NOTEBOOK
"НЕЙРОННЫЕ СЕТИ"

Глава 2

С34. Единичная функция активации с жестким ограничением *hardlim*.

Эта функция описывается соотношением

`a = hardlim(n) = 1(n)`

и равна 0, если n < 0,

и равна 1, если n ≥ 0.

Построим график этой функции в диапазоне значений входа от -5 до + 5:

```
n = -5:0.1:5;
plot(n,hardlim(n),'b+:');
```

С34. Линейная функция активации *purelin*.

Эта функция описывается соотношением

`a = purelin(n) = n.`

Построим график этой функции в диапазоне значений входа от -5 до + 5:

```
n=-5:0.1:5;
plot(n,purelin(n),'b+:');
```

С34. Логистическая функция активации *logsig*.

Эта функция описывается соотношением

`a = logsig(n) = 1/(1 + exp(-n)).`

Она принадлежит к классу сигмоидальных функций, и ее аргумент может принимать любое значение в диапазоне от -∞ до +∞, а выход изменяется в диапазоне от 0 до 1. Благодаря свойству дифференцируемости, эта функция часто используется в сетях с обучением на основе метода обратного распространения ошибки.

Построим график этой функции в диапазоне значений входа от -5 до + 5:

```
n=-5:0.1:5;
plot(n,logsig(n),'b+:');
```

С42. Формирование архитектуры нейронной сети.

Следующий оператор создает сеть с прямой передачей сигнала

```
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');
gensim(net)
```

Эта сеть использует один вектор входа с двумя элементами, имеющими допустимые границы значений [-1 2] и [0 5];

- имеет 2 слоя с 3 нейронами в первом слое и 1 нейроном во втором слое;

- используемые функции активации: tansig - в первом слое, purelin – во втором слое;
- используемая функция обучения traingd.

С42. Инициализация нейронной сети.

После того как сформирована архитектура сети, должны быть заданы начальные значения весов и смещений, или иными словами, сеть должна быть инициализирована. Такая процедура выполняется с помощью метода init для объектов класса network. Оператор вызова этого метода имеет вид

```
net = init(net);
```

Если мы хотим заново инициализировать веса и смещения в первом слое, используя функцию rands, то следует ввести следующую последовательность операторов:

```
net.layers{1}.initFcn = 'initwb';
net.inputWeights{1,1}.initFcn = 'rands';
net.biases{1,1}.initFcn = 'rands';
net.biases{2,1}.initFcn = 'rands';
net = init(net);
```

С43. Моделирование сети.

Статическая нейронная сеть характеризуется тем, что в ее состав не входят линии задержки и обратные связи.

Рассмотрим однослойную сеть с двухэлементным вектором входа и линейной функцией активации. Для задания такой сети предназначена М-функция newlin, которая требует указать минимальное и максимальное значения для каждого из элементов входа; в данном случае они равны –1 и 1, соответственно, а также количество слоев, в данном случае 1.

Формирование однослойной линейной сети net с двухэлементным входным сигналом со значениями от -1 до 1:

```
net = newlin([-1 1;-1 1],1);
```

Определим весовую матрицу и смещение, равными **W** = [1 2], b = 0, и зададим эти значения, используя описание структуры сети

```
net.IW{1,1} = [1 2];
net.b{1} = 0;
```

Предположим, что на сеть подается следующая последовательность из четырех векторов входа

$$\left\{\begin{bmatrix}-1\\0\end{bmatrix},\begin{bmatrix}0\\-1\end{bmatrix},\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}1\\-1\end{bmatrix}\right\}.$$

Поскольку сеть статическая, можно перегруппировать эту последовательность в следующий числовой массив

```
P = [-1 0 0 1; 0 -1 1 -1];
```

Теперь можно моделировать сеть

```
A = sim(net,P)
A =
    -1    -2     2    -1
```

Результат следует интерпретировать следующим образом. На вход сети подается последовательность из 4 входных сигналов, и сеть генерирует вектор выхода из 4 элементов.

Динамическая нейронная сеть характеризуется тем, что в ее состав входят линии задержки и/или обратные связи. Когда сеть содержит линии задержки, вход сети следует рассматривать как последовательность векторов, подаваемых на сеть в определенные моменты времени.

Создание однослойной линейной сети с линией задержки [0 1]:

```
net = newlin([-1 1],1,[0 1]);
```

Зададим следующую матрицу весов W=[1 2] и нулевое смещение:

```
net.IW{1,1} = [1 2];
net.biasConnect = 0;
```

Предположим, что входная последовательность имеет вид {-1, -1/2, 1/2, 1} и зададим ее в виде массива ячеек

```
P = {-1 -1/2 1/2 1};
```

Теперь можно моделировать сеть, используя метод sim:

```
A = sim(net,P)
A =
    [-1]    [-2.5000]    [-0.5000]    [2]
```

Если те же самые входы подать на сеть одновременно, то получим совершенно иную реакцию. Для этого сформируем следующий вектор входа

```
P = [-1 -1/2 1/2 1];
```

После моделирования получаем

```
A = sim(net,P)
A =
   -1.0000   -0.5000    0.5000    1.0000
```

Результат такой же как, если применить каждый вход к отдельной сети и вычислить ее выход. Поскольку начальные условия для элементов запаздывания не указаны, то по умолчанию они приняты нулевыми. В этом случае выход сети равен

$$[1\ \ 2]\begin{bmatrix} -1 & -1/2 & 1/2 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -1 & -1/2 & 1/2 & 1 \end{bmatrix}.$$

Если требуется моделировать реакцию сети для нескольких последовательностей сигналов на входе, то следует сформировать массив ячеек, размер каждой из которых совпадает с числом таких последовательностей. Пусть, например, требуется приложить к сети две последовательности

$$p_1(1) = [-1],\ p_1(2) = [-1/2],\ p_1(3) = [1/2],\quad p_1(4) = [1];$$
$$p_2(1) = [1],\ p_2(2) = [1/2],\quad p_2(3) = [-1/2],\ p_2(4) = [-1].$$

Вход P в этом случае должен быть массивом ячеек, каждая из которых содержит два элемента по числу последовательностей

```
P = {[-1,1],[-1/2,1/2],[1/2,-1/2],[1,-1]};
```

Теперь можно моделировать сеть:

```
A = sim(net,P); cat(1, A{:})
ans =
   -1.0000    1.0000
   -2.5000    2.5000
   -0.5000    0.5000
    2.0000   -2.0000
```

Глава 3

C52. Адаптация нейронных сетей.

Статические сети.

Воспользуемся следующей моделью однослойной линейной сети с двухэлементным вектором входа, значения которого находятся в интервале [–1 1] и нулевым параметром скорости настройки

```
clear
net = newlin([-1 1;-1 1],1, 0, 0);
```

Требуется адаптировать параметры сети так, чтобы она формировала линейную зависимость вида

Последовательный способ.

Рассмотрим случай последовательного представления обучающей последовательности. В этом случае входы и целевой вектор формируются в виде массива формата cell

```
P = {[-1; 1] [-1/3; 1/4] [1/2; 0] [1/6; 2/3]};
T = {-1 -5/12 1 1};
P1 = [P{:}], T1=[T{:}]
P1 =
   -1.0000   -0.3333    0.5000    0.1667
    1.0000    0.2500         0    0.6667
T1 =
   -1.0000   -0.4167    1.0000    1.0000
```

Сначала зададим сеть с нулевыми значениями начальных весов и смещений

```
net.IW{1} = [0 0];
net.b{1} = 0;
```

Выполним один цикл адаптации сети с нулевым параметром скорости настройки:

```
[net1,a,e] = adapt(net,P,T);
```

В этом случае веса не модифицируются, выходы сети остаются нулевыми, поскольку параметр скорости настройки равен нулю, адаптации сети не происходит. Погрешности совпадают со значениями целевой последовательности

```
net1.IW{1, 1}, a, e
ans =
     0     0
a =
    [0]    [0]    [0]    [0]
e =
    [-1]    [-0.4167]    [1]    [1]
```

Зададим значения параметров скорости настройки и весов входа и смещения

```
net.IW{1} = [0 0];
net.b{1} = 0;
net.inputWeights{1,1}.learnParam.lr = 0.2;
net.biases{1,1}.learnParam.lr = 0;
```

Выполним один цикл настройки:

```
[net1,a,e] = adapt(net,P,T);
net1.IW{1, 1}, a, e
ans =
```

```
    0.3454   -0.0694
a =
    [0]    [-0.1167]    [0.1100]    [-0.0918]
e =
    [-1]    [-0.3000]    [0.8900]    [1.0918]
```

Теперь выполним последовательную адаптацию сети в течение 30 циклов:

```
net = newlin([-1 1;-1 1],1, 0, 0);
net.IW{1} = [0 0];
net.b{1} = 0;
```

Зададим значения параметров скорости настройки для весов входа и смещения

```
net.inputWeights{1,1}.learnParam.lr = 0.2;
net.biases{1,1}.learnParam.lr = 0;
P = {[-1; 1] [-1/3; 1/4] [1/2; 0] [1/6; 2/3]};
T = {-1 -5/12 1 1};
for i=1:30,
    [net,a{i},e{i}] = adapt(net,P,T);
    W(i,:)=net.IW{1,1};
end
mse(cell2mat(e{30}))
ans =
    0.0017
```

Веса после 30 циклов:

```
W(30,:)
ans =
    1.9199    0.9250
cell2mat(a{30})
ans =
   -0.9944   -0.4086    0.9566    0.9301
cell2mat(e{30})
ans =
   -0.0056   -0.0081    0.0434    0.0699
```

Построим графики зависимости значений выходов сети и весовых коэффициентов в зависимости от числа итераций:

```
subplot(3,1,1)
plot(0:30,[[0 0 0 0];cell2mat(cell2mat(a'))],'k')
xlabel(''), ylabel('Выходы a(i)'),grid
subplot(3,1,2)
plot(0:30,[[0 0]; W],'k')
xlabel(''), ylabel('Веса входов w(i)'),grid
```

```
subplot(3,1,3)
for i=1:30, E(i) = mse(e{i}); end
semilogy(1:30, E,'+k')
xlabel(' Циклы'), ylabel('Ошибка'),grid
```

Групповой способ.

Рассмотрим случай группового представления обучающей последовательности. В этом случае входы и целевой вектор формируются в виде массива формата double:

```
clear
P = [-1 -1/3 1/2 1/6; 1 1/4 0 2/3];
T = [-1 -5/12 1 1];
```

Основной цикл адаптации сети выглядит следующим образом:

```
net3 = newlin([-1 1;-1 1],1, 0, 0.2);
net3.IW{1} = [0 0];
net3.b{1} = 0;
net3.inputWeights{1,1}.learnParam.lr = 0.2;
EE = 10; i=1;
while EE > 0.0017176
   [net3,a{i},e{i},pf] = adapt(net3,P,T);
   W(i,:) = net3.IW{1,1};
   EE = mse(e{i});
   ee(i)= EE;
   i = i+1;
end
```

Результатом адаптации являются следующие значения коэффициентов, значений выходов и среднеквадратической погрешности адаптации

```
W(63,:)
ans =
    1.9114    0.8477
cell2mat(a(63))
ans =
   -1.0030   -0.3624    1.0172    0.9426
EE = mse(e{63})
EE =
    0.0016
mse(e{1})
ans =
    0.7934
```

Процедура адаптации выходов и параметров нейронной сети иллюстрируется графиками:

```
subplot(3,1,1)
plot(0:63,[zeros(1,4); cell2mat(a')],'k') % Рис.3.2,а
xlabel(''), ylabel('Выходы a(i)'),grid
subplot(3,1,2)
plot(0:63,[[0 0]; W],'k') % Рис.3.2,б
xlabel(''), ylabel('Веса входов w(i)'),grid
subplot(3,1,3)
semilogy(1:63, ee,'+k') % Рис.3.2,в
xlabel('Циклы'), ylabel('Ошибка'),grid
```

С56. Динамические сети.

Эти сети характеризуются наличием линий задержки, и для них последовательное представление входов является наиболее естественным.

Последовательный способ.

Обратимся к линейной модели нейронной сети с одним входом и одним элементом запаздывания. Установим начальные условия на линии задержки, а также для весов и смещения равными нулю; а параметр скорости настройки равным 0.5.

```
clear
net = newlin([-1 1],1,[0 1],0.5);
Pi = {0};
net.IW{1} = [0 0];
net.biasConnect = 0;
```

Чтобы применить последовательный способ адаптации, представим входы и цели как массивы ячеек

```
P = {-1/2  1/3  1/5  1/4};
T = { -1   1/6  11/15 7/10};
```

Попытаемся адаптировать сеть формировать нужный выход на основе следующего соотношения

```
y(t) = 2p(t) + p(t-1).
```

Используем для этой цели М-функцию adapt и основной цикл адаптации сети с заданной погрешностью:

```
EE = 10; i = 1;
while EE > 0.0001
   [net,a{i},e{i},pf] = adapt(net,P,T);
```

```
    W(i,:)=net.IW{1,1};
    EE = mse(e{i});
    ee(i) = EE;
    i = i+1;
end
```

Сеть адаптировалась за 22 цикла. Результатом адаптации при заданной погрешности являются следующие значения коэффициентов, выходов нейронной сети и среднеквадратической погрешности

```
W(22,:)
ans =
    1.9830    0.9822
a{22}
ans =
    [-0.9896]    [0.1714]    [0.7227]    [0.6918]
EE
EE =
  7.7874e-005
```

Построим графики зависимости выходов системы и весовых коэффициентов от числа циклов обучения:

```
subplot(3,1,1)
plot(0:22,[zeros(1,4); cell2mat(cell2mat(a'))],'k') % Рис.3.3,а
xlabel(''), ylabel('Выходы a(i)'),grid
subplot(3,1,2)
plot(0:22,[[0 0]; W],'k') % Рис.3.3,б
xlabel(''), ylabel('Веса входов w(i)'),grid
subplot(3,1,3)
semilogy(1:22,ee,'+k') % Рис.3.3,в
xlabel('Циклы'), ylabel('Ошибка'),grid
```

С58. Обучение нейронных сетей

Статические сети.

Воспользуемся моделью однослойной линейной сети с двухэлементным вектором входа, значения которого находятся в интервале [–1 1] и нулевым параметром скорости настройки, как это было для случая адаптации:

```
clear
net = newlin([-1 1;-1 1],1, 0, 0);
net.IW{1} = [0 0];
net.b{1} = 0;
```

Требуется обучить параметры сети так, чтобы она формировала линейную зависимость вида

$$t = 2p_1 + p_2 \ .$$

Последовательный способ.

Представим обучающую последовательность в виде массивов ячеек

```
P = {[-1; 1] [-1/3; 1/4] [1/2; 0] [1/6; 2/3]};
T = {-1 -5/12 1 1};
```

Теперь все готово к обучению сети. Будем обучать ее с помощью функции train в течение 30 циклов. Для обучения и настройки параметров сети используем функции trainwb и learnwh, соответственно.

```
net.inputWeights{1,1}.learnParam.lr = 0.2;
net.biases{1}.learnParam.lr = 0;
net.trainParam.epochs = 30;
net1 = train(net,P,T);

TRAINB, Epoch 0/30, MSE 0.793403/0.
TRAINB, Epoch 25/30, MSE 0.00373997/0.
TRAINB, Epoch 30/30, MSE 0.00138167/0.
TRAINB, Maximum epoch reached.
```

Параметры сети после обучения равны следующим значениям

```
W = net1.IW{1}
W =
    1.9214    0.9260
y = sim(net1, P)
y =
    [-0.9954]    [-0.4090]    [0.9607]    [0.9376]
EE = mse([y{:}]-[T{:}])
EE =
    0.0014
```

Групповой способ.

Для этого представим обучающую последовательность в виде массивов формата double array:

```
P = [-1 -1/3 1/2 1/6; 1 1/4 0 2/3];
T = [-1 -5/12 1 1];
net1 = train(net,P,T);

TRAINB, Epoch 0/30, MSE 0.793403/0.
TRAINB, Epoch 25/30, MSE 0.00373997/0.
```

```
TRAINB, Epoch 30/30, MSE 0.00138167/0.
TRAINB, Maximum epoch reached.
```

Параметры сети после обучения равны следующим значениям

```
W = net1.IW{1}

W =
    1.9214    0.9260
y = sim(net1, P)

y =
   -0.9954   -0.4090    0.9607    0.9376
EE = mse(y-T)

EE =
    0.0014
```

Динамические сети.

Обучение динамических сетей выполняется аналогичным образом с использованием метода train.

Последовательный способ.

Обратимся к линейной модели нейронной сети с одним входом и одним элементом запаздывания.

Установим начальные условия для элемента запаздывания, весов и смещения, равными нулю; параметр скорости настройки, равным 0.5.

```
clear
net = newlin([-1 1],1,[0 1],0.5);
Pi = {0};
net.IW{1} = [0 0];
net.biasConnect = 0;
net.trainParam.epochs = 22;
```

Чтобы применить последовательный способ обучения, представим входы и цели как массивы ячеек

```
P = {-1/2  1/3  1/5  1/4};
T = { -1   1/6  11/15 7/10};
```

Используем для этой цели М-функцию train

```
net1 = train(net, P, T, Pi);
TRAINB, Epoch 0/22, MSE 0.513889/0.
TRAINB, Epoch 22/22, MSE 3.65137e-005/0.
TRAINB, Maximum epoch reached.
```

Параметры сети после обучения равны следующим значениям

```
W = net1.IW{1}
W =
    1.9883    0.9841
y = sim(net1, P)
y =
    [-0.9941]    [0.1707]    [0.7257]    [0.6939]
EE = mse([y{:}]-[T{:}])
EE =
  3.6514e-005
```

Градиентные алгоритмы обучения.

С67. Алгоритм GD.

Алгоритм GD, или алгоритм градиентного спуска используется для такой корректировки весов и смещений, чтобы минимизировать функционал ошибки, то есть обеспечить движение по поверхности функционала в направлении, противоположном градиенту функционала по настраиваемым параметрам.

Рассмотрим двухслойную нейронную сеть прямой передачи сигнала с сигмоидальным и линейным слоями для обучения ее на основе метода обратного распространения ошибки

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');
```

Последовательная адаптация.

Чтобы подготовить модель сети к процедуре последовательной адаптации на основе алгоритма GD, необходимо указать ряд параметров. В первую очередь, это имя функции настройки learnFcn, соответствующее алгоритму градиентного спуска, в данном случае – это M-функция learngd:

```
net.biases{1,1}.learnFcn = 'learngd';
net.biases{2,1}.learnFcn = 'learngd';
net.layerWeights{2,1}.learnFcn = 'learngd';
net.inputWeights{1,1}.learnFcn = 'learngd';
```

С функцией learngd связан лишь один параметр скорости настройки lr. Текущие приращения весов и смещений сети определяются умножением этого параметра на вектор градиента. Чем больше значение параметра, тем больше приращение на текущей итерации. Если параметр скорости настройки выбран

слишком большим, алгоритм может стать неустойчивым; если параметр слишком мал, то алгоритм может потребовать длительного счета.

При выборе функции learngd по умолчанию устанавливается следующее значение параметра скорости настройки

```
net.layerWeights{2,1}.learnParam
ans =
    lr: 0.0100
```

Увеличим значение этого параметра до 0.2

```
net.layerWeights{2,1}.learnParam.lr = 0.2;
```

Мы теперь почти готовы к обучению сети. Осталось задать обучающее множество. Это простое множество входов и целей определим следующим образом

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
```

Поскольку используется последовательный способ обучения, необходимо преобразовать массивы входов и целей в массивы ячеек

```
p = num2cell(p,1);
t = num2cell(t,1);
```

Последний параметр, который требуется установить при последовательном обучении, это число проходов net.adaptParam.passes

```
net.adaptParam.passes = 50;
```

Теперь можно выполнить настройку параметров, используя процедуру адаптации

```
[net,a,e] = adapt(net,p,t);
```

Чтобы проверить качество обучения, после окончания обучения смоделируем сеть.

```
a = sim(net,p)
a =
   -1.0127   -0.9960    1.0189    0.9813
mse(e)
ans =
  2.4546e-004
```

Групповое обучение.

Для обучения сети на основе алгоритма GD необходимо использовать M-функцию traingd взамен функции настройки learngd. В этом случае нет

необходимости задавать индивидуальные функции обучения для весов и смещений, а достаточно указать одну обучающую функцию для всей сети.

Вновь создадим ту же двухслойную нейронную сеть прямой передачи сигнала с сигмоидальным и линейным слоями для обучения по методу обратного распространения ошибки

```
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');
```

Функция traingd характеризуется следующими параметрами, заданными по умолчанию

```
net.trainParam
ans =
      epochs: 100
        goal: 0
          lr: 0.0100
    max_fail: 5
    min_grad: 1.0000e-010
        show: 25
        time: Inf
```

Установим новые значения параметров обучения, зададим обучающую последовательность в виде массива double и выполним процедуру обучения

```
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); % Рис.3.8
TRAINGD, Epoch 0/300, MSE 0.300839/1e-005, Gradient 0.884133/1e-
010
TRAINGD, Epoch 50/300, MSE 0.00576418/1e-005, Gradient
0.0975344/1e-010
TRAINGD, Epoch 100/300, MSE 6.76541e-005/1e-005, Gradient
0.0109646/1e-010
TRAINGD, Epoch 121/300, MSE 9.98702e-006/1e-005, Gradient
0.00422134/1e-010
TRAINGD, Performance goal met.
```

С69. Алгоритм GDM

Вновь рассмотрим двухслойную нейронную сеть прямой передачи сигнала с сигмоидальным и линейным слоями (рис. 3.7)

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingdm');
```

Последовательная адаптация.

Чтобы подготовить модель сети к процедуре последовательной адаптации на основе алгоритма GDM, необходимо указать ряд параметров. В первую очередь, это имя функции настройки learnFcn, соответствующее алгоритму градиентного спуска с возмущением, в данном случае – это М-функция learngdm:

```
net.biases{1,1}.learnFcn = 'learngdm';
net.biases{2,1}.learnFcn = 'learngdm';
net.layerWeights{2,1}.learnFcn = 'learngdm';
net.inputWeights{1,1}.learnFcn = 'learngdm';
```

С этой функцией связано два параметра - параметр скорости настройки lr и параметр возмущения mc.

При выборе функции learngdm по умолчанию устанавливаются следующие значения этих параметров

```
net.layerWeights{2,1}.learnParam
```

```
ans =
    lr: 0.0100
    mc: 0.9000
```

Увеличим значение параметра скорости обучения до 0.2

```
net.layerWeights{2,1}.learnParam.lr = 0.2;
```

Мы теперь почти готовы к обучению сети. Осталось задать обучающее множество и количество проходов, равное 50.

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
p = num2cell(p,1);
t = num2cell(t,1);
net.adaptParam.passes = 50;
tic, [net,a,e] = adapt(net,p,t); toc
elapsed_time =  4.78
a, mse(e)
```

```
elapsed_time =
    3.4600
elapsed_time =
    4.7800
a =
```

```
    [-0.9889]    [-1.0007]    [1.0182]    [0.9917]
ans =
  1.3116e-004
```

Эти результаты сравнимы с результатами работы алгоритма GD, рассмотренного ранее.

Групповое обучение.

Альтернативой последовательной адаптации является групповое обучение, которое основано на применении функции train. В этом режиме параметры сети модифицируются только после того, как реализовано все обучающее множество, и градиенты, рассчитанные для каждого элемента множества, суммируются, чтобы определить приращения настраиваемых параметров.

Для обучения сети на основе алгоритма GDM необходимо использовать M-функцию traingdm взамен функции настройки learngdm. Различие этих двух функций состоит в следующем. Алгоритм функции traingdm суммирует градиенты, рассчитанные на каждом цикле обучения, а параметры модифицируются только после того, как все обучающие данные будут представлены. Если проведено N циклов обучения и для функционала ошибки оказалось выполненным условие $J_N \geq 1.04\, J_{N-1}$, то параметр возмущения mc следует установить в нуль.

Вновь обратимся к той же сети

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingdm');
```

Функция traingdm характеризуется следующими параметрами, заданными по умолчанию

```
net.trainParam


ans =
      epochs: 100
        goal: 0
          lr: 0.0100
    max_fail: 5
          mc: 0.9000
    min_grad: 1.0000e-010
        show: 25
        time: Inf
```

По сравнению с функцией traingd здесь добавляется только один параметр возмущения mc.

Установим следующие значения параметров обучения

```
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.show = 50;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t);
TRAINGDM, Epoch 0/300, MSE 1.93032/1e-005, Gradient 3.53636/1e-010
TRAINGDM, Epoch 50/300, MSE 0.0176601/1e-005, Gradient
0.202784/1e-010
TRAINGDM, Epoch 100/300, MSE 0.000838074/1e-005, Gradient
0.0350156/1e-010
TRAINGDM, Epoch 146/300, MSE 9.1672e-006/1e-005, Gradient
0.00361773/1e-010
TRAINGDM, Performance goal met.
```

На рис. приведен график изменения ошибки обучения в зависимости от числа выполненных циклов.

```
a = sim(net,p)

a =
   -0.9949   -1.0027    1.0011    1.0014
```

Поскольку начальные веса и смещения инициализируются случайным образом, графики ошибок могут отличаться от одной реализации к другой и от графиков на рис. 3.8-3.9 книги.

C72. Алгоритм GDA

Вновь обратимся к той же нейронной сети (рис. 3.7), но будем использовать функцию обучения traingda

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingda');
```

Функция traingda характеризуется следующими параметрами по умолчанию:

```
net.trainParam

ans =
          epochs: 100
```

```
            goal: 0
              lr: 0.0100
          lr_inc: 1.0500
          lr_dec: 0.7000
        max_fail: 5
    max_perf_inc: 1.0400
        min_grad: 1.0000e-006
            show: 25
            time: Inf
```

Установим следующие значения этих параметров и выполним обучение:

```
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.show = 50;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); % Рис.3.10
TRAINGDA, Epoch 0/300, MSE 8.20691e-006/1e-005, Gradient
0.0024486/1e-006
TRAINGDA, Performance goal met.
```

Теперь выполним моделирование обученной нейронной сети:

```
a = sim(net,p)
a =
   -0.9989   -1.0007    0.9955    1.0032
```

С73. Алгоритм Rprop

Алгоритм Rprop использует функцию обучения trainrp

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'trainrp');
```

Функция trainrp характеризуется следующими параметрами, заданными по умолчанию:

```
net.trainParam


ans =
      epochs: 100
        show: 25
        goal: 0
        time: Inf
    min_grad: 1.0000e-006
    max_fail: 5
```

```
    delt_inc: 1.2000
    delt_dec: 0.5000
      delta0: 0.0700
    deltamax: 50
```

Установим следующие значения этих параметров:

```
net.trainParam.show = 10;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); % Рис.3.11
TRAINRP, Epoch 0/300, MSE 2.27506/1e-005, Gradient 3.27134/1e-006
TRAINRP, Epoch 10/300, MSE 0.0190078/1e-005, Gradient 0.28722/1e-
006
TRAINRP, Epoch 20/300, MSE 6.80914e-005/1e-005, Gradient
0.0217624/1e-006
TRAINRP, Epoch 23/300, MSE 6.8422e-006/1e-005, Gradient
0.00289721/1e-006
TRAINRP, Performance goal met.
a = sim(net,p)
a =
   -0.9957   -1.0019    0.9995    1.0023
```

Алгоритмы метода сопряженных градиентов.

С76. Алгоритм CGF

Алгоритм CGF использует функцию обучения traincgf

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traincgf');
```

Функция traincgf по умолчанию характеризуется следующими параметрами:

```
net.trainParam


ans =
       epochs: 100
         show: 25
         goal: 0
         time: Inf
     min_grad: 1.0000e-006
     max_fail: 5
    searchFcn: 'srchcha'
    scale_tol: 20
```

```
        alpha: 0.0010
         beta: 0.1000
        delta: 0.0100
         gama: 0.1000
      low_lim: 0.1000
       up_lim: 0.5000
      maxstep: 100
      minstep: 1.0000e-006
         bmax: 26
```

Установим следующие значения параметров:

```
net.trainParam.epochs = 300;
net.trainParam.show = 5;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); %Рис.3.12
```

```
TRAINCGF-srchcha, Epoch 0/300, MSE 0.711565/1e-005, Gradient
1.7024/1e-006
TRAINCGF-srchcha, Epoch 5/300, MSE 0.000975456/1e-005, Gradient
0.0253333/1e-006
TRAINCGF-srchcha, Epoch 10/300, MSE 3.06365e-005/1e-005, Gradient
0.00748491/1e-006
TRAINCGF-srchcha, Epoch 12/300, MSE 7.50883e-006/1e-005, Gradient
0.00189117/1e-006
TRAINCGF, Performance goal met.
a = sim(net,p)
a =
   -1.0021   -0.9963    1.0029    0.9981
```

С78. Алгоритм CGP

Алгоритм CGP использует функцию обучения traincgp

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traincgp');
```

Изменим установку следующих параметров:

```
net.trainParam.epochs = 300;
net.trainParam.show = 5;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); %Рис.3.13
```

```
TRAINCGP-srchcha, Epoch 0/300, MSE 3.6913/1e-005, Gradient
4.54729/1e-006
TRAINCGP-srchcha, Epoch 5/300, MSE 0.00160249/1e-005, Gradient
0.077521/1e-006
TRAINCGP-srchcha, Epoch 8/300, MSE 9.42009e-006/1e-005, Gradient
0.00863694/1e-006
TRAINCGP, Performance goal met.
a = sim(net,p)
a =
   -1.0042   -0.9976    0.9975    0.9972
```

C79. Алгоритм CGB

Рассмотрим работу этого алгоритма на примере следующей нейронной сети:

```
clear
   net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traincgb');
```

Изменим установку следующих параметров:

```
net.trainParam.epochs = 300;
net.trainParam.show = 5;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); % Рис.3.14
```

```
TRAINCGB-srchcha, Epoch 0/300, MSE 1.65035/1e-005, Gradient
2.94633/1e-006
TRAINCGB-srchcha, Epoch 5/300, MSE 0.000813691/1e-005, Gradient
0.0222647/1e-006
TRAINCGB-srchcha, Epoch 10/300, MSE 0.000142255/1e-005, Gradient
0.00672937/1e-006
TRAINCGB-srchcha, Epoch 15/300, MSE 1.38893e-005/1e-005, Gradient
0.00260293/1e-006
TRAINCGB-srchcha, Epoch 16/300, MSE 7.09083e-006/1e-005, Gradient
0.0025014/1e-006
TRAINCGB, Performance goal met.
a = sim(net,p)
a =
   -0.9985   -0.9987    0.9961    1.0030
```

C80. Алгоритм SCG

Алгоритм SCG использует функцию обучения trainrp

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'trainscg');
```

Функция trainrp по умолчанию характеризуется следующими параметрами:

```
net.trainParam

ans =
      epochs: 100
        show: 25
        goal: 0
        time: Inf
    min_grad: 1.0000e-006
    max_fail: 5
       sigma: 5.0000e-005
      lambda: 5.0000e-007
```

Изменим установки некоторых параметров:

```
net.trainParam.epochs = 300;
net.trainParam.show = 10;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); %Рис.3.15
TRAINSCG, Epoch 0/300, MSE 1.71149/1e-005, Gradient 2.6397/1e-006
TRAINSCG, Epoch 10/300, MSE 8.2375e-005/1e-005, Gradient
0.0100708/1e-006
TRAINSCG, Epoch 12/300, MSE 1.58889e-006/1e-005, Gradient
0.00321225/1e-006
TRAINSCG, Performance goal met.
a = sim(net,p)
a =
   -1.0007   -1.0009    1.0022    0.9994
```

Квазиньютоновы алгоритмы

С81. Алгоритм BFGS

Алгоритм BFGS использует функцию обучения trainbfg

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'trainbfg');
```

Установим параметры обучающей процедуры по аналогии с предшествующими примерами:

```
net.trainParam.epochs = 300;
net.trainParam.show = 5;
```

```
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); %Рис.3.16

TRAINBFG-srchbac, Epoch 0/300, MSE 0.469151/1e-005, Gradient
1.4258/1e-006
TRAINBFG-srchbac, Epoch 5/300, MSE 0.00212634/1e-005, Gradient
0.0981181/1e-006
TRAINBFG-srchbac, Epoch 10/300, MSE 2.12166e-005/1e-005, Gradient
0.0119488/1e-006
TRAINBFG-srchbac, Epoch 12/300, MSE 2.687e-007/1e-005, Gradient
0.000475672/1e-006
TRAINBFG, Performance goal met.
a = sim(net,p)
a =
   -0.9994   -1.0007    0.9998    1.0003
```

C82. Алгоритм OSS

Алгоритм OSS использует функцию обучения trainoss

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'trainoss');
```

Установим параметры обучающей процедуры по аналогии с предшествующими примерами:

```
net.trainParam.epochs = 300;
net.trainParam.show = 5;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=train(net,p,t); %Рис.3.17
TRAINOSS-srchbac, Epoch 0/300, MSE 2.15911/1e-005, Gradient
3.17681/1e-006
TRAINOSS-srchbac, Epoch 5/300, MSE 0.315332/1e-005, Gradient
1.12347/1e-006
TRAINOSS-srchbac, Epoch 10/300, MSE 0.0294864/1e-005, Gradient
0.553976/1e-006
TRAINOSS-srchbac, Epoch 15/300, MSE 0.000315263/1e-005, Gradient
0.0220646/1e-006
TRAINOSS-srchbac, Epoch 20/300, MSE 1.26224e-006/1e-005, Gradient
0.000784773/1e-006
```

```
TRAINOSS, Performance goal met.
a = sim(net,p)
a =
   -0.9987   -1.0006    0.9986    1.0010
```

С83. Алгоритм LM

Алгоритм LM использует функцию обучения trainlm:

```
clear
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'trainlm');
```

Функция trainlm по умолчанию характеризуется следующими параметрами:

```
net.trainParam


ans =
       epochs: 100
         goal: 0
     max_fail: 5
    mem_reduc: 1
     min_grad: 1.0000e-010
           mu: 0.0010
       mu_dec: 0.1000
       mu_inc: 10
       mu_max: 1.0000e+010
         show: 25
         time: Inf
```

Установим параметры обучающей процедуры по аналогии с предшествующими примерами:

```
net.trainParam.epochs = 300;
net.trainParam.show = 5;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net = train(net,p,t); %Рис.3.18


TRAINLM, Epoch 0/300, MSE 2.99886/1e-005, Gradient 7.55892/1e-010
TRAINLM,   Epoch   4/300,   MSE   4.88189e-009/1e-005,   Gradient
0.000243573/1e-010
TRAINLM, Performance goal met.
a = sim(net,p)
a =
   -1.0000   -1.0000    1.0001    1.0000
```

Расширение возможностей процедур обучения.

С89. Переобучение.

Рассмотрим нейронную сеть типа 1-30-1 с 1 входом, 30 скрытыми нейронами и 1 выходом, которую необходимо обучить аппроксимировать функцию синуса, если заданы ее приближенные значения, которые получены как сумма значений синуса и случайных величин, распределенных по нормальному закону

```
clear
net = newff([-1 1],[30,1],{'tansig','purelin'},'trainbfg');
net.trainParam.epochs = 500;
net.trainParam.show = 50;
net.trainParam.goal = 1e-5;
p = [-1:0.05:1];
t = sin(2*pi*p)+0.1*randn(size(p));
t1 = sin(2*pi*p);
[net,tr] = train(net,p,t); %Рис.3.19
```

```
TRAINBFG-srchbac, Epoch 0/500, MSE 7.31152/1e-005, Gradient
14.7762/1e-006
TRAINBFG-srchbac, Epoch 50/500, MSE 0.000354095/1e-005, Gradient
0.00393802/1e-006
TRAINBFG-srchbac, Epoch 100/500, MSE 1.86929e-005/1e-005, Gradient
0.000200169/1e-006
TRAINBFG-srchbac, Epoch 105/500, MSE 9.84507e-006/1e-005, Gradient
0.000336686/1e-006
TRAINBFG, Performance goal met.
```

Для того чтобы убедиться, что в данном случае мы сталкиваемся с явлением переобучения, построим графики аппроксимируемой функции, ее приближенных значений и результата аппроксимации:

```
an = sim(net,p);
figure(1); plot(p,t,'+',p,an,'-',p,t1,':') % Рис.3.20
legend('вход','выход','sin(2pi*t)'); grid on
```

С91. Метод регуляризации.

Рассмотрим нейронную сеть типа 1-5-1 с 1 входом, 5 скрытыми нейронами и 1 выходом. Эта сеть использует те же типы нейронов, как и предыдущая сеть, но количество скрытых нейронов в 6 раз меньше, предельное количество циклов обучения сокращено до 100, а параметр регуляризации равен 0.9998.

```
clear
net = newff([-1 1],[5,1],{'tansig','purelin'},'trainbfg');
net.trainParam.epochs = 100;
net.trainParam.show = 50;
net.trainParam.goal = 1e-5;
net.performFcn = 'msereg';
net.performParam.ratio = 0.9998;
p = [-1:0.05:1];
t = sin(2*pi*p)+0.1*randn(size(p));
t1 = sin(2*pi*p);
[net,tr] = train(net,p,t); %Рис.3.21,а

TRAINBFG-srchbac, Epoch 0/100, MSEREG 1.44168/1e-005, Gradient
4.02109/1e-006
TRAINBFG-srchbac, Epoch 50/100, MSEREG 0.00943887/1e-005, Gradient
0.00377816/1e-006
TRAINBFG-srchbac, Epoch 100/100, MSEREG 0.00747179/1e-005,
Gradient 0.00630789/1e-006
TRAINBFG, Maximum epoch reached, performance goal was not met.
an = sim(net,p);
figure(1); plot(p,t,'+',p,an,'-',p,t1,':') % Рис.3.21,б
legend('вход','выход','sin(2pi*t)'); grid on
```

Автоматическая регуляризация.

Вновь обратимся к нейронной сети типа 1-20-1, предназначенной для решения задачи аппроксимации функции синуса.

```
clear
net = newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
```

Функция trainbr по умолчанию характеризуется следующими параметрами:

```
net.trainParam
ans =
       epochs: 100
         show: 25
         goal: 0
         time: Inf
     min_grad: 1.0000e-010
     max_fail: 5
    mem_reduc: 1
           mu: 0.0050
       mu_dec: 0.1000
       mu_inc: 10
```

```
        mu_max: 1.0000e+010
```

Установим следующие значения этих параметров:

```
net = newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
net.trainParam.epochs = 50;
net.trainParam.show = 10;
randn('seed',192736547);
p = [-1:.05:1];
t = sin(2*pi*p)+0.1*randn(size(p));
net = init(net);
net = train(net,p,t); % Рис.3.22
TRAINBR, Epoch 0/50, SSE 69.5303/0, SSW 21463.6, Grad
1.19e+002/1.00e-010, #Par 6.10e+001/61
TRAINBR, Epoch 10/50, SSE 0.0765763/0, SSW 6859.43, Grad 8.53e-
003/1.00e-010, #Par 3.40e+001/61
TRAINBR, Epoch 20/50, SSE 0.0602212/0, SSW 5018.79, Grad 3.18e-
001/1.00e-010, #Par 3.43e+001/61
TRAINBR, Epoch 30/50, SSE 0.0577611/0, SSW 4530.75, Grad 1.59e-
002/1.00e-010, #Par 3.38e+001/61
TRAINBR, Epoch 40/50, SSE 0.0761021/0, SSW 2622.33, Grad 7.98e-
002/1.00e-010, #Par 3.17e+001/61
TRAINBR, Epoch 50/50, SSE 0.0800647/0, SSW 2355.29, Grad 3.70e-
002/1.00e-010, #Par 3.13e+001/61
TRAINBR, Maximum epoch reached.
```

Построим графики исследуемых функций:

```
an = sim(net,p);  t1 = sin(2*pi*p);
figure(1); plot(p,t,'+',p,an,'-',p,t1,':') % Рис.3.22
legend('вход','выход','sin(2pi*t)'); grid on
```

С94. Формирование представительной выборки

Сформируем обучающее подмножество на интервале входных значений от -1 до 1 с шагом 0.05 в виде суммы функции синуса и погрешности, описываемой случайной величиной, распределенной по нормальному закону с дисперсией 0.01

```
 clear
p = [-1:0.05:1];
t = sin(2*pi*p)+ 0.1*randn(size(p));
```

Затем сформируем контрольное подмножество. Определим входы в диапазоне от –0.975 до 0.975 и чтобы сделать задачу более реалистичной, добавим некоторую помеху, распределенную по нормальному закону

```
v.P = [-0.975:.05:0.975];
```

```
v.T = sin(2*pi*v.P)+0.1*randn(size(v.P));
```

Тестовое подмножество в данном примере не используется.

Вновь сформируем нейронную сети типа 1-20-1 и обучим ее. Обратите внимание, что контрольное подмножество в виде массива структуры передается функции обучения в качестве шестого входного параметра. В данном случае используется обучающая функция traingdx, хотя может быть применена и любая другая функция обучения

```
net = newff([-1 1],[20,1],{'tansig','purelin'},'traingdx');
net.trainParam.epochs = 300;
net.trainParam.show = 25;
net = init(net);
[net,tr] = train(net,p,t,[],[],v); % Рис.3.24
grid on
legend('контрольное подмножество', 'обучающее подмножество')
```

```
TRAINGDX, Epoch 0/300, MSE 12.9244/0, Gradient 20.8895/1e-006
TRAINGDX, Epoch 25/300, MSE 0.500121/0, Gradient 0.978807/1e-006
TRAINGDX, Epoch 50/300, MSE 0.191717/0, Gradient 0.32784/1e-006
TRAINGDX, Epoch 75/300, MSE 0.0734146/0, Gradient 0.116014/1e-006
TRAINGDX, Epoch 100/300, MSE 0.0218068/0, Gradient 0.0437681/1e-
006
TRAINGDX, Epoch 125/300, MSE 0.00574346/0, Gradient 0.0485314/1e-
006
TRAINGDX, Epoch 130/300, MSE 0.00574346/0, Gradient 0.0485314/1e-
006
TRAINGDX, Validation stop.
```

Построим графики исследуемых функций

```
an = sim(net,p);
t1 = sin(2*pi*p);
figure(2); plot(p,t,'+',p,an,'-',p,t1,':') % Рис.3.25
legend('вход','выход','sin(2pi*t)'); grid on
```

С96. Предварительная обработка и восстановление данных.

Регрессионный анализ.

Следующая последовательность операторов поясняет, как можно выполнен регрессионный анализ для сети, построенной на основе процедуры с прерыванием обучения

```
clear
```

```
p = [-1:0.05:1];
t = sin(2*pi*p)+ 0.1*randn(size(p));
v.P = [-0.975:.05:0.975];
v.T = sin(2*pi*v.P)+0.1*randn(size(v.P));
net = newff([-1 1],[20,1],{'tansig','purelin'},'traingdx');
net.trainParam.show = 25;
net.trainParam.epochs = 300;
net = init(net);
[net,tr]=train(net,p,t,[],[],v);
grid on
legend('контрольное подмножество', 'обучающее подмножество')
```

```
TRAINGDX, Epoch 0/300, MSE 8.33909/0, Gradient 13.9129/1e-006
TRAINGDX, Epoch 25/300, MSE 1.48273/0, Gradient 2.00314/1e-006
TRAINGDX, Epoch 50/300, MSE 0.361276/0, Gradient 0.543845/1e-006
TRAINGDX, Epoch 75/300, MSE 0.0769917/0, Gradient 0.147463/1e-006
TRAINGDX, Epoch 100/300, MSE 0.0181576/0, Gradient 0.0458443/1e-
006
TRAINGDX, Epoch 125/300, MSE 0.00440005/0, Gradient 0.0691844/1e-
006
TRAINGDX, Epoch 138/300, MSE 0.00427574/0, Gradient 0.033685/1e-
006
TRAINGDX, Validation stop.
```

```
a = sim(net,p); % Моделирование сети
figure(2);
[m,b,r] = postreg(a,t), grid on
```

```
m =
    0.9894
b =
   -0.0092
r =
    0.9956
```

Пример процедуры обучения.

Требуется разработать вычислительный инструмент, который может определять уровни липидных составляющих холестерина на основе измерений спектра крови. Имеется статистика измерения 21 показателя липидного спектра крови для 264 пациентов. Кроме того, известны уровни hdl, ldl и vldl липидных

составляющих холестерина, основанных на сепарации сыворотки. Необходимо определить состоятельность нового способа анализа крови, основанного на измерении ее спектра.

Данные измерений спектра должны быть загружены из MAT-файла choles_all и подвергнуты факторному анализу

```
clear, load choles_all
[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);
[ptrans,transMat] = prepca(pn, 0.001);
```

В этом случае сохраняются только те компоненты, которые объясняют 99.9 % изменений в наборе данных. Проверим, сколько же компонентов из первоначальных 21 являются состоятельными

```
[R,Q] = size(ptrans)
R =
     4
Q =
   264
```

Оказывается, что всего 4.

Теперь сформируем из исходной выборки обучающее, контрольное и тестовое множества. Для этого выделим половину выборки для обучающего множества ptr и по четверти для контрольного v и тестового t

```
iitst = 2:4:Q;
iival = 4:4:Q;
iitr = [1:4:Q 3:4:Q];
v.P = ptrans(:,iival); v.T = tn(:,iival);
t.P = ptrans(:,iitst); t.V = tn(:,iitst);
ptr = ptrans(:,iitr); ttr = tn(:,iitr);
```

Теперь необходимо сформировать и обучить нейронную сеть. Будем использовать сеть с 2 слоями, с функциями активации: в скрытом слое - гиперболический тангенс, в выходном слое - линейная функция. Такая структура эффективна для решения задач аппроксимации и регрессии. В качестве начального приближения установим 5 нейронов в скрытом слое. Сеть должна иметь 3 выходных нейрона, поскольку определено 3 цели. Для обучения применим алгоритм LM

```
net = newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm')
[net,tr]=train(net,ptr,ttr,[],[],v,t);
```

```
net =
    Neural Network object:
    architecture:
         numInputs: 1
         numLayers: 2
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]
     targetConnect: [0 1]

        numOutputs: 1  (read-only)
        numTargets: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)
    subobject structures:
            inputs: {1x1 cell} of inputs
            layers: {2x1 cell} of layers
           outputs: {1x2 cell} containing 1 output
           targets: {1x2 cell} containing 1 target
            biases: {2x1 cell} containing 2 biases
      inputWeights: {2x1 cell} containing 1 input weight
      layerWeights: {2x2 cell} containing 1 layer weight
    functions:
          adaptFcn: 'trains'
           initFcn: 'initlay'
        performFcn: 'mse'
          trainFcn: 'trainlm'
    parameters:
        adaptParam: .passes
         initParam: (none)
      performParam: (none)
        trainParam: .epochs, .goal, .max_fail, .mem_reduc,
                    .min_grad, .mu, .mu_dec, .mu_inc,
                    .mu_max, .show, .time
    weight and bias values:
                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors
    other:
          userdata: (user stuff)
TRAINLM, Epoch 0/100, MSE 1.66603/0, Gradient 576.465/1e-010
TRAINLM, Epoch 12/100, MSE 0.314181/0, Gradient 88.4175/1e-010
```

```
TRAINLM, Validation stop.
```

Обучение остановлено, потому что контрольная ошибка в 5 раз превысила ошибку обучения. Построим графики всех ошибок

```
figure(2)
plot(tr.epoch,tr.perf,'-',tr.epoch,tr.vperf,'--',tr.epoch,tr.tperf,':');
legend('Обучение', 'Контроль', 'Проверка'); grid on %Рис.3.27
```

Погрешности проверки на тестовом и контрольном множествах ведут себя одинаково, и нет заметных тенденций к переобучению.

Теперь следует выполнить анализ реакции сети. Используем весь набор данных (обучение, признание выборки представительной и тестовый) и оценим линейную регрессию между выходами сети и соответствующими целями. Сначала нужно перейти к ненормализованным выходам сети.

```
an = sim(net,ptrans); % Моделирование сети
a = poststd(an,meant,stdt); % Восстановление выходов
t = poststd(tn,meant,stdt); % Восстановление целей
for i=1:3
   figure(i)
   [m(i),b(i),r(i)] = postreg(a(i,:),t(i,:)); % Расчет регрессии
end
```

Первые два выхода сети (hdl и ldl составляющие) хорошо отслеживают целевое множество, значение близко к 0.9, и это означает, что эти липидные характеристики могут быть определены по измерениям спектра крови. Третий выход (vldl составляющая) восстанавливается плохо (коэффициент корреляции около 0.6), и это означает, что решение задачи должно быть продолжено.

Можно использовать другую архитектуру сети, увеличить количество нейронов в скрытом слое (больше скрытых слоев нейронов) или воспользоваться методом регуляризации. Хотя может оказаться, что восстановление составляющей vldl на основе измерения спектра крови вообще несостоятельно.

Глава 4. Персептроны

C104. Модель персептрона.

Функция

```
clear
net = newp([0 2],1);
```

создает персептрон с одноэлементным входом и одним нейроном; диапазон значений входа - [0 2].

Определим некоторые параметры персептрона, инициализируемые по умолчанию.

Веса входов:

```
inputweights = net.inputweights{1,1}
inputweights =
        delays: 0
       initFcn: 'initzero'
         learn: 1
      learnFcn: 'learnp'
    learnParam: []
          size: [1 1]
      userdata: [1x1 struct]
     weightFcn: 'dotprod'
```

Заметим, что функция настройки персептрона по умолчанию – learnp; вход функции активации вычисляется с помощью функции скалярного произведения dotprod; функция инициализации initzero используется для установки нулевых начальных весов.

Смещения:

```
biases = net.biases{1}
biases =
       initFcn: 'initzero'
         learn: 1
      learnFcn: 'learnp'
    learnParam: []
          size: 1
      userdata: [1x1 struct]
```

Нетрудно увидеть, что начальное смещение также установлено в ноль.

C105. Моделирование персептрона.

Рассмотрим однослойный персептрон с одним двухэлементным вектором входа, значения элементов которого изменяются в диапазоне от -2 до 2

```
clear, net = newp([-2 2;-2 2],1);
```

По умолчанию веса и смещение равны нулю и для того, чтобы установить желаемые значения, необходимо применить следующие операторы

```
net.IW{1,1}= [-1 1];
net.b{1} = [1];
```

В этом случае разделяющая линия имеет вид

L: $-p_1 + p_2 + 1 = 0$.

и соответствует рис. 4.1.

```
gensim(net) % Рис.4.4
```

Структурная схема модели персептрона показана на рис. 4.4

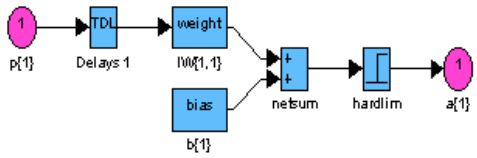

Рис. 4.4

Теперь определим, как откликается сеть на входные векторы p1 и p2, расположенные по разные стороны от разделяющей линии:

```
p1 = [1; 1];
a1 = sim(net,p1) % Моделирование сети net с входным вектором p1
a1 =
     1
p2 = [1; -1];
a2 = sim(net,p2) % Моделирование сети net с входным вектором p2

a2 =
     0
```

Персептрон правильно классифицировал эти два вектора.

Заметим, что можно было бы ввести последовательность двух векторов в виде массива ячеек и получить результат также в виде массива ячеек

```
p3 = {[1; 1] [1; -1]}
a3 = sim(net,p3) % Моделирование сети net при входном сигнале p3

p3 =
    [2x1 double]    [2x1 double]
a3 =
    [1]    [0]
```

C106. Инициализация параметров.

Для однослойного персептрона в качестве параметров нейронной сети в общем случае выступают веса входов и смещения. Допустим, что создается персептрон с двухэлементным вектором входа и одним нейроном

```
clear, net = newp([-2 2;-2 2],1);
```

Запросим характеристики весов входа

```
net.inputweights{1, 1}
ans =
        delays: 0
       initFcn: 'initzero'
         learn: 1
      learnFcn: 'learnp'
    learnParam: []
          size: [1 2]
      userdata: [1x1 struct]
     weightFcn: 'dotprod'
```

Из этого списка следует, что в качестве функции инициализации по умолчанию используется функция initzero, которая присваивает весам входа нулевые значения. В этом можно убедиться, если извлечь значения элементов матрицы весов и смещения

```
wts = net.IW{1,1}, bias = net.b{1}
wts =
     0     0
bias =
     0
```

Теперь переустановим значения элементов матрицы весов и смещения

```
net.IW{1,1} = [3, 4]; net.b{1} = 5;
wts = net.IW{1,1}, bias = net.b{1}
wts =
     3     4
bias =
     5
```

Для того чтобы вернуться к первоначальным установкам параметров персептрона, и предназначена функция init

```
net = init(net); wts = net.IW{1,1}, bias = net.b{1}


wts =
     0     0
bias =
     0
```

Можно изменить способ, каким инициализируется персептрон с помощью функции init. Для этого достаточно изменить тип функций инициализации, которые применяются для установки первоначальных значений весов входов и

смещений. Например, воспользуемся функцией инициализации rands, которая устанавливает случайные значения параметров персептрона.

```
% Задать функции инициализации весов и смещений
net.inputweights{1,1}.initFcn = 'rands';
net.biases{1}.initFcn = 'rands';
% Выполнить инициализацию ранее созданной сети с новыми функциями
net = init(net);
wts = net.IW{1,1}, bias = net.b{1}

wts =
   -0.1886    0.8709
bias =
   -0.6475
```

Видно, что веса и смещения выбраны случайным образом.

С108. Правила настройки параметров персептрона.

Рассмотрим простой пример персептрона с единственным нейроном и двухэлементным вектором входа

```
clear, net = newp([-2 2;-2 2],1);
```

Определим смещение b равным 0, а вектор весов **w** равным [1 -0.8]

```
net.b{1} = 0;
w = [1 -0.8];  net.IW{1,1} = w;
```

Обучающее множество зададим следующим образом

```
p = [1; 2]; t = [1];
```

Моделируя персептрон, рассчитаем выход и ошибку на первом шаге настройки

```
a = sim(net,p), e = t-a

a =
     0
e =
     1
```

Используя М-функцию настройки параметров learnp, найдем требуемое изменение весов

```
dw = learnp(w,p,[ ],[ ],[ ],[ ],e,[ ],[ ],[ ])
dw =
     1     2
```

Тогда новый вектор весов примет вид

```
w = w + dw
w =
    2.0000    1.2000
```

С110. Процедура адаптации.

. Вновь сформируем модель персептрона, изображенного на рис. 4.5

```
clear, net = newp([-2 2;-2 2],1);
```

Введем первый элемент обучающего множества

```
p = {[2; 2]}; t = {0};
```

Установим параметр passes (число проходов), равным 1, и выполним один шаг настройки

```
net.adaptParam.passes = 1;
[net,a,e] = adapt(net,p,t); a,e
a =
    [1]
e =
    [-1]
```

Скорректированные вектор весов и смещение равны

```
twts = net.IW{1,1}, tbiase = net.b{1}
twts =
    -2    -2
tbiase =
    -1
```

Это совпадает с результатами, полученными при ручном расчете. Теперь можно ввести второй элемент обучающего множества и т.д., то есть повторить всю процедуру ручного счета и получить те же результаты.

Но можно эту работу выполнить автоматически, задав сразу все обучающее множество и выполнив один проход

```
clear, net = newp([-2 2;-2 2],1);
net.trainParam.passes = 1;
p = {[2;2] [1;-2] [-2;2] [-1;1]};
t = {0 1 0 1};
```

Теперь обучим сеть.

```
[net,a,e] = adapt(net,p,t);
```

Возвращаются выход и ошибка

```
a, e
a =
    [1]    [1]    [0]    [0]
```

```
e =
    [-1]    [0]    [0]    [1]
```

Скорректированные вектор весов и смещение равны

```
twts = net.IW{1,1}, tbiase = net.b{1}
twts =
    -3    -1
tbiase =
     0
```

Моделируя полученную сеть по каждому входу, получим

```
a1 = sim(net,p)

a1 =
    [0]    [0]    [1]    [1]
```

Можно убедиться, что не все выходы равны целевым значениям обучающего множества. Это означает, что следует продолжить настройку персептрона.

Выполним еще один цикл настройки

```
[net,a,e] = adapt(net,p,t); a, e

a =
    [0]    [0]    [0]    [1]
e =
    [0]    [1]    [0]    [0]
twts = net.IW{1,1}, tbiase = net.b{1}
twts =
    -2    -3
tbiase =
     1
a1 = sim(net,p)

a1 =
    [0]    [1]    [0]    [1]
```

Теперь решение совпадает с целевыми выходами обучающего множества, и все входы классифицированы правильно.

Глава 5. Линейные сети

C117. Создание модели линейной сети.

Линейную сеть с одним нейроном, показанную на рис. 5.1, можно создать следующим образом

```
clear, net = newlin([-1 1; -1 1],1);
```

Первый входной аргумент задает диапазон изменения элементов вектора входа; второй аргумент указывает, что сеть имеет единственный выход. Начальные веса и смещение по умолчанию равны нулю.

Присвоим весам и смещению следующие значения

```
net.IW{1,1} = [2 3]; net.b{1} =[-4];
```

Теперь можно промоделировать линейную сеть для следующего предъявленного вектора входа

```
p = [5;6];
a = sim(net,p)


a =
    24
```

С118. Обучение линейной сети.

Процедура настройки.

В отличие от многих других сетей настройка линейной сети для заданного обучающего множества может быть выполнена посредством прямого расчета с использованием М-функции newlind.

Предположим, что заданы следующие векторы, принадлежащие обучающему множеству

```
clear, P = [1 -1.2];  T = [0.5 1];
```

Построим линейную сеть и промоделируем ее

```
net = newlind(P,T); net.IW{1,1}, net.b
Y = sim(net, P)
ans =
   -0.2273
ans =
    [0.7273]
Y =
    0.5000    1.0000
```

Выход сети соответствует целевому вектору.

Зададим следующий диапазон весов и смещений, рассчитаем критерий качества обучения и построим его линии уровня:

```
w_range=-1:0.1: 0; b_range=0.5:0.1:1;
ES = errsurf(P,T, w_range, b_range, 'purelin');
contour(w_range, b_range,ES,20)
```

```
hold on
plot(-2.2727e-001,7.2727e-001, 'x') % Рис.5.4.
hold off
```

На графике знаком ""x" отмечены оптимальные значения веса и смещения для данной сети.

С120. Процедура обучения.

Обратимся к тому же примеру, который использовался при рассмотрении процедуры адаптации, и выполним процедуру обучения.

```
clear, P = [1 -1.2];% Вектор входов
T= [0.5, 1]; % Вектор целей
```

% **Максимальное значение параметра обучения**

```
maxlr = 0.40*maxlinlr(P,'bias');
```

% **Создание линейной сети**

```
net = newlin([-2,2],1,[0],maxlr);
```

% Расчет функции критерия качества

```
w_range=-1:0.2:1; b_range=-1:0.2:1;
ES = errsurf(P,T, w_range, b_range, 'purelin');
```

% **Построение поверхности функции критерия качества**

```
surfc(w_range, b_range, ES) % Рис.5.5,a)
```

На рис. 5.5, а построена поверхность функции критерия качества в пространстве параметров сети. В процессе обучения траектория обучения будет перемещаться из начальной точки в точку минимума критерия качества. Выполним расчет и построим траекторию обучения линейной сети для заданных начальных значений веса и смещения.

% **Расчет траектории обучении**

```
x = zeros(1,50); y = zeros(1,50);
net.IW{1}=1; net.b{1}= -1;
x(1) = net.IW{1}; y(1) = net.b{1};
net.trainParam.goal = 0.001;
net.trainParam.epochs = 1;
```

% Цикл вычисления весов и смещения для одной эпохи

```
for i = 2:50,
   [net, tr] = train(net,P,T);
   x(i) = net.IW{1};
   y(i) = net.b{1};
end
```

```
TRAINB, Epoch 0/1, MSE 5.245/0.001.
TRAINB, Epoch 1/1, MSE 2.049/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 2.049/0.001.
TRAINB, Epoch 1/1, MSE 0.815178/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.815178/0.001.
TRAINB, Epoch 1/1, MSE 0.330857/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.330857/0.001.
TRAINB, Epoch 1/1, MSE 0.137142/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.137142/0.001.
TRAINB, Epoch 1/1, MSE 0.0580689/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.0580689/0.001.
TRAINB, Epoch 1/1, MSE 0.0250998/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.0250998/0.001.
TRAINB, Epoch 1/1, MSE 0.0110593/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.0110593/0.001.
TRAINB, Epoch 1/1, MSE 0.00495725/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.00495725/0.001.
TRAINB, Epoch 1/1, MSE 0.00225533/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.00225533/0.001.
TRAINB, Epoch 1/1, MSE 0.00103897/0.001.
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.00103897/0.001.
TRAINB, Epoch 1/1, MSE 0.000483544/0.001.
```

```
TRAINB, Maximum epoch reached.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.
```

```
TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.
```

```
TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

TRAINB, Epoch 0/1, MSE 0.000483544/0.001.
TRAINB, Performance goal met.
```

% Построение линий уровня и траектории обучении

```
clf, contour(w_range, b_range, ES, 20), hold on
plot(x, y,'-*'), grid on, hold off, % Рис.5.5,б
```

На рис. 5.5, б символами "*" отмечены значения веса и смещения на каждом шаге обучения; видно, что примерно за 10 шагов при заданной точности (пороговое значение критерия качества) 0.001 получим w = -0.22893, b = 0.70519. Это согласуется с решением, полученным с использованием процедуры адаптации.

Если не строить траектории процесса обучения, то можно выполнить обучение, обратившись к М-функции train только один раз:

```
net.IW{1}=1; net.b{1}= -1;
net.trainParam.epochs = 50;
net.trainParam.goal = 0.001;
[net, tr] = train(net,P,T);
net.IW, net.b

TRAINB, Epoch 0/50, MSE 5.245/0.001.
TRAINB, Epoch 11/50, MSE 0.000483544/0.001.
TRAINB, Performance goal met.

ans =
    [-0.2289]
ans =
    [0.7052]
```

Если повысить точность обучения до значения 0.00001, то получим следующие результаты

```
net.trainParam.goal = 0.00001;
[net, tr] = train(net,P,T);
net.IW, net.b
TRAINB, Epoch 0/50, MSE 0.000483544/1e-005.
TRAINB, Epoch 6/50, MSE 5.55043e-006/1e-005.
TRAINB, Performance goal met.
ans =
    [-0.2279]
ans =
    [0.7249]
```

Повышение точности на два порядка приводит к уточнению значений параметров во втором знаке.

C122. Применение линейных сетей.

Задача классификации векторов.

Определим линейную сеть с начальными значениями веса и смещения, используемыми по умолчанию, то есть нулевыми; зададим допустимую погрешность обучения, равную 0.1:

```
clear, p = [2 1 -2 -1;2 -2 2 1]; t = [0 1 0 1];
net = newlin( [-2 2; -2 2],1);
```

% Инициализация линейной сети с двумя входами и одним выходом

```
net.trainParam.goal= 0.1;
[net, tr] = train(net,p,t);
TRAINB, Epoch 0/100, MSE 0.5/0.1.
```

```
TRAINB, Epoch 25/100, MSE 0.181122/0.1.
TRAINB, Epoch 50/100, MSE 0.111233/0.1.
TRAINB, Epoch 64/100, MSE 0.0999066/0.1.
TRAINB, Performance goal met.
```

Пороговое значение функции качества достигается за 64 цикла обучения, а соответствующие параметры сети принимают значения:

```
weights = net.iw{1,1}, bias = net.b(1)


weights =
   -0.0615   -0.2194
bias =
    [0.5899]
```

Выполним моделирование созданной сети с векторами входа из обучающего множества и вычислим ошибки сети

```
A = sim(net, p)
err = t - sim(net,p)
A =
    0.0282    0.9672    0.2741    0.4320
err =
   -0.0282    0.0328   -0.2741    0.56800
```

Заметим, что погрешности сети весьма значительны. Попытка задать большую точность в данном случае не приводит к цели, поскольку возможности линейной сети ограничены.

С124. Фильтрация сигнала.

Рассмотрим конкретный пример цифрового фильтра, представленный на рис. 5.8.

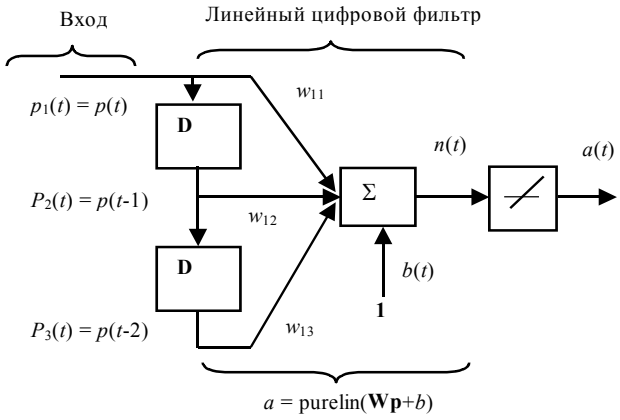

Рис. **Ошибка! Текст указанного стиля в документе отсутствует.**.1

Предположим, что входной сигнал принимает значения в диапазоне от 0 до 10, и сформируем линейную сеть с одним входом и одним выходом, используя М-функцию newlin:

```
clear, net = newlin([0,10],1);
```

Введем ЛЗ с двумя тактами запаздывания

```
net.inputWeights{1,1}.delays = [0 1 2];
```

определим следующие начальные значения весов и смещения

```
net.IW{1,1} = [7 8 9];  net.b{1} = [0];
```

зададим начальные условия для динамических блоков линии задержки

```
pi ={1 2}
```

```
pi =
    [1]    [2]
```

Последовательность их задания слева направо соответствует блокам запаздывания, расположенным на рисунке сверху вниз. Этим завершается формирование сети.

Теперь определим входной сигнал в виде следующей последовательности значений

```
p = {3 4 5 6}
```

```
p =
    [3]    [4]    [5]    [6]
```

и промоделируем эту сеть

```
[a,pf] = sim(net,p,pi)
```

```
a =
    [46]    [70]    [94]    [118]
pf =
    [5]    [6]
```

Для того чтобы получить желаемую последовательность сигналов на выходе, необходимо выполнить настройку сформированной сети. Предположим, что задана следующая желаемая последовательность для выхода фильтра

```
T = {10 20 30 40};
```

Выполним настройку параметров, используя процедуру адаптации adapt и 10 циклов обучения.

```
net.adaptParam.passes = 10;
[net,y,E pf,af] = adapt(net,p,T,pi); % Процедура адаптации
```

Выведем полученные значения весов, смещения и выходного сигнала:

```
wts = net.IW{1,1}, bias = net.b{1}, y
```

```
wts =
    0.5059    3.1053    5.7046
bias =
   -1.5993
y =
    [11.8558]    [20.7735]    [29.6679]    [39.0036]
```

Если продолжить процедуру настройки, то можно еще точнее приблизить выходной сигнал к желаемому

```
net.adaptParam.passes = 500;
[net,y,E,pf,af] = adapt(net,p,T,pi);  y
```

```
y =
    [10.0043]    [20.0018]    [29.9992]    [39.9977]
```

Таким образом, линейные динамические нейронные сети могут быть адаптированы для решения задач фильтрации временных сигналов. Для сетей такого класса часто используется название ADALINE (ADaptive LInear NEtwork) – адаптируемые линейные сети.

C125. Предсказание сигнала.

Попробуем применить сеть ADALINE для предсказания значений детерминированного процесса p(t). Обратимся к рис. 5.9.

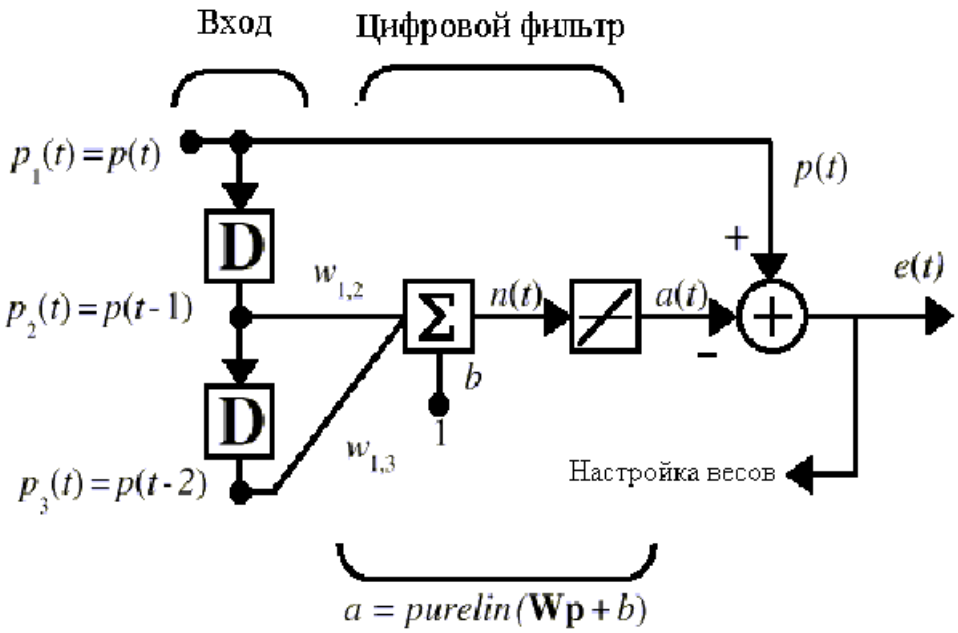

Рис. **Ошибка! Текст указанного стиля в документе отсутствует.**.2

Некоторый сигнал поступает на линию задержки так, что на ее выходе формируются 2 сигнала: p(t-1), p(t-2). Настройка сети реализуется с помощью М-функции adapt, которая изменяет параметры сети на каждом шаге с целью минимизировать погрешность e(t) = a(t) - p(t). Если эта погрешность нулевая, то выход сети a(t) точно равен p(t), и сеть выполняет предсказание должным образом.

Ниже приведен сценарий, который предназначен для решения задачи предсказания сигнала на один шаг вперед. Входной детерминированный процесс получен в результате прохождения ступенчатого сигнала через колебательное звено.

Поскольку для формирования входа применено динамическое звено второго порядка, то в сети ADALINE будет использована ЛЗ с двумя блоками. Запишем следующий сценарий для решения задачи предсказания сигнала.

```
clear
sys = ss(tf(1,[1 1 1])); % Формирование колебательного звена
% Реакция на ступенчатое входное воздействие
time = 0:0.2:10;
[y,time] = step(sys,0:0.2:10);
% Формирование обучающего множества
p = y(1:length(time)-2)';
t = y(3:length(time))';
```

```
time = time(1:length(time)-2);
```

% **Формирование нейронной сети**

```
net = newlin([-1  1],1,[1 2]);
P = num2cell(p);
T = num2cell(t);
```

% **Настройка нейронной сети**

```
pi = {0 0};
net.adaptParam.passes = 5;
[net,Y,E,Pf,Af] = adapt(net,P,T,pi);
Y1 = cat(1,Y{:});
```

% **Построение графиков**

```
figure(1), plot(time,Y1,'b:',time,p,'r-'), grid on
xlabel('ВремЯ, с'), ylabel('Процессы')
title('Обучение нейронной сети')
```

% **Моделирование нейронной сети**

```
x = sim(net,P);
x1 = cat(1,x{:});
figure(2), plot(time,x1,'b:+', time,p,'r-o'),grid on,
legend('выход', 'вход')
```

Найденные значения весов и смещения равны

```
net.IW{1,1}, net.b
ans =
    0.3343    0.3182
ans =
    [0.3585]
```

Глава 6. Радиальные базисные сети

С134. Радиальная базисная сеть с нулевой ошибкой

Пример создания, обучения и моделирования:

```
clear, P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609  .1336 ...
     -.2013 -.4344 -.5000 -.3930 -.1647  .0988  .3072  .3960 ...
      .3449  .1816  -.0312  -.2189  -.3201];
figure(1), plot(P,T,'*r','MarkerSize',4,'LineWidth',2),
grid on, hold on
```

% Создание сети

```
net = newrbe(P,T); % Создание радиальной базисной сети
```

```
Warning: Rank deficient, rank = 13  tol =   2.2386e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrbe.m (designrbe) at line
120
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrbe.m at line 103
net.layers{1}.size % Число нейронов в скрытом слое


ans =
    21
```

% Моделирование сети

```
V = sim(net,P);  % Векторы входа из обучающего множества
plot(P,V,'ob','MarkerSize',5, 'LineWidth',2)
p = [-0.75 -0.25 0.25 0.75];
v = sim(net,p);  % Новый вектор входа
plot(p,v,'+k','MarkerSize',10, 'LineWidth',2)
xlabel('P, p'), ylabel('T, v') %Рис.6.4
```

С135. Итерационная процедура формирования сети.

Применим функцию newrb для создания радиальной базисной сети:

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609  .1336 ...
     -.2013 -.4344 -.5000 -.3930 -.1647  .0988  .3072  .3960 ...
      .3449  .1816  -.0312  -.2189  -.3201];
plot(P,T,'*r','MarkerSize',4,'LineWidth',2), hold on
```

% Создание сети

```
GOAL = 0.01; % Допустимое значение функционала ошибки
net = newrb(P,T,GOAL); % Создание радиальной базисной сети
net.layers{1}.size % Число нейронов в скрытом слое
NEWRB, neurons = 0, SSE = 3.69051
ans =
     6
```

% Моделирование сети

```
V = sim(net,P);  % Векторы входа из обучающего множества
plot(P,V,'ob','MarkerSize',5, 'LineWidth',2)
p = [-0.75 -0.25 0.25 0.75];
v = sim(net,p);  % Новый вектор входа
plot(p,v,'+k','MarkerSize',10, 'LineWidth',2), grid on
xlabel('P, p'), ylabel('T, v') %Рис.6.5,a
```

С137. Примеры радиальных базисных сетей.

Демонстрационный пример demorb1 иллюстрирует применение радиальной базисной сети для решения задачи аппроксимации функции от одной переменной.

Представим функцию $f(x)$ следующим разложением

$$f(x) = \sum_{i=1}^{N} \alpha_i \varphi_i(x), \qquad \textbf{(Ошибка! Текст указанного стиля в документе отсутствует.}.1)$$

где $\varphi_i(x)$ – радиальная базисная функция.

Тогда идея аппроксимации может быть представлена графически следующим образом. Рассмотрим взвешенную сумму трех радиальных базисных функций, заданных на интервале [–3 3].

```
 clear, p = -3:.1:3;
a1 = radbas(p);
a2 = radbas(p - 1.5);
a3 = radbas(p + 2);
a = a1 + a2*1 + a3*0.5;
figure(1), clf,
plot(p,a1,p,a2,p,a3*0.5,'LineWidth',1.5),hold on
plot(p,a,'LineWidth',3,'Color',[1/3,2/3,2/3]),grid on,
legend('a1','a2','a3*0.5','a')
```

Приступим к формированию радиальной базисной сети. Сформируем обучающее множество и зададим допустимое значение функционала ошибки, равное 0.01, параметр влияния определим равным 1 и будем использовать итерационную процедуру формирования радиальной базисной сети

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609  .1336 ...
     -.2013 -.4344 -.5000 -.3930 -.1647  .0988  .3072  .3960 ...
      .3449  .1816  -.0312  -.2189  -.3201];
GOAL = 0.01;
SPREAD = 1;
net = newrb(P,T,GOAL,SPREAD); % Создание сети
net.layers{1}.size % Число нейронов в скрытом слое
NEWRB, neurons = 0, SSE = 3.69051
ans =
     6
```

Для заданных параметров нейронная сеть состоит из 6 нейронов и обеспечивает следующие возможности аппроксимации нелинейных зависимостей после обучения. Моделируя сформированную нейронную сеть, построим аппроксимацинную кривую на интервале [-1 1] с шагом 0.01 для нелинейной зависимости.

```
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = -1:.01:1;
Y = sim(net,X); % Моделирование сети
plot(X,Y,'LineWidth',2), grid on % Рис. 6.7
```

В демонстрационных примерах demorb3 и demorb4 исследуется влияние параметра SPREAD на структуру радиальной базисной сети и качество аппроксимации. В демонстрационном примере demorb3 параметр влияния SPREAD установлен равным 0.01. Это означает, что диапазон перекрытия входных значений составлет лишь ±0.01, а поскольку обучающие входы заданы с интервалом 0.1, то входные значения функциями активации не перекрываются.

```
GOAL = 0.01;
SPREAD = 0.01;
net = newrb(P,T,GOAL,SPREAD);
net.layers{1}.size % Число нейронов в скрытом слое
NEWRB, neurons = 0, SSE = 2.758
ans =
    19
```

Это приводит к тому, что, во-первых, увеличивается количество нейронов скрытого слоя с 6 до 19, а во-вторых, не обеспечиваетя необходимой гладкости аппроксимируемой функции

```
figure(2), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = -1:.01:1;
Y = sim(net,X); % Моделирование сети
plot(X,Y,'LineWidth',2), grid on % Рис. 6.8
```

Пример demorb4 иллюстрирует противоположный случай, когда параметр влияния SPREAD выбирается достаточно большим (в данном примере – 12 или больше), то все функции активации перекрываются и каждый базисный нейрон

выдает значение, близкое к 1, для всех значений входов. Это приводит к тому, что сеть не реагирует на входные значения. Функция newrb будет пытаться строить сеть, но не сможет обеспечить необходимой точности из-за вычислительных проблем.

```
GOAL = 0.01; SPREAD = 12;
net = newrb(P,T,GOAL,SPREAD);
net.layers{1}.size
NEWRB, neurons = 0, SSE = 3.6997
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
```

```
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 6  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 6  tol =   2.1368e-014.
> In  C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m  (solvelin2)  at  line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 5.304273e-020.
```

```
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
Warning: Rank deficient, rank = 5  tol =   2.2386e-014.
> In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (solvelin2) at line
243
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m (designrb) at line 200
  In C:\MATLAB6P1\toolbox\nnet\nnet\newrb.m at line 130
ans =
    21
```

В процессе вычислений возникают трудности с обращением матриц и об этом выдаются предупреждения; количество нейронов скрытого слоя устанавливается равным 21, а точность аппроксимации оказывается недопустимо низкой

```
figure(3), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = -1:.01:1;
Y = sim(net,X); % Моделирование сети
plot(X,Y,'LineWidth',2), grid on % Рис. 6.9
```

Вывод из выполненного исследования состоит в том, что параметр влияния SPREAD следует выбирать большим, чем шаг разбиения интервала задания обучающей последовательности, но меньшим размера самого интервала. Для данной задачи это означает, что параметр влияния SPREAD должен быть больше 0.1 и меньше 2.

C140. Сети GRNN.

Для создания нейронной сети GRNN предназначена M-функция newgrnn. Зададим следующее обучающее множество векторов входа и целей и построим сеть GRNN

```
clear, P = [4 5 6]; T = [1.5 3.6 6.7];
net = newgrnn(P,T);
net.layers{1}.size % Число нейронов в скрытом слое
```

```
ans =
     3
```

Эта сеть имеет 3 нейрона в скрытом слое. Промоделируем построенную сеть сначала для одного входа, а затем для последовательности входов из интервала [4 7]

```
p = 4.5; v = sim(net,p);
p1 = 4:0.1:7; v1 = sim(net,p1);
figure(1), clf
plot(P,T,'*g',p,v,'or',p1,v1,'-b','MarkerSize',8,'LineWidth',2)
grid on % Рис.6.11
```

Демонстрационная программа demogrn1 иллюстрирует, как сети GRNN решают задачи аппроксимации. Определим обучающее множество в виде массивов P и T.

```
clear, P = [1 2 3 4 5 6 7 8]; T = [0 1 2 3 2 1 2 1];
```

Примем значение параметра влияния SPREAD немного меньшим, чем шаг задания аргумента функции (в данном случае 1), чтобы построить аппроксимирующую кривую, близкую к заданным точкам. Как мы уже видели ранее, чем меньше значение параметра SPREAD, тем ближе точки аппроксимирующей кривой к заданным, но тем менее гладкой является сама кривая

```
spread = 0.7;
net = newgrnn(P,T,spread);
net.layers{1}.size % Число нейронов в скрытом слое

ans =
     8
A = sim(net,P);
figure(2), clf,
plot(P,T,'*b','markersize',10), hold on,
plot(P,A,'or','markersize',10); grid on % Рис.6.12
```

Моделирование сети для диапазона значений аргумента позволяет увидеть всю аппроксимирующую кривую, причем возможна экстраполяция этой кривой за пределы области ее определения. Для этого зададим интервал аргумента в диапазоне [-1 10]

```
P2 = -1:0.1:10; A2 = sim(net,P2);
figure(3), clf, plot(P2,A2,'-b','linewidth',2) % Рис.6.13
hold on, plot(P,T,'*r','markersize',10), grid on
```

С144. Сети PNN.

Для создания нейронной сети PNN предназначена М-функция newpnn. Определим 7 следующих векторов входа и соотнесем каждый из них к одному из 3 классов

```
clear, P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3]'; Tc = [1 1 2 2 3 3 3];
```

Вектор Tc назовем *вектором индексов классов*. Этому индексному вектору можно поставить в соответствие матрицу связности T в виде разреженной матрицы вида

```
T = ind2vec(Tc)
T =
   (1,1)        1
   (1,2)        1
   (2,3)        1
   (2,4)        1
   (3,5)        1
   (3,6)        1
   (3,7)        1
```

которая определяет принадлежность первых двух векторов классу 1, двух последующих– классу 2 и трех последних – классу 3. Полная матрица T имеет вид

```
T = full(T)
T =
     1     1     0     0     0     0     0
     0     0     1     1     0     0     0
     0     0     0     0     1     1     1
```

Массивы P и T задают обучающее множество, что позволяет выполнить формирование сети, промоделировать ее, используя массив входов P, и удостовериться, что сеть правильно решает задачу классификации на элементах обучающего множества. В результате моделирования сети формируется матрица связности, сответствующая массиву векторов входа. Для того чтобы преобразовать ее в индексный вектор, предназначена М-функция vec2ind.

```
net = newpnn(P,T);
net.layers{1}.size % Число нейронов в сети PNN

ans =
     7
Y = sim(net,P); Yc = vec2ind(Y)
```

```
Yc =
     1     1     2     2     3     3     3
```

Результат подтверждает правильность решения задачи классификации.

Выполним классификацию некоторого набора произвольных векторов p, не принадлежащих обучающему множеству, используя ранее созданную сеть PNN

```
p = [1 3; 0 1; 5 2]';
```

Выполняя моделирование сети для этого набора векторов, получаем

```
a = sim(net,p); ac = vec2ind(a)
ac =
     2     1     3
```

Фрагмент демонстрационной программы demopnn1 позволяет проиллюстрировать результаты классификации в графическом виде

```
figure(1), clf reset, drawnow
p1 = 0:.05:5;    p2 = p1;
   [P1,P2]=meshgrid(p1,p2); pp = [P1(:) P2(:)];
   aa = sim(net,pp'); aa = full(aa);
   m = mesh(P1,P2,reshape(aa(1,:),length(p1),length(p2)));
   set(m,'facecolor',[0.75 0.75 0.75],'LineStyle','none');
   hold on, view(3)
   m = mesh(P1,P2,reshape(aa(2,:),length(p1),length(p2)));
   set(m,'FaceColor',[0 1 0.5],'LineStyle','none');
   m = mesh(P1,P2,reshape(aa(3,:),length(p1),length(p2)));
   set(m,'FaceColor',[0 1 1],'LineStyle','none');
   plot3(P(1,:),P(2,:),ones(size(P,2))+0.1,'.','MarkerSize',20)
   plot3(p(1,:),p(2,:),1.1*ones(size(p,2)),'*','MarkerSize',10,...
'Color',[1 0 0]), hold off, view(2)
```

Результаты классификации представлены на рис. 6.15 и показывают, что 3 вектора, отмеченные звездочками, классифицируются сетью PNN, состоящей из 7 нейронов, абсолютно правильно.

Глава 7

C149. Слой Кохонена.

Формирование слоя Кохонена выполняется с помощью M-функции newc. Предположим, что задан массив из 4 двухэлементных векторов, которые надо разделить на два класса.

```
clear, p = [.1 .8 .1 .9; .2 .9 .1 .8]
```

```
p =
    0.1000    0.8000    0.1000    0.9000
    0.2000    0.9000    0.1000    0.8000
```

В этом простом примере нетрудно увидеть, что одна пара векторов расположена вблизи точки (0,0), а другая – вблизи точки (1, 1). Сформируем слой Кохонена с двумя нейронами для анализа двухэлементных векторов входа с диапазоном значений от 0 до 1

```
net = newc([0 1; 0 1],2);
```

Первый аргумент указывает диапазон входных значений, второй определяет количество нейронов в слое. Начальные значения элементов матрицы весов задаются как среднее максимального и минимального значений, то есть в центре интервала входных значений; это реализуется по умолчанию с помощью М-функции midpoint при создании сети. Убедимся, что это действительно так

```
wts = net.IW{1,1}
wts =
    0.5000    0.5000
    0.5000    0.5000
```

Определим характеристики слоя Кохонена

```
net.layers{1}
ans =
     dimensions: 2
    distanceFcn: ''
      distances: []
        initFcn: 'initwb'
    netInputFcn: 'netsum'
      positions: [0 1]
           size: 2
    topologyFcn: 'hextop'
    transferFcn: 'compet'
       userdata: [1x1 struct]
```

Из этого описания следует, что сеть использует функцию евклидова расстояния dist, функцию инициализации initwb, функцию обработки входов netsum, функцию активации compet и функцию описания топологии hextop.

Характеристики смещений следующие

```
net.biases{1}
ans =
       initFcn: 'initcon'
         learn: 1
      learnFcn: 'learncon'
```

```
    learnParam: [1x1 struct]
          size: 2
      userdata: [1x1 struct]
```

Смещения задаются функцией initcon и для инициализированной сети равны

```
net.b{1}
ans =
    5.4366
    5.4366
```

Функцией настройки смещений является функция learncon, обеспечивающая настройку с учетом параметра активности нейронов.

Элементы структурной схемы слоя Кохонена показаны на рис. 7.2, *а-б* и могут быть получены с помощью оператора

```
gensim(net)
```

Они наглядно поясняют архитектуру и функции, используемые при построении слоя Кохонена

Обучение сети

Реализуем 10 циклов обучения. Для этого можно использовать функции train или adapt

```
net.trainParam.epochs = 10; net = train(net,p);
net.adaptParam.passes = 10; [net,y,e] = adapt(net,mat2cell(p));
TRAINR, Epoch 0/10
TRAINR, Epoch 10/10
TRAINR, Maximum epoch reached.
```

Заметим, что для сетей с конкурирующим слоем по умолчанию используется обучающая функция trainwb1, которая на каждом цикле обучения случайно выбирает входной вектор и предъявляет его сети; после этого производится коррекция весов и смещений.

Выполним моделирование сети после обучения

```
a = sim(net,p); ac = vec2ind(a)
ac =
     2     1     2     1
```

Видим, что сеть обучена классификации векторов входа на два кластера: первый расположен в окрестности вектора (0, 0), второй – в окрестности вектора (1, 1). Результирующие веса и смещения равны

```
wts1 = net.IW{1,1}, b1 = net.b{1}
wts1 =
    0.6161    0.6161
```

```
    0.3673    0.3839
b1 =
    5.4366
    5.4365
```

С153. Пример.

Сформируем координаты случайных точек и построим план их расположения на плоскости:

```
clear, c = 8; n = 6; % Число кластеров, векторов в кластере
d = 0.5; % Среднеквадратическое отклонение от центра кластера
x = [-10 10;-5 5]; % Диапазон входных значений
[r,q] = size(x); minv = min(x')'; maxv = max(x')';
v = rand(r,c).*((maxv - minv)*ones(1,c) + minv*ones(1,c));
t = c*n; % Число точек
v = [v v v v v v]; v=v+randn(r,t)*d; % Координаты точек
P = v;
figure(1), clf, plot(P(1,:), P(2,:),'+k') % Рис.7.3
xlabel('P(1,:)'), ylabel('P(2,:)'), hold on, grid on
```

Применим конкурирующую сеть из восьми нейронов для того, чтобы распределить их по классам

```
net = newc([-2 12;-1 6], 8 ,0.1);
w0 = net.IW{1}; b0 = net.b{1}; c0 = exp(1)./b0;
```

После обучения в течение 50 циклов получим

```
tic, net.trainParam.epochs = 50;
[net,TR] = train(net,P); toc
w = net.IW{1}; bn = net.b{1}; cn = exp(1)./bn;
TRAINR, Epoch 0/50
TRAINR, Epoch 25/50
TRAINR, Epoch 50/50
TRAINR, Maximum epoch reached.
elapsed_time =
    20
```

Нанесем на график векторов входа центры кластеризации и отметим их красными кружочками:

```
plot(w(:,1),w(:,2),'or'),
title('Векторы входа и центры кластеризации')
```

С155. Карта Кохонена.

Создание сети.

Для создания самоорганизующейся карты Кохонена предусмотрена М-функция newsom. Допустим, что требуется создать сеть для обработки двухэлементных векторов входа с диапазоном изменения элементов от 0 до 2 и от 0 до 1, соответственно. Предполагается использовать гексагональную сетку размера 2×3. Тогда для формирования такой нейронной сети достаточно воспользоваться оператором

```
clear, net = newsom([0 2; 0 1], [2 3]);
net.layers{1}
ans =
     dimensions: [2 3]
    distanceFcn: 'linkdist'
      distances: [6x6 double]
        initFcn: 'initwb'
    netInputFcn: 'netsum'
      positions: [2x6 double]
           size: 6
    topologyFcn: 'hextop'
    transferFcn: 'compet'
       userdata: [1x1 struct]
```

Из анализа характеристик этой сети следует, что она использует по умолчанию гексагональную топологию hextop и функцию расстояния linkdist.

Для обучения сети зададим следующие 12 двухэлементных векторов входа

```
P = [0.1 0.3 1.2 1.1 1.8 1.7 0.1 0.3 1.2 1.1 1.8 1.7; ...
0.2 0.1 0.3 0.1 0.3 0.2 1.8 1.8 1.9 1.9 1.7 1.8];
```

Построим на топографической карте начальное расположение нейронов карты Кохонена и вершины векторов входа

```
figure(1), clf,
plotsom(net.iw{1,1},net.layers{1}.distances), hold on
plot(P(1,:),P(2,:),'*k','markersize',10), grid on %(рис. 7.11)
```

Обучение сети

Зададим количество циклов обучения, равным 200

```
tic, net.trainParam.epochs = 200;
net = train(net,P); toc
figure(2), plot(P(1,:),P(2,:),'*','markersize',10), hold on
plotsom(net.iw{1,1},net.layers{1}.distances)
```

```
TRAINR, Epoch 0/200
TRAINR, Epoch 100/200
TRAINR, Epoch 200/200
TRAINR, Maximum epoch reached.
elapsed_time =
   14.6100
```

Положение нейронов и их нумерация определяется массивом весовых векторов, который для данного примера имеет вид

```
net.IW{1}
ans =
    1.2009    1.8200
    0.7003    1.4810
    1.0334    1.0099
    0.4370    0.5749
    1.5505    0.2334
    1.0627    0.2000
```

Если промоделировать карту Кохонена на массиве обучающих векторов входа, то будет получен следующий выход сети

```
a = sim(net,P)
a =
   (4,1)        1
   (4,2)        1
   (6,3)        1
   (6,4)        1
   (5,5)        1
   (5,6)        1
   (2,7)        1
   (2,8)        1
   (1,9)        1
   (1,10)       1
   (1,11)       1
   (1,12)       1
```

Если сформировать произвольный вектор входа, то карта Кохонена должна указать его принадлежность тому или иному кластеру

```
a = sim(net,[1.5; 1])
a =
   (3,1)        1
```

С165. Одномерная карта Кохонена.

Рассмотрим 100 двухэлементных входных векторов единичной длины, распределенных равномерно в пределах от 0° до 90°.

```
clear, angles = 0:0.5*pi/99:0.5*pi;
P = [sin(angles); cos(angles)];
figure(1), clf, plot(P(1,1:10:end), P(2,1:10:end),'*b'),
hold on, grid on
```

На графике входных векторов символом * отмечено положение каждого десятого вектора.

Сформируем самоорганизующуюся карту Кохонена в виде одномерного слоя из 10 нейронов и выполним обучение в течение 50 циклов

```
net = newsom([0 1;0 1], [10]);
net.trainParam.epochs = 50;
tic, [net, tr] = train(net,P); toc
a = sim(net,P);
plotsom(net.IW{1,1},net.layers{1}.distances) % Рис.7.13,а
figure(2), clf, bar(sum(a')) % Рис.7.13,б
TRAINR, Epoch 0/50
TRAINR, Epoch 25/50
TRAINR, Epoch 50/50
TRAINR, Maximum epoch reached.
elapsed_time =
   36.1500
```

Сеть подготовлена к кластеризации входных векторов. Определим, к какому кластеру будет отнесен вектор [1; 0]

```
a = sim(net,[1;0])
a =
  (10,1)        1
```

Как и следовало ожидать, он отнесен к кластеру с номером 10.

С166. Двумерная карта Кохонена

Этот пример демонстрирует обучение двумерной карты Кохонена. Сначала создадим обучающий набор случайных двумерных векторов, элементы которых распределены по равномерному закону в интервале [-1 1].

```
clear, P = rands(2,100);
figure(1), clf, plot(P(1,:),P(2,:),'+') % Рис.7.14
```

Для кластеризации векторов входа создадим самоорганизующуюся карту Кохонена размера 3×4 с 12 нейронами, размещенными на гексагональной сетке

```
net = newsom([-1 1; -1 1],[3,4]);
net.trainParam.epochs = 20;
tic, net = train(net,P); toc
TRAINR, Epoch 0/20
```

```
TRAINR, Epoch 20/20
TRAINR, Maximum epoch reached.
elapsed_time =
    15
figure(1), clf, plotsom(net.IW{1,1},net.layers{1}.distances)
```

Определим принадлежность нового вектора одному из кластеров карты Кохонена

```
a = sim(net,[0.5;0.3])
hold on, plot(0.5,0.3,'*k')
a =
   (4,1)        1
```

Промоделируем обученную карту Кохонена, используя массив векторов входа

```
a = sim(net,P);
figure(2), clf, bar(sum(a'))  %Рис.7.16
```

С169. LVQ-сети.

Создание сети.

Этот пример правильно работает в версии MATLAB 5.3 (R11), однако в версиях MATLAB 6 , 6.1 (R12, R12.1) он выполняется неправильно. Поэтому в нижеследующем тексте операторы языка MATLAB не оформлены в виде ячеек входа ИС Notebook. Пользователю необходимо это выполнить самостоятельно.

Предположим, что задано 10 векторов входа, и необходимо создать сеть, которая, во-первых, группирует эти вектора в 4 кластера, а во-вторых, соотносит эти кластеры к одному из двух выходных классов. Для этого следует использовать LVQ-сеть, первый конкурирующий слой которой имеет 4 нейрона по числу кластеров, а второй линейный слой – 2 нейрона по числу классов.

Зададим обучающую последовательность в следующем виде

```
clear, P = [-3 -2 -2  0 0 0 0  2 2 3; 0 1 -1 2 1 -1 -2 1 -1 0];
Tc = [1 1 1 2 2 2 2 1 1 1];
```

Из структуры обучающей последовательности следует, что 3 первых и 3 последних ее вектора относятся к классу 1, а 4 промежуточных - к классу 2. Построим расположение векторов входа:

```
I1 = find(Tc==1); I2 = find(Tc==2);
figure(1), clf, axis([-4,4,-3,3]), hold on
plot(P(1,I1),P(2,I1),'+r')
```

```
plot(P(1,I2),P(2,I2),'xb') %Рис.7.18
```

Преобразуем вектор индексов Tc в массив целевых векторов

```
T = full(ind2vec(Tc))
```

```
T =
     1     1     1     0     0     0     0     1     1     1
     0     0     0     1     1     1     1     0     0     0
```

Процентные доли входных векторов в каждом классе равны 0.6 и 0.4, соответственно. Теперь подготовлены все данные, необходимые для вызова функции newlvq. Вызов может быть реализован с использованием следующего оператора

```
net = newlvq(minmax(P),4,[.6 .4],0.05,'learnlv2');
net.inputWeights{1}
```

Процедура обучения

Для обучения сети применим М-функцию train, задав количество циклов обучения, равным 2000, и значение параметра скорости обучения 0.05

```
net.trainParam.epochs = 2000;
net.trainParam.show = 100;
net.trainParam.lr = 0.05;
tic, net = train(net,P,T); toc
```

В результате обучения получим следующие весовые коэффициенты нейронов конкурирующего слоя, которые определяют положения центров кластеризации

```
V = net.IW{1,1}
```

Построим картину распределения входных векторов по кластерам

```
I1 = find(Tc==1); I2 = find(Tc==2);
figure(1), axis([-4,4,-3,3]), hold on
P1 = P(:,I1); P2 = P(:,I2);
plot(P1(1,:),P1(2,:),'+k')
plot(P2(1,:),P2(2,:),'xb')
plot(V(:,1),V(:,2),'or')% Рис.7.19
```

В свою очередь, массив весов линейного слоя указывает, как центры кластеризации распределяются по классам

```
net.LW{2}
```

Чтобы проверить функционирование сети, подадим на ее вход массив обучающих векторов P

```
Y = sim(net,P), Yc = vec2ind(Y)
```

Теперь построим границу, разделяющую области точек, принадлежащих двум классам. Для этого покроем сеткой прямоугольную область и определим принадлежность каждой точки тому или иному классу. Текст соответствующего сценария и вспомогательной М-функции приведен ниже

```
x = -4:0.2:4;
y = -3:0.2:3;
P = mesh2P(x,y);
Y = sim(net,P);
Yc = vec2ind(Y);
I1 = find(Yc==1); I2 = find(Yc==2);
plot(P(1,I1),P(2,I1),'+k'), hold on
plot(P(1,I2),P(2,I2),'*b') %Рис.7.20
```

Текст М-функции необходимо разместить в папке work системы MATLAB

```
function P = mesh2P(x,y)
% Вычисление массива координат прямоугольной сетки

[X,Y]= meshgrid(x,y);
P = cat(3,X,Y);
[n1,n2,n3] = size(P);
P = permute(P,[3 2 1]);
P = reshape(P, [n3 n1*n2]);
```

Результат работы этого сценария представлен на рис. 7.20. Здесь же отмечены вычисленные ранее центры кластеризации для синтезированной LVQ-сети. Анализ рисунка подтверждает, что граница между областями не является прямой линией

Наряду с процедурой обучения, можно применить и процедуру адаптации в течение 200 циклов для 10 векторов, что равносильно 2000 циклам обучения с использованием функции train.

```
net.adaptparam.passes = 200;
```

Обучающая последовательность при использовании функции adapt должна быть представлена в виде массивов ячеек

```
clear, P = [-3 -2 -2  0 0 0 0  2 2 3; 0 1 -1 2 1 -1 -2 1 -1 0];
Tc = [1 1 1 2 2 2 2 1 1 1];
T = full(ind2vec(Tc))
net = newlvq(minmax(P),4,[.6 .4]);
Pseq = con2seq(P);
Tseq = con2seq(T);
```

```
net = adapt(net,Pseq,Tseq);
net.IW{1,1}
```

Промоделируем сеть, используя массив входных векторов обучающей последовательности

```
Y = sim(net,P);
Yc = vec2ind(Y)
```

Результаты настройки параметров сети в процессе адаптации практически совпадают с результатами обучения.

Единственное, что можно было бы напомнить при этом, что при обучении векторы входа выбираются в случайном порядке и поэтому в некоторых случаях обучение может давать лучшие результаты, чем адаптация.

Можно было бы и процедуру адаптации реализовать с использованием случайной последовательности входов, например, следующим образом. Сформируем 2000 случайных векторов и выполним лишь один цикл адаптации

```
TS = 2000;
ind = floor(rand(1,TS)*size(P,2))+1;
Pseq = con2seq(P(:,ind));
Tseq = con2seq(T(:,ind));
net.adaptparam.passes = 1;
net = adapt(net,Pseq,Tseq);
net.IW{1,1}
Y = sim(net,P);
Yc = vec2ind(Y)
```

Глава 8

С175. Сети Элмана.

Сеть Элмана исследуется на примере задачи детектирования амплитуды гармонического сигнала. Пусть известно, что на вход нейронной сети поступают выборки из некоторого набора синусоид. Требуется выделить значения амплитуд этих синусоид.

Далее рассматриваются выборки из набора двух синусоид с амплитудами 1.0 и 2.0:

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

Целевыми выходами такой сети являются векторы

```
t1 = ones(1,20);
```

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

Сформируем набор векторов входа и целевых выходов

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

Сформируем обучающую последовательность в виде массивов ячеек

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

Создание сети

Для создания сети Элмана предназначена М-функция newelm. Решаемая задача требует, чтобы сеть Элмана на каждом шаге наблюдения значений выборки могла выявить единственный ее параметр – амплитуду синусоиды. Это означает, что сеть должна иметь один вход и один выход:

```
R = 1; % Число элементов входа
S2 = 1;% Число нейронов выходного слоя
```

Рекуррентный слой может иметь любое число нейронов и чем сложнее задача, тем большее количество нейронов требуется. Остановимся на 10 нейронах рекуррентного слоя

```
S1 = 10;% Число нейронов рекуррентного слоя
```

Элементы входа для данной задачи изменяются в диапазоне от-2 до 2. Для обучения используется метод градиентного спуска с возмущением и адаптацией параметра скорости настройки, реализованный в виде М-функции traingdx

```
net = newelm([-2 2],[S1 S2],{'tansig','purelin'},'traingdx');
gensim(net)
```

Обучение сети.

Для обучения сети Элмана используется М-функция train. Ее входными аргументами являются обучающие последовательности Pseq и Tseq, в качестве метода обучения используется метод обратного распространения ошибки с возмущением и адаптацией параметра скорости настройки. Количество циклов обучения принимается равным 1000, периодичность вывода результатов –20 циклов, конечная погрешность обучения 0.01:

```
net.trainParam.epochs = 1000;
net.trainParam.show = 25;
net.trainParam.goal = 0.01;
[net,tr] = train(net,Pseq,Tseq);
TRAINGDX, Epoch 0/1000, MSE 5.54748/0.01, Gradient 9.68599/1e-006
```

```
   TRAINGDX, Epoch 25/1000, MSE 0.351376/0.01, Gradient 0.603403/1e-
006
   TRAINGDX, Epoch 50/1000, MSE 0.272563/0.01, Gradient 0.127855/1e-
006
   TRAINGDX, Epoch 75/1000, MSE 0.250764/0.01, Gradient 0.0547544/1e-
006
   TRAINGDX, Epoch 100/1000, MSE 0.238683/0.01, Gradient
0.0289263/1e-006
   TRAINGDX, Epoch 125/1000, MSE 0.22393/0.01, Gradient 0.364115/1e-
006
   TRAINGDX, Epoch 150/1000, MSE 0.220146/0.01, Gradient 0.185611/1e-
006
   TRAINGDX, Epoch 175/1000, MSE 0.217272/0.01, Gradient
0.0602147/1e-006
   TRAINGDX, Epoch 200/1000, MSE 0.15495/0.01, Gradient 0.184134/1e-
006
   TRAINGDX, Epoch 225/1000, MSE 0.113434/0.01, Gradient 0.200365/1e-
006
   TRAINGDX, Epoch 250/1000, MSE 0.0604263/0.01, Gradient 0.08513/1e-
006
   TRAINGDX, Epoch 275/1000, MSE 0.053735/0.01, Gradient 0.135704/1e-
006
   TRAINGDX, Epoch 300/1000, MSE 0.035034/0.01, Gradient 0.17518/1e-
006
   TRAINGDX, Epoch 325/1000, MSE 0.0297003/0.01, Gradient
0.0846544/1e-006
   TRAINGDX, Epoch 350/1000, MSE 0.024135/0.01, Gradient
0.0646931/1e-006
   TRAINGDX, Epoch 375/1000, MSE 0.0234717/0.01, Gradient
0.048488/1e-006
   TRAINGDX, Epoch 400/1000, MSE 0.0345563/0.01, Gradient
0.101825/1e-006
   TRAINGDX, Epoch 425/1000, MSE 0.0565671/0.01, Gradient
0.298326/1e-006
   TRAINGDX, Epoch 450/1000, MSE 0.0483179/0.01, Gradient
0.0997117/1e-006
   TRAINGDX, Epoch 475/1000, MSE 0.0466496/0.01, Gradient
0.0843978/1e-006
   TRAINGDX, Epoch 500/1000, MSE 0.0450599/0.01, Gradient
0.0702556/1e-006
   TRAINGDX, Epoch 525/1000, MSE 0.0412523/0.01, Gradient
0.0507589/1e-006
```

```
   TRAINGDX,    Epoch    550/1000,    MSE    0.0362202/0.01,    Gradient
0.0372358/1e-006
   TRAINGDX,    Epoch    575/1000,    MSE    0.0300758/0.01,    Gradient
0.0201178/1e-006
   TRAINGDX,    Epoch    600/1000,    MSE    0.0260255/0.01,    Gradient
0.214156/1e-006
   TRAINGDX,    Epoch    625/1000,    MSE    0.0251267/0.01,    Gradient
0.128905/1e-006
   TRAINGDX,    Epoch    650/1000,    MSE    0.0245318/0.01,    Gradient
0.0354351/1e-006
   TRAINGDX,    Epoch    675/1000,    MSE    0.0239542/0.01,    Gradient
0.0166466/1e-006
   TRAINGDX,    Epoch    700/1000,    MSE    0.0224075/0.01,    Gradient
0.0146542/1e-006
   TRAINGDX, Epoch 725/1000, MSE 0.0209583/0.01, Gradient 0.18643/1e-
006
   TRAINGDX,    Epoch    750/1000,    MSE    0.0199925/0.01,    Gradient
0.045114/1e-006
   TRAINGDX,    Epoch    775/1000,    MSE    0.0197382/0.01,    Gradient
0.020261/1e-006
   TRAINGDX,    Epoch    800/1000,    MSE    0.0191773/0.01,    Gradient
0.0131769/1e-006
   TRAINGDX,    Epoch    825/1000,    MSE    0.0175978/0.01,    Gradient
0.012473/1e-006
   TRAINGDX,    Epoch    850/1000,    MSE    0.0174953/0.01,    Gradient
0.123555/1e-006
   TRAINGDX, Epoch 875/1000, MSE 0.017082/0.01, Gradient 0.043115/1e-
006
   TRAINGDX,    Epoch    900/1000,    MSE    0.0168296/0.01,    Gradient
0.0103317/1e-006
   TRAINGDX,    Epoch    925/1000,    MSE    0.0162647/0.01,    Gradient
0.00979225/1e-006
   TRAINGDX,    Epoch    950/1000,    MSE    0.0161786/0.01,    Gradient
0.134446/1e-006
   TRAINGDX,    Epoch    975/1000,    MSE    0.0158085/0.01,    Gradient
0.0564908/1e-006
   TRAINGDX,    Epoch   1000/1000,    MSE    0.0157046/0.01,    Gradient
0.0103259/1e-006
   TRAINGDX, Maximum epoch reached, performance goal was not met.
```

Проверка сети

Будем использовать для проверки сети входы обучающей последовательности

```
figure(2)
a = sim(net,Pseq);
time = 1:length(p);
plot(time, t, '--', time, cat(2,a{:}))
axis([1 80 0.8 2.2]), grid on %Рис.8.3
```

Обладает ли построенная сеть Элмана свойством обобщения? Попробуем проверить это, выполнив следующие исследования.

Подадим на сеть набор сигналов, составленный из двух синусоид с амплитудами 1.6 и 1.2, соответственно

```
p3 = sin(1:20)*1.6;
t3 = ones(1,20)*1.6;
p4 = sin(1:20)*1.2;
t4 = ones(1,20)*1.2;
pg = [p3 p4 p3 p4];
tg = [t3 t4 t3 t4];
pgseq = con2seq(pg);
figure(3)
a = sim(net,pgseq);
time = 1:length(pg);
plot(time, tg, '--', time, cat(2,a{:}))
axis([1 80 0.8 2.2]) %Рис.8.4
grid on
```

С181. Сети Хопфилда.

Требуется разработать сеть Хопфилда с двумя устойчивыми точками в вершинах трехмерного куба

```
T = [-1 -1 1; 1 -1 1]'
T =
    -1     1
    -1    -1
     1     1
```

Синтез сети

Выполним синтез сети, используя М-функцию newhop

```
net = newhop(T);
gensim(net)
```

Удостоверимся, что разработанная сеть имеет устойчивые состояния в этих двух точках. Выполним моделирование сети Хопфилда, приняв во внимание, что эта сеть не имеет входов и содержит рекуррентный слой; в этом случае

целевые векторы определяют начальное состояние слоя Ai, а второй аргумент функции sim определяется числом целевых векторов

```
Ai = T;
[Y,Pf,Af] = sim(net,2,[],Ai); Y
```

С186. Пример.

Рассмотрим сеть Хопфилда с четырьмя нейронами и определим 4 точки равновесия в виде следующего массива целевых векторов

```
T = [1 -1; -1 1;  1 1; -1 -1]'
T =
     1    -1     1    -1
    -1     1     1    -1
```

На рис. показаны эти 4 точки равновесия на плоскости состояний сети Хопфилда

```
figure(1), clf, plot(T(1,:), T(2,:),'*r') %Рис.8.7
axis([-1.1 1.1 -1.1 1.1])
title('Точки равновесиЯ сети Хопфилда')
xlabel('a(1)'), ylabel('a(2)')
```

Рассчитаем веса и смещения модифицированной сети Хопфилда, использую М-функцию newhop

```
net = newhop(T);
W= net.LW{1,1}
b = net.b{1,1}
W =
    1.1618         0
         0    1.1618
b =
  1.0e-016 *
         0
   -0.1797
```

Проверим, принадлежат ли вершины квадрата сети Хопфилда:

```
Ai = T;
[Y,Pf,Af] = sim(net,4,[],Ai)

Y =
     1    -1     1    -1
    -1     1     1    -1
Pf =
     []
```

```
Af =
     1    -1     1    -1
    -1     1     1    -1
```

Как и следовало ожидать, выходы сети равны целевым векторам.

Теперь проверим поведение сети при случайных начальных условиях.

```
 figure(1), clf, plot(T(1,:), T(2,:),'*r', 0,0,'rh'),
hold on, axis([-1.1 1.1 -1.1 1.1])
xlabel('a(1)'), ylabel('a(2)')
new = newhop(T);
[Y,Pf,Af] = sim(net,4,[],T);
for i =1:20
   a = {rands(2,1)};
   [Y,Pf,Af] = sim(net,{1,20},{},a);
   record = [cell2mat(a) cell2mat(Y)];
   start = cell2mat(a);
   plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:))
end
```

Глава 9

Аппроксимация и фильтрация сигналов.

C188. Предсказание стационарного сигнала.

Задан гармонический сигнал с круговой частотой 4π 1/с (2 Гц) и длительностью 5 с. Дискретный сигнал T получен в результате квантования исходного сигнала по времени с частотой 40 Гц (такт дискретности 0.025 с)

```
time = 0:0.025:5; T = sin(time*4*pi);
stairs(time,T);
axis([0 5 -1 1]), xlabel('time, c'), ylabel('T')
```

Сформируем обучающее множество следующим образом. Входная последовательность P1 определена на интервале от 0 до 1 с и имеет длину Q1, а каждый вектор входа состоит из пяти компонентов, соответствующих запаздывающим значениям сигнала T; целевой вектор T1 сформирован из значений сигнала T, начиная с шестого; контрольное подмножество T2 формируется из значений сигнала T на интервале от 3 до 5 с

```
Q = length(T); h = 0.025; Q1 = 1/h;
P1 = zeros(5,Q1);
P1(1,1:Q1) = T(1,1:Q1);
P1(2,2:Q1) = T(1,1:(Q1-1));
```

```
P1(3,3:Q1) = T(1,1:(Q1-2));
P1(4,4:Q1) = T(1,1:(Q1-3));
P1(5,5:Q1) = T(1,1:(Q1-4));
T1 = T(1,6:(Q1+5)); T2 = T(1,3/h:Q);
```

Сеть для решения этой задачи должна состоять из одного нейрона с пятью входами:

```
net = newlind(P1,T1); gensim(net) % Рис. 9.2
```

Выполним проверку сети, используя входную последовательность обучающего подмножества и сравнивая выход сети с фактическим значением сигнала T

```
figure(1), clf, Q1 = 40;
a = sim(net,P1(:,1:Q1)); t1 = 6:Q1+5;
plot(time(t1),a,'*r', time(1:Q1+5),T(1,1:Q1+5))
xlabel('Time, c'); grid on
```

Проверим работу сети, используя контрольное множество T2. Определим длину входной последовательности N1, равную 20, и построим реакцию сети:

```
N1 = 20;
Tt = T2(1,1:N1);
P2(1,:) = Tt(1,:);
P2(2,2:end) = Tt(1,1:end-1);
P2(3,3:end) = Tt(1,1:end-2);
P2(4,4:end) = Tt(1,1:end-3);
P2(5,5:end) = Tt(1,1:end-4);
a = sim(net,P2);
figure(2), clf
h1 = plot(time(1:size(P2, 2)-5), a(1:end-5), '*'); hold on
h2 = plot(time(1:size(P2, 2)-5), Tt(6:end), 'r'); grid on
```

Вычислим погрешность сети, используя информацию из описания графических объектов Line с дескрипторами h1 и h2

```
y1 = get(h1,'YData'); y2 = get(h2,'YData');
minlength = min(length(y1), length(y2));
e = y1(1:minlength) - y2(1:minlength);
nre = sqrt(mse(e))
nre =    0.4774
```

C192. Слежение за нестационарным сигналом.

Задана дискретная выборка T из гармонического сигнала длительностью 6 с, частота которого удваивается по истечении 4 с. Частота квантования для

интервала времени от 0 до 4 с составляет 20 Гц, а для интервала от 4.05 до 6 с - 40 Гц.

```
time1 = 0:0.05:4; time2 = 4.05:0.025:6; time = [time1 time2];
T = [sin(time1*4*pi) sin(time2*8*pi)];
```

Поскольку при синтезе сети будут использоваться адаптивные алгоритмы настройки, сформируем обучающую последовательность {P, T} в виде массива ячеек, при этом последовательность входов P должна совпадать с последовательностью целевых выходов T (задача слежения):

```
T = con2seq(T); P = T;
```

Построим график гармонической последовательности:

```
figure(1), clf, plot(time, cat(2,P{:}))
```

Для решения задачи воспользуемся однослойной линейной нейронной сетью, которая предсказывает текущее значение сигнала по 5 его предшествующим значениям. Сеть состоит только из одного нейрона, так как требуется только одно значение выходного сигнала T, которое генерируется на каждом шаге. Для создания такой сети предназначена M-функция newlin; параметр скорости настройки выберем равным 0.1:

```
lr = 0.1;
delays = [1 2 3 4 5];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
[net,a,e] = adapt(net,P,T);
```

Сформированная нейронная сеть имеет следующие весовые коэффициенты и смещение

```
net.IW{1}, net.b
```

```
ans =
  Columns 1 through 4
    0.3942    0.1068   -0.1559   -0.3148
  Column 5
   -0.3452
ans =
    [-4.5457e-006]
```

Построим график выходного сигнала и сравним его с целевым:

```
y = sim(net,P); figure(1), clf
plot(time,cat(2,y{:}), time,cat(2,T{:}),'.r') % Рис.9.10
axis([0 6 -1.5 1.5]), grid on
```

Построим также график сигнала ошибки

```
figure(2), clf, plot(time,cat(2,e{:})), grid on % Рис. 9.11
```

С194. Моделирование стационарного фильтра.

Одно из полезных применений нейронных сетей – это создание моделей динамических систем по наблюдаемым входным и выходным сигналам и их применение для последующего моделирования таких систем.

Допустим, что на вход фильтра подается входной сигнал вида r(t) = sin(10*sin(t)*t), заданный массивом значений R с тактом квантования 0.025 с на интервале 5 с

```
time = 0:0.025:5;
R = sin(sin(time).*time*10);
figure(1), clf, plot(time,R)
axis([0 5 -1 1]); grid on
```

Определим фильтр второго порядка, функционирование которого в системе MATLAB описывается следующей M-функцией

```
Y = filter([1 0.5 -1.5],1,R);
```

и построим график сигнала на его выходе:

```
figure(2), clf, plot(time,Y), axis([0 5 -2 2]); grid on
```

Задача нейронной сети – сформировать такую линейную модель, которая в процессе обучения определяет параметры фильтра, а затем использует их для моделирования при произвольных значениях входа.

Определим следующую обучающую последовательность: в качестве целевого выхода T примем массив Y, а входную последовательность P зададим на основе текущего и двух предшествующих значений входа R

```
T = Y; Q = size(R,2); P = zeros(3,Q);
P(1,1:Q) = R(1,1:Q);
P(2,2:Q) = R(1,1:(Q-1));
P(3,3:Q) = R(1,1:(Q-2));
```

Нейронная сеть должна иметь только один нейрон, потому что динамическая система имеет только один выход. Нейрон должен иметь три входа, чтобы получить текущий и два запаздывающих значения входного сигнала. M-функция newlind позволяет выполнить синтез такой нейронной сети

```
net = newlind(P,T); net.IW{1}, net.b
ans =
    1.0000    0.5000   -1.5000
ans =
    [-3.5239e-017]
```

Нетрудно убедиться, что сеть точно определяет параметры фильтра.

Для проверки функционирования сети подадим входную последовательность P и сравним с целевой последовательностью T:

```
a = sim(net,P); figure(1), clf
plot(time,T, 'Color', [0 0.8 0.8], 'LineWidth',3),hold on
plot(time,a, 'k'), grid on, axis([0 5 -2 2]); % Рис.9.14
```

Сеть выполняет поставленную задачу. Погрешность моделирования находится в пределах точности компьютера при вычислениях с плавающей точкой

```
e = T-a;
figure(2), clf, plot(time, e), grid on % Рис.9.15
```

С197. Моделирование нестационарного фильтра.

На вход фильтра подается входной сигнал вида r(t) = sin(8sin(4t)*t), заданный массивом значений R с тактом квантования 0.005 с на интервале 6 с

```
time1 = 0:0.005:4; time2 = 4.005:0.005:6; time = [time1 time2];
R = sin(sin(time*4).*time*8); figure(1), clf,
plot(time,R), axis([0 6 -1.1 1.1]); grid on
```

Нестационарный линейный фильтр может быть представлен в системе MATLAB следующим образом

```
steps1 = length(time1);
[Y1,state] = filter([1 -0.5],1,R(1:steps1));
steps2 = length(time2);
Y2 = filter([0.9 -0.6],1,R((1:steps2)+steps1),state); Y = [Y1 Y2];
figure(2), clf, plot(time,Y), grid on
```

Определим следующую обучающую последовательность: в качестве целевого выхода T примем массив Y, а входную последовательность P зададим на основе текущего и предшествующего значений входа R. Для использования алгоритмов адаптации представим обучающие последовательности в виде массивов ячеек

```
T = con2seq(Y); P = con2seq(R);
```

Сеть создается с помощью функции newlin, которая генерирует веса и смещение для линейного нейрона с двумя входами. На входе сети используется линия задержки на 1 такт; параметр скорости настройки принят равным 0.5

```
lr = 0.5; delays = [0 1];
```

```
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
[net,a,e] = adapt(net,P,T);
```

Сформированная нейронная сеть имеет следующие весовые коэффициенты и смещение

```
net.IW{1}, net.b
ans =
    0.9000   -0.6000
ans =
    [-3.1400e-013]
```

Нетрудно убедиться, что они соответствуют коэффициентам второго фильтра.

Построим график погрешности сети

```
figure(2), clf, plot(time, cat(2,e{:})), grid on % Рис. 9.18
```

Из анализа графика следует, что нейронной сети требуется 2.5 с для настройки на реакцию первого фильтра и немногим более 0.2 с для настройки на реакцию второго фильтра. Этого объясняется тем, что фактические настройки параметров сети стационарны и соответствуют значениям параметров второго фильтра.

С199. Распознавание образов.

Требуется создать нейронную сеть для распознавания 26 символов латинского алфавита. В качестве датчика предполагается использовать систему распознавания, которая выполняет оцифровку каждого символа, находящегося в поле зрения. В результате каждый символ будет представлен шаблоном размера 5×7. Проектируемая нейронная сеть должна точно распознавать идеальные векторы входа и с максимальной точностью воспроизводить зашумленные векторы. М-функция prprob определяет 26 векторов входа, каждый из которых содержит 35 элементов, этот массив называется *алфавитом*. М-функция формирует выходные переменные alphabet и targets, которые определяют массивы алфавита и целевых векторов. Массив targets определяется как eye(26).

Определим шаблон для символа A, который является первым элементом алфавита

```
clear, [alphabet, targets] = prprob;
i = 1; ti = alphabet(:, i);
letter{i} = reshape(ti, 5, 7)'; letter{i}
```

```
ans =
     0     0     1     0     0
     0     1     0     1     0
     0     1     0     1     0
     1     0     0     0     1
     1     1     1     1     1
     1     0     0     0     1
     1     0     0     0     1
```

Инициализация сети

Вызовем М-файл prprob, который формирует массив векторов входа alphabet размера 35×26 с шаблонами символов алфавита и массив целевых векторов targets:

```
[alphabet,targets] = prprob;
[R,Q] = size(alphabet); [S2,Q] = size(targets);
```

Двухслойная нейронная сеть создается с помощью команды newff.

```
S1 = 10;
net=newff(minmax(alphabet),[S1 S2],...
{'logsig' 'logsig'}, 'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
gensim(net)
```

Обучение в отсутствии шума

Сеть первоначально обучается в отсутствии шума с максимальным числом циклов обучения – 5000, либо до достижения допустимой средней квадратичной погрешности, равной 0.1

```
P = alphabet; T = targets;
net.performFcn = 'sse'; net.trainParam.goal = 0.1;
net.trainParam.show = 20; net.trainParam.epochs = 500;
net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T); %Рис.9.23
```

```
TRAINGDX, Epoch 0/500, SSE 168.447/0.1, Gradient 46.0382/1e-006
TRAINGDX, Epoch 20/500, SSE 25.7528/0.1, Gradient 1.9299/1e-006
TRAINGDX, Epoch 40/500, SSE 25.2952/0.1, Gradient 0.415125/1e-006
TRAINGDX, Epoch 60/500, SSE 25.5128/0.1, Gradient 0.3371/1e-006
TRAINGDX, Epoch 80/500, SSE 25.5261/0.1, Gradient 0.339056/1e-006
TRAINGDX, Epoch 100/500, SSE 25.4204/0.1, Gradient 0.407/1e-006
TRAINGDX, Epoch 120/500, SSE 24.8589/0.1, Gradient 0.594/1e-006
TRAINGDX, Epoch 140/500, SSE 23.0034/0.1, Gradient 0.723414/1e-006
```

```
TRAINGDX, Epoch 160/500, SSE 14.4322/0.1, Gradient 1.29897/1e-006
TRAINGDX, Epoch 180/500, SSE 1.62053/0.1, Gradient 0.482638/1e-006
TRAINGDX, Epoch 199/500, SSE 0.0903872/0.1, Gradient 0.0562929/1e-
006
TRAINGDX, Performance goal met.
```

Обучение в присутствии шума

Чтобы спроектировать нейронную сеть, не чувствительную к воздействию шума, обучим ее с применением двух идеальных и двух зашумленных копий векторов алфавита. Целевые векторы состоят из четырех копий векторов. Зашумленные векторы имеют шум со средним значением 0.1 и 0.2. Это обучает нейрон правильно распознавать зашумленные символы и в то же время хорошо распознавать идеальные векторы.

При обучении с шумом максимальное число циклов обучения сократим до 300, а допустимую погрешность увеличим до 0.6

```
netn = net; netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
T = [targets targets targets targets];
for pass = 1:10
      P = [alphabet, alphabet, ...
      (alphabet + randn(R,Q)*0.1), ...
      (alphabet + randn(R,Q)*0.2)];
      [netn,tr] = train(netn,P,T);
end % Рис.9.24
```

```
TRAINGDX, Epoch 0/300, SSE 4.38727/0.6, Gradient 3.91707/1e-006
TRAINGDX, Epoch 20/300, SSE 2.44694/0.6, Gradient 1.70548/1e-006
TRAINGDX, Epoch 40/300, SSE 1.93829/0.6, Gradient 0.880157/1e-006
TRAINGDX, Epoch 60/300, SSE 1.26772/0.6, Gradient 0.620832/1e-006
TRAINGDX, Epoch 80/300, SSE 0.743827/0.6, Gradient 0.285485/1e-006
TRAINGDX, Epoch 89/300, SSE 0.595819/0.6, Gradient 0.201143/1e-006
TRAINGDX, Performance goal met.

TRAINGDX, Epoch 0/300, SSE 1.69802/0.6, Gradient 2.60629/1e-006
TRAINGDX, Epoch 20/300, SSE 1.10048/0.6, Gradient 0.780206/1e-006
TRAINGDX, Epoch 40/300, SSE 0.91056/0.6, Gradient 0.516545/1e-006
TRAINGDX, Epoch 60/300, SSE 0.712442/0.6, Gradient 0.284097/1e-006
TRAINGDX, Epoch 73/300, SSE 0.597462/0.6, Gradient 0.196257/1e-006
TRAINGDX, Performance goal met.
```

```
TRAINGDX, Epoch 0/300, SSE 1.84056/0.6, Gradient 3.45909/1e-006
TRAINGDX, Epoch 20/300, SSE 1.0796/0.6, Gradient 0.935168/1e-006
TRAINGDX, Epoch 40/300, SSE 0.833965/0.6, Gradient 0.581924/1e-006
TRAINGDX, Epoch 60/300, SSE 0.599226/0.6, Gradient 0.295716/1e-006
TRAINGDX, Performance goal met.

TRAINGDX, Epoch 0/300, SSE 1.60648/0.6, Gradient 2.3488/1e-006
TRAINGDX, Epoch 20/300, SSE 1.07786/0.6, Gradient 0.902587/1e-006
TRAINGDX, Epoch 40/300, SSE 0.862175/0.6, Gradient 0.456114/1e-006
TRAINGDX, Epoch 60/300, SSE 0.701691/0.6, Gradient 0.271165/1e-006
TRAINGDX, Epoch 74/300, SSE 0.592602/0.6, Gradient 0.188876/1e-006
TRAINGDX, Performance goal met.

TRAINGDX, Epoch 0/300, SSE 3.77602/0.6, Gradient 4.17101/1e-006
TRAINGDX, Epoch 20/300, SSE 1.73397/0.6, Gradient 2.35782/1e-006
TRAINGDX, Epoch 40/300, SSE 0.985143/0.6, Gradient 0.749831/1e-006
TRAINGDX, Epoch 60/300, SSE 0.699861/0.6, Gradient 0.364809/1e-006
TRAINGDX, Epoch 71/300, SSE 0.594703/0.6, Gradient 0.226427/1e-006
TRAINGDX, Performance goal met.

TRAINGDX, Epoch 0/300, SSE 1.42439/0.6, Gradient 2.20268/1e-006
TRAINGDX, Epoch 20/300, SSE 0.886985/0.6, Gradient 0.784242/1e-006
TRAINGDX, Epoch 40/300, SSE 0.702066/0.6, Gradient 0.433875/1e-006
TRAINGDX, Epoch 54/300, SSE 0.594342/0.6, Gradient 0.318662/1e-006
TRAINGDX, Performance goal met.

TRAINGDX, Epoch 0/300, SSE 2.09651/0.6, Gradient 3.81564/1e-006
TRAINGDX, Epoch 20/300, SSE 0.991083/0.6, Gradient 1.83845/1e-006
TRAINGDX, Epoch 40/300, SSE 0.652035/0.6, Gradient 0.526766/1e-006
TRAINGDX, Epoch 49/300, SSE 0.597725/0.6, Gradient 0.519457/1e-006
TRAINGDX, Performance goal met.

TRAINGDX, Epoch 0/300, SSE 0.923138/0.6, Gradient 1.46303/1e-006
TRAINGDX, Epoch 20/300, SSE 0.690035/0.6, Gradient 0.571227/1e-006
TRAINGDX, Epoch 36/300, SSE 0.597901/0.6, Gradient 0.448717/1e-006
TRAINGDX, Performance goal met.

TRAINGDX, Epoch 0/300, SSE 3.00343/0.6, Gradient 2.66202/1e-006
TRAINGDX, Epoch 20/300, SSE 2.10058/0.6, Gradient 1.27402/1e-006
TRAINGDX, Epoch 40/300, SSE 1.32981/0.6, Gradient 1.08293/1e-006
TRAINGDX, Epoch 60/300, SSE 0.841513/0.6, Gradient 0.605556/1e-006
TRAINGDX, Epoch 76/300, SSE 0.599021/0.6, Gradient 0.247759/1e-006
TRAINGDX, Performance goal met.
```

```
TRAINGDX, Epoch 0/300, SSE 3.92135/0.6, Gradient 4.29976/1e-006
TRAINGDX, Epoch 20/300, SSE 2.68635/0.6, Gradient 1.74033/1e-006
TRAINGDX, Epoch 40/300, SSE 2.19335/0.6, Gradient 0.95226/1e-006
TRAINGDX, Epoch 60/300, SSE 1.76432/0.6, Gradient 0.439475/1e-006
TRAINGDX, Epoch 80/300, SSE 1.48833/0.6, Gradient 0.207324/1e-006
TRAINGDX, Epoch 100/300, SSE 1.32874/0.6, Gradient 0.103221/1e-006
TRAINGDX, Epoch 120/300, SSE 1.2109/0.6, Gradient 0.0501167/1e-006
TRAINGDX, Epoch 140/300, SSE 1.12095/0.6, Gradient 0.0461831/1e-
006
TRAINGDX, Epoch 160/300, SSE 0.348182/0.6, Gradient 0.958822/1e-
006
TRAINGDX, Performance goal met.
```

Повторное обучение в отсутствии шума

Поскольку нейронная сеть обучалась в присутствии шума, то имеет смысл повторить ее обучение без шума, чтобы гарантировать, что идеальные векторы входа классифицируются правильно.

```
netn.trainParam.goal = 0.1;
netn.trainParam.epochs = 500;
net.trainParam.show = 5;
[netn, tr] = train(netn,P,T);
```

```
TRAINGDX, Epoch 0/500, SSE 0.348182/0.1, Gradient 0.958822/1e-006
TRAINGDX, Epoch 20/500, SSE 0.289733/0.1, Gradient 0.661362/1e-006
TRAINGDX, Epoch 40/500, SSE 0.236086/0.1, Gradient 0.272537/1e-006
TRAINGDX, Epoch 60/500, SSE 0.195025/0.1, Gradient 0.117262/1e-006
TRAINGDX, Epoch 80/500, SSE 0.166933/0.1, Gradient 0.0653886/1e-
006
TRAINGDX, Epoch 100/500, SSE 0.142052/0.1, Gradient 0.0398789/1e-
006
TRAINGDX, Epoch 120/500, SSE 0.1158/0.1, Gradient 0.0258157/1e-006
TRAINGDX, Epoch 132/500, SSE 0.0985149/0.1, Gradient 0.0199491/1e-
006
TRAINGDX, Performance goal met.
```

Эффективность функционирования системы

Эффективность нейронной сети будем оценивать следующим образом. Рассмотрим две структуры нейронной сети: сеть 1, обученную на идеальных последовательностях, и сеть 2, обученную на зашумленных последовательностях. Проверка функционирования производится на 100 векторах входа при различных уровнях шума.

Приведем фрагмент сценарий appcr1, который выполняет эти операции

```
tic, noise_range = 0:.05:.5; max_test = 20;
network1 = []; network2 = []; T = targets;
 for noiselevel = noise_range
    errors1=0; errors2=0;
   for i=1:max_test
    P = alphabet + randn(35,26)*noiselevel;
    A = sim(net,P);   AA = compet(A);
    errors1 = errors1 + sum(sum(abs(AA - T)))/2;
    An = sim(netn,P);  AAn = compet(An);
    errors2 = errors2 + sum(sum(abs(AAn - T)))/2; echo off
  end
  network1 = [network1 errors1/26/100];
  network2 = [network2 errors2/26/100];
end
toc
elapsed_time =
   26.2000
```

Тестирование реализуется следующим образом. Шум со средним значением 0 и стандартным отклонением от 0 до 0.5 с шагом 0.05 добавляется к векторам входа. Для каждого уровня шума формируется 100 зашумленных последовательностей для каждого символа и вычисляется выход сети. Выходной сигнал обрабатывается М-функцией compet с той целью, чтобы только один из 26 элементов вектора выхода После этого оценивается количество ошибочных классификаций и вычисляется процент ошибки.

Соответствующий график погрешности сети от уровня входного шума показан на рис. 9.25.

```
figure(1),clf
plot(noise_range,network1*100,'--',noise_range,network2*100);
grid on
```

Проверим работу нейронной сети для распознавания символов. Сформируем зашумленный вектор входа для символа J:

```
noisyJ = alphabet(:,10) + randn(35,1)*0.2;
plotchar(noisyJ); % Зашумленный символ J (рис.9.26)

A2 = sim(net,noisyJ); A2 = compet(A2);
answer = find(compet(A2) == 1)
plotchar(alphabet(:,answer)); % Распознанный символ J
```

```
answer =    10
```

Нейронная сеть выделила 10 правильных элементов и восстановила символ J без ошибок.

Глава 11

Модели сетей

C245. NETWORK

Пример:

Создадим шаблон нейронной сети с двумя входами (numInputs = 2), тремя слоями (numLayers = 3) и следующими матрицами связности:

BiasConnect = [1; 0; 0] размера numLayers×1;

inputConnect = [1 1; 0 0; 0 0] размера numLayers×numInputs;

layerConnect = [0 0 0; 1 0 0; 0 1 0] размера numLayers×numLayers;

outputConnect = [0 0 1] размера 1× numLayers;

targetConnect = [0 0 1] размера 1× numLayers.

```
net = network(2,3,[1; 0; 0],[1 1; 0 0; 0 0],[0 0 0; 1 0 0; 0 1 0],
...
[0 0 1], [0 0 1]);
gensim(net) % Рис.11.1
```

Введем линии задержки для входов 1 и 2, а также для слоя 3

```
net.inputWeights{1,1}.delays = [0 1];
net.inputWeights{1,2}.delays = [1 2];
net.layerWeights{3,2}.delays = [0 1 2];
gensim(net)
```

Установим параметры нейронной сети и векторов входа:

```
net.inputs{1}.range = [0 1];
net.inputs{2}.range = [0 1];
net.b{1}=-1/4;
net.IW{1,1} = [ 0.5 0.5 ]; net.IW{1,2} = [ 0.5 0.25];
net.LW{2,1} = [ 0.5 ];
net.LW{3,2} = [ 0.5 0.25 1];
P = [0.5 1; 1 0.5];
gensim(net)
```

C248. NEWP

Создать персептрон с одним нейроном, входной вектор которого имеет два элемента, значения которых не выходят за пределы диапазона:

```
clear, net = newp([0 1; 0 1],1);
gensim(net) % Рис.11.6
```

Определим следующую последовательность двухэлементных векторов входа P, составленных из 0 и 1:

```
P = {[0; 0] [0; 1] [1; 0] [1; 1]};
```

Обучим персептрон выполнять операцию ЛОГИЧЕСКОЕ И. С этой целью для полного набора входных векторов сформируем последовательность целей

```
P1 = cat(2, P{:}); T1 = num2cell(P1(1,:)&P1(2,:))
T1 =
    [0]    [0]    [0]    [1]
```

Применим процедуру адаптации, установив число проходов равным 10:

```
net.adaptParam.passes = 10; net = adapt(net,P,T1);
```

Вектор весов и смещение можно определить следующим образом

```
net.IW{1}, net.b{1}

ans =
     2     1
ans =
    -3
```

Таким образом, разделяющая линия имеет вид

$$L: 2p_1 + p_2 - 3 = 0.$$

Промоделируем спроектированную нейронную сеть, подав входную обучающую последовательность

```
Y = sim(net,P)

Y =
    [0]    [0]    [0]    [1]
```

Настройка параметров сети выполнена правильно.

Обучим персептрон выполнять операцию НЕИСКЛЮЧАЮЩЕЕ ИЛИ.

С этой целью для полного набора входных векторов P сформируем последовательность целей

```
P1 = cat(2, P{:}); T2 = num2cell(P1(1, :) | P1(2, :))

T2 =
```

```
    [0]    [1]    [1]    [1]
```

Применим процедуру обучения, установив число циклов равным 20:

```
net.trainParam.epochs = 20; net = train(net,P,T2);
```

```
TRAINC, Epoch 0/20
TRAINC, Epoch 2/20
TRAINC, Performance goal met.
```

Вектор весов и смещение можно определить следующим образом

```
net.IW{1}, net.b{1}, net.IW{1}, net.b{1}
```

```
ans =
     2     2
ans =
    -2
ans =
     2     2
ans =
    -2
```

Таким образом, разделяющая линия имеет вид

$$L: 2p_1 + 2p_2 - 2 = 0.$$

Промоделируем спроектированную нейронную сеть, подав входную обучающую последовательность

```
Y = sim(net,P)
```

```
Y =
    [0]    [1]    [1]    [1]
```

Обучение и настройка сети выполнены правильно.

C250. NEWLIN

Сформировать линейный слой, который для заданного входа воспроизводит заданный отклик системы.

Архитектура линейного слоя: линия задержки типа [0 1 2], один нейрон, вектор входа с элементами из диапазона [-1 1], параметр скорости настройки 0.01.

```
net = newlin([-1 1], 1, [0 1 2], 0.01);
gensim(net)  %Рис.11.7
```

Сформируем следующие обучающие последовательности векторов входа и цели

```
P1 = {0 -1 1 1 0 -1 1 0 0 1}; T1 = {0 -1 0 2 1 -1 0 1 0 1};
```

```
P2 = {1 0 -1 -1 1 1 1 0 -1}; T2 = {2 1 -1 -2 0 2 2 1  0};
```

Выполним обучение, используя только обучающие последовательности P1 и T1:

```
net = train(net,P1,T1);
```

```
TRAINB, Epoch 0/100, MSE 0.9/0.
TRAINB, Epoch 25/100, MSE 0.0959309/0.
TRAINB, Epoch 50/100, MSE 0.0298875/0.
TRAINB, Epoch 75/100, MSE 0.0108788/0.
TRAINB, Epoch 100/100, MSE 0.00403078/0.
TRAINB, Maximum epoch reached.
```

Соответствующие значения весов и смещения следующие

```
net.IW{1}, net.b{1}
```

```
ans =
    0.9198    0.9301   -0.0876
ans =
    0.0396
```

Выполним моделирование сети для всех значений входа, объединяющих векторы P1 и P2

```
Y1 = sim(net,[P1 P2]);
```

Теперь выполним обучение сети на всем объеме обучающих данных, соответствующем объединению векторов входа {[P1 P2]} и векторов целей {[T1 T2]}.

```
net = init(net);
P3 = [P1 P2]; T3 = [T1 T2];
net.trainParam.epochs = 200;
net.trainParam.goal = 0.01;
net = train(net,P3,T3);
```

```
TRAINB, Epoch 0/200, MSE 1.47368/0.01.
TRAINB, Epoch 25/200, MSE 0.0478548/0.01.
TRAINB, Epoch 50/200, MSE 0.0438463/0.01.
TRAINB, Epoch 75/200, MSE 0.0437006/0.01.
TRAINB, Epoch 100/200, MSE 0.0436917/0.01.
TRAINB, Epoch 125/200, MSE 0.0436911/0.01.
TRAINB, Epoch 150/200, MSE 0.0436911/0.01.
TRAINB, Epoch 175/200, MSE 0.0436911/0.01.
TRAINB, Epoch 200/200, MSE 0.0436911/0.01.
```

```
TRAINB, Maximum epoch reached.
```

В этом случае процедура обучения не достигает предельной точности в течение 200 циклов обучения и, судя по виду, кривой имеет статическую ошибку.

Значения весов и смещений несколько изменяются

```
net.IW{1}, net.b{1}
ans =
    0.9242    0.9869    0.0339
ans =
    0.0602
```

Результаты моделирования представлены в виде зависимости Y3.

```
Y3 = sim(net,[P1 P2]);
figure(1), clf, plot(cat(2,Y1{:}),'r'), hold on, grid on
stairs(cat(2,T3{:}),'k'), plot(cat(2,Y3{:}),'b')
```

C253. NEWLIND

Требуется сформировать линейный слой, который обеспечивает для заданного входа P выход, близкий к цели T, если заданы следующие обучающие последовательности

```
P = 0:3; T = [0.0 2.0 4.1 5.9];
```

Анализ данных подсказывает, что требуется найти аппроксимирующую кривую, которая близка к зависимости $t = 2p$. Применение линейного слоя LIND в данном случае вполне оправдано.

```
net = newlind(P,T); gensim(net) % Рис.11.11
```

Значения весов и смещений равны

```
net.IW{1}, net.b{1}
ans =
    1.9800
ans =
    0.0300
```

Выполним моделирование сформированного линейного слоя

```
Y = sim(net,P)
Y =
    0.0300    2.0100    3.9900    5.9700
```

Многослойные сети.

C255. NEWFF

Создать нейронную сеть, чтобы обеспечить следующее отображение последовательности входа P в последовательность целей T:

```
P = [0 1 2 3 4 5 6 7 8 9 10]; T = [0 1 2 3 4 3 2 1 2 3 4];
```

Архитектура нейронной сети: двухслойная сеть с прямой передачей сигнала; первый слой - 5 нейронов с функцией активации tansig; второй слой - 1 нейрон с функцией активации purelin; диапазон изменения входа [0 10].

```
net = newff([0 10],[5 1],{'tansig' 'purelin'});
gensim(net)
```

Выполним моделирование сети и построим графики сигналов выхода и цели:

```
Y = sim(net,P); figure(1), clf
plot(P, T, P, Y, 'o'), grid on %Рис.11.13
```

Обучим сеть в течение 50 циклов:

```
net.trainParam.epochs = 50;
net = train(net,P,T);
TRAINLM, Epoch 0/50, MSE 0.0154002/0, Gradient 0.190957/1e-010
TRAINLM, Epoch 25/50, MSE 0.0153319/0, Gradient 0.0398632/1e-010
TRAINLM, Epoch 50/50, MSE 0.0153054/0, Gradient 0.017265/1e-010
TRAINLM, Maximum epoch reached, performance goal was not met.
```

Выполним моделирование сформированной двухслойной сети, используя обучающую последовательность входа

```
Y = sim(net,P);
figure(2), clf, plot(P,T,P,Y,'o'), grid on % Рис.11.15
```

C258. NEWFFTD

Создать нейронную сеть, чтобы обеспечить следующее отображение последовательности входа P в последовательность целей T:

```
P = {1 0 0 1 1 0 1 0 0 0 0 1 1 0 0 1};
T = {1 -1 0 1 0 -1 1 -1 0 0 0 1 0 -1 0 1};
```

Архитектура нейронной сети: двухслойная сеть с прямой передачей сигнала и линией задержки [0 1]; первый слой - 5 нейронов с функцией активации tansig; второй слой - 1 нейрон с функцией активации purelin; диапазон изменения входа [0 10].

```
net = newfftd([0 1],[0 1],[5 1],{'tansig' 'purelin'});
gensim(net)
```

Обучим сеть в течение 50 циклов и промоделируем, используя в качестве теста обучающую последовательность входа:

```
net.trainParam.epochs = 50;
net = train(net,P,T); Y = sim(net,P)
TRAINLM, Epoch 0/50, MSE 3.10515/0, Gradient 40.7492/1e-010
TRAINLM, Epoch 4/50, MSE 5.31364e-031/0, Gradient 1.09916e-014/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.

Y =
  Columns 1 through 3
    [1]    [-1.0000]    [-3.8858e-016]
  Columns 4 through 6
    [1]    [1.9429e-015]    [-1.0000]
  Columns 7 through 9
    [1]    [-1.0000]    [-3.8858e-016]
  Columns 10 through 11
    [-3.8858e-016]    [-3.8858e-016]
  Columns 12 through 14
    [1]    [1.9429e-015]    [-1.0000]
  Columns 15 through 16
    [-3.8858e-016]    [1]
```

C260. NEWCF

Создать каскадную нейронную сеть, чтобы обеспечить следующее отображение последовательности входа P в последовательность целей T:

```
P = [0 1 2 3 4 5 6 7 8 9 10]; T = [0 1 2 3 4 3 2 1 2 3 4];
```

Архитектура нейронной сети: каскадная двухслойная сеть с прямой передачей сигнала; первый слой - 5 нейронов с функцией активации tansig; второй слой - 1 нейрон с функцией активации purelin; диапазон изменения входа [0 10].

```
net = newcf([0 10],[5 1],{'tansig' 'purelin'});
gensim(net)
```

Обучим сеть в течение 50 циклов:

```
net.trainParam.epochs = 50; net = train(net,P,T);

TRAINLM, Epoch 0/50, MSE 3.01162/0, Gradient 96.4205/1e-010
TRAINLM, Epoch 19/50, MSE 1.63682e-024/0, Gradient 1.1003e-012/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.
```

Выполним моделирование каскадной двухслойной сети, используя обучающую последовательность входа

```
Y = sim(net,P);
figure(1), clf, plot(P,T,P,Y,'or'), grid on
```

Радиальные базисные сети

C263. NEWRB

Создадим радиальную базисную сеть для следующей обучающей последовательности при средней квадратичной ошибки 0.1:

```
P = 0:3; T = [0.0 2.0 4.1 5.9];
net = newrb(P,T,0.1); net.layers{1}.size
gensim(net)
```

```
NEWRB, neurons = 0, SSE = 1.80858
ans =
     3
```

Сформированная радиальная базисная сеть имеет 3 нейрона с функцией активации radbas.

Выполним моделирование сети для нового входа

```
figure(1), clf,
plot(P,T,'*r','MarkerSize',2,'LineWidth',2), hold on
V = sim(net,P);  % Векторы входа из обучающего множества
plot(P,V,'ob','MarkerSize',8, 'LineWidth',2), grid on
P1 = 0.5:2.5; Y = sim(net,P1)
plot(P1,Y,'+k','MarkerSize',10, 'LineWidth',2)% Рис.11.21
```

```
Y =
    1.0346    2.8817    5.5053
```

C265. NEWRBE

Создадим радиальную базисную сеть с нулевой ошибкой для следующей обучающей последовательности:

```
P = 0:3; T = [0.0 2.0 4.1 5.9];
net = newrbe(P,T); net.layers{1}.size
ans =
     4
```

Сформированная радиальная базисная сеть с нулевой ошибкой имеет 4 нейрона в первом слое. Сравните с предыдущим случаем, когда число нейронов было равно трем.

Выполним моделирование сети для нового входа

```
figure(1), clf, plot(P,T,'*r','MarkerSize',2,'LineWidth',2)
hold on, grid on
V = sim(net,P);  % Векторы входа из обучающего множества
plot(P,V,'ob','MarkerSize',8, 'LineWidth',2)
P1 = 0.5:2.5; Y = sim(net,P1)
plot(P1,Y,'+k','MarkerSize',10, 'LineWidth',2)% Рис.11.22
```

```
Y =
    1.0346    2.8817    5.5053
```

C267. NEWGRNN

Создадим обобщенную регрессионную сеть с входами P и целями T:

```
P = 0:3; T = [0.0 2.0 4.1 5.9];
net = newgrnn(P,T); gensim(net)
```

Выполним моделирование сети для нового входа и построим график:

```
figure(1), clf, plot(P,T,'*r','MarkerSize',2,'LineWidth',2)
hold on, grid on
V = sim(net,P);  % Векторы входа из обучающего множества
plot(P,V,'ob','MarkerSize',8, 'LineWidth',2)
P1 = 0.5:2.5; Y = sim(net,P1);
plot(P1,Y,'+k','MarkerSize',10, 'LineWidth',2)% Рис.11.25
Y = sim(net, 0:0.5:3)
Y =
  Columns 1 through 4
    0.8104    1.3759    2.1424    3.0300
  Columns 5 through 7
    3.9030    4.6345    5.1615
```

Из анализа результатов моделирования следует, что на границах интервала расхождения существенны.

Если уменьшить значение параметра влияния до 0.1, то мы получим аппроксимацию высокой точности

```
net = newgrnn(P,T,0.1); Y = sim(net, 0:0.5:3)
```

```
Y =
  Columns 1 through 4
```

```
    0.0000    1.0000    2.0000    3.0500
  Columns 5 through 7
    4.1000    5.0000    5.9000
```

C269. NEWPNN

Задача классификации определена множествами входа P и индексов класса Tc:

```
P = [1 2 3 4 5 6 7]; Tc = [1 2 3 2 2 3 1];
```

Индексы класса преобразуем в вектор целей и сформируем вероятностную нейронную сеть:

```
T = ind2vec(Tc);
net = newpnn(P,T); gensim(net)
```

Выполним моделирование сети, используя обучающее множество входов

```
Y = sim(net,P); Yc = vec2ind(Y)
```

```
Yc =
     1     2     3     2     2     3     1
```

Самоорганизующиеся сети

C271. NEWC

Зададим массив входа P в виде четырех двухэлементных векторов:

```
P = [.1 .8 .1 .9; .2 .9 .1 .8];
```

Необходимо определить центры группирования (кластеризации) близких векторов. В этой задаче интуитивно ясно, что существует два таких центра. Поэтому сформируем слой Кохонена с двумя нейронами и определим диапазон расположения входных векторов в интервале [0 1].

```
net = newc([0 1; 0 1],2); gensim(net)
```

Выполним настройку параметров слоя, чтобы решить задачу для заданного входа

```
net = train(net,P);
```

```
TRAINR, Epoch 0/100
TRAINR, Epoch 25/100
TRAINR, Epoch 50/100
TRAINR, Epoch 75/100
TRAINR, Epoch 100/100
TRAINR, Maximum epoch reached.
```

Центры кластеризации расположены в точках

```
w = net.IW{1}
w =
    0.8031    0.8031
    0.1536    0.1969
```

Построим на плоскости входных векторов точки кластеризации и сами входные векторы

```
figure(1), clf,
plot(P(1,:),P(2,:)','+b','MarkerSize',8, 'Linewidth',2) % Рис.7.3
title(' Векторы входа'), xlabel('P(1,:)'), ylabel('P(2,:)')
hold on, grid on, axis([0 1,0 1])
plot(w(:,1), w(:,2)', 'or', 'MarkerSize',8, 'Linewidth',2)
```

Моделирование сети с определением близости проверяемых точек к центрам кластеризации можно реализовать следующим образом

```
P1 = [0.2:0.1:0.7; 0.2:0.1:0.7]
Y = sim(net,P1); Yc = vec2ind(Y)
```

```
P1 =
  Columns 1 through 4
    0.2000    0.3000    0.4000    0.5000
    0.2000    0.3000    0.4000    0.5000
  Columns 5 through 6
    0.6000    0.7000
    0.6000    0.7000
Yc =
     2     2     2     1     1     1
```

C273. NEWSOM

Задан случайный вектор входа, элементы которого расположены в диапазонах [0 2] и [0 1]. Построить двумерную самоорганизующуюся карту Кохонена с числом нейронов [3 5] для классификации входных векторов:

```
P = [rand(1,400)*2; rand(1,400)];
net = newsom([0 2; 0 1],[3 5]); gensim(net)
```

Построим топологию двумерной карты Кохонена

```
figure(1), clf,
plotsom(net.layers{1}.positions), grid on %Рис.11.29
```

Затем реализуется процедура обучения. Следует отметить, что процедура длится достаточно долго, поэтому для иллюстрации ограничимся 3 циклами:

```
net.trainparam.epochs = 3; tic, net = train(net,P); toc
```

```
figure(2), clf, plot(P(1,: ),P(2,: ),'.g','MarkerSize',10)
hold on, plotsom(net.iw{1,1},net.layers{1}.distances)
TRAINR, Epoch 0/3
TRAINR, Epoch 3/3
TRAINR, Maximum epoch reached.
elapsed_time =
   10.3800
```

Промоделируем обученную карту Кохонена на массиве векторов входа

```
a = sim(net,P);
figure(2), clf, bar(sum(a')) %Рис.11.31
```

Сети – классификаторы входных векторов.

C276. NEWLVQ

Векторы входа P и выходные классы Tc, представленные ниже, определяют задачу классификации, которая будет решена LVQ сетью:

```
P = [-3 -2 -2 0 0  0  0 +2 +2 +3;  0 +1 -1 2 1 -1 -2 +1 -1  0];
Tc = [1  1  1 2 2  2  2  1  1  1];
```

Целевые классы Tc преобразуем к целевым векторам T, создадим сеть LVQ со входами P, четырьмя нейронами и долями распределения по классам [0.6 0.4]

```
T = ind2vec(Tc);
net = newlvq(minmax(P),4,[.6 .4]); gensim(net) %
```

Выполним обучение сети:

```
net = train(net,P,T);
TRAINR, Epoch 0/100
TRAINR, Epoch 2/100
TRAINR, Performance goal met.
```

Промоделируем сеть на векторе входа

```
Y = sim(net,P); Yc = vec2ind(Y)
Yc =
  Columns 1 through 7
     1     1     1     2     2     2     2
  Columns 8 through 10
     1     1     1
```

Рекуррентные сети

C277. NEWLM

Зададим вход P в виде случайной булевой последовательности из нулей и единиц; выходом сети должна быть такая булева последовательность T,

элементы которой принимают значение 1 только в том случае, когда в последовательности P встретились две единицы подряд:

```
P = round(rand(1,20))
T = [0 (P(1:end-1)+P(2:end)==2)]
```

```
P =
  Columns 1 through 7
     0     1     0     0     0     1     1
  Columns 8 through 14
     1     1     0     0     0     1     0
  Columns 15 through 20
     1     0     0     1     1     0
T =
  Columns 1 through 7
     0     0     0     0     0     0     1
  Columns 8 through 14
     1     1     0     0     0     0     0
  Columns 15 through 20
     0     0     0     0     1     0
```

Требуется построить сеть, которая распознает во входном сигнале две единицы, следующие подряд. Сначала эти последовательности представим в виде массивов ячеек:

```
Pseq = con2seq(P); Tseq = con2seq(T);
```

Создадим сеть Элмана с диапазоном входного сигнала от 0 до 1 с 10 скрытыми и одним выходным нейронами:

```
net = newelm([0 1],[10 1],{'tansig','logsig'});  gensim(net)
```

Затем обучим сеть с допустимой средней квадратичной ошибкой 0.001 и смоделируем ее:

```
net.trainParam.goal = 0.001; net.trainParam.epochs = 1000;
net = train(net,Pseq,Tseq);
Y = sim(net,Pseq); Y1 = seq2con(Y);
E = round(T-Y1{1})
TRAINGDX, Epoch 0/1000, MSE 0.187402/0.001, Gradient 0.189387/1e-
006
TRAINGDX, Epoch 25/1000, MSE 0.1745/0.001, Gradient 0.176499/1e-
006
TRAINGDX, Epoch 50/1000, MSE 0.143774/0.001, Gradient 0.152248/1e-
006
TRAINGDX, Epoch 75/1000, MSE 0.0774428/0.001, Gradient
0.128802/1e-006
```

```
   TRAINGDX,   Epoch   100/1000,   MSE   0.0225923/0.001,   Gradient
0.0456008/1e-006
   TRAINGDX,   Epoch   125/1000,   MSE   0.00349329/0.001,   Gradient
0.00901132/1e-006
   TRAINGDX,   Epoch   140/1000,   MSE   0.000923021/0.001,   Gradient
0.00261185/1e-006
   TRAINGDX, Performance goal met.
   E =
     Columns 1 through 7
        0     0     0     0     0     0     0
     Columns 8 through 14
        0     0     0     0     0     0     0
     Columns 15 through 20
        0     0     0     0     0     0
```

C280. NEWHOP

Создадим сеть Хопфилда с двумя устойчивыми точками в трехмерном пространстве

```
clear, T = [-1 -1 1; 1 -1 1]';
net = newhop(T); gensim(net)
```

Проверим, что сеть устойчива в этих точках и используем их как начальные условия для линии задержки. Если сеть устойчива, можно ожидать, что выходы Y будут те же самые

```
Ai = T; [Y,Pf,Af] = sim(net,2,[],Ai); Y

Y =
    -1     1
    -1    -1
     1     1
```

Таким образом, вершины гиперкуба являются устойчивыми точками равновесия сети Хопфилда.

Проверим сходимость сети при произвольном входном векторе Ai

```
Ai = {[-0.9; -0.8; 0.7]};
[Y,Pf,Af] = sim(net,{1 5},{},Ai); Y{1}
ans =
    -1
    -1
     1
```

Сеть Хопфилда обеспечила переход к устойчивому положению равновесия, ближайшему к заданному входу.

Функции активации

Персептрон

C282. HARDLIM

Информация о функции активации hardlim:

```
name = hardlim('name')
dname = hardlim('deriv')
inrange = hardlim('active')
outrange = hardlim('output')
```

```
name =
Hard Limit
dname =
dhardlim
inrange =
     0     0
outrange =
     0     1
```

Зададим следующий вектор входа функции активации с жесткими ограничениями для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = hardlim(N), dA_dN = dhardlim(N,A)
```

```
A =
     1
     1
     0
dA_dN =
     0
     0
     0
```

C283. HARDLIMS

Информация о функции активации hardlims:

```
name = hardlims('name')
dname = hardlims('deriv')
inrange = hardlims('active')
```

```
outrange = hardlims('output')
```

```
name =
Symmetric Hard Limit
dname =
dhardlms
inrange =
     0     0
outrange =
    -1     1
```

Зададим следующий вектор входа симметричной функции активации с жесткими ограничениями для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = hardlims(N),   dA_dN = dhardlms(N,A)
```

```
A =
     1
     1
    -1
dA_dN =
     0
     0
     0
```

Линейные сети

C284. PURELIN

Информация о функции активации purelin:

```
name = purelin('name')
dname = purelin('deriv')
inrange = purelin('active')
outrange = purelin('output')
```

```
name =
Linear
dname =
dpurelin
inrange =
  -Inf   Inf
```

```
outrange =
  -Inf   Inf
```

Зададим следующий вектор входа линейной функции активации для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = purelin(N),   A_dN = dpurelin(N,A)
A =
    0.1000
    0.8000
   -0.7000
A_dN =
     1
     1
     1
```

C285. POSLIN

Информация о функции активации poslin:

```
name = poslin('name')
dname = poslin('deriv')
inrange = poslin('active')
outrange = poslin('output')
```

```
name =
Positive Linear
dname =
dposlin
inrange =
     0   Inf
outrange =
     0   Inf
```

Зададим следующий вектор входа положительной линейной функции активации для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = poslin(N),  dA_dN = dposlin(N,A)
A =
    0.1000
    0.8000
         0
dA_dN =
     1
     1
```

```
     0
```

C286. SATLIN

Информация о функции активации satlin:

```
name = satlin('name')
dname = satlin('deriv')
inrange = satlin('active')
outrange = satlin('output')
name =
Saturating Linear
dname =
dsatlin
inrange =
     0     1
outrange =
     0     1
```

Зададим следующий вектор входа линейной функции активации с ограничениями для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = satlin(N),  dA_dN = dsatlin(N,A)

A =
    0.1000
    0.8000
         0
dA_dN =
     1
     1
     0
```

C288. SATLINS

Информация о функции активации satlins:

```
name = satlins('name')
dname = satlins('deriv')
inrange = satlins('active')
outrange = satlins('output')

name =
Symmetric Saturating Linear
dname =
dsatlins
```

```
inrange =
    -1     1
outrange =
    -1     1
```

Зададим следующий вектор входа симметричной линейной функции активации с ограничениями для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = satlins(N),  dA_dN = dsatlins(N,A)
A =
    0.1000
    0.8000
   -0.7000
dA_dN =
     1
     1
     1
```

Радиальные базисные сети

C289. RADBAS

Информация о функции активации radbas:

```
name = radbas('name')
dname = radbas('deriv')
inrange = radbas('active')
outrange = radbas('output')
name =
Radial Basis
dname =
dradbas
inrange =
    -2     2
outrange =
     0     1
```

Следующая последовательность команд создает график функции активации radbas:

```
n = -3:0.1:3; a = radbas(n);
figure(1), clf, plot(n,a), grid on
```

Зададим следующий вектор входа радиальной базисной функции активации для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = radbas(N), dA_dN = dradbas(N,A)
A =
```

```
    0.9900
    0.5273
    0.6126
dA_dN =
   -0.1980
   -0.8437
    0.8577
```

C290. TRIBAS

Информация о функции активации tribas:

```
name = tribas('name')
dname = tribas('deriv')
inrange = tribas('active')
outrange = tribas('output')
name =
Triangle Basis
dname =
dtribas
inrange =
    -1     1
outrange =
     0     1
```

Построим график функции активации tribas:

```
n = -2:0.1:2; a = tribas(n);
figure(1), clf, plot(n,a), grid on
```

Зададим следующий вектор входа треугольной функции активации для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = tribas(N),  dA_dN = dtribas(N,A)
A =
    0.9000
    0.2000
    0.3000
dA_dN =
    -1
    -1
     1
```

Самоогранизующиеся сети

C292. COMPET

Информация о функции активации compet:

```
name = compet('name')
```

```
dname = compet('deriv')
inrange = compet('active')
outrange = compet('output')
```

```
name =
Competitive
dname =
     ''
inrange =
  -Inf   Inf
outrange =
     0     1
```

Зададим следующий вектор входа конкурирующей функции активации, вычислим выход и представим входы и выходы в виде столбцовых диаграмм

```
n = [0; 1; -0.5; 0.5]; a = compet(n);
figure(1), clf
subplot(2,1,1), bar(n), ylabel('n')
subplot(2,1,2), bar(a), ylabel('a')
```

C293. SOFTMAX

Информация о функции активации softmax:

```
name = softmax('name')
dname = softmax('deriv')
inrange = softmax('active')
outrange = softmax('output')
```

```
name =
Soft Max
dname =
     ''
inrange =
  -Inf   Inf
outrange =
     0     1
```

Зададим следующий вектор входа конкурирующей функции активации с мягким максимумом, вычислим выход и представим входы и выходы в виде столбцовых диаграмм

```
n = [0; 1; -0.5; 0.5]; a = softmax(n);
figure(1), clf
subplot(2,1,1), bar(n), ylabel('n')
subplot(2,1,2), bar(a), ylabel('a')
```

Рекуррентные сети

C294. LOGSIG

Информация о функции активации logsig:

```
name = logsig('name')
dname = logsig('deriv')
inrange = logsig('active')
outrange = logsig('output')
```

```
name =
Log Sigmoid
dname =
dlogsig
inrange =
    -4     4
outrange =
     0     1
```

Построим график функции активации logsig:

```
n = -5:0.1:5; a = logsig(n);
figure(1), clf, plot(n,a), grid on
```

Зададим следующий вектор входа логистической функции активации для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = logsig(N), dA_dN = dlogsig(N,A)
```

```
A =
    0.5250
    0.6900
    0.3318
dA_dN =
    0.2494
    0.2139
    0.2217
```

C296. TANSIG

Информация о функции активации tansig:

```
name = tansig('name')
```

```
dname = tansig('deriv')
inrange = tansig('active')
outrange = tansig('output')
```

```
name =
Tan Sigmoid
dname =
dtansig
inrange =
    -2     2
outrange =
    -1     1
```

Следующая последовательность команд строит график функции активации tansig:

```
n = -3:0.1:3; a = tansig(n);
figure(1), clf, plot(n,a), grid on
```

Зададим следующий вектор входа гиперболической тангенциальной функции активации для слоя из 3 нейронов и рассчитаем векторы выхода A и производной dA_dN:

```
N = [0.1; 0.8; -0.7]; A = tansig(N),  dA_dN = dtansig(N,A)
```

```
A =
    0.0997
    0.6640
   -0.6044
dA_dN =
    0.9901
    0.5591
    0.6347
```

Синаптические функции

Функции взвешивания и расстояний

C298. DOTPROD

Вычислим вектор взвешенных входов Z, если заданы случайные матрица весов W и вектор входа P:

```
W = rand(4,3), P = rand(3,1);  P'
Z = dotprod(W,P); Z'
W =
    0.5786    0.8690    0.1990
```

```
    0.0143    0.3324    0.3554
    0.6843    0.5951    0.4652
    0.8174    0.8755    0.3119
ans =
    0.0856    0.3270    0.1833
ans =
    0.3702    0.1751    0.3384    0.4134
```

Определим имя M-функции, которая вычисляет производную взвешенной функции в виде скалярного произведения

```
df = dotprod('deriv')
df =
ddotprod
```

Вычислим производные скалярного произведения по каждому аргументу

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

dZ_dP =
    0.5786    0.8690    0.1990
    0.0143    0.3324    0.3554
    0.6843    0.5951    0.4652
    0.8174    0.8755    0.3119
dZ_dW =
    0.0856
    0.3270
    0.1833
```

C299. NORMPROD

Определим случайную весовую матрицу W и вектор входа P и рассчитаем соответствующий взвешенный вход Z:

```
W = rand(4,3), P = rand(3,1); P'
Z = normprod(W,P); Z'

W =
    0.7504    0.3715    0.3784
    0.7498    0.8191    0.5556
    0.2033    0.3740    0.2942
    0.7711    0.4528    0.4704
ans =
    0.6137    0.9855    0.4298
ans =
    0.4876    0.7423    0.3055    0.5528
```

C300. DIST

Вычислим вектор взвешенных входов Z, если заданы случайные матрица весов W размера 4×3 и вектор входа P размера 3×1:

```
W = rand(4,3); P = rand(3,1);
Z = dist(W,P)
Z =
    0.7709
    0.7344
    0.9168
    0.4496
```

Рассмотрим сеть с топологией, для которой задана случайная матрица координат для 10 нейронов в трехмерном пространстве:

```
pos = rand(3,10);
figure(1), clf, plotsom(pos)
```

Рассчитаем матрицу евклидовых расстояний между ними:

```
D = dist(pos)
D =
  Columns 1 through 4
         0    1.1593    0.4322    0.6111
    1.1593         0    0.7624    0.9547
    0.4322    0.7624         0    0.5944
    0.6111    0.9547    0.5944         0
    0.5083    1.0085    0.3617    0.7094
    0.8022    0.4755    0.4894    0.4819
    0.8231    0.9197    0.6729    0.3889
    0.5363    0.8468    0.2398    0.8280
    0.6459    0.6005    0.2508    0.5793
    0.6291    0.8130    0.4113    0.4157
  Columns 5 through 8
    0.5083    0.8022    0.8231    0.5363
    1.0085    0.4755    0.9197    0.8468
    0.3617    0.4894    0.6729    0.2398
    0.7094    0.4819    0.3889    0.8280
         0    0.7149    0.6493    0.3683
    0.7149         0    0.4978    0.6846
    0.6493    0.4978         0    0.8611
    0.3683    0.6846    0.8611         0
    0.4189    0.3303    0.5338    0.3930
    0.3993    0.4132    0.2775    0.5846
  Columns 9 through 10
    0.6459    0.6291
```

```
    0.6005    0.8130
    0.2508    0.4113
    0.5793    0.4157
    0.4189    0.3993
    0.3303    0.4132
    0.5338    0.2775
    0.3930    0.5846
         0    0.2916
    0.2916         0
```

C302. NEGDIST

Вычислим вектор взвешенных входов Z, если заданы случайные матрица весов W размера 4×3 и вектор входа P размера 3×1:

```
W = rand(4,3); P = rand(3,1);
Z = negdist(W,P)
Z =
   -0.2495
   -0.6759
   -0.6959
   -0.3220
```

C303. MANDIST

Вычислим вектор взвешенных входов Z, если заданы случайные матрица весов W размера 4×3 и вектор входа P размера 3×1:

```
W = rand(4,3); P = rand(3,1);
Z = mandist(W,P)
Z =
    1.1527
    0.8322
    0.5810
    0.8248
```

Рассмотрим сеть с топологией, для которой задана случайная матрица координат для 10 нейронов в трехмерном пространстве:

```
pos = rand(3,10);
figure(1), clf, plotsom(pos)
```

Рассчитаем матрицу евклидовых расстояний между ними:

```
D = mandist(pos)
D =
  Columns 1 through 4
         0    1.4762    1.2734    1.7250
    1.4762         0    1.0259    1.1297
```

```
    1.2734    1.0259         0    0.4516
    1.7250    1.1297    0.4516         0
    1.0001    0.5871    0.9674    1.3264
    1.5999    0.5968    0.7643    0.9726
    1.5179    0.6554    0.9023    1.1435
    1.9948    0.7160    1.1132    1.1711
    1.2926    1.5432    0.5173    0.5361
    1.0572    0.4190    0.7406    1.1921
  Columns 5 through 8
    1.0001    1.5999    1.5179    1.9948
    0.5871    0.5968    0.6554    0.7160
    0.9674    0.7643    0.9023    1.1132
    1.3264    0.9726    1.1435    1.1711
         0    0.8495    0.9874    1.1058
    0.8495         0    0.2168    0.3950
    0.9874    0.2168         0    0.5011
    1.1058    0.3950    0.5011         0
    1.4847    1.0976    1.0155    1.4925
    0.2881    0.5613    0.6993    0.9376
  Columns 9 through 10
    1.2926    1.0572
    1.5432    0.4190
    0.5173    0.7406
    0.5361    1.1921
    1.4847    0.2881
    1.0976    0.5613
    1.0155    0.6993
    1.4925    0.9376
         0    1.2304
    1.2304         0
```

C305. BOXDIST

Пусть задан массив случайных координат трехмерного пространства, в которых размещены 10 нейронов.

```
pos = rand(3,10);
figure(1), clf, plotsom(pos)
```

Требуется вычислить массив максимальных координатных смещений между этими нейронами:

```
d = boxdist(pos)
```

```
d =
  Columns 1 through 4
```

```
         0    0.3489    0.6079    0.6925
    0.3489         0    0.7883    0.6322
    0.6079    0.7883         0    0.2023
    0.6925    0.6322    0.2023         0
    0.5053    0.6620    0.6220    0.6318
    0.6334    0.3989    0.7898    0.7598
    0.4530    0.6334    0.1549    0.3527
    0.6165    0.3256    0.6768    0.6865
    0.3630    0.2419    0.8707    0.7146
    0.5207    0.7011    0.6201    0.6299
  Columns 5 through 8
    0.5053    0.6334    0.4530    0.6165
    0.6620    0.3989    0.6334    0.3256
    0.6220    0.7898    0.1549    0.6768
    0.6318    0.7598    0.3527    0.6865
         0    0.6635    0.6192    0.5456
    0.6635         0    0.7472    0.2296
    0.6192    0.7472         0    0.6739
    0.5456    0.2296    0.6739         0
    0.7444    0.6407    0.7158    0.5675
    0.2628    0.7026    0.6173    0.5006
  Columns 9 through 10
    0.3630    0.5207
    0.2419    0.7011
    0.8707    0.6201
    0.7146    0.6299
    0.7444    0.2628
    0.6407    0.7026
    0.7158    0.6173
    0.5675    0.5006
         0    0.7835
    0.7835         0
```

C306. LINKDIST

Пусть задан массив случайных координат трехмерного пространства, в которых размещены 10 нейронов. Требуется вычислить массив расстояний связи между этими нейронами:

```
pos = rand(3,10);
figure(1), clf, plotsom(pos)
d = linkdist(pos)

d =
```

```
Columns 1 through 7
   0     1     1     1     1     1     1
   1     0     1     1     1     1     1
   1     1     0     1     1     1     1
   1     1     1     0     1     1     1
   1     1     1     1     0     1     2
   1     1     1     1     1     0     1
   1     1     1     1     2     1     0
   1     1     1     1     1     1     1
   1     1     1     1     1     1     1
   1     1     1     1     1     1     1
Columns 8 through 10
   1     1     1
   1     1     1
   1     1     1
   1     1     1
   1     1     1
   1     1     1
   1     1     1
   0     1     2
   1     0     1
   2     1     0
```

Функции накопления

C307. NETSUM

Вычислить функцию накопления потенциала для следующих взвешенных входов:

```
Z1 = [ 1 2 4;  3  4 1]; Z2 = [-1 2 2; -5 -6 1];
N = netsum(Z1,Z2)
N =
     0     4     6
    -2    -2     2
```

Вычислить функцию накопления с учетом вектора смещения b. Поскольку массивы Z1 и Z2 составлены из трех векторов, то с помощью функции concur должен быть создан массив из трех копий вектора смещения b для того, чтобы все размерности совпадали:

```
b = [0; -1]; N = netsum(Z1,Z2,concur(b,size(Z1,2)))


N =
     0     4     6
    -3    -3     1
```

Определим две весовых матрицы входа для слоя с тремя нейронами:

```
Z1 = [0; 1; -1]; Z2 = [1; 0.5; 1.2];
```

Вычислить вход нейрона N с помощью функции netsum и затем найти производные по каждому из взвешенных входов:

```
N = netsum(Z1,Z2)
dN_dZ1 = dnetsum(Z1,N)
dN_dZ2 = dnetsum(Z2,N)
```

```
N =
    1.0000
    1.5000
    0.2000
dN_dZ1 =
     1
     1
     1
dN_dZ2 =
     1
     1
     1
```

C309. NETPROD

Вычислить функцию накопления для следующих взвешенных входов:

```
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
```

Вычислить функцию накопления с учетом вектора смещения b. Поскольку массивы Z1 и Z2 составлены из трех векторов, то с помощью функции concur должен быть создан массив из трех копий вектора смещения b для того, чтобы все размерности совпадали:

```
b = [0; -1]; N = netprod(Z1,Z2,concur(b,size(Z1,2)))
```

```
N =
     0     0     0
    15    24    -1
```

Определим два взвешенных входа для слоя с тремя нейронами:

```
Z1 = [0; 1; -1]; Z2 = [1; 0.5; 1.2];
```

Вычислить вход нейрона N с помощью функции netprod и затем найти производные по каждому из взвешенных входов:

```
N = netprod(Z1,Z2); N'
dN_dZ1 = dnetprod(Z1,N); dN_dZ1'
dN_dZ2 = dnetprod(Z2,N); dN_dZ2'
ans =
         0    0.5000   -1.2000
Warning: Divide by zero.
> In C:\MATLAB6P1\toolbox\nnet\nnet\dnetprod.m at line 43
ans =
       NaN    0.5000    1.2000
ans =
     0     1    -1
```

Функции инициализации

C310. INIT

Сформировать персептрон с одним нейроном, вход которого имеет 2 элемента со значениями в диапазонах [0 1] и [–2 2]:

```
net = newp([0 1; -2 2], 1);
net.initFcn
net.layers{1}.initFcn
net.inputWeights{1}.initFcn
net.biases{1}.initFcn
ans =
initlay
ans =
initwb
ans =
initzero
ans =
initzero
```

Выведем значения установленных весов и смещений:

```
net.IW{1,1}, net.b{1}
ans =
     0     0
ans =
     0
```

Обучим персептрон на следующих обучающих множествах

```
P = [0 1 0 1; 0 0 1 1]; T = [0 0 0 1];
net = train(net,P,T); net.IW{1,1}, net.b{1}
TRAINC, Epoch 0/100
TRAINC, Epoch 6/100
TRAINC, Performance goal met.

ans =
     1     2
ans =
    -3
```

Для того чтобы возвратиться к начальным значениям весов и смещений, характерных для данной сети, и предназначена функция init :

```
net = init(net); net.IW{1,1}, net.b{1}
ans =
     0     0
ans =
     0
```

C314. INITZERO

Присвоить нулевые значения матрице весов и вектору смещения для слоя с 5 нейронами и вектором входа, элементы которого принимают значения в диапазонах [0 1], [-2 2]:

```
W = initzero(5, [0 1; -2 2]),  b = initzero(5, [1 1])

W =
     0     0
     0     0
     0     0
     0     0
     0     0
b =
     0
     0
     0
     0
     0
```

C315. MIDPOINT

Присвоить нулевые значения матрице весов и вектору смещения для слоя с 5 нейронами и вектором входа, элементы которого принимают значения в диапазонах [0 1], [-2 2]:

```
W = midpoint(5,[0 1; -2 2])
W =
    0.5000         0
    0.5000         0
    0.5000         0
    0.5000         0
    0.5000         0
```

C316. RANDS

Сформируем с помощью функции rands массивы случайных величин различных размеров:

```
v = rands(4)
M = rands(2,3)
W = rands(4,[0 1; -2 2])
v =
    0.9003
   -0.5377
    0.2137
   -0.0280
M =
    0.7826   -0.0871    0.6428
    0.5242   -0.9630   -0.1106
W =
    0.2309   -0.6475
    0.5839   -0.1886
    0.8436    0.8709
    0.4764    0.8338
```

C317. RANDNC

Сформируем случайный массив из четырех нормированных трехэлементных столбцов:

```
M = randnc(3,4)


M =
   -0.1499   -0.2452   -0.6491    0.2650
    0.6575    0.5220   -0.5343   -0.5817
   -0.7384   -0.8169   -0.5415   -0.7690
```

C318. RANDNR

Создадим случайную матрицу из трех нормированных четырехэлементных строк:

```
W = randnr(3,4)


W =
   -0.6415    0.5715    0.4582    0.2278
    0.5866   -0.0808    0.0598    0.8036
   -0.0957   -0.1419   -0.5186   -0.8377
```

C318. INITCON

Начальные значения смещений рассчитаем для слоя с 5 нейронами:

```
b = initcon(5)
b =
   13.5914
   13.5914
   13.5914
   13.5914
   13.5914
```

Функции адаптации и обучения

Функции адаптации

C325. TRAINS

Создать на основе персептрона нейронную сеть с вектором входа из двух элементов со значениями из диапазона [-2 2] и выполнить адаптивное обучение, используя функцию trains:

Формирование персептрона:

```
net = newp([-2 2;-2 2],1);
```

Формирование векторов входа и цели:

```
P = {[2;2] [1;-2] [-2;2] [-1;1]}; T = {[0] [1] [0] [1]};
```

Адаптивное обучение с использованием трех циклов:

```
net.adaptFcn = 'trains';  net.adaptParam.passes = 3;
[net,a,e]=adapt(net,P,T);  a
a =
    [0]    [1]    [0]    [1]
```

Выход сети после адаптивного обучения полностью совпадает с вектором целей.

Сформировать линейную динамическую нейронную сеть с одним выходом, линией задержки на входе [0 1], диапазон изменения элементов входа [-1 1] и адаптивно обучить ее за 80 проходов с параметром скорости обучения 0.01.

Формирование сети:

```
net = newlin([-1 1],1,[0 1],0.01);
```

Формирование векторов входа и цели:

```
P1 = {0 -1 1 1 0 -1 1 0 0 1}; T1 = {0 -1 0 2 1 -1 0 1 0 1};
```

Адаптивное обучение:

```
net.trainFcn='trains'; net.trainParam.passes = 80;

[net,TR]=train(net,P1,T1);
```

Моделирование сети при заданном входе:

```
Y = sim(net,P1); Y = [Y{:}]
```

```
Y =
  Columns 1 through 4
    0.0201   -0.9411    0.0282    1.9344
  Columns 5 through 8
    0.9732   -0.9411    0.0282    0.9732
  Columns 9 through 10
    0.0201    0.9813
```

Результат моделирования близок к вектору целей. При этом среднеквадратическая ошибка составляет

```
E = mse(Y-[T1{:}])
```

```
E =
    0.0015
```

Функции обучения

C331. TRAINB

Сформировать динамическую линейную нейронную сеть с одним выходом, линией задержки на входе [0 1], диапазон изменения элементов входа [-1 1] и адаптивно обучить ее за 200 циклов с параметром скорости обучения 0.01.

Формирование сети:

```
net = newlin([-1 1],1,[0 1],0.01);
```

Формирование векторов входа и цели:

```
P1 = {0 -1 1 1 0 -1 1 0 0 1}; T1 = {0 -1 0 2 1 -1 0 1 0 1};
```

Групповое обучение:

```
net.trainFcn='trainb'; net.trainParam.epochs = 200;

[net,TR]=train(net,P1,T1);
```

```
TRAINB, Epoch 0/200, MSE 0.9/0.
TRAINB, Epoch 25/200, MSE 0.0999258/0.
TRAINB, Epoch 50/200, MSE 0.0146419/0.
TRAINB, Epoch 75/200, MSE 0.00216069/0.
TRAINB, Epoch 100/200, MSE 0.000318915/0.
TRAINB, Epoch 125/200, MSE 4.70722e-005/0.
```

```
TRAINB, Epoch 150/200, MSE 6.94794e-006/0.
TRAINB, Epoch 175/200, MSE 1.02553e-006/0.
TRAINB, Epoch 200/200, MSE 1.5137e-007/0.
TRAINB, Maximum epoch reached.
```

Выполним моделирование сети при заданном входе:

```
Y = sim(net,P1); Y = [Y{:}]
```

```
Y =
  Columns 1 through 4
    0.0002   -0.9994    0.0003    1.9993
  Columns 5 through 8
    0.9997   -0.9994    0.0003    0.9997
  Columns 9 through 10
    0.0002    0.9998
```

Результат моделирования близок к вектору целей. При этом среднеквадратическая ошибка составляет

```
E = mse(Y-[T1{:}])
```

```
E =
  1.5137e-007
```

C334. TRAINC

Сформировать нейронную сеть на основе персептрона с одним выходом и вектором входа из двух элементов, принимающих значения в диапазоне [–2 2].

Формирование сети:

```
net = newp([-2 2; -2 2],1);
```

Формирование векторов входа и цели:

```
P = [ 2  1 -2 -1;  2 -2  2  1]; T = [ 0  1  0  1];
```

Обучение с циклическим представлением входа:

```
net.trainFcn='trainc';
[net,TR]=train(net,P,T);
```

```
TRAINC, Epoch 0/100
TRAINC, Epoch 3/100
TRAINC, Performance goal met.
```

Выполним моделирование сети при заданном входе:

```
Y = sim(net, P)
```

```
Y =
     0     1     0     1
```

Найдем значения весов и смещения:

```
net.IW{1,1}, net.b{1}

ans =
    -2    -3
ans =
     1
```

C337. TRAINR

Сформировать самоорганизующуюся нейронную сеть для разделения векторов входа на два класса. Векторы входа состоят из двух элементов со значениями из диапазона [0 1]. Сеть имеет два выхода (по числу классов) и обучается с помощью функции trainr.

Формирование сети:

```
clear, net = newc([0 1; 0 1], 2);
```

Формирование векторов входа:

```
P = [.1 .8 .1 .9;  .2 .9 .1 .8];
```

Обучение с циклическим представлением входа:

```
net.trainFcn ='trainr';
net = train(net,P);
TRAINR, Epoch 0/100
TRAINR, Epoch 25/100
TRAINR, Epoch 50/100
TRAINR, Epoch 75/100
TRAINR, Epoch 100/100
TRAINR, Maximum epoch reached.
```

Выполним моделирование сети, и значения выхода преобразуем в индексы классов:

```
Y = sim(net,P); Yc = vec2ind(Y)

Yc =
     2     1     2     1
```

Градиентные алгоритмы обучения

C344. TRAINGD

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию traingd.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'traingd');
```

Обучение сети:

```
net.trainParam.epochs=1000; net.trainParam.show=50;
net.trainParam.goal = 0.01;
[net,TR] = train(net,P,T);
TRAINGD, Epoch 0/1000, MSE 0.015653/0.01, Gradient 0.0317272/1e-
010
TRAINGD, Epoch 50/1000, MSE 0.0151675/0.01, Gradient 0.0305964/1e-
010
TRAINGD, Epoch 100/1000, MSE 0.0147153/0.01, Gradient
0.0295528/1e-010
TRAINGD, Epoch 150/1000, MSE 0.0142928/0.01, Gradient
0.0285863/1e-010
TRAINGD, Epoch 200/1000, MSE 0.0138969/0.01, Gradient
0.0276881/1e-010
TRAINGD, Epoch 250/1000, MSE 0.0135251/0.01, Gradient 0.026851/1e-
010
TRAINGD, Epoch 300/1000, MSE 0.013175/0.01, Gradient 0.0260685/1e-
010
TRAINGD, Epoch 350/1000, MSE 0.0128447/0.01, Gradient
0.0253352/1e-010
TRAINGD, Epoch 400/1000, MSE 0.0125324/0.01, Gradient
0.0246461/1e-010
TRAINGD, Epoch 450/1000, MSE 0.0122366/0.01, Gradient
0.0239972/1e-010
TRAINGD, Epoch 500/1000, MSE 0.011956/0.01, Gradient 0.0233848/1e-
010
TRAINGD, Epoch 550/1000, MSE 0.0116892/0.01, Gradient
0.0228057/1e-010
TRAINGD, Epoch 600/1000, MSE 0.0114354/0.01, Gradient 0.022257/1e-
010
TRAINGD, Epoch 650/1000, MSE 0.0111935/0.01, Gradient
0.0217363/1e-010
```

```
TRAINGD,    Epoch    700/1000,    MSE    0.0109625/0.01,    Gradient
0.0212413/1e-010
TRAINGD,    Epoch    750/1000,    MSE    0.0107419/0.01,    Gradient
0.0207702/1e-010
TRAINGD, Epoch 800/1000, MSE 0.0105308/0.01, Gradient 0.020321/1e-
010
TRAINGD,    Epoch    850/1000,    MSE    0.0103287/0.01,    Gradient
0.0198922/1e-010
TRAINGD,    Epoch    900/1000,    MSE    0.0101349/0.01,    Gradient
0.0194824/1e-010
TRAINGD,    Epoch    937/1000,    MSE    0.0099965/0.01,    Gradient
0.0191906/1e-010
TRAINGD, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
Y =
  Columns 1 through 4
    0.1034    0.0460    0.0968    0.8716
  Columns 5 through 6
    0.8966    0.8968
```

C346. TRAINGDA

Заданы следующие обучающие последовательности входов P и целей T:

```
clear, P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию traingda.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'traingda');
```

Обучение сети:

```
net.trainParam.epochs = 500; net.trainParam.goal = 0.01;
[net,TR] = train(net,P,T); grid on

TRAINGDA, Epoch 0/500, MSE 0.428879/0.01, Gradient 0.133123/1e-006
TRAINGDA, Epoch 25/500, MSE 0.419751/0.01, Gradient 0.143599/1e-
006
```

```
TRAINGDA, Epoch 50/500, MSE 0.379232/0.01, Gradient 0.169519/1e-
006
TRAINGDA, Epoch 75/500, MSE 0.0778148/0.01, Gradient 0.110517/1e-
006
TRAINGDA, Epoch 100/500, MSE 0.033615/0.01, Gradient 0.0718189/1e-
006
TRAINGDA, Epoch 125/500, MSE 0.0209789/0.01, Gradient 0.074307/1e-
006
TRAINGDA, Epoch 150/500, MSE 0.0141581/0.01, Gradient
0.0685576/1e-006
TRAINGDA, Epoch 167/500, MSE 0.00993758/0.01, Gradient
0.0257991/1e-006
TRAINGDA, Performance goal met.
```

Выведем график изменения параметра скорости настройки в процессе обучения:

```
figure(2), clf, plot(TR.lr, 'LineWidth',2), grid on
title('Параметр скорости настройки')
xlabel('Число циклов')
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
  Columns 1 through 4
    0.0216    0.0303    0.1558    0.8271
  Columns 5 through 6
    0.9506    0.9596
```

C348. TRAINGDM

Заданы следующие обучающие последовательности входов P и целей T:

```
clear, P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию traingdm и увеличим параметр скорости настройки до 0.1.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'traingdm');
```

Обучение сети:

```
net.trainParam.epochs = 500; net.trainParam.goal = 0.01;
net.trainParam.lr = 0.1;
[net,TR] = train(net,P,T);

TRAINGDM, Epoch 0/500, MSE 0.394818/0.01, Gradient 0.718973/1e-010
TRAINGDM, Epoch 25/500, MSE 0.0108355/0.01, Gradient 0.024793/1e-
010
TRAINGDM, Epoch 50/500, MSE 0.0103841/0.01, Gradient 0.0266315/1e-
010
TRAINGDM, Epoch 59/500, MSE 0.00994986/0.01, Gradient
0.0249995/1e-010
TRAINGDM, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
Y =
  Columns 1 through 4
    0.0741    0.0436    0.1312    0.8831
  Columns 5 through 6
    0.8964    0.8965
```

C351. TRAINGDX

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию traingdx.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'traingdx');
```

Обучение сети:

```
net.trainParam.epochs = 500; net.trainParam.goal = 0.01;
[net,TR] = train(net,P,T);

TRAINGDX, Epoch 0/500, MSE 0.416432/0.01, Gradient 0.61579/1e-006
TRAINGDX, Epoch 25/500, MSE 0.30052/0.01, Gradient 0.457879/1e-006
```

```
TRAINGDX, Epoch 50/500, MSE 0.125779/0.01, Gradient 0.353204/1e-006
TRAINGDX, Epoch 75/500, MSE 0.0136136/0.01, Gradient 0.0293882/1e-006
TRAINGDX, Epoch 90/500, MSE 0.00995653/0.01, Gradient 0.0227043/1e-006
TRAINGDX, Performance goal met.
```

Выведем график изменения параметра скорости настройки в процессе обучения:

```
figure(2), clf, plot(TR.lr, 'LineWidth',2), grid on
title('Параметр скорости настройки')
xlabel('Число циклов')
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
  Columns 1 through 4
    0.0796    0.0404    0.1178    0.8777
  Columns 5 through 6
    0.8928    0.8929
```

C353. TRAINRP

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию trainrp.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'trainrp');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.01;
[net,TR] = train(net,P,T);
```

```
TRAINRP, Epoch 0/50, MSE 0.46999/0.01, Gradient 0.140837/1e-006
```

```
TRAINRP, Epoch 9/50, MSE 0.00869996/0.01, Gradient 0.0321288/1e-
006
TRAINRP, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
  Columns 1 through 4
    0.0365    0.0340    0.0686    0.8333
  Columns 5 through 6
    0.9065    0.9079
```

Алгоритмы метода сопряженных градиентов

C356. TRAINCGF

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию traincgf.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'traincgf');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.001;
[net,TR] = train(net,P,T); grid on
```

```
TRAINCGF-srchcha, Epoch 0/50, MSE 0.165524/0.001, Gradient
0.293981/1e-006
TRAINCGF-srchcha, Epoch 6/50, MSE 0.000964101/0.001, Gradient
0.00271925/1e-006
TRAINCGF, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
```

```
  Columns 1 through 4
    0.0077    0.0121    0.0524    0.9579
  Columns 5 through 6
    0.9760    0.9780
```

C358. TRAINCGP

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию traincgp.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'traincgp');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.001;
[net,TR] = train(net,P,T);  grid on

TRAINCGP-srchcha,  Epoch  0/50,  MSE  0.394041/0.001,  Gradient
0.721591/1e-006
TRAINCGP-srchcha,  Epoch  4/50,  MSE  0.00026955/0.001,  Gradient
0.00334169/1e-006
TRAINCGP, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)

Y =
  Columns 1 through 4
    0.0062    0.0051    0.0374    0.9912
  Columns 5 through 6
    0.9938    0.9938
```

C361. TRAINCGB

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию traincgb.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'traincgb');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.001;
[net,TR] = train(net,P,T);  grid on

TRAINCGB-srchcha, Epoch 0/50, MSE 0.42501/0.001, Gradient
0.558356/1e-006
TRAINCGB-srchcha, Epoch 4/50, MSE 0.00025582/0.001, Gradient
0.00131645/1e-006
TRAINCGB, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
Y =
  Columns 1 through 4
    0.0151    0.0150    0.0323    0.9950
  Columns 5 through 6
    0.9977    0.9977
```

C363. TRAINSCG

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию trainscg.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'trainscg');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.001;
[net,TR] = train(net,P,T); grid on
```

```
TRAINSCG, Epoch 0/50, MSE 0.184004/0.001, Gradient 0.345587/1e-006
TRAINSCG, Epoch 10/50, MSE 0.00179225/0.001, Gradient
0.00384079/1e-006
TRAINSCG, Epoch 12/50, MSE 0.000755243/0.001, Gradient
0.00273395/1e-006
TRAINSCG, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
  Columns 1 through 4
    0.0077    0.0129    0.0381    0.9526
  Columns 5 through 6
    0.9820    0.9830
```

Квазиньютоновы алгоритмы обучения

C365. TRAINBFG

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию trainbfg.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'trainbfg');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.0001;
[net,TR] = train(net,P,T); grid on
```

```
TRAINBFG-srchbac,  Epoch  0/50,  MSE  0.475987/0.0001,  Gradient
0.339126/1e-006
TRAINBFG-srchbac,  Epoch  10/50,  MSE  0.00426485/0.0001,  Gradient
0.0770971/1e-006
TRAINBFG-srchbac,  Epoch  13/50,  MSE  5.71314e-006/0.0001,  Gradient
0.000107212/1e-006
TRAINBFG, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
  Columns 1 through 4
    0.0000    0.0000    0.0006    0.9944
  Columns 5 through 6
    0.9989    0.9990
```

C368. TRAINOSS

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию trainoss.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'trainoss');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.0001;
[net,TR] = train(net,P,T); grid on
```

```
TRAINOSS-srchbac,  Epoch  0/50,  MSE  0.468812/0.0001,  Gradient
0.199095/1e-006
TRAINOSS-srchbac,  Epoch  10/50,  MSE  0.0080093/0.0001,  Gradient
0.116508/1e-006
TRAINOSS-srchbac,  Epoch  20/50,  MSE  0.00339965/0.0001,  Gradient
0.0136916/1e-006
```

```
TRAINOSS-srchbac, Epoch 26/50, MSE 4.15302e-005/0.0001, Gradient
0.000220306/1e-006
TRAINOSS, Performance goal met.
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
  Columns 1 through 4
    0.0002    0.0002    0.0031    0.9875
  Columns 5 through 6
    0.9935    0.9936
```

C370. TRAIMLM

Заданы следующие обучающие последовательности входов P и целей T:

```
P = [0 1 2 3 4 5]; T = [0 0 0 1 1 1];
```

Поскольку соответствие между входом и целью носит явно выраженный нелинейный характер, то будем использовать нейронную сеть с нелинейными сигмоидальными функциями активации. Выберем двухслойную нейронную сеть с прямой передачей сигнала; в первом слое используем 2 нейрона с функцией активации tansig, а во втором - 1 нейрон с функцией активации logsig. Для обучения сети применим функцию trainlm.

Формирование сети:

```
net = newff([0 5],[2 1],{'tansig','logsig'},'trainlm');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.0001;
[net,TR] = train(net,P,T); grid on
```

```
TRAINLM, Epoch 0/50, MSE 0.463676/0.0001, Gradient 1.60204/1e-010
TRAINLM, Epoch 10/50, MSE 0.000125314/0.0001, Gradient
0.00115041/1e-010
TRAINLM, Epoch 11/50, MSE 1.68681e-005/0.0001, Gradient
0.000171152/1e-010
TRAINLM, Performance goal met.
```

Данный алгоритм имеет адаптивный параметр mu, изменение которого построено в виде графика:

```
figure(2), clf, plot(TR.mu, 'LineWidth',2), grid on
title('Параметр адаптации')
```

```
xlabel('Число циклов')
```

Выполним моделирование сети:

```
Y = sim(net,P)
```

```
Y =
  Columns 1 through 4
    0.0006    0.0006    0.0049    0.9924
  Columns 5 through 6
    0.9970    0.9970
```

С373. TRAINBR

Рассмотрим задачу аппроксимации синусоидальной функции, которая зашумлена нормально распределенным шумом:

```
P = [-1:.05:1]; T = sin(2*pi*P) + 0.1*randn(size(P));
```

Сформируем для решения этой задачи двухслойную нейронную сеть прямой передачи сигнала. Вход сети принимает значения в диапазоне от –1 до 1. Первый слой имеет 20 нейронов с функцией активации tansig, второй слой имеет один нейрон с функцией активации purelin . В качестве обучающей используем функцию trainbr.

Формирование сети

```
net = newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
```

Обучение сети:

```
net.trainParam.epochs = 50; net.trainParam.show = 10;
net = train(net,P,T);  grid on
```

```
TRAINBR, Epoch 0/50, SSE 35.495/0, SSW 21461.6, Grad
6.54e+001/1.00e-010, #Par 6.10e+001/61
TRAINBR, Epoch 10/50, SSE 0.165045/0, SSW 3033.99, Grad 2.17e-
001/1.00e-010, #Par 2.94e+001/61
TRAINBR, Epoch 20/50, SSE 0.186866/0, SSW 1579.57, Grad 3.73e-
001/1.00e-010, #Par 2.56e+001/61
TRAINBR, Epoch 30/50, SSE 0.202243/0, SSW 988.328, Grad 1.13e-
001/1.00e-010, #Par 2.24e+001/61
TRAINBR, Epoch 40/50, SSE 0.215021/0, SSW 677.114, Grad 4.84e-
002/1.00e-010, #Par 1.99e+001/61
TRAINBR, Epoch 50/50, SSE 0.226781/0, SSW 389.632, Grad 1.32e-
001/1.00e-010, #Par 1.64e+001/61
TRAINBR, Maximum epoch reached.
```

Выполним моделирование сети и построим графики исследуемых функций

```
Y = sim(net,P);
figure(2), clf, h1=plot(P,Y,'LineWidth',2);
hold on, set(h1,'Color',[1/2,1/2,0])
plot(P,T,'+r','MarkerSize',6,'LineWidth',2), grid on
legend('выход', 'вход')
```

Функции оценки качества обучения

C377. SSE

Получим информацию о данной функции и ее производной:

```
sse('name'), sse('deriv'),  sse('pnames')
```

```
ans =
Sum squared error
ans =
dsse
ans =
     {}
```

Сформируем двухслойную нейронную сеть прямой передачи с одноэлементным входом, изменяющимся в диапазоне [–10 10], которая имеет 4 скрытых нейрона с функцией активации tansig и 1 нейрон на выходе с функцией активации purelin :

```
net = newff([-10 10],[4 1],{'tansig','purelin'});
```

Зададим векторы входа и целей

```
P = [-10 -5 0 5 10]; T = [ 0 0 1 1 1];
```

Промоделируем исходную нейронную сеть и вычислим ее ошибку

```
Y = sim(net, P); E = T-Y
E =
  Columns 1 through 4
   -1.5958   -1.1879    1.3390    1.7340
  Column 5
    2.6642
```

Вычислим функционал качества sse

```
net.performFcn = 'sse'; perf = sse(E)
```

```
perf =
   15.8554
```

Теперь вычислим градиенты функционала качества.

Градиент функционала по вектору ошибки вычисляется следующим образом

```
dPerf_dE = dsse('e',E)
```

```
dPerf_dE =
  Columns 1 through 4
   -3.1916   -2.3757    2.6779    3.4681
  Column 5
    5.3284
```

Для вычислений градиента функционала по вектору настраиваемых параметров сформируем этот вектор, который объединяет веса и смещения сети

```
X = [net.IW{1}; net.b{1}]'
```

```
X =
  Columns 1 through 4
    0.5600   -0.5600    0.5600    0.5600
  Columns 5 through 8
   -5.6000    1.8667    1.8667    5.6000
```

Градиент функционала по вектору параметров

```
dPerf_dX = dsse('x',E,X)
dPerf_dX =
  Columns 1 through 7
     0     0     0     0     0     0     0
  Column 8
     0
```

Этот градиент равен нулевому вектору, поскольку функционал качества не зависит явным образом от параметров сети.

C378. MSE

Получим информацию о данной функции и ее производной:

```
mse('name'), mse('deriv'), mse('pnames')
```

```
ans =
Mean squared error
ans =
dmse
ans =
     {}
```

Сформируем двухслойную нейронную сеть прямой передачи с одноэлементным входом, изменяющимся в диапазоне [–10 10], которая имеет 4 скрытых нейрона с функцией активации tansig и 1 нейрон на выходе с функцией активации purelin :

```
net = newff([-10 10],[4 1],{'tansig','purelin'});
```

Зададим векторы входа и целей

```
P = [-10 -5 0 5 10]; T = [ 0 0 1 1 1];
```

Промоделируем исходную нейронную сеть и вычислим ее ошибку

```
Y = sim(net, P); E = T-Y
```

```
E =
  Columns 1 through 4
    0.8520    0.8881    2.1218    3.0940
  Column 5
    2.2752
```

Вычислим функционал качества mse

```
net.performFcn = 'mse'; perf = mse(E)
perf =
    4.1533
```

Теперь вычислим градиенты функционала качества.

Для вычисления градиентов сформируем вектор настраиваемых параметров (веса и смещения):

```
X = [net.IW{1}; net.b{1}]'
```

Градиент функционала по вектору ошибки вычисляется следующим образом

```
dPerf_dE = dmse('e',E,X,perf)
dPerf_dE =
  Columns 1 through 4
    0.3408    0.3553    0.8487    1.2376
  Column 5
    0.9101
```

Градиент функционала по вектору настраиваемых параметров:

```
dPerf_dX = dmse('x',E,X)
dPerf_dX =
  Columns 1 through 7
     0     0     0     0     0     0     0
  Column 8
     0
```

Этот градиент равен нулевому вектору, поскольку функционал качества не зависит явным образом от параметров сети.

C379. MSEREG

Получим информацию о данной функции и ее производной

```
msereg('name'),        msereg('deriv'),        msereg('pnames'),
msereg('pdefaults')
ans =
Mean squared error with regularization
ans =
dmsereg
ans =
    'ratio'
ans =
    ratio: 0.9000
```

Это единственный функционал качества, который состоит из двух слагаемых: среднеквадратической ошибки с весом ratio и штрафной функции, оцениваемой суммой квадратов весов и смещений с весом 1-ratio.

Сформируем двухслойную нейронную сеть прямой передачи с одноэлементным входом, изменяющимся в диапазоне [–10 10], которая имеет 4 скрытых нейрона с функцией активации tansig и 1 нейрон на выходе с функцией активации purelin:

```
net = newff([-10 10],[4 1],{'tansig','purelin'});
```

Зададим векторы входа и целей

```
P = [-10 -5 0 5 10]; T = [ 0 0 1 1 1];
```

Промоделируем исходную нейронную сеть и вычислим ее ошибку

```
Y = sim(net, P); E = T-Y
E =
  Columns 1 through 4
   -1.9956   -1.7524    0.0534    0.9007
  Column 5
    1.2061
```

Вычислим функционал качества msereg

```
net.performFcn = 'msereg'; net.performParam.ratio = 0.9;
perf = msereg(E,net)

perf =
    2.2333
```

Теперь вычислим градиенты функционала качества.

Для вычисления градиентов сформируем вектор настраиваемых параметров (веса и смещения):

```
X = [net.IW{1}; net.b{1}]'
```

```
X =
  Columns 1 through 4
    0.5600   -0.5600    0.5600    0.5600
  Columns 5 through 8
   -5.6000    1.8667    1.8667    5.6000
```

Градиент функционала по вектору ошибки вычисляется следующим образом

```
dPerf_dX = dmsereg('e',E,X,perf,net.performParam)
dPerf_dX =
  Columns 1 through 4
   -0.7184   -0.6309    0.0192    0.3243
  Column 5
    0.4342
```

Градиент функционала по вектору параметров

```
dPerf_dX = dmsereg('x',E,X,perf,net.performParam)
```

```
dPerf_dX =
  Columns 1 through 4
   -0.0140    0.0140   -0.0140   -0.0140
  Columns 5 through 8
    0.1400   -0.0467   -0.0467   -0.1400
```

Этот градиент не равен нулевому вектору, поскольку функционал качества зависит явным образом от параметров сети.

C381. MAE

Получим информацию о данной функции и ее производной

```
mae('name'), mae('deriv'), mae('pnames')
```

```
ans =
Mean absolute error
ans =
dmae
ans =
     {}
```

Сформируем двухслойную нейронную сеть прямой передачи с одноэлементным входом, изменяющимся в диапазоне [–10 10], которая имеет 4 скрытых нейрона с функцией активации tansig и 1 нейрон на выходе с функцией активации purelin:

```
net = newff([-10 10],[4 1],{'tansig','purelin'});
```

Зададим векторы входа и целей

```
P = [-10 -5 0 5 10]; T = [ 0 0 1 1 1];
```

Промоделируем исходную нейронную сеть и вычислим ее ошибку

```
Y = sim(net, P); E = T-Y
```

```
E =
  Columns 1 through 4
    0.3265    0.0113    0.6208   -0.3710
  Column 5
    0.1998
```

Вычислим функционал качества mae

```
net.performFcn = 'mae'; perf = mae(E)
```

```
perf =
    0.3059
```

Теперь вычислим градиенты функционала качества.

Для вычисления градиентов сформируем вектор настраиваемых параметров (веса и смещения):

```
X = [net.IW{1}; net.b{1}]'
X =
  Columns 1 through 4
    0.5600   -0.5600    0.5600    0.5600
  Columns 5 through 8
   -5.6000    1.8667    1.8667    5.6000
```

Градиент функционала по вектору ошибки вычисляется следующим образом

```
dPerf_dE = dmae('e',{E},X); [dPerf_dE{:}]
ans =
     1     1     1    -1     1
```

Градиент функционала по вектору параметров

```
dPerf_dX = dmae('x',{E},{X})
dPerf_dX =
     0
```

Функции настройки параметров

C383. LEARNP

Определим сеть со случайными векторами входа p и ошибки e:

```
p = rand(2,1); e = rand(3,1);
```

Вызов функции learnp можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnp([],p,[],[],[],[],e,[],[],[],[],[])
dW =
    0.0252    0.0198
    0.0156    0.0123
    0.1798    0.1412
```

C384. LEARNPN

Определим сеть со случайными векторами входа p и ошибки e:

```
p = rand(2,1); e = rand(3,1);
```

Вызов функции learnpn можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnpn([],p,[],[],[],[],e,[],[],[],[],[])

dW =
    0.4122    0.1003
    0.3301    0.0803
    0.6055    0.1473
```

C385. LEARNWH

Определим сеть со случайными векторами входа p и ошибки e с 2-элементным входом и тремя нейронами и параметром скорости настройки lr:

```
p = rand(2,1); e = rand(3,1); lp.lr = 0.5;
```

Вызов функции learnwh можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnwh([],p,[],[],[],[],e,[],[],[],lp,[])

dW =
    0.0071    0.0042
    0.3130    0.1875
    0.1695    0.1015
```

C387. LEARNGD

Допустим, что на некотором шаге настройки слоя с 3 нейронами и двухэлементным входом известен случайный вектор градиента gW, а параметр скорости настройки задан равным 0.5:

```
gW = rand(3,2); lp.lr = 0.5;
```

Тогда вызов функции learngd можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learngd([],[],[],[],[],[],[],gW,[],[],lp,[])
```

```
dW =
    0.3077    0.3691
    0.3960    0.0881
    0.4609    0.2029
```

C388. LEARNGDM

Допустим, что на некотором шаге настройки слоя с 3 нейронами и двухэлементным входом известен случайный вектор градиента gW, а параметры скорости настройки и возмущения заданы равными 0.6 и 0.8, соответственно:

```
gW = rand(3,2); lp.lr = 0.6; lp.mc = 0.8;
```

Тогда вызов функции learngdm можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
ls = []; [dW,ls] = learngdm([],[],[],[],[],[],[],gW,[],[],lp,ls)
dW =
    0.5613    0.5362
    0.5501    0.0347
    0.2462    0.2117
ls =
    dw: [3x2 double]
ls.dw
ans =
    0.5613    0.5362
    0.5501    0.0347
    0.2462    0.2117
```

C389. LEARNLV1

Определим слой нейронной сети с двухэлементным входом и тремя нейронами сеть со случайными массивами входа **p**, весов **w** и выхода **a**; зададим также градиент функционала по выходу gA и параметр скорости настройки lr:

```
p = rand(2,1); w = rand(3,2);
```

```
a = compet(negdist(w,p));
gA = [-1;1;1]; lp.lr = 0.5;
```

Вызов функции learnlv1 можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnlv1(w,p,[],[],a,[],[],[],gA,[],lp,[])
dW =
         0         0
         0         0
   -0.3072    0.0945
```

C391. LEARNLV2

Определим слой нейронной сети с двухэлементным входом и тремя нейронами сеть со случайными массивами входа **p**, весов **w** и выхода **a**; зададим также градиент функционала по выходу gA и параметр скорости настройки lr:

```
p = rand(2,1); w = rand(3,2);
n = negdist(w,p); a = compet(n);
gA = [-1;1;1]; lp.lr = 0.5; lp.window = 0.25;
```

Вызов функции learnlv2 можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnlv2(w,p,[],n,a,[],[],[],gA,[],lp,[])

dW =
   -0.2149    0.1641
         0         0
    0.2254   -0.1108
```

C392. LEARNK

Определим слой Кохонена сети с двухэлементным входом и тремя нейронами со случайными массивами входа **p**, весов **w** и выхода **a**; зададим также параметр скорости настройки lr:

```
p = rand(2,1); a = rand(3,1); w = rand(3,2);
lp.lr = 0.5;
```

Вызов функции learnk можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnk(w,p,[],[],a,[],[],[],[],[],lp,[])

dW =
   -0.0884   -0.0187
   -0.3146    0.1216
```

```
    -0.1501    0.1838
```

C394. LEARNCON

Определим слой Кохонена сети с двухэлементным входом и тремя нейронами со случайными массивами выхода **a**, вектора смещений **b**; зададим также параметр скорости настройки lr:

```
a = rand(3,1); b = rand(3,1);
lp.lr = 0.5;
```

Вызов функции learnk можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
db = learncon(b,[],[],[],a,[],[],[],[],[],lp,[])
```

```
db =
    0.2902
    0.5015
    0.1399
```

C395. LEARNIS

Определим слой Кохонена с двухэлементным входом и тремя нейронами со случайными массивами входа **p**, весов **w** и выхода **a**; зададим также параметр скорости настройки lr:

```
p = rand(2,1); a = rand(3,1); w = rand(3,2);
lp.lr = 0.5;
```

Вызов функции learnis можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnis(w,p,[],[],a,[],[],[],[],[],lp,[])
```

```
dW =
    0.0866   -0.1146
   -0.0862   -0.1876
   -0.0367   -0.0836
```

C396. LEARNOS

Определим слой Кохонена с двухэлементным входом и тремя нейронами со случайными массивами входа **p**, весов **w** и выхода **a**; зададим также параметр скорости настройки lr:

```
p = rand(2,1); a = rand(3,1); w = rand(3,2);
lp.lr = 0.5;
```

Вызов функции learnos можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnos(w,p,[],[],a,[],[],[],[],[],lp,[])
dW =
    0.0055   -0.0042
   -0.0521   -0.0244
    0.0272    0.0262
```

C397. LEARNSOM

Определим карту Кохонена с расположением нейронов на гексагональной сетке размера 2×3, а также расстояния между ними; зададим случайные массивы входа **p**, выхода **a** и весов **w**, а также параметры процедуры настройки.

```
p = rand(2,1); a = rand(6,1); w = rand(6,2);
pos = hextop(2,3); d = linkdist(pos);
lp.order_lr = 0.9; lp.order_steps = 1000;
lp.tune_lr = 0.02; lp.tune_nd = 1;
```

Вызов функции learnsom можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
ls = []; [dW,ls] = learnsom(w,p,[],[],a,[],[],[],[],d,lp,ls)


dW =
   -1.0999    1.4042
    0.3702   -0.4255
    0.4251    1.0936
   -0.8669    0.7965
   -0.6354    0.0720
    0.5719    0.7607
ls =
      step: 1
    nd_max: 2
```

C399. LEARNH

Определим нейронную сеть с двухэлементным входом и тремя нейронами со случайными массивами входа **p** и выхода **a**; зададим также параметр скорости настройки lr:

```
p = rand(2,1); a = rand(3,1);
lp.lr = 0.5;
```

Вызов функции learnh можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnh([],p,[],[],a,[],[],[],[],[],lp,[])
dW =
    0.2319    0.0320
    0.1367    0.0189
    0.0994    0.0137
```

C400. MAXLINLR

Зададим числовой массив входа, состоящий из четырех двухэлементных векторов и вычислим максимальные значения параметра скорости настройки для линейной сети без и со смещением:

```
P = [ 1 2 -4 7; 0.1 3 10 6];
lr = maxlinlr(P) %Без смещения
lr = 0.0069
lr = maxlinlr(P,'bias') %Со смещением
lr =
    0.0069
lr =
    0.0069
lr =
    0.0067
```

C401. LEARNHD

Определим сеть со случайными входом P, выходом A и весами W для слоя с 2-элементным входом и тремя нейронами; зададим также параметры скорости настройки lr и затухание dr.

```
p = rand(2,1); a = rand(3,1); w = rand(3,2);
lp.lr = 0.5; lp.dr = 0.05;
```

Вызов функции learnhd можно организовать следующим образом, поскольку не все входные аргументы требуются для вызова этой функции:

```
dW = learnhd(w,p,[],[],a,[],[],[],[],[],lp,[])

dW =
    0.0736    0.0618
    0.0317    0.0187
    0.1174    0.0576
```

Функции одномерного поиска

C404. SRCHGOL

Заданы векторы входа **p** и целей **t**, требуется сформировать нейронную сеть, выходы которой близки к заданным целям:

```
p = [0 1 2 3 4 5]; t = [0 0 0 1 1 1];
```

Выберем архитектуру двухслойной сети с прямой передачей сигнала; зададим диапазон входов от 0 до 10, первый слой имеет 2 нейрона с функцией активации tansig, второй – 1 нейрон с функцией активации logsig. Используем функции обучения traincgf и поиска одномерного экстремума srchgol.

```
net = newff([0 10],[2 1],{'tansig','logsig'},'traincgf');
net.trainParam.searchFcn = 'srchgol';
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.0001;
net = train(net,p,t); grid on
a = sim(net,p)
TRAINCGF-srchgol, Epoch 0/50, MSE 0.486862/0.0001, Gradient
0.0813759/1e-006
TRAINCGF-srchgol, Epoch 9/50, MSE 1.90795e-010/0.0001, Gradient
1.77906e-008/1e-006
TRAINCGF, Performance goal met.

a =
  Columns 1 through 4
    0.0000    0.0000    0.0000    1.0000
  Columns 5 through 6
    1.0000    1.0000
```

C405. SRCHBRE

Заданы векторы входа **p** и целей **t**, требуется сформировать нейронную сеть, выходы которой близки к заданным целям:

```
p = [0 1 2 3 4 5]; t = [0 0 0 1 1 1];
```

Выберем архитектуру двухслойной сети с прямой передачей сигнала; зададим диапазон входов от 0 до 10, первый слой имеет 2 нейрона с функцией активации tansig, второй – 1 нейрон с функцией активации logsig. Используем функции обучения traincgf и поиска одномерного экстремума srchbre.

```
net = newff([0 10],[2 1],{'tansig','logsig'},'traincgf');
net.trainParam.searchFcn = 'srchbre';
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.0001;
net = train(net,p,t); grid on
a = sim(net,p)
```

```
TRAINCGF-srchbre,  Epoch  0/50,  MSE  0.464364/0.0001,  Gradient
0.0566042/1e-006
TRAINCGF-srchbre,  Epoch  10/50,  MSE  0.000607763/0.0001,  Gradient
0.00140522/1e-006
TRAINCGF-srchbre,  Epoch  15/50,  MSE  7.54063e-005/0.0001,  Gradient
0.000251973/1e-006
TRAINCGF, Performance goal met.

a =
  Columns 1 through 4
    0.0023    0.0027    0.0156    0.9861
  Columns 5 through 6
    0.9986    0.9989
```

C406. SRCHHYB

Заданы векторы входа **p** и целей **t**, требуется сформировать нейронную сеть, выходы которой близки к заданным целям:

```
p = [0 1 2 3 4 5]; t = [0 0 0 1 1 1];
```

Выберем архитектуру двухслойной сети с прямой передачей сигнала; зададим диапазон входов от 0 до 10, первый слой имеет 2 нейрона с функцией активации tansig, второй – 1 нейрон с функцией активации logsig. Используем функции обучения traincgf и поиска одномерного экстремума srchhyb.

```
net = newff([0 10],[2 1],{'tansig','logsig'},'traincgf');
net.trainParam.searchFcn = 'srchhyb';
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.0001;
net = train(net,p,t); grid on
a = sim(net,p)
```

```
TRAINCGF-srchhyb,  Epoch  0/50,  MSE  0.297493/0.0001,  Gradient
0.211072/1e-006
TRAINCGF-srchhyb,  Epoch  7/50,  MSE  6.47144e-016/0.0001,  Gradient
3.86955e-014/1e-006
TRAINCGF, Performance goal met.

a =
  Columns 1 through 4
    0.0000    0.0000    0.0000    1.0000
  Columns 5 through 6
    1.0000    1.0000
```

C408. SRCHCHA

Заданы векторы входа **p** и целей **t**, требуется сформировать нейронную сеть, выходы которой близки к заданным целям:

```
p = [0 1 2 3 4 5]; t = [0 0 0 1 1 1];
```

Выберем архитектуру двухслойной сети с прямой передачей сигнала; зададим диапазон входов от 0 до 10, первый слой имеет 2 нейрона с функцией активации tansig, второй – 1 нейрон с функцией активации logsig. Используем функции обучения traincgf и поиска одномерного экстремума srchcha.

```
net = newff([0 10],[2 1],{'tansig','logsig'},'traincgf');
net.trainParam.searchFcn = 'srchcha';
net.trainParam.epochs = 50; net.trainParam.show = 10;
net.trainParam.goal = 0.0001;
net = train(net,p,t); grid on
a = sim(net,p)
```

```
TRAINCGF-srchcha, Epoch 0/50, MSE 0.466016/0.0001, Gradient
0.054143/1e-006
TRAINCGF-srchcha, Epoch 9/50, MSE 8.2629e-005/0.0001, Gradient
0.000961218/1e-006
TRAINCGF, Performance goal met.

a =
  Columns 1 through 4
    0.0084    0.0090    0.0185    0.9996
  Columns 5 through 6
    1.0000    1.0000
```

C409. SRCHBAC

Заданы векторы входа **p** и целей **t**, требуется сформировать нейронную сеть, выходы которой близки к заданным целям:

```
p = [0 1 2 3 4 5]; t = [0 0 0 1 1 1];
```

Выберем архитектуру двухслойной сети с прямой передачей сигнала; зададим диапазон входов от 0 до 10, первый слой имеет 2 нейрона с функцией активации tansig, второй – 1 нейрон с функцией активации logsig. Используем функции обучения traincgf и поиска одномерного экстремума srchbac.

```
net = newff([0 10],[2 1],{'tansig','logsig'},'traincgf');
net.trainParam.searchFcn = 'srchbac';
net.trainParam.epochs = 50; net.trainParam.show = 10;
```

```
net.trainParam.goal = 0.0001;
net = train(net,p,t); grid on
a = sim(net,p)
```

```
TRAINCGF-srchbac, Epoch 0/50, MSE 0.427245/0.0001, Gradient
0.147317/1e-006
TRAINCGF-srchbac, Epoch 9/50, MSE 7.04485e-005/0.0001, Gradient
0.00175321/1e-006
TRAINCGF, Performance goal met.

a =
  Columns 1 through 4
    0.0049    0.0049    0.0052    0.9813
  Columns 5 through 6
    0.9993    0.9993
```

Масштабирование и восстановление данных

C410. PREMNMX

Следующие операторы выполняют нормировку данных так, чтобы значения входа и цели попадали в интервал [-1,1]:

```
p = [-10 -7.5 -5 -2.5 0 2.5 5 7.5 10];
t = [0 7.07 -10 -7.07 0 7.07 10 7.07 0];
[pn,minp,maxp,tn,mint,maxt] = premnmx(p,t)
```

```
pn =
  Columns 1 through 4
   -1.0000   -0.7500   -0.5000   -0.2500
  Columns 5 through 8
         0    0.2500    0.5000    0.7500
  Column 9
    1.0000
minp =
   -10
maxp =
    10
tn =
  Columns 1 through 4
         0    0.7070   -1.0000   -0.7070
  Columns 5 through 8
         0    0.7070    1.0000    0.7070
  Column 9
```

```
         0
mint =
   -10
maxt =
    10
```

C411. PRESTD

Задана следующая обучающая последовательность векторов входа и целей. Требуется выполнить ее приведение к нормальному закону распределения с параметрами [0 1]

```
p = [-0.92 0.73 -0.47  0.74 0.29; -0.08 0.86 -0.67 -0.52 0.93];
t = [-0.08 3.4 -0.82 0.69 3.1];
[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t)


pn =
  Columns 1 through 4
   -1.3389    0.8836   -0.7328    0.8971
   -0.2439    1.0022   -1.0261   -0.8272
  Column 5
    0.2910
    1.0950
meanp =
    0.0740
    0.1040
stdp =
    0.7424
    0.7543
tn =
  Columns 1 through 4
   -0.7049    1.1285   -1.0947   -0.2992
  Column 5
    0.9704
meant =
    1.2580
stdt =
    1.8982
```

C412. PREPCA

Зададим массив двухэлементных векторов входа и выполним их факторный анализ, удерживая только те компоненты вектора, дисперсия которых превышает 2% общей дисперсии. Сначала с помощью функции prestd приведем

входные данные к нормальному закону распределения, а затем применим функцию prepca.

```
P = [-1.5 -0.58 0.21 -0.96 -0.79;  -2.2 -0.87 0.31 -1.4 -1.2];
[pn,meanp,stdp] = prestd(P)

pn =
  Columns 1 through 4
   -1.2445    0.2309    1.4978   -0.3785
   -1.2331    0.2208    1.5108   -0.3586
  Column 5
   -0.1058
   -0.1399
meanp =
   -0.7240
   -1.0720
stdp =
    0.6236
    0.9148
[ptrans,transMat] = prepca(pn,0.02)

ptrans =
  Columns 1 through 4
    1.7519   -0.3194   -2.1274    0.5212
  Column 5
    0.1738
transMat =
   -0.7071   -0.7071
```

C413. POSTMNNMX

В этом примере сначала с помощью функции premnmx выполняется масштабирование обучающей последовательности к диапазону [-1 1], затем создается и обучается нейронная сеть прямой передачи, выполняется ее моделирование и восстановление выхода с помощью функции postmnmx.

```
P = [-0.92 0.73 -0.47  0.74 0.29; -0.08 0.86 -0.67 -0.52 0.93];
t = [-0.08 3.40 -0.82  0.69 3.10];
[pn,minp,maxp,tn,mint,maxt] = premnmx(P,t);
net = newff(minmax(pn),[5 1],{'tansig' 'purelin'},'trainlm');
net = train(net,pn,tn); grid on
an = sim(net,pn)
```

```
TRAINLM, Epoch 0/100, MSE 4.01275/0, Gradient 12.3803/1e-010
TRAINLM, Epoch 5/100, MSE 4.2096e-026/0, Gradient 6.29848e-013/1e-
010
TRAINLM, Minimum gradient reached, performance goal was not met.

an =
  Columns 1 through 4
   -0.6493    1.0000   -1.0000   -0.2844
  Column 5
    0.8578
a = postmnmx(an,mint,maxt)

a =
  Columns 1 through 4
   -0.0800    3.4000   -0.8200    0.6900
  Column 5
    3.1000
```

C414. POSTSTD

В этом примере сначала с помощью функции prestd выполняется масштабирование обучающей последовательности к нормальному закону распределения с параметрами [0 1], затем создается и обучается нейронная сеть прямой передачи, выполняется ее моделирование и восстановление выхода с помощью функции poststd.

```
p = [-0.92 0.73 -0.47  0.74 0.29; -0.08 0.86 -0.67 -0.52 0.93];
t = [-0.08 3.40 -0.82 0.69 3.10];
[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);
net = newff(minmax(pn),[5 1],{'tansig' 'purelin'},'trainlm');
net = train(net,pn,tn); grid on
an = sim(net,pn)

TRAINLM, Epoch 0/100, MSE 2.34864/0, Gradient 9.59914/1e-010
TRAINLM, Epoch 4/100, MSE 1.24883e-022/0, Gradient 4.57729e-
011/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.

an =
  Columns 1 through 4
   -0.7049    1.1285   -1.0947   -0.2992
  Column 5
    0.9704
```

```
a = poststd(an,meant,stdt)


a =
  Columns 1 through 4
   -0.0800    3.4000   -0.8200    0.6900
  Column 5
    3.1000
```

C415. POSTREG

В данном примере с помощью функции prestd нормализуется множество обучающих данных, на нормализованных данных вычисляются главные составляющие преобразования, создается и обучается сеть, используя pca данные, сеть моделируется. Затем выход сети с помощью функции poststd денормализуется и вычисляется линейная регрессия между выходом (ненормализованным) сети и целями, чтобы проверить качество обучения сети.

```
P = [-0.92 0.73 -0.47  0.74 0.29; -0.08 0.86 -0.67 -0.52 0.93];
T = [-0.08 3.40 -0.82 0.69 3.10];
[pn,meanp,stdp,tn,meant,stdt] = prestd(P,T);
[ptrans,transMat] = prepca(pn,0.02);
net = newff(minmax(ptrans),[5 1],{'tansig' 'purelin'},'trainlm');
net = train(net,ptrans,tn); grid on
an = sim(net,ptrans)

TRAINLM, Epoch 0/100, MSE 1.06854/0, Gradient 6.56843/1e-010
TRAINLM, Epoch 4/100, MSE 7.84424e-030/0, Gradient 3.04851e-
015/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.

an =
  Columns 1 through 4
   -0.7049    1.1285   -1.0947   -0.2992
  Column 5
    0.9704
a = poststd(an,meant,stdt)


a =
  Columns 1 through 4
   -0.0800    3.4000   -0.8200    0.6900
  Column 5
    3.1000
```

```
figure(1), clf
[m,b,r] = postreg(a,t), grid on % Рис.11.61

m =
    1.0000
b =
  5.0387e-016
r =
     1
```

C417. TRAMNMX

Следующие операторы масштабируют обучающую последовательность к диапазону [-1 1], формируют и обучают нейронную сеть прямой передачи

```
p = [-10 -7.5 -5 -2.5 0 2.5 5 7.5 10];
t = [0 7.07 -10 -7.07 0 7.07 10 7.07 0];
[pn,minp,maxp,tn,mint,maxt] = premnmx(p,t);
net = newff(minmax(pn),[5 1],{'tansig' 'purelin'},'trainlm');
net = train(net,pn,tn); grid on

TRAINLM, Epoch 0/100, MSE 0.685331/0, Gradient 10.0713/1e-010
TRAINLM, Epoch 10/100, MSE 6.32948e-022/0, Gradient 4.72377e-
011/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.
```

Если в дальнейшем к обученной сети будут приложены новые входы, то они должны быть масштабированы с помощью функции tramnmx. Выход сети должен быть восстановлен с помощью функции postmnmx.

```
p2 = [4 -7]; pn = tramnmx(p2,minp,maxp);
an = sim(net,pn)

an =
    0.9440    0.0067
a = postmnmx(an,mint,maxt)

a =
    9.4399    0.0671
```

C417. TRASTD

Следующие операторы масштабируют обучающую последовательность к нормальному закону распределения с параметрами [0 1], формируют и обучают нейронную сеть прямой передачи

```
p = [-0.92 0.73 -0.47  0.74 0.29;  -0.08 0.86 -0.67 -0.52 0.93];
t = [-0.08 3.4 -0.82 0.69 3.1];
[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);
net = newff(minmax(pn),[5 1],{'tansig' 'purelin'},'trainlm');
net = train(net,pn,tn); grid on

TRAINLM, Epoch 0/100, MSE 4.09289/0, Gradient 14.841/1e-010
TRAINLM, Epoch 5/100, MSE 5.42342e-032/0, Gradient 1.24667e-
015/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.
```

Если в дальнейшем к обученной сети будут приложены новые входы, то они должны быть масштабированы с помощью функции trastd. Выход сети должен быть восстановлен с помощью poststd.

```
p2 = [1.5 -0.8; 0.05 -0.3];
pn = trastd(p2,meanp,stdp);
an = sim(net,pn)

an =
    1.1259   -0.9903
a = poststd(an,meant,stdt)

a =
    3.3951   -0.6218
```

C418. TRAPCA

Следующие операторы выполняют главный факторный анализ обучающей последовательности, удерживая только те компоненты, которые имеют дисперсию, превышающую значение 0.02.

```
P = [-1.5 -0.58 0.21 -0.96 -0.79;  -2.2 -0.87 0.31 -1.40 -1.20];
t = [-0.08 3.4 -0.82 0.69 3.1];
[pn,meanp,stdp,tn,meant,stdt] = prestd(P,t);
[ptrans,transMat] = prepca(pn,0.02)
```

```
ptrans =
  Columns 1 through 4
    1.7519   -0.3194   -2.1274    0.5212
  Column 5
    0.1738
transMat =
   -0.7071   -0.7071
net = newff(minmax(ptrans),[5 1],{'tansig' 'purelin'},'trainlm');
net = train(net,ptrans,tn); grid on

TRAINLM, Epoch 0/100, MSE 3.14416/0, Gradient 8.99022/1e-010
TRAINLM, Epoch 7/100, MSE 1.80389e-027/0, Gradient 2.06364e-
014/1e-010
TRAINLM, Minimum gradient reached, performance goal was not met.
```

Если в дальнейшем к сети будут приложены новые входы, то они должны быть масштабированы с помощью функций trastd и trapca. Выход сети должен быть восстановлен с помощью poststd:

```
p2 = [1.50 -0.8;       0.05 -0.3];
p2n = trastd(p2,meanp,stdp);
p2trans = trapca(p2n,transMat)

p2trans =
   -3.3893   -0.5106
an = sim(net,p2trans)

an =
   -0.9733    0.9177
a = poststd(an,meant,stdt)

a =
   -0.5895    3.0000
```

Вспомогательные функции

C420. CALCA

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, тремя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2] (рис. 11.62).

```
net = newlin([0 1],3,[0 2 4]); net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2];
```

```
gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 8 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3 0.6 0.4 0.7 0.2 0.1};
Pi = {0.2 0.3 0.4 0.1}; Pc = [Pi P];
Pd = calcpd(net,8,1,Pc)

Pd(:,:,1) =
    [3x1 double]
Pd(:,:,2) =
    [3x1 double]
Pd(:,:,3) =
    [3x1 double]
Pd(:,:,4) =
    [3x1 double]
Pd(:,:,5) =
    [3x1 double]
Pd(:,:,6) =
    [3x1 double]
Pd(:,:,7) =
    [3x1 double]
Pd(:,:,8) =
    [3x1 double]
```

Сформируем вектор начальных условий на линии задержки выхода слоя для каждого из трех нейронов:

```
Ai = {[0.5; 0.1; 0.2] [0.6; 0.5; 0.2]};
```

Применяя функцию calca, рассчитаем сигналы в слое на каждом временном шаге TS:

```
[Ac,N,LWZ,IWZ,BZ] = calca(net,Pd,Ai,1,8)
Ac =
  Columns 1 through 2
    [3x1 double]    [3x1 double]
  Columns 3 through 4
    [3x1 double]    [3x1 double]
  Columns 5 through 6
    [3x1 double]    [3x1 double]
  Columns 7 through 8
    [3x1 double]    [3x1 double]
  Columns 9 through 10
```

```
    [3x1 double]    [3x1 double]
N =
  Columns 1 through 2
    [3x1 double]    [3x1 double]
  Columns 3 through 4
    [3x1 double]    [3x1 double]
  Columns 5 through 6
    [3x1 double]    [3x1 double]
  Columns 7 through 8
    [3x1 double]    [3x1 double]
LWZ(:,:,1) =
    [3x1 double]
LWZ(:,:,2) =
    [3x1 double]
LWZ(:,:,3) =
    [3x1 double]
LWZ(:,:,4) =
    [3x1 double]
LWZ(:,:,5) =
    [3x1 double]
LWZ(:,:,6) =
    [3x1 double]
LWZ(:,:,7) =
    [3x1 double]
LWZ(:,:,8) =
    [3x1 double]
IWZ(:,:,1) =
    [3x1 double]
IWZ(:,:,2) =
    [3x1 double]
IWZ(:,:,3) =
    [3x1 double]
IWZ(:,:,4) =
    [3x1 double]
IWZ(:,:,5) =
    [3x1 double]
IWZ(:,:,6) =
    [3x1 double]
IWZ(:,:,7) =
    [3x1 double]
IWZ(:,:,8) =
    [3x1 double]
BZ =
```

```
    [3x1 double]
```

C422. CALCA1

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, тремя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2] (рис. 11.62).

```
net = newlin([0 1],3,[0 2 4]); net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2]; gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 3 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3}; Pi = {0.2 0.3 0.4 0.1};
Pc = [Pi P]; Pd = calcpd(net,3,1,Pc)
```

```
Pd(:,:,1) =
    [3x1 double]
Pd(:,:,2) =
    [3x1 double]
Pd(:,:,3) =
    [3x1 double]
```

Сформируем вектор начальных условий на линии задержки выхода слоя для каждого из трех нейронов

```
Ai = {[0.5; 0.1; 0.2] [0.6; 0.5; 0.2]};
```

Применяя функцию calca1, рассчитаем сигналы в слое на первом шаге по времени:

```
[A1,N1,LWZ1,IWZ1,BZ1] = calca1(net,Pd(:,:,1),Ai,1)
```

```
A1 =
    [3x1 double]
N1 =
    [3x1 double]
LWZ1 =
    [3x1 double]
IWZ1 =
    [3x1 double]
BZ1 =
    [3x1 double]
```

Теперь можно вычислить новые состояния на ЛЗ, используя массивы Ai и A, и рассчитать сигналы слоя на втором шаге по времени

```
Ai2 = [Ai(:,2:end) A1];
[A2,N2,LWZ2,IWZ2,BZ2] = calca1(net,Pd(:,:,2),Ai2,1)
```

```
A2 =
    [3x1 double]
N2 =
    [3x1 double]
LWZ2 =
    [3x1 double]
IWZ2 =
    [3x1 double]
BZ2 =
    [3x1 double]
```

C423. CALCPD

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, тремя нейронами и линией задержки на входе с параметрами [0 2 4]

```
net = newlin([0 1],3,[0 2 4]); gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 3 шагов по времени и вектор начальных условий на линии задержки Pi:

```
P = {0 0.1 0.3}; Pi = {0.2 0.3 0.4 0.1};
```

Запаздывающие входы (значения входов после прохождения через линию задержки) рассчитываются с помощью функции calcpd после их объединения в вектор Pc

```
Pc = [Pi P]; Pd = calcpd(net,3,1,Pc)
```

```
Pd(:,:,1) =
    [3x1 double]
Pd(:,:,2) =
    [3x1 double]
Pd(:,:,3) =
    [3x1 double]
```

Теперь можно просмотреть значения запаздывающих входов для двух первых шагов

```
Pd{1,1,1}, Pd{1,1,2}
```

```
ans =
         0
```

```
    0.4000
    0.2000
ans =
    0.1000
    0.1000
    0.3000
```

C424. CALCE

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, двумя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2]:

```
net = newlin([0 1],2,[0 2 4]);
net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2]; gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 5 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3 0.6 0.4}; Pi = {0.2 0.3 0.4 0.1};
Pc = [Pi P]; Pd = calcpd(net,5,1,Pc);
```

Сформируем вектор начальных условий на линии задержки выхода слоя для каждого из двух нейронов и рассчитаем сигналы в слое на 5 шагах по времени:

```
Ai = {[0.5; 0.1] [0.6; 0.5]};
[Ac,N,LWZ,IWZ,BZ] = calca(net,Pd,Ai,1,5);
```

Определим цели слоя для двух нейронов для каждого из 5 временных шагов и рассчитаем ошибки слоя:

```
Tl = {[0.1;0.2] [0.3;0.1], [0.5;0.6] [0.8;0.9], [0.5;0.1]};
El = calce(net,Ac,Tl,5)
```

```
El =
  Columns 1 through 2
    [2x1 double]    [2x1 double]
  Columns 3 through 4
    [2x1 double]    [2x1 double]
    [2x1 double]
```

Просмотрим ошибки слоя 1 на временном шаге 2:

```
El{1,2}
```

```
ans =
    0.3000
```

```
    0.1000
```

C425. CALCE1

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, двумя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2]:

```
net = newlin([0 1],2,[0 2 4]);
net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2]; gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 5 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3 0.6 0.4}; Pi = {0.2 0.3 0.4 0.1};
Pc = [Pi P]; Pd = calcpd(net,5,1,Pc);
```

Сформируем вектор начальных условий на линии задержки выхода слоя для каждого из двух нейронов и рассчитаем сигналы в слое на 5 шагах по времени:

```
Ai = {[0.5; 0.1] [0.6; 0.5]};
[A1,N1,LWZ1,IWZ1,BZ1] = calca1(net,Pd(:,:,1),Ai,1)

A1 =
    [2x1 double]
N1 =
    [2x1 double]
LWZ1 =
    [2x1 double]
IWZ1 =
    [2x1 double]
BZ1 =
    [2x1 double]
```

Определим цели слоя для двух нейронов для каждого из 5 временных шагов и рассчитаем ошибки слоя на первом шаге:

```
Tl = {[0.1;0.2] [0.3;0.1], [0.5;0.6] [0.8;0.9], [0.5;0.1]};
El = calce1(net,A1,Tl(:,1))
El =
    [2x1 double]
```

Просмотрим ошибку слоя на первом шаге:

```
El{1}

ans =
```

```
    0.1000
    0.2000
```

Теперь можно вычислить новые состояния на ЛЗ, используя массивы Ai и A, и рассчитать сигналы слоя на втором шаге по времени

```
Ai2 = [Ai(:,2:end) A1];
[A2,N2,LWZ2,IWZ2,BZ2] = calca1(net,Pd(:,:,2),Ai2,1);
El = calce1(net,A2,Tl(:,2)); El{1}

ans =
    0.3000
    0.1000
```

C426. FORMX

Создадим однослойную сеть с тремя нейронами и двухэлементным вектором входа со значениями из диапазонов [0 1] и [-1 1]:

```
net = newff([0 1; -1 1],[3]); gensim(net)
```

Выведем значения массивов весов и смещений:

```
b = net.b, b{1}

b =
    [3x1 double]
ans =
   -3.8200
   -1.1587
   -1.3436
iw = net.iw,   iw{1}

iw =
    [3x2 double]
ans =
    2.7902   -1.9834
    2.3173   -2.1301
   -2.1626   -2.1704
lw = net.lw
lw =
    {[]}
```

Объединим массивы весов и смещений в общий вектор

```
x = formx(net,net.b,net.iw,net.lw); x'
ans =
  Columns 1 through 4
```

```
    2.7902    2.3173   -2.1626   -1.9834
  Columns 5 through 8
   -2.1301   -2.1704   -3.8200   -1.1587
  Column 9
   -1.3436
```

C428. GETX

Создадим однослойную сеть с тремя нейронами и двухэлементным вектором входа со значениями из диапазонов [0 1] и [-1 1]:

```
net = newff([0 1; -1 1],[3]); gensim(net)
```

Выведем значения массивов весов и смещений:

```
net.iw{1,1}, net.b{1}
```

```
ans =
   -1.0809   -2.3639
    4.7917   -0.3739
   -1.9788   -2.2138
ans =
    2.9653
   -2.3959
   -1.4355
```

Эти же значения можно вывести в виде объединенного вектора, который содержится в описании нейронной сети:

```
x = getx(net); x'
```

```
ans =
  Columns 1 through 4
   -1.0809    4.7917   -1.9788   -2.3639
  Columns 5 through 8
   -0.3739   -2.2138    2.9653   -2.3959
  Column 9
   -1.4355
```

C428. SETX

Создадим однослойную сеть с тремя нейронами и двухэлементным вектором входа со значениями из диапазонов [0 1] и [-1 1]:

```
net = newff([0 1; -1 1],[3]); gensim(net)
net.iw, net.b
```

```
ans =
```

```
    [3x2 double]
ans =
    [3x1 double]
```

Сеть имеет 6 весовых коэффициентов и 3 элемента смещений, то есть всего 9 значений. Зададим этим элементам случайные значения и включим их в описание нейронной сети:

```
net = setx(net,rand(9,1));
```

Эти значения можно вывести на экран с помощью команды

```
getx(net)
ans =
    0.6252
    0.7334
    0.3759
    0.0099
    0.4199
    0.7537
    0.7939
    0.9200
    0.8447
```

C429. CALCPERF

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, двумя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2]:

```
net = newlin([0 1],2,[0 2 4]);
net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2]; gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 5 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3 0.6 0.4}; Pi = {0.2 0.3 0.4 0.1}; Pc = [Pi P];
Pd = calcpd(net,5,1,Pc);
```

сформируем вектор начальных условий на линии задержки выхода слоя для каждого из двух нейронов и массив векторов целей на 5 шагах по времени:

```
Ai = {[0.5; 0.1] [0.6; 0.5]};
Tl = {[0.1;0.2] [0.3;0.1], [0.5;0.6] [0.8;0.9], [0.5;0.1]};
```

Извлечем объединенный вектор весов и смещений из описания сети

```
X = getx(net);
```

Вычислим функционал качества и сигналы в сети

```
[perf,El,Ac,N,BZ,IWZ,LWZ] = calcperf(net,X,Pd,Tl,Ai,1,5);
```

Выведем значения функционала качества и массива ошибок слоя

```
perf, cat(2, El{:})
perf =
    0.2470
ans =
  Columns 1 through 4
    0.1000    0.3000    0.5000    0.8000
    0.2000    0.1000    0.6000    0.9000
  Column 5
    0.5000
    0.1000
```

C430. CALCGX

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, двумя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2]:

```
net = newlin([0 1],2,[0 2 4]);
net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2]; gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 5 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3 0.6 0.4}; Pi = {0.2 0.3 0.4 0.1}; Pc = [Pi P];
Pd = calcpd(net,5,1,Pc);
```

Сформируем вектор начальных условий на линии задержки выхода слоя для каждого из двух нейронов и массив векторов целей на 5 шагах по времени:

```
Ai = {[0.5; 0.1] [0.6; 0.5]};
Tl = {[0.1;0.2] [0.3;0.1], [0.5;0.6] [0.8;0.9], [0.5;0.1]};
```

Извлечем из описания сети объединенный вектор весов и смещений сети и вычислим функционал качества и сигналы сети

```
X = getx(net);
[perf,El,Ac,N,BZ,IWZ,LWZ] = calcperf(net,X,Pd,Tl,Ai,1,5);
```

В заключение используем функцию calcgx, чтобы вычислить градиент функционала по объединенному вектору весов и смещений:

```
[gX,normgX] = calcgx(net,X,Pd,BZ,IWZ,LWZ,N,Ac,El,perf,1,5); gX'
ans =
```

```
  Columns 1 through 4
    0.1720    0.1540    0.0600    0.0420
  Columns 5 through 8
    0.0780    0.0800    0.0120    0.0240
  Columns 9 through 12
    0.0100    0.0200    0.0460    0.0320
  Columns 13 through 16
    0.0320    0.0140    0.4400    0.3800
normgX
normgX =
    0.6440
```

C432. CALCJX

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, двумя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2]:

```
net = newlin([0 1],2, [0 2 4]); net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2]; gensim(net)
```

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 5 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3 0.6 0.4}; Pi = {0.2 0.3 0.4 0.1}; Pc = [Pi P];
Pd = calcpd(net,5,1,Pc);
```

Зададим два начальных значения запаздывающих выходов слоя для каждого из двух нейронов и цели слоя для двух нейронов на пять шагов по времени:

```
Ai = {[0.5; 0.1] [0.6; 0.5]};
Tl = {[0.1;0.2] [0.3;0.1], [0.5;0.6] [0.8;0.9], [0.5;0.1]};
```

Извлечем из описания сети объединенный вектор весов и смещений сети и вычислим функционал качества и сигналы сети

```
X = getx(net);
[perf,El,Ac,N,BZ,IWZ,LWZ] = calcperf(net,X,Pd,Tl,Ai,1,5);
```

Теперь можно применить функцию calcjx, чтобы вычислить якобиан функционала качества по объединенной матрице весов и смещений

```
jX = calcjx(net,Pd,BZ,IWZ,LWZ,N,Ac,1,5);
figure(1), clf, spy(jX)
```

C433. CALCJEJJ

Создадим линейную сеть с одним входом, изменяющимся в диапазоне от 0 до 1, двумя нейронами и линией задержки на входе с параметрами [0 2 4]; в сети используется обратная связь с линией задержки [1 2]:

```
net = newlin([0 1],2, [0 2 4]); net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2]; gensim(net)
net.iw, net.lw, net.b
ans =
    [2x3 double]
ans =
    [2x4 double]
ans =
    [2x1 double]
```

Данная сеть имеет 16 настраиваемых параметров: 6 элементов весовой матрицы входа, 8 элементов весовой матрицы в обратной связи и 2 элемента вектора смещения.

Вычислим вектор запаздывающих входов Pd, если заданы реализация вектора входа P для 5 шагов по времени, вектор начальных условий на линии задержки входов Pi:

```
P = {0 0.1 0.3 0.6 0.4}; Pi = {0.2 0.3 0.4 0.1}; Pc = [Pi P];
Pd = calcpd(net,5,1,Pc);
```

Зададим два начальных значения запаздывающих выходов слоя для каждого из двух нейронов и цели слоя для двух нейронов на 5 шагов по времени:

```
Ai = {[0.5; 0.1] [0.6; 0.5]};
Tl = {[0.1;0.2] [0.3;0.1], [0.5;0.6] [0.8;0.9], [0.5;0.1]};
```

Извлечем из описания сети объединенный вектор весов и смещений сети и вычислим функционал качества и сигналы сети

```
X = getx(net);
[perf,El,Ac,N,BZ,IWZ,LWZ] = calcperf(net,X,Pd,Tl,Ai,1,5);
```

В заключение используем функцию calcjejj, задав коэффициент экономии памяти равным 2

```
tic, [je,jj,normje] = calcjejj(net,Pd,BZ,IWZ,LWZ,N,Ac,El,1,5,2);
toc, figure(1), clf, spy(je)
elapsed_time =
    0.7700
```

Результаты будут одинаковыми при любом значении коэффициента экономии памяти, однако время вычислений будет расти. Увеличим коэффициент экономии памяти до значения 4:

```
tic, [je,jj,normje] = calcjejj(net,Pd,BZ,IWZ,LWZ,N,Ac,El,1,5,4);
toc, figure(2), clf, spy(je)
elapsed_time =
    0.9800
```

Операции с массивами данных

C436. CELL2MAT

```
C = {[1 2] [3]; [4 5; 6 7] [8; 9]};
figure(1), clf, cellplot(C), M = cell2mat(C)
M =
     1     2     3
     4     5     8
     6     7     9
```

C437. COMBVEC

Рассмотрим следующие 2 выборки P1 и P2 и рассчитаем объединенную выборку P:

```
P1 = [1 2 3; 4 5 6]; P2 = [7 8; 9 10];
P = combvec(P1,P2)
P =
     1     2     3     1     2     3
     4     5     6     4     5     6
     7     7     7     8     8     8
     9     9     9    10    10    10
```

Добавим выборку P3

```
P3 = [4 5 6];
P = combvec(P,P3)
P =
  Columns 1 through 7
     1     2     3     1     2     3     1
     4     5     6     4     5     6     4
     7     7     7     8     8     8     7
     9     9     9    10    10    10     9
     4     4     4     4     4     4     5
  Columns 8 through 14
     2     3     1     2     3     1     2
     5     6     4     5     6     4     5
     7     7     8     8     8     7     7
```

```
     9     9    10    10    10     9     9
     5     5     5     5     5     6     6
  Columns 15 through 18
     3     1     2     3
     6     4     5     6
     7     8     8     8
     9    10    10    10
     6     6     6     6
```

C438. CON2SEQ, SEQ2CON

Преобразуем числовой массив P размера 2×3, соответствующий групповому представлению данных, в массив ячеек S размера 1×3, содержащих числовые массивы размера 2×1 и соответствующий последовательному представлению данных:

```
P = [1 4 2; 2 5 3]; S = con2seq(P)
S =
  Columns 1 through 2
    [2x1 double]    [2x1 double]
    [2x1 double]
```

Преобразуем массив ячеек P размера Q×1, содержащих массивы размера R×m*TS, который соответствует структуре группового представления, в массив ячеек S размера Q×TS, содержащих массивы размера R×m, и который соответствует структуре последовательного представления данных.

```
P = { [1 2; 1 2]; [3 4; 3 4]; [5 6; 5 6] }; S = con2seq(P,2)
S =
    [2x1 double]    [2x1 double]
    [2x1 double]    [2x1 double]
    [2x1 double]    [2x1 double]
```

Этому массиву соответствует следующее описание

```
cell2mat(S), figure(1), clf, cellplot(S)
ans =
     1     2
     1     2
     3     4
     3     4
     5     6
     5     6
```

Преобразуем массив ячеек S размера Q×TS, содержащих числовые массивы размера R×m, в массив ячеек P размера Q×1 . При этом каждая ячейка содержит числовой массив размера R×m*TS:.

```
S = {[1; 1] [5; 4] [1; 2];  [3; 9] [4; 1] [9; 8]}
P = seq2con(S)
S =
  Columns 1 through 2
    [2x1 double]    [2x1 double]
    [2x1 double]    [2x1 double]
    [2x1 double]
    [2x1 double]
P =
    [2x3 double]
    [2x3 double]
```

Сформируем числовой массив P, соответствующий групповому представлению

```
P = cell2mat(P)
P =
     1     5     1
     1     4     2
     3     4     9
     9     1     8
```

C439. CONCUR

Функция concur создает три копии вектора смещения для данного слоя нейронной сети:

```
b = [1; 3; 2; -1]; B = concur(b,3)
B =
     1     1     1
     3     3     3
     2     2     2
    -1    -1    -1
```

Двухслойная нейронная сеть имеет 2 вектора смещения, которые для применения функции concur необходимо объединить в вектор ячеек

```
b1 = [1; 3; 2; -1]; b2 = [3; 2; -1]; b = {b1; b2}
B = concur(b,3)
b =
    [4x1 double]
    [3x1 double]
B =
```

```
    [4x3 double]
    [3x3 double]
```

C440. IND2VEC, VEC2IND

Преобразовать вектор индексов классов в матрицу связности

```
ind = [1 3 2 3], vec = ind2vec(ind), vec = full(vec)
ind =
     1     3     2     3
vec =
   (1,1)        1
   (3,2)        1
   (2,3)        1
   (3,4)        1
vec =
     1     0     0     0
     0     0     1     0
     0     1     0     1
```

Преобразовать матрицу связности в вектор индексов классов

```
vec = [1 0 0 0; 0 0 1 0; 0 1 0 1];
ind = vec2ind(vec)
ind =
     1     3     2     3
```

C441. MAT2CELL

Преобразовать числовой массив M размера 3×4 в массив ячеек с разбиением mrow = [2 1], ncol = [1 2 1]:

```
M = [1 2 3 4; 5 6 7 8; 9 10 11 12];
C = mat2cell(M,[2 1],[1 2 1])
C =
  Columns 1 through 2
    [2x1 double]    [2x2 double]
    [         9]    [1x2 double]
    [2x1 double]
    [        12]
[C{1,:}; C{2,:}]
ans =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

C441. MINMAX

Вычислить минимальные и максимальные значения элементов в строках массива P:

```
P = [0 1 2; -1 -2 -0.5]; pr = minmax(P)
pr =
          0    2.0000
   -2.0000   -0.5000
```

C442. NORMC, NORMR

Нормировать матрицу M по строкам и столбцам

```
M = [1 2; 3 4];
normr(M), normc(M)
ans =
    0.4472    0.8944
    0.6000    0.8000
ans =
    0.3162    0.4472
    0.9487    0.8944
```

C442. PNORMC

Выполнить псевдонормировку стобцов матрицы

```
M = [0.1 0.6; 0.3 0.1];
pM = pnormc(M)
pM =
    0.1000    0.6000
    0.3000    0.1000
    0.9487    0.7937
```

C443. QUANT

Округлить элементы массива P с точностью до 0.1

```
P = [1.333 4.756 -3.897]; qP = quant(P,0.1)
qP =
    1.3000    4.8000   -3.9000
```

C443. SUMSQR

Вычислить сумму квадратов всех элементов массива M:

```
M = [ 1 2 3; 4 5 6]; s = sumsqr(M)
s =
    91
```

Графические утилиты

C443. PLOTV

Отобразить векторы в виде линий

```
P = [-0.4 0.7 0.2; -0.5 0.1 0.5];
figure(1), clf, plotv(P,'-'); grid on %Рис.11.64
```

C444. PLOTVEC

Отобразить векторы входа в виде маркеров

```
P =[ 0.1000    1.0000    0.5000    0.7000
    -1.0000    2.0000    0.5000    0.1000
     1.0000    0.1000    0.7000    0.5000];
t = minmax(P)'; figure(1), clf, axis(t(:)'), c=[1 2 3 4]';
plotvec(P,c,'o'), grid on %Рис.11.65
```

C445. PLOTPV

Отобразить векторы входа и целей персептрона в виде маркеров

```
P = [0 0 1 1; 0 1 0 1; 1 0 0 1];  T = [0 0 0 1];
figure(1), clf, plotpv(P,T),grid on,
title('Векторы входов и целей') %Рис.11.66
```

C446. PLOTPC

Определим векторы входов и целей и отобразим их в двумерном пространстве входов

```
P = [0 0 1 1; 0 1 0 1]; T = [0 0 0 1];
figure(1), clf, plotpv(P,T), grid on, hold on
```

Создадим персептрон с входами P, зададим произвольные значения весам и смещениям и построим разделяющую линию в пространстве входов:

```
net = newp(minmax(P),1);
net.iw{1,1} = [-1.2 -1]; net.b{1} = 1.3;
h = plotpc(net.iw{1,1},net.b{1});
set(h,'Color',[1/2,1/2,0],'LineWidth',2)
```

Построиv разделяющую плоскость в трехмерном пространстве

```
P = [0 0 1 1; 0 1 0 1; 1 0 0 1]; T = [0 0 0 1];
figure(1), clf, plotpv(P,T), grid on, hold on
```

Следующие функции создают персептрон с входами, соответствующими значениям вектора P, назначают значения его весам и смещениям и строят разделяющую плоскость:

```
net = newp(minmax(P),1);
net.iw{1,1} = [-1.2 -1 -0.5]; net.b{1} = 1.3;
h = plotpc(net.iw{1,1}, net.b{1});
```

C448. HINTONW

Зададим случайную матрицу весов и построим для нее диаграмму Хинтона, используя значения дополнительных аргументов по умолчанию

```
W = rands(4,5); hintonw(W), % Рис.11.69
```

C449. HINTONWB

Зададим случайные матрицу весов и вектор смещений и построим для них диаграмму Хинтона, используя значения дополнительных аргументов по умолчанию

```
W = rands(4,5); b = rands(4,1);
hintonwb(W,b), % Рис.11.70
```

C450. PLOTPERF

Зададим 8 значений вектора входа P, соответствующий им вектор целей T, а также контрольное подмножество в виде векторов VV.P и VV.T:

```
P = 1:8; T = sin(P); VV.P = P; VV.T = T+rand(1,8)*0.1;
```

Создадим и обучим двухслойную сеть прямой передачи с 4-мя нейронами в первом слое с функцией активации tansig и одним нейроном во втором слое также с функцией активации tansig

```
net = newff(minmax(P),[4 1],{'tansig','tansig'});
[net,tr] = train(net,P,T,[],[],VV); grid on
TRAINLM, Epoch 0/100, MSE 1.45434/0, Gradient 2.89033/1e-010
TRAINLM, Epoch 25/100, MSE 0.358863/0, Gradient 0.12179/1e-010
TRAINLM, Epoch 50/100, MSE 0.0103365/0, Gradient 0.0194758/1e-010
TRAINLM, Epoch 70/100, MSE 0.00721783/0, Gradient 0.0138325/1e-010
TRAINLM, Validation stop.
```

В процеесе выполнения процедуры train для построения графика точности обучения также применяется функция plotperf и во многих случаях этого бывает достаточно для оценки процедуры обучения.

Однако функция plotperf позволяет оформить графики результатов обучения и по завершении этой процедуры, используя дополнительные аргументы. Например, выполняя обучение с предельной точностью, заданной по умолчанию, на заключительном графике можно указать требуемую точность и оценить длительность обучения

```
plotperf(tr, 0.005) %Рис.11.71
```

C451. ERRSURF, PLOTES

Вычислить и построить график поверхности ошибки для нейрона

```
p = [3 2]; t = [0.4 0.8];
wv = -4:0.4:4; bv = wv;
ES = errsurf(p,t,wv,bv,'logsig');
plotes(wv,bv,ES,[60 30]) % Рис.11.72
```

C452. PLOTEP

Построить траекторию обучения в пространстве настраиваемых параметров. Script-файл, приведенный ниже, описывает сценарий построения такой траектории. Этот сценарий необходимо скопировать и выполнить в рабочем окне системы MATLAB:

```
Script
 %   Задание обучающей последовательности
 P = 1:8; T = sin(P);
 %   Построение поверхности ошибок
 w_range = -1:0.2:1;  b_range = -1:0.2:1;
 ES = errsurf(P, T, w_range, b_range, 'purelin');
 plotes(w_range, b_range,ES);
 %    Формирование нейронной сети
 maxlr = 0.40*maxlinlr(P, 'bias');
 net = newlin([-2 2], 1, [0], maxlr);
 %    Задание начальной точки
 subplot(1, 2, 2);
 h = text(sum(get(gca, 'xlim'))*0.5, ...
 sum(get(gca, 'ylim'))*0.5, '*Укажите точку*');
 [net.IW{1,1} net.b{1}] = ginput(1);
 delete(h);
% Построение траектории обучения
 limiting = net.trainParam.epochs;
 limloop = limiting+1;
 net.trainParam.epochs = 1;
 net.trainParam.goal = .001;
 net.trainParam.show = Inf;
 h = plotep(net.IW{1}, net.b{1}, mse(T-sim(net, P)));
 [net, tr] = train(net, P, T);
 r = tr;
 epoch = 1; cont = 1;
 while (length(r.epoch)<limloop & cont)
```

```
      epoch = epoch+1;
      [net, tr]=train(net, P, T);
      if length(tr.epoch)>1
         h = plotep(net.IW{1,1}, net.b{1}, tr.perf(2), h);
         r.epoch = [r.epoch epoch];
         r.perf    = [r.perf tr.perf(2)];
         r.vperf  = [r.vperf NaN];
         r.tperf  = [r.tperf NaN];
      else
         cont = 0;
      end;
   end; % Рис.11.73
```

Информация о сети и ее топологии

C454. DISP, DISPLAY

Создадим персептрон и выведем на экран его свойства:

```
net = newp([-1 1; 0 2],3); display(net)
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: {1x1 cell} of inputs
            layers: {1x1 cell} of layers
           outputs: {1x1 cell} containing 1 output
           targets: {1x1 cell} containing 1 target
            biases: {1x1 cell} containing 1 bias
      inputWeights: {1x1 cell} containing 1 input weight
      layerWeights: {1x1 cell} containing no layer weights
    functions:
          adaptFcn: 'trains'
```

```
        initFcn: 'initlay'
    performFcn: 'mae'
      trainFcn: 'trainc'
parameters:
    adaptParam: .passes
     initParam: (none)
  performParam: (none)
    trainParam: .epochs, .goal, .show, .time
weight and bias values:
            IW: {1x1 cell} containing 1 input weight matrix
            LW: {1x1 cell} containing no layer weight matrices
             b: {1x1 cell} containing 1 bias vector
other:
      userdata: (user stuff)
```

C455. PLOTSOM

Рассмотрим способы отображения двумерных и трехмерных карт Кохонена; для этого с помощью М-фунеции gridtop рассчитаем сетку с прямоугольной топологией

```
pos = gridtop(4,3);
figure(1), clf, plotsom(pos),
title('Позиции нейронов'), grid on   %Рис.11.74,а
```

Для отображения двумерной карты Кохонена в пространстве весов зададим случайную матрицу весов W размера 12x2 и вычислим матрицу расстояний D на выбранной сетке

```
W = rand(12,2); D = dist(pos);
figure(2), clf,
plotsom(W,D), title('Позиции нейронов в пространстве весов'),
grid on      %Рис.11.74,б
```

Выполним аналогичные построения для трехмерной карты Кохонена

```
pos = gridtop(4,3,3);
figure(1), clf, plotsom(pos), plotsom(pos)
title('Позиции нейронов'), grid on   %Рис.11.75,а
D = dist(pos); W = rand(36,3);
plotsom(W, D), figure(2), clf,
plotsom(W,D), title('Позиции нейронов в пространстве весов'),
grid on %Рис.11.75,б
```

C457. GRIDTOP

Рассчитать положения нейронов на 4-мерной сетке с прямоугольной топологией размера 5x4x3x2 и выполнить попытку построить график

```
pos = gridtop(5,4,3,2);
figure(1), clf, plotsom(pos), plotsom(pos)
title('Позиции нейронов'), grid on   %Рис.11.76
```

```
Warning - PLOTSOM only shows first three dimensions.
Warning - PLOTSOM only shows first three dimensions.
```

C457. HEXTOP

Рассчитать положения нейронов на 3-мерной сетке с гексагональной топологией размера 5x4x3 с 60 нейронами и построить график их расположения

```
pos = hextop(5,4,3);
figure(1), clf, plotsom(pos), plotsom(pos)
title('Позиции нейронов'), grid on   %Рис.11.77
```

C458. RANDTOP

Рассчитать положения нейронов на 3-мерной сетке с гексагональной топологией размера 5x4x3 с 60 нейронами и построить график их расположения

```
pos = randtop(5,4,3);
figure(1), clf, plotsom(pos), plotsom(pos)
title('Позиции нейронов'), grid on   %Рис.11.78
```

Моделирование нейронных сетей и система Simulink

Функции моделирования сети

C459. SIM

Создадим нейронную сеть персептрона с одним слоем и двухэлементным входом с диапазоном значений [0 1] и одним нейроном

```
net = newp([0 1;0 1],1);
```

Теперь можно промоделировать персептрон, подавая различные последовательности векторов входа: один вектор с двумя элементами, группа из двух векторов с тремя элементами, последовательность из 3 векторов, что соответствует различным формам представления данных

```
p1 = [.2; .9]; a1 = sim(net,p1)
p2 = [.2 .5 .1 ;.9 .3 .7]; a2 = sim(net,p2)
p3 = {[.2; .9] [.5; .3] [.1; .7]}; a3 = sim(net,p3)
a1 =
     1
```

```
a2 =
     1     1     1
a3 =
    [1]    [1]    [1]
```

В данном случае в качестве результата выводятся только выходы нейрона.

Создадим динамическую однослойную линейную сеть с двумя нейронами, трехэлементным входом с диапазоном значений [0 2] и линией задержки на входе [0 1]:

```
net = newlin([0 2;0 2;0 2],2,[0 1]);
```

Линейный слой моделируется с последовательностью из двух векторов входа, для заданных по умолчанию начальных условия на линиях задержки

```
p1 = {[2; 0.5; 1] [1; 1.2; 0.1]};
[y1,pf] = sim(net,p1)
y1 =
    [2x1 double]    [2x1 double]
pf =
    [3x1 double]
```

Затем этот слой моделируется еще для 3 векторов, используя состояния на элементах задержки как новые начальные условия

```
p2 = {[0.5; 0.6; 1.8] [1.3; 1.6; 1.1] [0.2; 0.1; 0]};
[y2,pf] = sim(net,p2,pf)
y2 =
  Columns 1 through 2
    [2x1 double]    [2x1 double]
    [2x1 double]
pf =
    [3x1 double]
```

Создадим двухслойную сеть Элмана с одноэлементным входом с диапазоном значений [0 1], имеющую 3 нейрона с функцией активации tansig в слое 1 и 2 нейрона с функцией активации purelin в слое 2. Сеть Элмана имеет линию задержки [0 1] при переходе от слоя 1 к слою 2:

```
net = newelm([0 1],[3 2],{'tansig','purelin'});
```

Сеть моделируется для трехэлементного вектора входа, используя заданные по умолчанию начальные условия на линии задержки

```
p1 = {0.2 0.7 0.1};
[y1,pf,af,e,perf] = sim(net,p1)
y1 =
  Columns 1 through 2
```

```
    [2x1 double]    [2x1 double]
    [2x1 double]
pf =
   Empty cell array: 1-by-0
af =
    [3x1 double]
    [2x1 double]
e =
  Columns 1 through 2
    [2x1 double]    [2x1 double]
    [2x1 double]
perf =
    0.5221
[e{:}]
ans =
    0.3072    0.7206    0.7108
    0.7558    0.8468    0.8517
```

Выполним еще один шаг моделирования, но теперь для четырехэлементного вектора входа, используя состояния на элементах задержки как новые начальные условия

```
p2 = {0.1 0.9 0.8 0.4};
[y1,pf,af,e,perf] = sim(net,p2,pf,af)
y1 =
  Columns 1 through 2
    [2x1 double]    [2x1 double]
  Columns 3 through 4
    [2x1 double]    [2x1 double]
pf =
   Empty cell array: 1-by-0
af =
    [3x1 double]
    [2x1 double]
e =
  Columns 1 through 2
    [2x1 double]    [2x1 double]
  Columns 3 through 4
    [2x1 double]    [2x1 double]
perf =
    0.6336
[e{:}]
ans =
    0.7274    0.7144    0.7448    0.7554
```

```
    0.8536    0.8453    0.8526    0.8568
```

Построение моделей нейронных сетей

C465. GENSIM

Создадим однослойную линейную сеть, реализующую следующее соотношение между входом и целью

```
P = [1 2 3 4 5];  T = [1 3 5 7 9];
net = newlind(P,T);
```

Для того чтобы сформировать S-модель этой сети, используем команду

```
gensim(net,-1) %Рис.11.84
```

Теперь рассмотрим пример моделирования динамической сети. Обратимся к сети Элмана, связанной с детектированием амплитуды гармонического сигнала.

Сформируем и обучим сеть Элмана

```
clear, p1 = sin(1:20); p2 = sin(1:20)*2;
t1 = ones(1,20); t2 = ones(1,20)*2;
p = [p1 p2 p1 p2]; t = [t1 t2 t1 t2];
Pseq = con2seq(p); Tseq = con2seq(t);
R = 1; % Число элементов входа
S2 = 1;% Число нейронов выходного слоя
S1 = 10;% Число нейронов рекуррентного слоя
net = newelm([-2 2],[S1 S2],{'tansig','purelin'},'traingdx');
net.trainParam.epochs = 500; net.trainParam.show = 25;
net.trainParam.goal = 0.05;
[net,tr] = train(net,Pseq,Tseq);
TRAINGDX, Epoch 0/500, MSE 9.2038/0.05, Gradient 21.0604/1e-006
TRAINGDX, Epoch 25/500, MSE 0.322819/0.05, Gradient 0.533849/1e-
006
TRAINGDX, Epoch 50/500, MSE 0.269378/0.05, Gradient 0.174894/1e-
006
TRAINGDX, Epoch 75/500, MSE 0.219944/0.05, Gradient 0.14799/1e-006
TRAINGDX, Epoch 100/500, MSE 0.0492742/0.05, Gradient 0.162553/1e-
006
TRAINGDX, Performance goal met.
gensim(net)
```

Создадим S-модель этой сети, дополнив ее блоком To Workspace, который позволяет записать результаты моделирования в виде массива yout в рабочую область системы MATLAB (рис.11.88, а).

Задав значение входного сиграла (рис. 11.89,а), выполним моделирование сети и выведем результат в виде графика изменения выхода сети, используя данные из рабочей области системы MATLAB.

```
stairs(tout(1:5:end),yout) %Рис.11.89,б
```

C470. GENSIMM

Сформируем M-файл для S-модели сети Элмана

```
p1 = sin(1:20); p2 = sin(1:20)*2;
t1 = ones(1,20); t2 = ones(1,20)*2;
p = [p1 p2 p1 p2]; t = [t1 t2 t1 t2];
Pseq = con2seq(p); Tseq = con2seq(t);
R = 1; % Число элементов входа
S2 = 1;% Число нейронов выходного слоя
S1 = 10;% Число нейронов рекуррентного слоя
net = newelm([-2 2],[S1 S2],{'tansig','purelin'},'traingdx');
netelm = gensimm(net)
netelm =
C:\WINDOWS\TEMP\matlab_nnet\tp354077.m
```

Выведем текст M-файла:

```
type(netelm)
function [perf,El,Ac,N,LWZ,IWZ,BZ] = tp354077(net,Pd,Ai,Tl,Q,TS)
%TP354077 Temporary network simulation file.
%
% [perf,El,Ac,N,LWZ,IWZ,BZ] = tp354077(net,Pd,Ai,Tl,Q,TS)
%   net  - Neural network.
%   Pd     -  numInputs-by-numLayers-by-TS  cell  array  of  delayed
inputs.
%   Ai    - numLayers-by-numLayerDelays cell array of layer delay
conditions.
%   Tl   - numLayers-by-TS cell array of layer targets.
%   Q    - number of concurrent simulations.
%   TS   - number of time steps.
% returns:
%   perf - network performance:
%   El   - numLayers-by-TS cell array of layer errors:
%   Ac     - numLayers-by-(numLayerDelays+TS) cell array of layer
outputs:
%   N    - numLayers-by-TS cell array of net inputs:
%   LWZ    -  numLayers-by-numLayers-by-TS  cell  array  of  weighed
layer outputs:
```

```
  %   IWZ    -  numLayers-by-numInputs-by-TS  cell  array  of  weighed
inputs:
  %   BZ   - numLayers-by-1 cell array of expanded biases:

  % Input weights
  IW1_1 = net.IW{1,1};

  % Layer weights
  LW1_1 = net.LW{1,1};
  LW2_1 = net.LW{2,1};

  % Biases
  QOnes = ones(1,Q);
  B1 = net.b{1}(:,QOnes);
  B2 = net.b{2}(:,QOnes);
  BZ = {B1; B2};

  % Signals
  El = cell(2,TS);
  Ac = [Ai cell(2,TS)];
  N = cell(2,TS);
  IWZ = cell(2,1,TS);
  LWZ = cell(2,2,TS);

  for ts=1:TS;
    tsc = ts + 1;

    % Simulate Layer 1
    IWZ{1,1,ts} = IW1_1*Pd{1,1,ts};
    LWZ{1,1,ts} = LW1_1*Ac{1,tsc-1};
    N{1,ts} = IWZ{1,1,ts}+LWZ{1,1,ts}+B1;
    Ac{1,tsc} = tansig(N{1,ts});

    % Simulate Layer 2
    LWZ{2,1,ts} = LW2_1*Ac{1,tsc};
    N{2,ts} = LWZ{2,1,ts}+B2;
    Ac{2,tsc} = N{2,ts};
    El{2,ts} = Tl{2,ts} - Ac{2,tsc};
  end;

  perf = mse(El,net,net.trainParam);
```

Оглавление