2,1)}; % Задание случайного вектора
» [y,Pf,Af] = sim(H,1 50,{ },a);
» plot(T(1,:),T(2,:),'r*')
» axis([-1.1 1.1 -1.1 1.1]);
» record = [cell2mat(a) cell2mat(y)];
» start = cell2mat(a);
» hold on;
» plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:))
» xlabel('a(1)'); ylabel('a(2)');
» title('Результат работы сети Хопфилда');
```

Результат иллюстрируется рис. 4.15.

4.3.5. Классификация с помощью персептрона. Следующий пример иллюстрирует решение задачи классификации с помощью персептрона. Исходные входные векторы (с указанием их принадлежности к одному из двух классов) и результат настройки персептрона (с именем **My_net**) представлены на рис. 4.16.

```
» % Задание входных векторов с указанием их принадлежности
» % одному из двух классов
» P = [-0.5 -0.5 +0.3 -0.1;-0.5 +0.5 -0.5 +1.0];
» T = [1 1 0 0];
» plotpv(P,T); % Графическое представление исходных
векторов
» % Создание персептрона с указанием границ изменений
входов и 1 нейроном
» My_net = newp([-1 1;-1 1],1);
» E = 1;
» My_net = init(My_net); % Инициализация персептрона
» % Организация цикла адаптивной настройки персептрона
» % с выводом графика разделяющей линии
» while (sse(E))
  [My_net,Y,E] = adapt(My_net,P,T);
```

```
linehandle = plotpc(My_net.IW{1},My_net.b{1});
drawnow;
end;
```

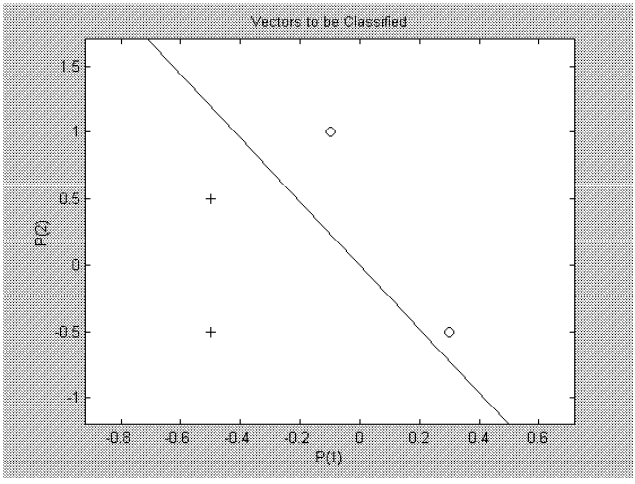

Рис. 4.16. Исходные входные векторы и разделяющая линия

4.3.6. Адаптивный линейный прогноз. В предыдущем примере настройка НС производилась адаптивно. Отличие такой настройки от выполняемой, например, с помощью метода обратного распространения ошибки, заключается в том, что векторы обучающей выборки поступают на вход сети не все «одновременно», а последовательно, по одному, при этом после предъявления очередного вектора производится корректировка весов и смещений и может быть произведен опрос сети, затем все повторяется. Адаптивная настройка особенно удобна при работе НС в «реальном» масштабе времени.

Рассмотрим пример задачи с прогнозированием значений сигнала (по 5 предыдущим значениям) с использованием указанной настройки.

Предположим, что исходный сигнал определен на интервале времени от 0 до 6 с, при этом при $0 \leqslant t < 4$ с он описывается соотношением $x(t) = \sin(4\pi t)$, а при $4 \leqslant t \leqslant 6$ с — соотношением $x(t) = \sin(8\pi t)$. График такого сигнала приведен на рис. 4.17.

```
» time1 = 0:0.05:4; % от 0 до 4 секунд
» time2 = 4.05:0.024:6; % от 4 до 6 секунд
```

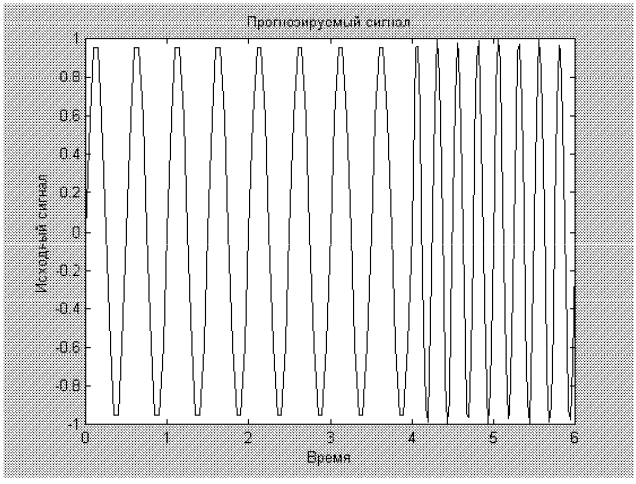

Рис. 4.17. График прогнозируемого сигнала

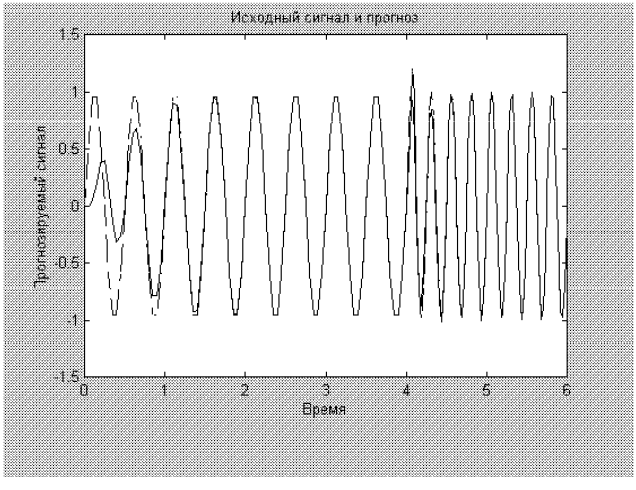

Рис. 4.18. Исходный сигнал и прогноз

```
» time = [time1 time2];
» % T определяет исходный сигнал:
» T = con2seq([sin(time1*4*pi) sin(time2*8*pi)]);
» % График исходного сигнала:
» plot(time,cat(2,T{:}))
» xlabel('Время');
» ylabel('Исходный сигнал');
» title('Прогнозируемый сигнал');
```

Для прогноза значений сигнала создадим линейную НС.

```
» % Входной и целевой прогнозируемый сигнал одинаковы:
» P = T;
» % Задание коэффициента обучения
» lr = 0.1;
» % Для прогноза используются 5 предыдущих значений
» delays = [1 2 3 4 5];
» % Создание и настройка линейной НС
» net = newlin(minmax(cat(2,P{:})),1,delays,lr); % Создание НС
» [net,y,e] = adapt(net,P,T); % Адаптивная настройка сети
» % Графики исходного сигнала и прогноза
» plot(time,cat(2,y{:}),time,cat(2,T{:}),'--')
» xlabel('Время');
» ylabel('Прогнозируемый сигнал');
» title('Исходный сигнал и прогноз');
```

Исходный сигнал и прогноз для этого примера приведены на рис. 4.18. Как видно из рис. 4.18, полученный результат можно считать удовлетворительным.

4.3.7. Использование сети Элмана. Рассмотрим задачу восстановления формы сигнала с использованием рекуррентной сети Элмана. Пусть имеется два синусоидальных сигнала — один с единичной амплитудой, другой — с амплитудой, равной двум:

```
p1 = sin(1:20);
p2 = sin(1:20)*2.
```

Пусть целевым сигналом будет сигнал, составленный из их амплитудных значений

```
t1 = ones(1,20);
t2 = ones(1,20)*2
```

при этом данные амплитуды чередуются, так что входные и целевые значения могут быть представлены в форме

```
p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2].
```

Преобразуем эти значения в последовательности:

```
Pseq = con2seq(p);
Tseq = con2seq(t).
```

После этого можно непосредственно перейти с проектированию НС. В рассматриваемой случае имеем, очевидно, один вход и один выход, т. е. в сети должны присутствовать один входной элемент и один выходной нейрон. Число нейронов в скрытом слое может быть, вообще говоря, любым (оно зависит от сложности задачи); примем, что этот слой содержит 10 нейронов. Дальнейшее решение задачи иллюстрируется ниже, при этом на рис. 4.19 приведен график изменения ошибки сети в процессе ее обучения, а на рис. 4.20 — результаты тестирования сети.

```
» % Задание исходных данных
» p1 = sin(1:20);
» t1 = ones(1,20);
» p2 = sin(1:20)*2;
» t2 = ones(1,20)*2;
» p = [p1 p2 p1 p2];
» t = [t1 t2 t1 t2];
» Pseq = con2seq(p);
» Tseq = con2seq(t);
» % Создание сети Элмана с диапазоном входа [-2, 2], 10
нейронами
» % скрытого слоя, одним выходным нейроном,
» % функцией активации в виде гиперболического тангенса
» % для нейронов скрытого слоя, линейной функцией
активации
» % для выходного нейрона, функцией обучения с адаптацией
» % коэффициента обучения
» net = newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx');
» % Задание параметров обучения:
» net.trainParam.epochs = 500; % Число циклов обучения
» net.trainParam.goal = 0.01; % Целевое значение функции
ошибки
» net.performFcn = 'sse'; % Задание вида функции ошибки
» % Обучение сети. По умолчанию промежуточные результаты обуче-
ния
» % выводятся через 25 циклов
» [net,tr] = train(net,Pseq,Tseq);
TRAINGDX, Epoch 0/500, SSE 443.798/0.01,
Gradient 387.439/1e-006
TRAINGDX, Epoch 25/500, SSE 22.1356/0.01,
Gradient 7.76533/1e-006
TRAINGDX, Epoch 50/500, SSE 20.0141/0.01,
Gradient 2.17503/1e-006
```

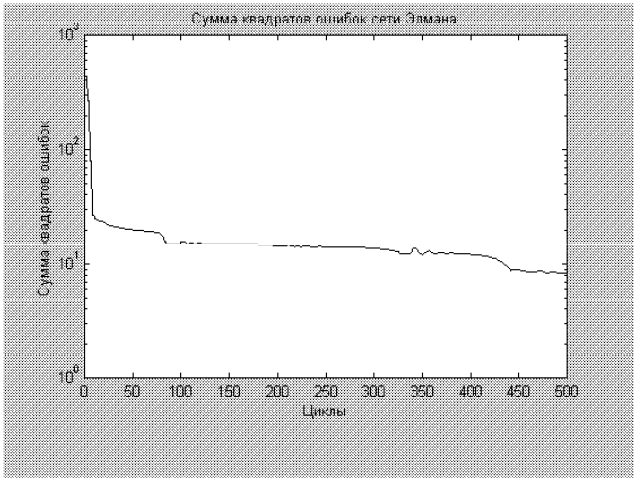

Рис. 4.19. Изменение ошибки сети в процессе обучения

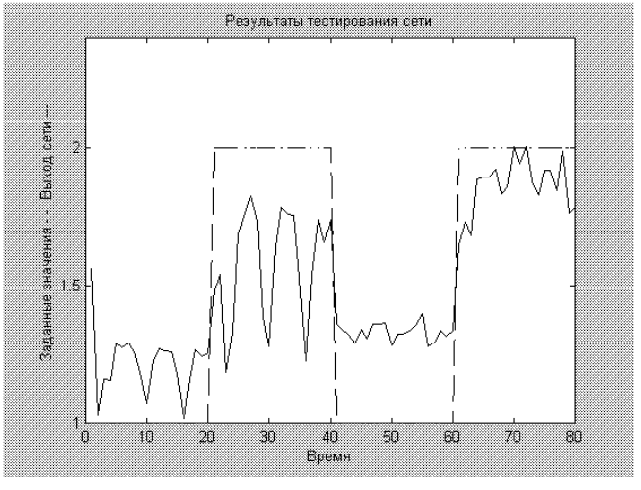

Рис. 4.20. Результаты тестирования сети

```
    TRAINGDX, Epoch 75/500, SSE 18.9825/0.01,
Gradient 1.26454/1e-006
    TRAINGDX, Epoch 100/500, SSE 15.8484/0.01,
Gradient 9.11439/1e-006
    TRAINGDX, Epoch 125/500, SSE 15.1932/0.01,
Gradient 2.92454/1e-006
    TRAINGDX, Epoch 150/500, SSE 15.1337/0.01,
Gradient 2.72534/1e-006
    TRAINGDX, Epoch 175/500, SSE 14.9634/0.01,
Gradient 2.75817/1e-006
    TRAINGDX, Epoch 200/500, SSE 14.3391/0.01,
Gradient 3.06463/1e-006
    TRAINGDX, Epoch 225/500, SSE 14.2136/0.01,
Gradient 3.24916/1e-006
    TRAINGDX, Epoch 250/500, SSE 14.1567/0.01,
Gradient 3.26074/1e-006
    TRAINGDX, Epoch 275/500, SSE 14.0799/0.01,
Gradient 3.1542/1e-006
    TRAINGDX, Epoch 300/500, SSE 13.7704/0.01,
Gradient 3.27526/1e-006
    TRAINGDX, Epoch 325/500, SSE 12.6771/0.01,
Gradient 3.72479/1e-006
    TRAINGDX, Epoch 350/500, SSE 12.1381/0.01,
Gradient 5.86736/1e-006
    TRAINGDX, Epoch 375/500, SSE 12.315/0.01,
Gradient 3.92575/1e-006
    TRAINGDX, Epoch 400/500, SSE 12.1414/0.01,
Gradient 3.89053/1e-006
    TRAINGDX, Epoch 425/500, SSE 11.1606/0.01,
Gradient 3.93421/1e-006
    TRAINGDX, Epoch 450/500, SSE 8.91345/0.01,
Gradient 5.55002/1e-006
    TRAINGDX, Epoch 475/500, SSE 8.56053/0.01,
Gradient 3.87387/1e-006
    TRAINGDX, Epoch 500/500, SSE 8.34089/0.01,
Gradient 3.7055/1e-006
    TRAINGDX, Maximum epoch reached, performance goal was
not met.
    » % Построение графика функции ошибки
    » semilogy(tr.epoch,tr.perf);
    » title('Сумма квадратов ошибок сети Элмана');
    » xlabel('Циклы');
    » ylabel('Сумма квадратов ошибок');
    » % Тестирование сети
```

```
» a = sim(net,Pseq);
» time = 1:length(p);
» time = 1:length(p);
» plot(time,t,'--',time,cat(2,a{:}))
» title('Результаты тестирования сети');
» xlabel('Время');
» ylabel('Заданные значения -- Выход сети ---')
```

4.3.8. Задача классификации: применение сети встречного распространения. Предположим, поставлена следующая задача классификации: задан набор из 10 векторов, представленных в виде столбцов матрицы

$$P = \begin{bmatrix} -3 & -2 & -2 & 0 & 0 & 0 & 0 & 2 & 2 & 3 \\ 0 & 1 & -1 & 2 & 1 & -1 & -2 & 1 & -1 & 0 \end{bmatrix},$$

а также задан вектор-строка, указывающий принадлежность каждого вектора к одному из двух классов:

Tc = [1 1 1 2 2 2 2 1 1 1].

Требуется: построить автоматический классификатор подобных векторов, используя приведенные данные как обучающую выборку.

Решение подобной задачи проведем с применением сети встречного распространения так, как это показано ниже.

```
» P = [-3 -2 -2 0 0 0 0 +2 +2 +3; 0 +1 -1 +2 +1 -1 -2 +1 -1 0];
» C = [1 1 1 2 2 2 2 1 1 1];
» T = ind2vec(C); % Преобразование вектора C в матрицу T с двумя строками
» % Создание новой сети встречного распространения требует 4-х параметров:
» % 1) матрицы минимальных и максимальных значений входных элементов,
» % 2) числа скрытых нейронов,
» % 3) вектор с элементами, указывающими долю каждого из классов,
» % 4) величины коэффициента обучения
» net = newlvq(minmax(P),4,[.6 .4],0.1); % Создание сети
» net.trainParam.epochs = 150; % Задание числа циклов обучения
» net.trainParam.show = Inf; % Запрет на выдачу промежуточных результатов
» net = train(net,P,T); % Обучение НС
TRAINWB1, Epoch 150/150
TRAINWB1, Maximum epoch reached.
» Y = sim(net,P) % Тестирование сети
Y =
  1   1   1   0   0   0   0   1   1   1
  0   0   0   1   1   1   1   0   0   0
```

Как видно из результатов тестирования, классификация элементов обучающей выборки произведена точно (три первых и три последних вектора отнесены к первому классу, остальные — ко второму).

4.3.9. Создание и использование самоорганизующейся карты. Как отмечалось, самоорганизующиеся карты можно рассматривать как усовершенствованную модификацию слоя конкурирующих нейронов (слоя Кохонена). От последнего данный вид НС отличается тем, что:

1) нейроны распределяются некоторым пространственным образом (по одному из трех возможных вариантов: в узлах прямоугольной решетки, гексагональной решетки или случайно);

2) на этапе самообучения корректируются веса не только нейрона-«победителя», но и группы нейронов в его некоторой пространственной окрестности.

Назначения самоорганизующихся карт такое же, как и у слоя Кохонена — выявление в режиме самообучения центров кластеров входных векторов.

Создание и использование самоорганизующейся карты рассмотрим на примере кластеризации двумерных векторов (исходные данные приведены на рис. 4.21).

```
» P = randss(2,1000); % Задание случайных двумерных
входных векторов
» plot(P(1,:),P(2,:),'+r') % Визуальное изображение входных
векторов
» Создание НС с 5×6 = 30 нейронами; все установки — по
умолчанию
» net = newsom([0 1; 0 1],[5 6]);
» net.trainParam.epochs = 1000; % Задание числа циклов
настройки
» net.trainParam.show = 200; % Задание периодичности
вывода информации
» net = train(net,P); % Настройка сети
TRAINWB1, Epoch 0/1000
TRAINWB1, Epoch 200/1000
TRAINWB1, Epoch 400/1000
TRAINWB1, Epoch 600/1000
TRAINWB1, Epoch 800/1000
TRAINWB1, Epoch 1000/1000
TRAINWB1, Maximum epoch reached.
```

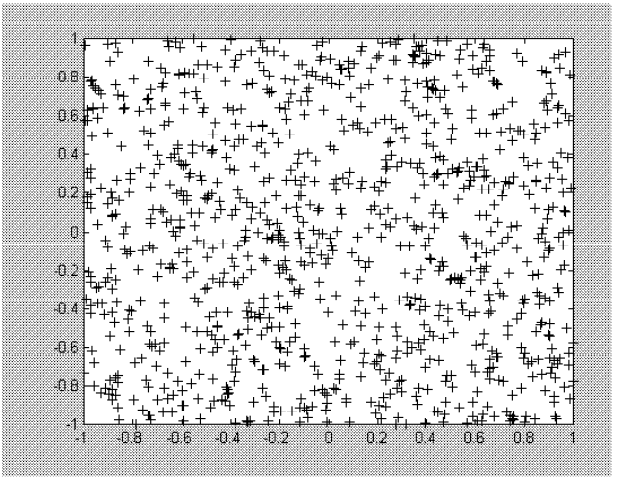

Рис. 4.21. Исходные данные

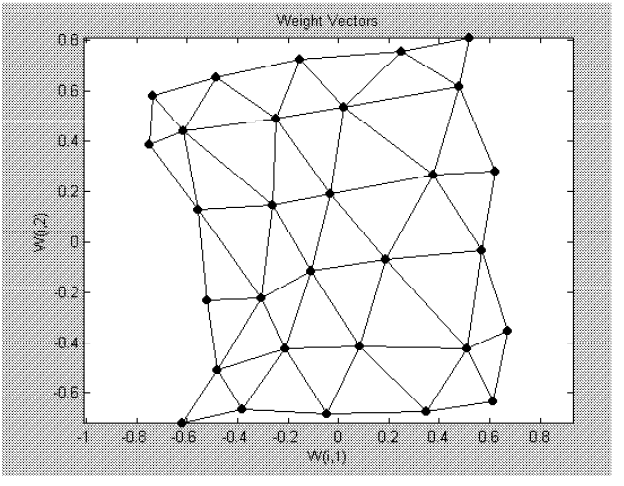

Рис. 4.22. Выявленные центры кластеров

```
» plotsom(net.iw{1,1},net.layers{1}.distances); % Выявленные
центры кластеров
» p = [0.5; 0.3];
» a = sim(net,p) % Опрос сети
a =
   (11,1)    1
```

Предъявленный на этапе тестирования вектор отнесен НС к 11-му классу.

Выявленные центры кластеров представлены на рис. 4.22.

Другие примеры доступны через главное меню MATLAB (пункт Help/Examples and Demos, раздел Toolboxes/Neural Networks).

4.3.10. Использование Simulink при построении нейронных сетей. Пакет Neural Network Toolbox содержит ряд блоков, которые либо могут быть непосредственно использованы для построения нейронных сетей в среде Simulink, либо применяться вместе с рассмотренной выше функцией **gensim**.

Для вызова отмеченной набора блоков, в командной строке необходимо набрать команду **neural**, после выполнения который появляется окно вида рис. 4.23. Каждый из представленных на

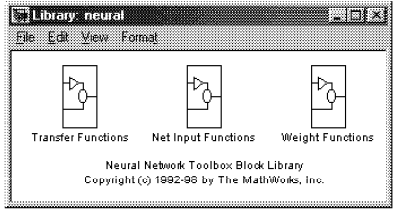

Рис. 4.23. Основные нейросетевые блоки Simulink

рис. 4.23 блоков в свою очередь является набором (библиотекой) некоторых блоков. Рассмотрим их.

Блоки функций активации (Transfer Functions). Двойной щелчок левой кнопки мыши на блоке Transfer Functions приводит к появлению библиотеки функций активации (рис. 4.24). Каждый из этих блоков данной библиотеки преобразует подаваемый на него вектор в соответствующий вектор той же размерности (табл. 2.1).

Блоки преобразования входов сети. Проводя аналогичную рассмотренной операцию, но с блоком Net Input Functions, придем к библиотеке блоков вида рис. 4.25.

Блоки данной библиотеки реализуют рассмотренные выше функции преобразования входов сети.

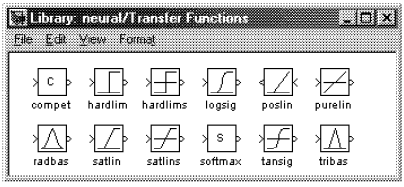

Рис. 4.24. Библиотека функций активации

Блоки весовых коэффициентов. Точно так же (но щелкая левой кнопкой мыши по иконке с надписью Weight Functions) придем к библиотеке блоков (рис. 4.26), реализующих некоторые функции весов и расстояний.

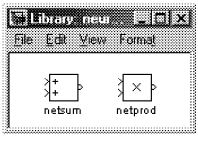

Рис. 4.25. Библиотека блоков преобразований сигналов

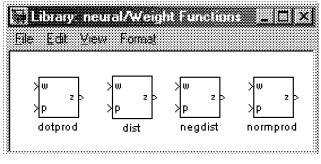

Рис. 4.26. Библиотека блоков весовых коэффициентов

Отметим, что при задании конкретных числовых значений, при операции со всеми приведенными блоками ввиду особенностей Simulink векторы необходимо представлять как столбцы, а не как строки (как это было до сих пор).

Формирование нейросетевых моделей. Основной функцией для формирование нейросетевых моделей в Simulink является функция **gensim**, записываемая в форме

gensim(net,st),

где **net** — имя созданной НС, **st** — интервал дискретизации (если НС не имеет задержек, ассоциированных с ее входами или слоями, значение данного аргумента устанавливается равным -1).

В качестве примера использования средств Simulink рассмотрим следующий.

Пусть входной и целевой векторы имеют вид

p = [1 2 3 4 5];

t = [1 3 5 7 9].

Создадим линейную НС и протестируем ее:

```
» p = [1 2 3 4 5];
» t = [1 3 5 7 9];
» net = newlind(p,t);
» y = sim(net,p)
y =
   1.0000    3.0000    5.0000    7.0000    9.0000
```

Затем запустим Simulink командой

```
» gensim(net,-1)
```

Это приведет к открытию блок-диаграммы (рис. 4.27).

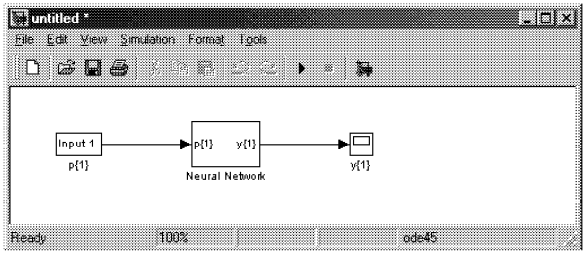

Рис. 4.27. Созданная нейросетевая модель Simulink

Для проведения тестирования модели щелкнем дважды по левой иконке (с надписью Input 1, т.е. Вход 1), что приведет к открытию диалогового окна (рис. 4.28).

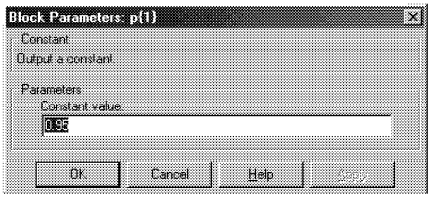

Рис. 4.28. Диалоговое окно задания входа НС

В данном случае блок Input 1 является стандартным блоком задания константы (**Constant**). Изменим значение по умолчанию на 2 и нажмем кнопку **OK**. Затем нажмем кнопку **Start** в меню моделирования. Расчет нового значения сетью производится практически мгновенно. Для его вывода необходимо дважды щелкнуть

мышью по правой иконке (блоку $y(1)$). Результат вычислений отображается рис. 4.29 — он равен 3, как и требуется.

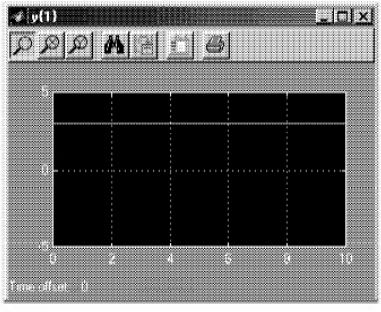

Рис. 4.29. Окно с выходом НС

Отметим, что, дважды щелкая левой кнопкой мыши по блоку Neural Network, затем — по блоку Layer 1, можно получить детальную графическую информацию о структуре сети (рис. 4.30).

С созданной сетью можно проводить различные эксперименты, возможные в среде Simulink; вообще с помощью команды **gensim** осуществляется интеграция созданных нейросетей в блок-диаграм-

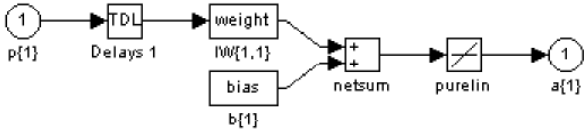

Рис. 4.30. Структура созданной НС

мы этого пакета — с использованием имеющихся при этом инструментов моделирования раз- личных систем (например, встраивание нейросетевого регулятора в систему управления и моделирование последней и т. п.).