Лабораторная работа №5

**6. Изучение основных функций пакета Neural Network Toolbox для нейронных сетей кластеризации данных**

Цель: Ознакомиться с основными командами создания, обучения и применения нейронных сетей кластеризации и классификации данных в Neural Network Toolbox с помощью сетей Кохонена.

**Порядок выполнения лабораторной работы**

1. Ознакомиться с демонстрационными программами нейронных сетей Кохонена и LVQ, для чего в командной строке набрать

>>help nndemos

demogrn1 - GRNN function approximation.

demopnn1 - PNN classification.

Learning Vector Quantization

demolvq1 - Learning vector quantization.

1. Для запуска демонстрационных программ нейронных сетей Кохонена выбрать demoс1(настройка слоя Кохонена), demosm1(одномерная карта Кохонена), demosm2(двумерная карта Кохонена), классификация векторов (LVQ-cеть demolvq1).
2. Создать m-файл, моделирующий нейронные сети Кохонена в соответствие с заданием (таблица 1) по методике, рассмотренной в приложении А, используя входной вектор обучающей выборки, Р.
3. Использовать гексагональную сетку размера 6×6. Для создания самоорганизующейся карты Кохонена использовать М-функцию newsom. Для оценки качества кластеризации использовать среднеквадратическое отклонение от центра кластера. Для тестирования создать не менее 10 тестовых векторов, не участвующих в обучении сети Кохонена.
4. Выполнить кластеризации с применением нейронных сетей Кохонена в соответствие с заданием (Приложение А). Создать и выполнить моделирование нейронной Кохонена, согласно согласованному с преподавателем, приведенному в приложении А.
5. Результаты кластеризации вывести в виде графика кластеров, а также точки тестовых векторов.
6. Для лаб.4 разработать нейросетевую экспертную систему на базе сети LVQ. В качестве обучающей выборки использовать созданные таблицы входных и выходных векторов в 7 семестре при моделировании нейросетевой ЭС на базе РБФ сети.
7. Для создания сети LVQ обучающую последовательность использовать сеть Кохонена. Для обучения использовать таблицы входных тестовых векторов.
8. По результатам выполнения лабораторной работы составить отчёт, включающий m-файлы, моделирующие нейронную сети Кохонена и сеть LVQ. Сравнить нейросетевой ЭС на базе РБФ сети и сети LVQ.

**Приложение А**

**Выбор датасетов для задачи кластеризации**

Сайты, специализирующиеся на обзорах и рейтингах датасетов, обычно предоставляют полезную информацию о качестве данных, размере выборки, типе объектов и признаков. Такие обзоры помогут вам выбрать наиболее подходящие датасеты для вашей задачи.

Вот несколько популярных сайтов, где можно найти обзоры и рейтинги доступных датасетов:

**KDnuggets**: на этом сайте вы найдете различные обзоры и рейтинги датасетов в области машинного обучения и анализа данных. Здесь вы найдете обзоры таких тематических наборов данных, как демографические данные, социальные сети и медицинские записи.

**UCI Machine Learning Repository**: этот репозиторий содержит коллекцию датасетов, которые были изначально использованы в исследованиях и статьях в области машинного обучения. Здесь вы найдете много различных датасетов для классификации и кластеризации людей.

**Data.gov**: в этом государственном репозитории вы найдете широкий выбор открытых государственных данных, которые могут быть использованы для классификации и кластеризации людей. На сайте предоставляется информация о каждом датасете, включая его описание, количество объектов и признаков.

**Приложение А**

**Методические материалы по моделированию**

**Нейронные сети Кохонена**

### **1.Пример.**

1.1 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.1*

*xlabel('P(1,:)'), ylabel('P(2,:)'), hold on, grid on*



Рис.1

1.2. 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 рис.2.*

*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*

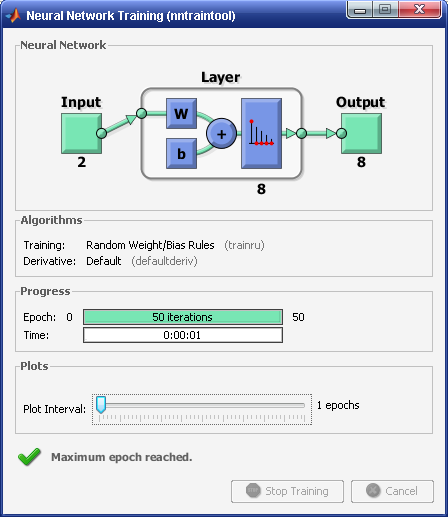


Рис.2

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

*plot(w(:,1),w(:,2),'or'),*

*title('Векторы входа и центры кластеризации')*

**

Рис.2

### **2 Карта Кохонена**

### **2.1 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. 3)*



Рис.3

### **2.2 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 % рис.4*

*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*



Рис.4

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

*net.IW{1}*

*ans =*

*1.0000 0.5000*

*1.0000 0.5000*

*1.0000 0.5000*

*1.0000 0.5000*

*1.0000 0.5000*

*1.0000 0.5000*

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

*a = sim(net,P)*

*a =*

*1 1 1 1 1 1 1 1 1 1 1 1*

*0 0 0 0 0 0 0 0 0 0 0 0*

*0 0 0 0 0 0 0 0 0 0 0 0*

*0 0 0 0 0 0 0 0 0 0 0 0*

*0 0 0 0 0 0 0 0 0 0 0 0*

*0 0 0 0 0 0 0 0 0 0 0 0*

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

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

### *a =*

### *1*

### *0*

### *0*

### *0*

### *0*

### *0*

### **3. Одномерная карта Кохонена**.

3.1 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 (рис.5) входных векторов символом \* отмечено положение каждого десятого вектора.



Рис.5

3.2 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. 6*



Рис.6

*TRAINR, Epoch 0/50*

*TRAINR, Epoch 25/50*

*TRAINR, Epoch 50/50*

*TRAINR, Maximum epoch reached.*

*elapsed\_time =*

*36.1500*

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

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

a =

1

0

0

0

0

0

0

0

0

0

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

### **4. Двумерная карта Кохонена**

4.1 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*



Рис.7

4.2 Для кластеризации векторов входа создадим самоорганизующуюся карту Кохонена размера 3×4 с 12 нейронами, размещенными на гексагональной сетке (рис.8).

*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*

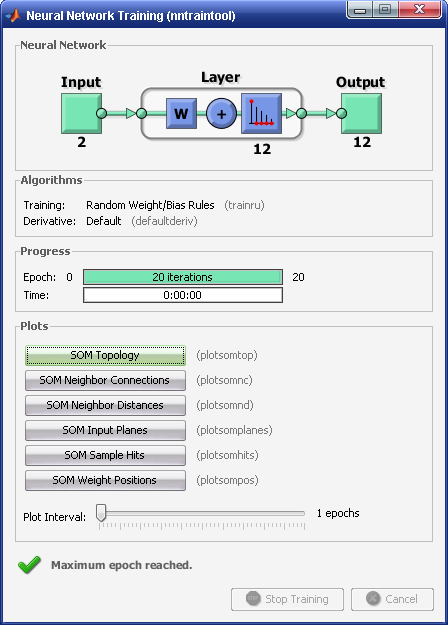
**

Рис.8

*figure(1), clf, plotsom(net.IW{1,1},net.layers{1}.distances) % рис.9*

**

Рис.9

4.3 Определим принадлежность нового вектора одному из кластеров карты Кохонена (рис.10).

*a = sim(net,[0.5;0.3])*

*hold on, plot(0.5,0.3,'\*k')*

*a =*

*(4,1) 1*

**

Рис.10

4.4 Промоделируем обученную карту Кохонена, используя массив векторов входа

*a = sim(net,P);*

*figure(2), clf, bar(sum(a')) %Рис. 11*



Рис.11

### **5. LVQ-сети**

5.1 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.

5.2 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.

5.3 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];*

5.4 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.12*



Рис.12

5.5 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*

5.6 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 c использованием следующего оператора

*net = newlvq(minmax(P),4,[.6 .4],0.05,'learnlv2');*

*net.inputWeights{1}*

*ans =*

*Neural Network Weight*

*delays: 0*

*initFcn: 'midpoint'*

*initSettings: .midpoint*

*learn: true*

*learnFcn: 'learnlv2'*

*learnParam: .lr, .window*

*size: [4 2]*

*weightFcn: 'negdist'*

*weightParam: (none)*

*userdata: (your custom info)*

Процедура обучения

5.7 Для обучения сети (рис.13) 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*

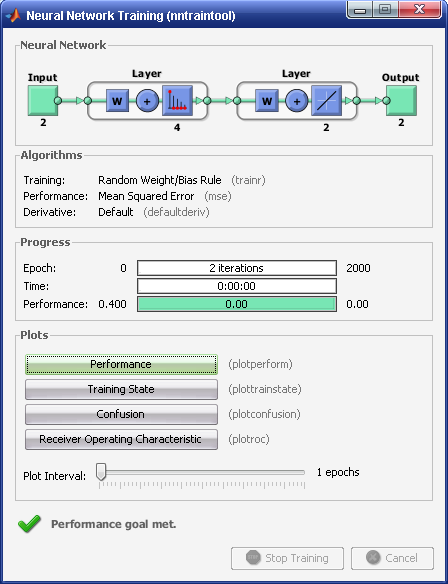
**

Рис.13

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

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

*V =*

*-0.5559 -0.0392*

*0.4295 -0.0489*

*0.0846 -0.2430*

*0.0978 0.3488*

5.9 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.14



Рис.14

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

net.LW{2}

ans =

1 1 0 0

0 0 1 1

**Нейронные сети Кохонена**

### Слой Кохонена.

Формирование слоя Кохонена выполняется с помощью М-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

### 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')

### 

### 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

### 

### 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

Cеть 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.

### 

### **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

### **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 c 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)