НЕЙРОСЕТЕВОЕ МОДЕЛИРОВАНИЕ В MATLAB

Лабораторный практикум

Лабораторная работа № 06-1. Исследование конкурентных слоев Кохонена

Лабораторная работа № 06-2. Исследование самоорганизующихся карт Кохонена

Лабораторная работа № 06-3. Исследование LVQ-сетей

Лабораторная работа № 06-1

Исследование конкурентных слоев Кохонена

Цель работы: изучение архитектуры самоорганизующихся нейронных слоев Кохонена и специальных функций для их создания, инициализации, взвешивания, накопления, активации, настройки весов и смещений, адаптации и обучения; ознакомление с демонстрационными примерами и их скриптами, а также приобретение навыков построения самоорганизующихся слоев для исследования топологической структуры данных, их объединением в кластеры (группы) и распределением по классам.

Теоретические сведения

Самоорганизующийся слой Кохонена – это однослойная нейронная сеть с конкурирующей передаточной функцией **`compet`**, которая анализирует выходные значения нейронов слоя и выдаёт в качестве результата наибольшее из этих значений (значение нейрона-победителя).

Инициализация весов входов производится с помощью функции средних значений

```
W = midpoint (S,PR),
```

где **`S`** – число нейронов в слое Кохонена;

`PR` – матрица размера `R×2`, задающая диапазоны $\left[P^j_{\min}, P^j_{\max}\right]$ изменения **`R`** элементов входного вектора;

`W` – матрица весов размера `S×R` для входов слоя, столбцы которой имеют значения $\frac{\left(P^j_{\min} + P^j_{\max}\right)}{2}$.

Инициализация весов смещений нейронов слоя производится с помощью функции равных смещений

```
B = initcon(S,PR),
```

которое каждому нейрону задает одно и то же смещение, равное **exp(1)*S**. Например, для **S=5** это смещение равно **5*2.71828==1.359740*10[1].**

Взвешивание входов слоя Кохонена реализуется в виде отрицательного евклидова расстояния между каждой строкой $\mathbf{W}_i$ матрицы весов **W** и каждым столбцом $\mathbf{P}_j$ матрицы входов **P**, которое вычисляется функцией **negdist(W,P)**. Суммирование взвешенных входов со смещениями производится функцией **netsum**.

Формирование самоорганизующегося слоя Кохонена осуществляется функцией (для версий с R2010b)

```
net = competlayer(S,KLr,clr),
```

и функцией (для версий ранее R2010b)

```
net = newc(PR,S,KLr,clr),
```

где **KLr** – параметр функции настройки весов, значение по умолчанию которого равно **0.01**;

clr – параметр функции настройки смещений, значение по умолчанию которого равно **0.001**.

Эта функция формирует однослойную сеть с **R** нейронами и **R** входами. Веса входов устанавливаются равными половине диапазона соответствующего вектора входа для всех нейронов. Также для всех нейронов устанавливается одно и то же смещение, равное **e*s**. Выходы нейронов поступают на конкурирующую передаточную функцию **compet**, которая определяет победителя. Номер активного нейрона-победителя **I*** определяет ту группу (кластер), к которой наиболее близок входной вектор.

Для того чтобы сформированная таким образом сеть решала задачу кластеризации данных, необходимо предварительно настроить ее веса и смещения по обучающей последовательностью векторов с помощью функций настройки **learnk** и **learncon** соответственно, используя процедуру адаптации **adapt** или процедуру обучения **train**.

Функция **learnk** рассчитывает приращение весов **dW** в зависимости от вектора входа **P**, выхода **a**, весов **w** и параметра скорости настройки **lr** в соответствии с правилом Кохонена:

$$\mathbf{dw} = \begin{cases} \mathbf{lr} * (\mathbf{p}' - \mathbf{w}), \mathbf{a_j} \neq \mathbf{0}; \\ \mathbf{0}, \mathbf{a_j} = \mathbf{0}. \end{cases}$$

Таким образом, вектор веса, наиболее близкий к вектору входа, модифицируется так, чтобы расстояние между ними стало ещё меньше. Результат такого обучения заключается в том, что победивший нейрон, вероятно, выиграет конкуренцию и в том случае, когда будет представлен новый входной вектор, близкий к предыдущему, и его победа менее вероятна, когда будет представлен вектор, существенно отличающийся от предыдущего. Когда на вход сети поступает всё большее и большее число векторов, нейрон, являющийся ближайшим, снова корректирует свой весовой вектор **w**. В конечном счёте, если в слое имеется достаточное количество нейронов, то каждая группа близких векторов окажется связанной с одним из нейронов слоя. В этом и заключается свойство самоорганизации слоя Кохонена.

Одно из ограничений всякого конкурирующего слоя состоит в том, что некоторые нейроны оказываются незадействованными, или “мертвыми”. Это происходит оттого, что нейроны, имеющие начальные весовые векторы, значительно удаленные от векторов входа, никогда не выигрывают конкуренции, независимо от продолжительности обучения. Для ликвидации нечувствительности таких нейронов используют положительные смещения, которые добавляются к отрицательным расстояниям удаленных нейронов. Функция **learncon** производит такую корректировку смещений следующим образом.

В начале процедуры настройки сети всем нейронам присваивается одинаковый характер активности $\mathbf{C_0}$ **= 1/S**. В процессе настройки эта величина для активных нейронов увеличивается, а для неактивных нейронов уменьшается:

$$\Delta\mathbf{C} = \mathbf{lr} * (\mathbf{a} - \mathbf{c}).$$

Нетрудно убедиться, что для всех нейронов, кроме нейрона- победителя, приращения отрицательны. Функция **learncon** рассчитывает приращения вектора смещений следующим образом: $\mathbf{\Delta b = exp(1 - log(c)) - b}$.

Увеличение смещений для неактивных нейронов позволяет расширить диапазон покрытия входных значений, и неактивный нейрон начинает формировать кластер, что улучшает кластеризацию входных данных.

Практические задания

В зависимости от версии MATLAB используйте для создания конкурентной сети функции **competlayer** или **newc**.

Пример 1. Пусть заданы 28 случайных векторов, изображённых на графике крестами (рис. 1). Оцифровав данный график, можно получить массив входных данных $P(1,:),P(2,:)$ (табл. 1).

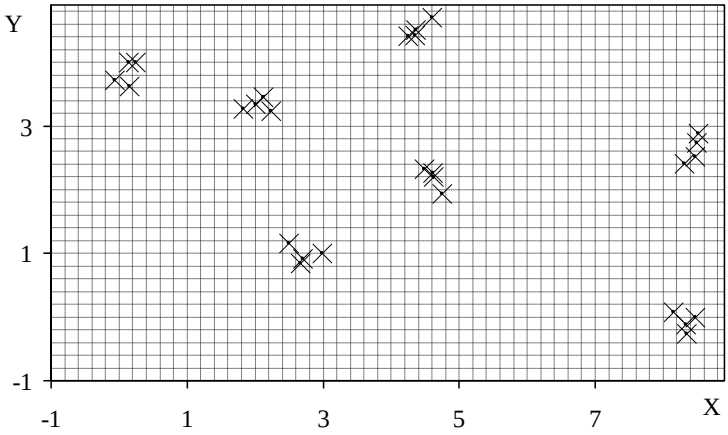

Рис. 1. Распределение входных векторов

Таблица 1. Массив входных данных

P(1,:)	3.0	-0.1	4.3	4.6	8.5	1.8	8.3	2.5	0.2	4.6

P(2,:)	1.0	3.7	4.4	2.3	2.7	3.3	-0.1	1.2	3.6	4.7
P(1,:)	4.5	8.5	2.2	8.2	2.7	0.2	4.4	4.6	8.3	2.0
P(2,:)	2.3	2.9	3.2	0.1	0.8	4.0	4.5	2.2	2.4	3.3
P(1,:)	8.5	2.7	0.1	4.3	4.7	8.5	2.1	8.3		
P(2,:)	0.0	0.9	4.0	4.4	1.9	2.5	3.5	-0.3		

Следующий алгоритм демонстрирует процедуру обучения самоорганизующейся нейронной сети Кохонена.

```
plot(P(1,:), P(2,:), '+m');
title('Input vectors');
xlabel('P(1,:)');
ylabel('P(2,:)');
hold on;
nclusters = 7;
nvectors = 4;
a1 = -10;
a2 = +10;
b1 = -5;
b2 = +5;
net = newc([a1 a2; b1 b2],nclusters,0.1,0.0005);
wo = net.IW{1};
bo = net.b{1};
co = exp(1)./bo;
net.trainParam.epochs=49;
net.trainParam.show=7;
net = train(net,P);
w = net.IW{1};
bn = net.b{1};
cn = exp(1)./bn;
plot(w(:,1),w(:,2),'kp');
```

На рис. 2 представлены исходные данные (кресты) и полученные центры кластеризации (звёзды).

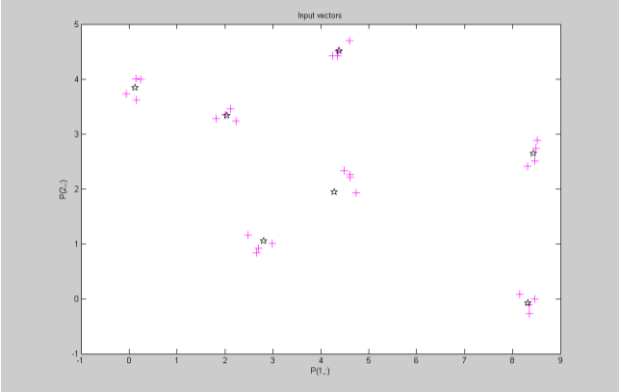

Рис. 2. Распределение входных данных (кресты) и положение центров кластеризации (звёзды)

Задание 1а. Изучить работу следующих функций

competlayer	Competitive layer (R2010b)
newc	Competitive layer (R2010b)
configure	Configure the network object and also initializes the weights and biases of the network
view	View neural network
train	Train neural network
trainru	Unsupervised random order weight/bias training
learnk	Kohonen weight learning function
learncon	Conscience bias learning function
genFunction	Generate MATLAB function for simulating neural network
negdist	Negative distance weight function

compet	Competitive transfer function

Задание 1b. Изучить примеры справочной и демонстрационной системы MATLAB по созданию, обучению и работе с конкурирующим слоем.

Задание 2. Создать слой Кохонена для двух векторов входа, проанализировать его структурную схему и параметры вычислительной модели, произвести обучение сети и моделирование, выполнив следующие команды:

```
P = [.1 .8 .1 .9; .2 .9 .1 .8]; % для обучения
слоя;
net = newc([01;01],2);    % создание слоя;
gensim (net);          % структура слоя;
net =  train(net,P);   % обучение слоя;
w = net.Iw{1,1};       % веса после обучения;
b =net.b{1}; % смещение после обучения;
plot(P(1,:),P(2,:),'+k')
title('Inputs'),xlabel('P(1,:)'),ylabel('P(2,:)')
hold on
plot (w, 'or')
P1= [0.2:0.1:0.7; 0.2:0.1:0.7];
y = sim(net,P1)
yc = vec2ind(Y)
```

Задание 3. Создать слой Кохонена, который для 48 случайных векторов формирует 8 кластеров, выполнив следующие команды:

```
c = 8;                % – число кластеров;
n =6;                 % – число векторов в классе;
x = [-10 10; -5 5];   % – диапазон входов;
[r,q] = size(x);      % – r число строк; q – число
столбцов;
minU = min(x')'       % минимальные значения;
```

```
maxU = max(x')'      % максимальные значения;
v = rand(r,c)*((maxU - minU)*
ones(1,c))+minU*ones(1,c);
t = c*n              % число точек;
v = [v v v v v v];   % 48 двухэлементных векторов;
v =v+randn(r,t)*d;   % координаты точек;
P = v; % векторы с отклонениями (нормал. закон);
plot (P(1,:),P(2,:), '+k')
title('Inputs'),xlabel('P(1,:)'),ylabel('P(2,:)')
net =newc([-2 12; -1 6],8 0.1);
wo = net.IW{1,1}     % веса после инициализации;
bo = net.b{1} % смещения после инициализации;
co = exp(1)/b0 % начальная активность;
net.trainParam.epochs = 500; % обучение;
net =  train(net, P) ,
a = sim(net, P), ac = vec2ind(a)
net.IW{1} % веса после обучения;
bn = net.b{1} % смещения после обучения;
cn = exp(1)/bn % активность после обучения;
net = newc([-2 12; -1 6], 8, 0.1);
co = exp(1)./net.b{1} % начальная активность;
net.adaptParam.passes = 500;
[net,y,e] = adapt(net,mat2cell(p)) % адаптация;
a =   sim(net, P) % моделирование после
ac = vec2ind(a) % адаптации.
```

Задание 4. Построить график приращений вектора смещений и проанализировать пример **democ1**.

Задание 5. С помощью слоя Кохонена произвести кластеризацию оценок абитуриентов.

Лабораторная работа № 06-2

Исследование самоорганизующихся карт Кохонена

Цель работы: изучение архитектуры самоорганизующихся нейронных сетей в виде карт Кохонена и специальных функций для создания карты и её топологии, взвешивания, накопления, настройки весов (размещение нейронов), адаптации и обучения; ознакомление с демонстрационными примерами и их скриптами, а также приобретение навыков построения самоорганизующихся карт для решения задач кластеризации входных векторов.

Теоретические сведения

Самоорганизующаяся карта Кохонена – это однослойная нейронная сеть без смешения с конкурирующей функцией **compet**, имеющая определенную топологию размещения нейронов в **N**-мерном пространстве. В отличие от слоя Кохонена карта Кохонена после обучения поддерживает такое топологическое свойство, когда близким входным векторам соответствуют близко расположенные активные нейроны.

Первоначальная топология размещения нейронов в карте Кохонена формируется при создание карты с помощью функции **newsom**, одним из параметров которого является имя топологической функции **gridtop**, **nextop** или **randtop**, что соответствует размещению нейронов в узлах либо прямоугольной, либо гексагональной сетки, либо в узлах сетки со случайной топологией.

Расстояния между нейронами и векторами входов вычисляются с помощью следующих функций:

dist – евклидово расстояние **d=sqrt((pos$_i$-p$_j$).^2)**;

boxdist – максимальное координатное смещение **d=max(abs(pos$_i$-p$_j$))**;

mandist – расстояние суммарного координатного смещения **d=sum(abs(pos$_i$-p$_j$))**;

linkdist – расстояние связи

$$d_{ij} = \begin{cases} 1, dist(pos_i - p_j) \le 1; \% - \text{евклидово пространство;} \\ 2, \forall k, d_{ik_1} = d_{kj} = 1; \% - \text{один промежуточный;} \\ 3, \forall (k_1, k_2), d_i k_1 = d_{k_1 k_{21}} = d_{k_1 j} = 1; \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ N, \forall (k_1, k_2 \dots k_n), d_i k_1 = d_{k_1 k_2} = \dots = d_{knj} = 1; \\ S, \text{в остальных случаях} \end{cases}$$

Формирование самоорганизующейся карты Кохонена осуществляется функцией (для версий с R2010b)

```
net = selforgmap(dimensions, coverSteps,
initNeighbor, topologyFcn, distanceFcn),
```

dimensions	Row vector of dimension sizes (default = [8 8])
coverSteps	Number of training steps for initial covering of the input space (default = 100)
initNeighbor	Initial neighborhood size (default = 3)
topologyFcn	Layer topology function (default = 'hextop')
distanceFcn	Neuron distance function (default = 'linkdist')

и функцией (для версий ранее R2010b)

net=newsom(PR,[d_1,d_2,…],tfcn, dfcn, olr, osteps, tlr, tnd)),

где **Pr** – массив размера **R×2** минимальных значений векторов входа;

d_1, d_2…– число нейронов по *i*-й размерности карты. По умолчанию – двумерная карта с числом нейронов **5*8**;

tfcn – функция топологии карты, по умолчанию **nextop**;

dfcn – функция расстояния, по умолчанию **linkdist**;

olr – параметр скорости обучения на этапе размещения, по умолчанию **0.9**;

osteps – число циклов обучения на этапе подстройки, по умолчанию **1000**;

tlr – параметр скорости на этапе подстройки, по умолчанию 0.02;

tnd – размер окрестности на этапе подстройки, по умолчанию **1**.

Настройка карты Кохонена производится по каждому входному вектору независимо от того, применяется метод адаптации или метод обучения. В любом случае функция **learnsom** выполняет настройку карты нейронов.

Прежде всего определяется нейрон-победитель и корректируется его вектор весов и векторы соседних нейронов согласно соотношению

```
dw=lr*A2*(p' - w),
```

где **lr** – параметр скорости обучения, равный **olr** для этапа упорядочения нейронов и **tlr** для этапа подстройки;

A2 – массив соседства для нейронов, расположенных в окрестности нейрона-победителя ***i***:

$$A2(i,q)=\begin{cases}1, a(i,q)=1;\\ 0.5, a(j,q)=1 \,\&\, D(i,j)\le nd;\\ 0, \text{в остальных случаях.}\end{cases}$$

Здесь **a(i,q)** – элемент выхода нейронной сети;

D(i,j) – расстояние между нейронами i и j;

nd – размер окрестности нейрона-победителя.

Т.о., вес нейрона-победителя изменяется пропорционально половинному параметру скорости обучения, а веса соседних нейронов – пропорционально половинному значению этого параметра.

Весь процесс обучения карты Кохонена делится на два этапа:

1) этап упорядоченности векторов весовых коэффициентов в пространстве признаков;

2) этап подстройки весов нейронов по отношению к набору векторов входа.

На этапе упорядочения используется фиксированное количество шагов. Начальный размер окрестности назначается равным максимальному расстоянию между нейронами для выбранной топологии и затем уменьшается до величины, используемой на следующем этапе, и вычисляется по следующей формуле:

nd=1.00001+(max(d)-1)(1-s/S),

где **max(d)** – максимальное расстояние между нейронами; **s** – номер текущего шага, а **S** – количество циклов на этапе упорядочения.

Параметр скорости обучения изменяется по правилу

lr =tlr+(olr-tlr)(1-s/S).

На этапе подстройки, который продолжается в течение оставшейся части процедуры обучения, размер окрестности остается постоянным и равным

nd=tnd+0.00001,

а параметр скорости обучения изменяется по следующему правилу

lr= tlr*S/s.

Параметр скорости обучения продолжает уменьшаться, но очень медленно. Малое значение окрестности и медленное уменьшение параметра скорости обучения хорошо настраивают сеть при сохранении размещения, найденного на предыдущем этапе. Число шагов на этапе подстройки должно значительно превышать число шагов на этапе размещения. На этом этапе происходит тонкая настройка весов нейронов по отношению к набору векторов входов.

Нейроны карты Кохонена будут упорядочиваться так, чтобы при равномерной плотности векторов входа нейроны также были распределены равномерно. Если векторы входа распределены неравномерно, то и нейроны будут иметь тенденцию распределяться в соответствии с плотностью размещения векторов входа.

Таким образом, при обучении карты Кохонена решается не только задачи кластеризации входных векторов, но и выполняется частичная классификация.

Практические задания

В зависимости от версии MATLAB используйте для создания самоорганизующейся карты Кохонена функции **selforgmap** или **newsom**.

Задание 1a. Изучить работу следующих функций

nnstart	Neural network getting started GUI
view	View neural network
selforgmap	Self-organizing map

train	Train neural network
plotsomhits	Plot self-organizing map sample hits
plotsomnc	Plot self-organizing map neighbor connections
plotsomnd	Plot self-organizing map neighbor distances
plotsomplanes	Plot self-organizing map weight planes
plotsompos	Plot self-organizing map weight positions
plotsomtop	Plot self-organizing map topology
genFunction	Generate MATLAB function for simulating neural network

Задание 1b. Изучить примеры по созданию, обучению и работе с сетями Кохонена помощи *Neural Network Clustering Tool* (команда >> **nctool**).

Задание 1c. Изучить примеры справочной и демонстрационной системы MATLAB по созданию, обучению и работе с сетями Кохонена (**demosm1, demosm2)**.

Задание 2a. Рассчитать положение нейронов на четырехмерной сетке с прямоугольной топологией размера 5*4*3*2 и сделать попытку построить график расположения нейронов, выполнив следующие команды:

```
pos=gridtop(5,4,3,2)
```

% массив координат узлов размера N*S,

% где N – количество измерений, равное 4,

% а $S = \prod_{i=1}^{N} \dim i$ **– количество узлов сетки,**

% равное произведению числа нейронов по

% каждому измерению 5*4*3*2=120;

```
plotsom(pos)  % – вывод только трех размерностей.
```

Задание 2b. Рассчитать положение нейронов на двухмерной сетке с прямоугольной топологией размера **3*2** и построить график расположения нейронов, выполнив следующие команды:

```
pos =gridtop(2,3) % 0   1   0   1   0  1
                  % 0   0   1   1   2  2
plotsom(pos)  % – плоский график.
```

Задание 2c. Рассчитать положение нейронов на двухмерной сетке с прямоугольной топологией размера 3*2 и построить график расположения нейронов, выполнив следующие команды:

```
pos =gridtop(3,2)  % 0   1   0   1   0  1
                   % 0   0   1   1   2  2
plotsom(pos)            % плоский график.
```

Задание 2d. Рассчитать положение нейронов на трехмерной сетке с гексагональной топологией размера **5*4*3** с **60** нейронами и построить график их расположения, выполнив следующие команды:

```
pos =hextop(5,4,3) % массив размера 3*60;
plotsom(pos)  % построение графика.
```

Задание 2e. Сформировать гексагональную сетку размером **2*3** и построить график, выполнив команды:

```
pos=hextop(2,3) % 0  1.0  0.5  1.5  0  1.0
          % 0  0  0.866  0.866  1.7321  1.7321
plotsom(pos)
```

Задание 2f. Создать сетку размера **2*3** со случайным расположением узлов и построить график расположения нейронов, выполнив следующие действия:

```
pos=randtop(2,3) % 0.062 0.647 0.49 и т. д.
                 % 0 0.122 0.904 и т. д.
plotsom(pos)  % построение графика.
```

Задание 2g. Создать сетку размера 5*4*3 со случайным расположением узлов и построить график расположения нейронов, выполнив следующие действия:

```
pos=randtop(5,4,3)  % создание сетки ;
plotsom(pos)    % построение графика.
```

Задание 3a. Вычислить евклидово расстояние между нейронами сети с теологией, для которой задана матрица для 10 нейронов в трехмерном пространстве, выполнив следующие команды:

```
pos=rand(3,10) % случайная матрица координат;
d=dist(pos) % евклидово расстояние между нейронами
```

Задание 3b. Вычислить расстояние максимального смещения координат нейронов, размещенных в трехмерном пространстве, выполнив команды:

```
pos = rand(3,10); % случайная матрица координат
d = boxdist(pos)
% максимальное координатное смещение,
% которые для векторов x и – y вычисляются
% следующим образом: d = max(abs(x-y)).
```

Задание 3c. Вычислить суммарные координатные смещения для сетки из 10 нейронов в трехмерном пространстве со случайной матрицей координат, выполнив следующие действия:

```
pos = rand(3,10) % случайные координаты для 10
нейронов.
d = mandist(pos)
```

```
% суммарные координатные смещения,
% которые для векторов x и – y вычисляются
% следующим образом: d = sum(abs(x-y)).
```

Задание 3d. Вычислить расстояние связи между нейронами, распределёнными случайным образом в трехмерном пространстве, выполнив следующие команды:

```
pos = rand(3,10)
% массив случайных координат для 10 % нейронов.
d = linkdist(pos)
% расстояния связи между нейронами,
% определяемые следующим образом:
                % 0 1 1 1 1 1 1 1 1 1 1
                % 1 0 1 1 1 1 1 1 1 1 1
                % 1 1 0 1 1 1 1 1 1 1 1
                % 1 1 1 0 2 1 2 1 1 1 1
                % и т.д.
```

Задание 4. Создать гексагональную карту Кохонена размером **2x3**, проанализировать ее структурную схему и параметры вычислительной модели, произвести обучение карты и ее моделирование, а также построить необходимые графики, выполнив следующие команды:

```
net = newsom([0 2; 0 1],[2 3]); % – два входа.
net, net.layers{1}    % вычислительная модель.
P=[.1 .3 1.2 1.1 1.8 1.7 .1 .3 1.2 1.1 1.8 1.7;…
.2 .1 .3 .1 .3 .2 1.8 1.8 1.9 1.9 1.7 1.8];
plotsom(net.IW{1,1}, net.layers{1}.distances)
hold on
plot(P(1,:),P(2,:),'*k','MarkerSize',10)
net.trainParam.epochs = 2000;
net.trainParam.show = 100;
net = train(net,P);
```

```
plot(P(1,:),P(2,:),'*','MarkerSize',10)
hold on
plotsom(net.IW{1,1}, net.layers{1}.distances)
net.IW{1,1}
a = sim(net,[1.5;1]   % – a = (3,1) 1.
```

Задание 5. Создать одномерную карту Кохонена из **10** нейронов, обучить её на последовательности из **100** двухэлементных векторов единичной длины, распределенных равномерно в пределах от **0** до **90º**, построить график распределения векторов по кластерам и выполнить моделирование сети для одного вектора входа, выполнив следующие команды:

```
angels=0 : 0.5*pi/99 : 0.5*pi;
p=[sin(angels); cos(angels)];
plot(P(1, 1:10:end), P(2, 1:10:end), '*b')
hold on
net=newsom([0 1 ;0 1], [10]);
net.trainparam.epochs=2000;
net.trainparam.show=100;
[net,tr]=train(net,P);
plotsom(net.iw{1,1}, net.layers{1}.distances)
a=sim(net,[1;0])% – отнесен к 10-му кластеру.
figure(2)
a=sim(net,P)  % – моделирование на обучающем
              %   множестве и построение
              %   столбцовой диаграммы.
bar(sum(a'))
```

Задание 6. Создать двухмерную карту Кохонена размеров **5*6** с гексагональной топологией, обучить на последовательности из **1000** двухэлементных случайных векторов, элементы которых распределены по равномерному закону в интервале **[-1; 1],** и выполнить моделирование, используя команды:

```
P=rands(2,1000)
```

```
plot(P(1,:), P(2,:), '+')
net=newsom([-1 1;-1 1],[5,6]);
net.trainparam.epochs=1000;
net.trainparam.show=100;
net=train(net,P);
plotsom(net.IW{1,1}, net.layers{1}.distances)
a=sim(net, P); bar(sum(a'))      % столбцы;
a=sim(net, [0.5; 0.3], hold on
plot(0.5, 0.3, '*k').
```

Задание 7. Загрузите набор реальных данных, связанных с задачами кластеризации, из какого-либо хранилища реальных данных, постройте сеть Кохонена, решите задачу. Составьте отчет с подробным описанием каждого этапа. Отчет должен также содержать указатели на хранилище данных, указатель на данные, описание файлов данных, представление данных, код, графическую информацию. Пакет документов: фалы данных, m-файл(-ы), текстовый файл MS Word, файл Pdf, презентация в PowerPoint.

Задание 8. Задача восстановления форм трехмерных объектов по трехмерному облаку точек.

Облако точек (*point cloud*) — набор вершин в трёхмерной системе координат. Эти вершины, как правило, определяются координатами X, Y и Z и, как правило, предназначены для представления внешней поверхности объекта.

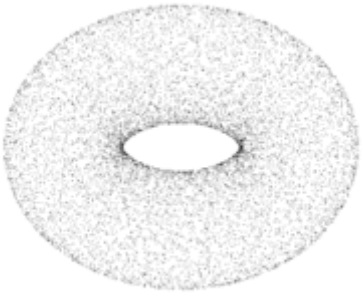

Рис.3. Облако точек поверхности тороида

См. также
http://conf.nsc.ru/files/conferences/ym2013/pdf/175192/ru/tex/abstracts_175192_ru.pdf

Лабораторная работа № 06-3

Исследование LVQ-сетей

Цель работы: изучение архитектуры самоорганизующихся нейронных сетей типа **LVQ** и специальных функций для их создания, настройки весов и обучения; ознакомление с демонстрационными примерами и их скриптами, а также приобретение навыков построения таких сетей для решения задач классификации входных векторов.

Теоретические сведения

Самоорганизующиеся нейронные сети типа **LVQ (Learning Vector Quantization)**, или сети для классификации входных векторов, состоят из двух слоёв без смещения. Входной слой является конкурирующим и используется для кластеризации векторов. Выходной слой является линейным с передаточной функцией **purelin** и обеспечивает соотнесение кластеров с целевыми классами, заданными пользователем. На каждый кластер во входном слое должен быть задан один нейрон, причём количество классов не должно быть больше количества кластеров.

Поскольку заранее известно, как кластеры первого слоя соотносятся с целевыми классами второго слоя, то это позволяет заранее задать элементы матрицы весов **LW** второго слоя. Однако чтобы найти правильный кластер для каждого вектора обучающего множества, необходимо выполнить процедуру обучения сети.

Для создания LVQ-сетей используется функция (для версий с R2010b)

```
net=lvqnet(hiddenSize,lvqLR,lvqLF)
```

с параметрами

hiddenSize	Size of hidden layer (default = 10)
lvqLR	LVQ learning rate (default = 0.01)
lvqLF	LVQ learning function (default = '**learnlv1**')

или функция (для версий ранее R2010b)

```
net = newlvg(PR, S1, PC, LR, LF),
```

где **PR** – массив размером **R×2** минимальных и максимальных значений R векторов входа;

S1 – число нейронов входного слоя (число кластеров);

PC – вектор размером **1×S2**, указывающий распределение по долям каждого класса из набора классов **S2**;

LR – параметр скорости настройки, по умолчанию 0.01;

LF – функция настройки параметров, по умолчанию **learnlv2**.

В результате выполнения функции **newlog** создаётся двухслойная сеть. Первый слой использует функции взвешивания **negdist**, накопления **netsum** и передаточную функцию **compet**. Второй слой использует функции взвешивания **dotprod**, накопления **netsum** и передаточную функцию **purelin**. Слои не имеют смещений. Веса первого слоя инициализируются с помощью функции **midpoint**; веса второго слоя устанавливаются так, чтобы каждому нейрону на выходе соответствовал единственный нейрон первого слоя. Адаптация и обучение выполняются с помощью функций **adaptwb**, которая модифицирует веса первого слоя, используя функцию настройки **learnlv1**.

LVQ – сеть обучается на основе множества пар – выход обучающей последовательности:

$$\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}.$$

Каждый целевой вектор имеет единственный элемент, равный единице, а остальные элементы равны нулю.

Предположим, что при задании вектора входа **p(q)** весовые коэффициенты нейрона **i*** входного слоя наиболее близки к вектору входа **p(q)** и нейрон **i*** выигрывает конкуренцию. Тогда конкурирующая функция **compet** выдаст единицу в качестве **i***-го элемента вектора выхода $\mathbf{a}^1$ первого слоя сети, причём все другие элементы $\mathbf{a}^1$ будут равны нулю. Во втором, линейном слое, произведение матри-

цы весов **LW×a**[1] выявляет некоторый столбец **LW**, в котором единичное значение указывает на класс **k***. Таким образом, сеть связывает вектор входа **p(q)** с классом **k***. Это назначение может оказаться либо правильным, либо ошибочным. Поэтому в процессе обучения необходимо откорректировать строку **i*** матрицы **IW** таким образом, чтобы приблизить её к вектору **p(q)**, если назначение правильное, и удалить от вектора **p(q)** если назначение неправильное:

i*IW(q) = i*IW(q-1)+LR(p(q)-i*IW(q-1)), $a^2_{k*} = t_{k*} = 1$;

i*IW(q) = i*IW(q-1)-LR(p(q)-i*IW(q-1)), $(a^2_{k*} = 1) \neq (t_{k*} = 0)$.

Это правило гарантирует, что при правильной классификации нейрон-победитель приближается к векторам входа данного класса, а при неправильной классификации удаляется от них. Оно различается функцией настройки весов слоя **LVQ**-сети **learnlv1**. Другая функция настройки весов **learnlv2** позволяет улучшить настройку параметров. Она производит корректировку двух весовых векторов, близких к входному. Два весовых вектора с евклидовыми расстояниям **di** и **dj** до вектора входа **p** считаются близкими, если выполняется условие

$$\min(di^*/dj^*, dj^*/di^*) > 0.5 \div 0.7.$$

Если при этом строка **i*** принадлежит к области в пространстве признаков, соответствующей требуемому классу, а строка **j*** не принадлежит, то корректировка весов производится следующим образом:

$$\begin{cases} \mathbf{i*IW(q) = i*IW(q-1) + lr(p(q)) - i*IW(q-1);} \\ \mathbf{j*IW(q) = i*IW(q-1) + lr(p(q)) - j*IW(q-1).} \end{cases}$$

Практические задания

В зависимости от версии MATLAB используйте для создания самоорганизующейся карты Кохонена функции **lvqnet** или **newlvq**.

Задание 1а. Изучить работу следующих функций

Function	Description

competlayer	Create a competitive layer.
learnk	Kohonen learning rule.
selforgmap	Create a self-organizing map.
learncon	Conscience bias learning function.
boxdist	Distance between two position vectors.
dist	Euclidean distance weight function.
linkdist	Link distance function.
mandist	Manhattan distance weight function.
gridtop	Gridtop layer topology function.
hextop	Hexagonal layer topology function.
randtop	Random layer topology function.
lvqnet	Create a learning vector quantization network.
learnlv1	LVQ1 weight learning function.
learnlv2	LVQ2 weight learning function.

Задание 1b. Изучить примеры справочной и демонстрационной системы MATLAB по созданию, обучению и работе с сетями Кохонена (**demolvq1**).

Задание 1. Создать нейронную **LVQ**-сеть для обучающей последовательности двухэлементных векторов, имеющих 4 нейрона во входном слое и 2 нейрона в выходном с распределением [0.6 0.4], проанализировать её структурную схему и значения параметров вычисленной модели, обучить сеть и промоделировать её на обучающей последовательности, выполнив следующие команды:

```
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 ];  % индексы клас-
сов;
T = ind2vec(Tc); % разряженная целевая матрица;
```

```
T = full(T); % полная целевая матрица;
net = newlvq(minmax(P), 4, [0.6 0.4]);
% параметры вычисли тельной модели
gensim(net); % структурная схема LVQ-сети;
net = train(net, P, T);
% обучение сети со значениями
%параметров по умолчанию;
Y = sim(net, P) % моделирование LVQ-сети;
Yc = vec2ind(Y) % – индексы классов, которые
                  %  получила сеть;
% Сравнить Yc и Tc.
```

Задание 2. Повторить первое задание для всевозможных векторов индексов **Tc** и выявить случаи несовпадения **Yc** и **Tc**, т. е. случаи неправильной классификации.

Задание 3. Создать нейронную **LVQ**-сеть с теми же параметрами, что и в первом задании, обучить сеть, промоделировать её, построить график распределения входных векторов по кластерам и разделяющую линию областей точек, принадлежащих разным классам, выполнив следующие команды:

```
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 ]; % индексы классов;
T = full(ind2vec(Tc));
net = newlvg(minmax(P), 4, [0.6 0.4]);
net.inputWeights{1, 1}
net.IW{1, 1} % веса входного слоя после иниц.;
net.LW{2,1} % веса выходного слоя после иниц.;
net.b{1}, net.b{2}
net.trainParam.epoch = 2000;
net.trainParam.show = 100;
net.trainParam.lr = 0.05;
net = train(net, P, T);
```

```
 net.IW{1, 1} % веса выходного слоя после обуч.;
 net.LW{2, 1} % веса выходного слоя после обуч.;
 I1 =find(Tc == 1); % вектор индексов 1-го клас-
са;
 I2 =find(Tc == 2); % вектор индексов 2-го клас-
са;
 axis([-4, 4, -3, 3]) % диапазоны для X и Y;
 P1 = P(: , I1) % векторы первого класса;
 P2 = P(: , I2)  % векторы второго класса;
 V = net.IW{1, 1} % веса выходного слоя;
 plot(P1(1, :), P1(2, :), '+ k'), hold on
 plot(P2(1, :), P2(2, :), '×b'), hold on
 plot(V(:, 1), V(:, 2), 'or')
 Y = sim(net, P)
 Yc=vec2ind(Y)
 % Построение разделяющей линии для классов:
 function P = mesh2P(x, y) % – начало М-функции;
 %Вычисление массива к-т 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]);
 % конец М-функции.
 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, I2), '+k'), hold on
 plot(P(1, I1),P(2, I2), '*b').
```

Задание 4. Создать нейронную **LVQ**-сеть для разбиения двухэлементных векторов на 8 кластеров и 4 класса, обучить сеть, промоделировать её, построить график распределения векторов по кластерам и разделяющую границу векторов, модифицируя команды 3-го задания.

Список литературы

1. MW_nnet.pdf. H.Demuth, M.Beale. Neural Network Toolbox. For use with MATLAB. 840 p.
2. Медведев В.С., Потемкин В.Г. Нейронные сети. MATLAB 6. М.: Диалог МИФИ, 2002. – 496с.
3. Голубева Л.Л., Малевич А.Э., Щеглова Н.Л. Курс лекций. Компьютерная математика. Числовой пакет Matlab. Минск, БГУ, 2007
4. Голубева Л.Л., Малевич А.Э., Щеглова Н.Л. Лабораторный практикум. Компьютерная математика. Числовой пакет Matlab. Минск, БГУ, 2008
5. Сивохин, А. В. Искусственные нейронные сети: Лаб. практикум / А. В. Сивохин, А. А. Лушников, С. В. Шибанов. – Пенза: Изд-во Пенз. гос. ун-та, 2004.