МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»

ПРАКТИКУМ ПО КУРСУ «НЕЙРОСЕТЕВЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ ИНФОРМАЦИИ»

Учебно-методическое пособие для вузов

Издательско-полиграфический центр
Воронежского государственного университета
2013

Утверждено научно-методическим советом факультета компьютерных наук 21 декабря 2012 г., протокол № 3

Составители: А.А. Сирота, Е.Ю. Митрофанова, М.А. Дрюченко

Рецензент к.т.н., доцент кафедры программного обеспечения и администрирования информационных систем факультета ПММ Ворониной И.Е.

Учебно-методическое пособие подготовлено преподавателями факультета ФКН Воронежского государственного университета.

Рекомендуется для выпускников факультета ФКН дневной и вечерней формы обучения.

Для специальностей: 230200 - Информационные системы, 230201 - Информационные системы и технологии, 230400 - Информационные системы и технологии

Лабораторная работа №1

«Изучение свойств простейшего искусственного персептрона»

Цель работы: изучение свойств простейших нейронных сетей.

1. Рабочая среда Matlab. Для реализации лабораторной работы необходим пакет Matlab версии не ниже 6.0 и начальные знания относительно языка системы.

Рабочая среда Matlab содержит:

- меню;
- панель инструментов с кнопками и раскрывающимся списком;
- окна с вкладками Workspace и Current Directory для просмотра переменных и установки текущего каталога;
- окно Command Window, служащее для ввода команд и вывода результата;
- окно Command History, предназначенное для просмотра и повторного выполнения ранее введенных команд (окно Command History может быть не пустым, если до этого пакет Matlab использовался);
- строка состояния с кнопкой Start.

Перед началом выполнения лабораторной работы в соответствующем разделе создается рабочая папка пользователя, в которой далее будут размещаться файлы, отражающие результаты выполнения лабораторных работ. После запуска системы данная папка устанавливается в окне «Current Directory» путем выбора из списка рабочих папок файловой системы. Для этого используется кнопка «…», открывающая стандартное окно проводника файловой системы, в котором можно изменить текущий дисковый накопитель или раздел диска, а также войти в нужную директорию. Работа оформляется в виде m- файла сценария (script file), который сохраняется в созданной рабочей папке.

Элементы рабочей среды следует располагать так, чтобы одновременно была возможность смотреть код сценария и окно Command Window, для просмотра результатов работы.

2. Формирование моделей нейронных сетей

2.1. Основные стандартные функции среды. Основные функции среды, используемые при выполнении работы, представлены в таблице 1.

Таблица 1 – Основные функции, используемые при выполнении работы

Функция	Синтаксис	Назначение
newp	**net = newp(PR,S,TF,LF)**, где PR — матрица минимальных и максимальных R входных элементов; S — количество нейронов (при создании одного нейрона S=1); TF — функция активации нейрона (transfer function); LF — имя функции обучения нейрона.	Формирует нейронную сеть – элементарный персептрон.
learnp	**[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)** **[db,LS] = learnp(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)** **info = learnp(code)** Детальное описание синтаксиса дано в документации (см. doc learnp).	Настраивает веса и смещение персептрона. Эта функция не имеет параметров настройки.
adapt	**[net, Y, E] = adapt(net, P, T),** где net — имя сети; P – матрица входных векторов сети, упакованных в виде столбцов матрицы; T — матрица (вектор) желаемого выхода. Функция возвращает параметры адаптированной сети net.adaptParam: net — измененная сеть,Y — выход сети, E — ошибки сети.	Адаптирует нейрон к условиям задачи в инкрементальном режиме (incremental learning), при этом веса сети корректируются каждый раз после предъявления одного входного вектора.
train	**[net,tr,Y,E,Pf,Af] = train(net,P,T,Pi,Ai,VV,TV)** Детальное описание синтаксиса дано в документации (см. doc train).	Обучает нейронную сеть в пакетном режиме (batch learning), при этом веса корректируются каждый раз после предъявления всей обучающей выборки. Каждая обучающая итерация называется эпохой (epoch).
sim	**Y = sim(net,P),** где net — сеть,Y — выходы сети, P — входы сети.	Выполняет моделирование работы нейронной сети.

Продолжение таблицы 1

init	**net = init(net),** где net – имя нейронной сети	Выполняет начальную инициализацию нейронной сети, используя функцию определяемую свойством net.initFcn и значения параметров, определяемые свойством net.initParam.

2.2. Структура данных, используемых при описании нейронных сетей. Структура данных нейронной сети дает ее полное описание. Обучение сети осуществляется в соответствии со следующими параметрами, значения которых либо устанавливаются пользователем, либо используются по умолчанию (таблица 2).

Таблица 2 –Структура данных описания нейронной сети

Структура данных	Комментарий
net.trainParam.epochs=100 (по умолч.)	Максимальное количество эпох обучения.
net.trainParam.goal = 0	Целевой значение ошибки.
net.trainParam.max_fail= 5	Максимальное значение количества эпох с увеличивающейся ошибкой перед остановом обучения при обучении сети в режиме валидации (с проверочной выборкой, validation set).
net.trainParam.min_grad= 1e-10	Минимальное значение градиента.
net.trainParam.show= 25	Количество эпох между точками, выводимыми на графике изменения ошибки обучения.
net.trainParam.time= inf	Максимальное время обучения в секундах.
Tr	Структура данных, содержащая значения о степени обученности НС в текущую эпоху.
tr.epochs	Номер эпохи.
tr.perf	Уровень ошибки обучения.
tr.vperf	Уровень ошибки по проверочной выборке.
tr.tperf	Уровень ошибки по тестовой выборке.

2.3. Активационные функции. Функция активации, или передаточная функция нейрона, преобразует каждый элемент n_i, вектора входа нейрона в элемент a_i вектора выхода $a_i = f(n_i)$. Ниже представлены основные типы этих функций (таблица 3).

Таблица 3 – Активационные функции

Функция	Выполняемое преобразование
hardlim(N)	Вычисляет пороговую функцию (1, если N > 0 и 0 – в противном случае)
logsig(N)	Вычисляет непрерывную униполярную функцию «логистический сигмоид»
tansig(N)	Вычисляет непрерывную биполярную функцию «гиперболический тангенс»
purelin(N)	Вычисляет линейную функцию на выходе нейрона

3. Изучение свойств простейшего искусственного нейрона

3.1. Создание простого двухвходового персептрона. Выполняются следующие действия

```
clear all;
%создадим персептрон с двуэлементным входом и одним нейроном; диапазон
%значений входа - [0 1] и [-2 2].
 net=newp([0 1;-2 2],1)
```

В результате в командном окне Matlab появятся результат

```
net =

   Neural Network object:

   architecture:

       numInputs: 1
       numLayers: 1
     biasConnect: [1]
    inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]

      numOutputs: 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
            biases: {1x1 cell} containing 1 bias
      inputWeights: {1x1 cell} containing 1 input weight
      layerWeights: {1x1 cell} containing no layer weights

    functions:

          adaptFcn: 'trains'
         divideFcn: (none)
       gradientFcn: 'calcgrad'
           initFcn: 'initlay'
         performFcn: 'mae'
          trainFcn: 'trainc'

    parameters:

        adaptParam: .passes
       divideParam: (none)
     gradientParam: (none)
         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 information)
```

3.2. Создание нейрона, выполняющего функцию логического «и» (с использованием для обучения функции adapt). Для создания нейрона, выполняющего функцию логического «и» необходимо создать простейшую нейронную сеть (см. п.3.1). После того как сформирована архитектура сети, должны быть заданы начальные значения весов и смещений, то есть сеть должна быть инициализирована при помощи функции init (таблица 1).

После того как сеть инициализирована необходимо задать входной вектор, содержащий те значения, на основе которых нейронная сеть будет обучаться. В качестве обучающих данных используются векторы P, содержащие все возможные комбинации 0 и 1 ([0;0] [0;1] [1;0] [1;1]) и заданные в виде массива ячеек (cell). После выполнения данных функции моделируется нейронная сеть (функция sim) для исследования и получения конечного результата.

```
clear all;
net=newp([0 1;-2 2],1);
%инициализация
net=init(net);
%входные векторы
P1={[0;0] [0;1] [1;0] [1;1]};
y1=sim(net,P1);
disp(y1);
```

В командном окне появится результат:

```
y1 =
    [1]    [1]    [1]    [1]
```

Полученный результат является неверным (что очевидно из таблицы истинности для логического «и»). Для того чтобы получить верный результат нейронную сеть необходимо обучить, использую не только входные значения, но и соответствующие им выходные значения, внесенные в целевой вектор T1, также заданный в виде массива ячеек. Каждая реализация процесса настройки с использованием обучающего множества называется проходом. Данный процесс можно выполнить при помощи специальной функции адаптации adapt. При каждом проходе данная функция использует обучающее множество, вычисляет выход, погрешность и выполняет подстройку параметров персептрона (весовых коэффициентов).

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

%входные векторы
P1={[0;0] [0;1] [1;0] [1;1]};
%требуемые целевые выходы
T1={0,0,0,1};

net.adaptParam.passes=20;
net=adapt(net,P1,T1);

%моделирование сети с входным сигналом P1
y1=sim(net,P1);
disp(y1);
```

В командном окне появится результат:

```
y1 =
    [0]    [0]    [0]    [1]
```

Данный результат свидетельствует о том, что обученная сеть правильно воспроизводит логическую функцию «и».

Для тестирования обученной нейронной сети создадим другой входной вектор и выполним моделирование сети с этими входными данными.

```
%входные векторы
P2={[0;0] [1;1]};
%моделирование сети с входным сигналом P2
y2=sim(net,P2);
disp(y2);
```

В командном окне появится результат:

```
y2 =
    [0]    [1]
```

3.3. Создание нейрона, выполняющего функцию логического «и» (с использованием для обучения функции train). При использовании процедуры train настройка сети выполняется в результате проходов всего обучающего множества следующим образом. В данном примере входные и выходные обучающие данные заданы в виде простых матриц и векторов.

```
clear all;
net=newp([0 1;-2 2],1);
net=init(net);
%матрица входных векторов
P2=[0 0 1 1;
    0 1 0 1];
% требуемые целевые выходы
T2=[0 0 0 1];
net.trainParam.epochs=20;
net=train(net,P2,T2);
y2=sim(net,P2);
disp(y2);
```

В командном окне появится результат:

```
y2 =
     0     0     0     1
```

Кроме того строится кривая изменения ошибки при обучении (рисунок 1)

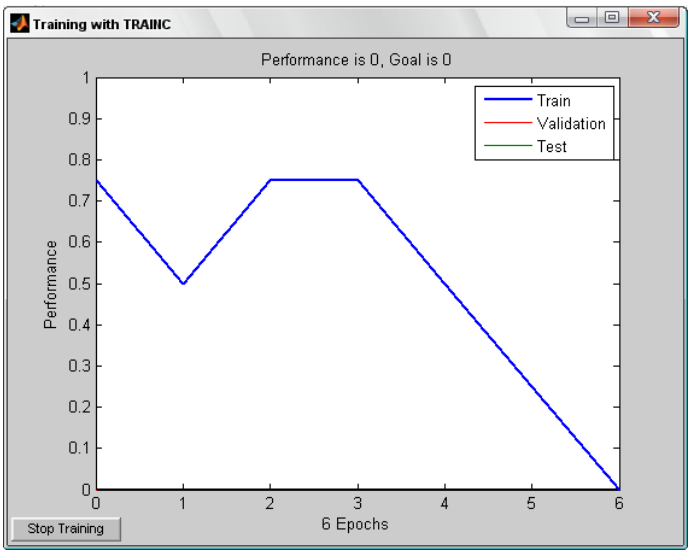


Рисунок 1 – Зависимость изменения ошибки обучения для логического «и» при использовании функции train от номера эпохи

Для тестирования обученной нейронной сети создадим новый входной вектор и смоделируем сеть с заданными входными данными.

```
%матрица входных векторов
P2=[0 1 0;
    0 1 0];
%моделирование сети с входным сигналом P2
y2=sim(net,P2);
disp(y2);
```

В командном окне появится результат:

```
y2 =
   0  1  0
```

4. Визуализация функций активации. Функция активации (активационная функция, функция возбуждения) – функция, вычисляющая выходной сигнал искусственного нейрона. Для ее визуализации используем стандартную функцию среды plot.

Функция hardlim. Простая кусочно-линейная функция. Если входное значение меньше порогового, то значение функции активации равно минимальному допустимому, иначе – максимально допустимому.

```
% hardlim
n = -2:0.01:2;
b = hardlim(n);
figure;
plot(n,b);
```

В результате получим (рисунок 2):

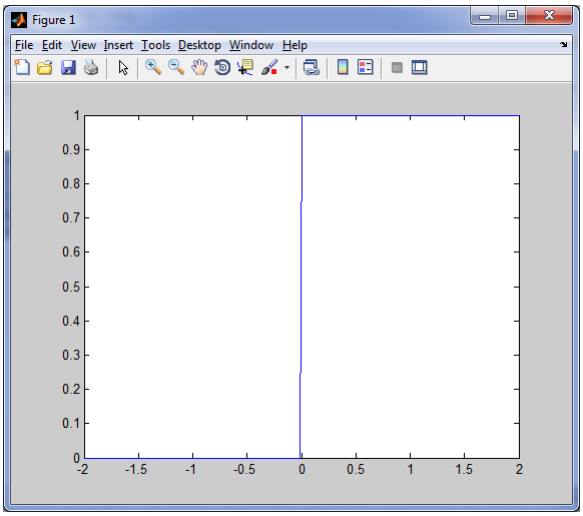


Рисунок 2 – Активационная функция hardlim

Функция logsig. Данная функция принадлежит к классу сигмоидальных функций, и ее аргумент может принимать любое значение в диапазоне от $-\infty$ до $+\infty$, а выход изменяется в диапазоне от 0 до 1. Благодаря свойству дифференцируемости (нет точек разрыва) эта функция часто используется в сетях с обучением на основе метода обратного распространения ошибки.

```
% logsig
n = -2:0.01:2;
b = logsig(n);
figure;
plot(n,b);
```

В результате получим (рисунок 3):

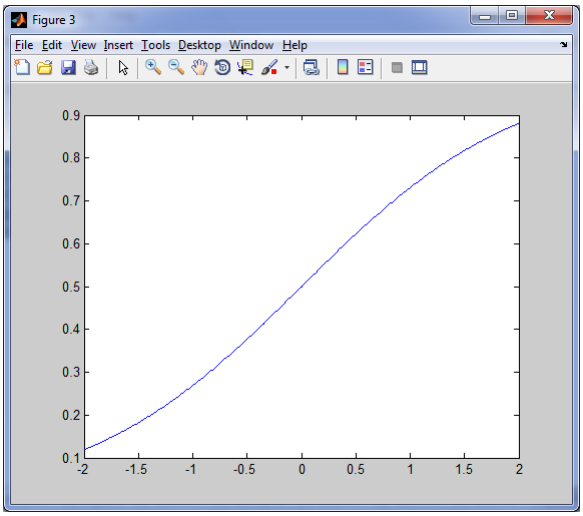


Рисунок 3 - Активационная функция logsig

Функция tansig. Сигмоидная функция в виде гиперболического тангенса имеет один входной аргумент возвращает выходные векторы со значениями в диапазоне от -1 до 1.

```
%tansig
n = -2:0.01:2;
b = tansig(n);
figure;
plot(n,b);
```

В результате получим (рисунок 4):

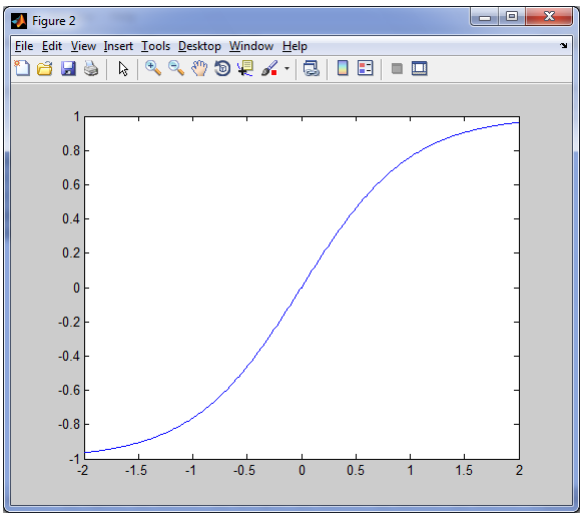


Рисунок 4- Активационная функция tansig

Функция purelin. Функция активации вычисляет выход нейрона как линейную функцию входного сигнала. Функция имеет один входной аргумент (матрицу входных векторов -столбцов) и возвращает выходные векторы без изменений.

```
%purelin
n =-2:0.01:2;
b = purelin(n);
figure;
plot(n,b);
```

В результате получим (рисунок 5):

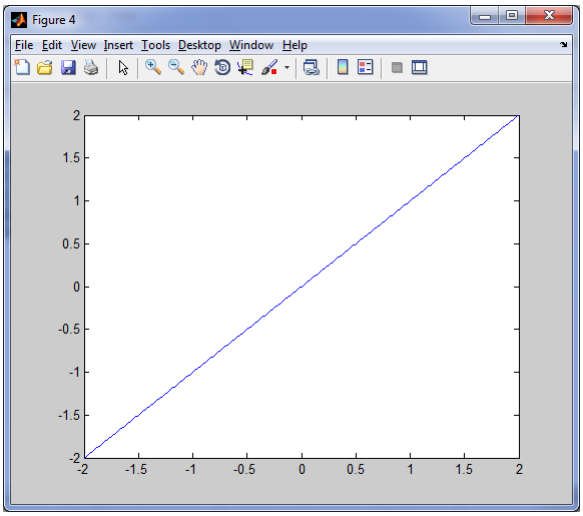


Рисунок 5- Активационная функция purelin

Задания для самостоятельной работы

1. Выполнить обучение нейрона выполнению функции логического «или» (с использованием функции adapt).
2. Провести экспериментальное исследование влияния значения параметра net.adaptParam.passes на результат.
3. Выполнить обучения нейрона выполнению функции логического «или» (с использованием функции train).
4. Провести экспериментальное исследование значения net.trainParam.epochs.
5. Выполнить попытку обучения нейрона выполнению функции исключающего «или» (с использованием функций adapt и train).

Лабораторная работа №2

«Нейросететевой классификатор данных с линейной разделяющей границей»

Цель работы: изучить возможности простейших нейронных сетей для классификации гауссовских случайных векторов.

1. Графическое отображение целевых векторов и построение линейной разделяющей линии. Для графического отображения принадлежности входных векторов разным классов образов используется функция plotpv, а для отображения разделяющей границы – функция plotpc (таблица 4).

Таблица 4 –Графическое отображение векторов и разделяющей границы

Функция	Синтаксис	Назначение
plotpv	**plotpv(P,T)**, где P – входные векторы данных; S – выходной целевой вектор, определяющий принадлежность входных векторов разным классам.	Графическое представление координат обучающих входных векторов и их принадлежности классам образов
plotpc	**plotpc(W,b)**, где W – матрица весов; b – вектор смещений	Строит разделяющую линию или плоскость

```
%матрица входных векторов
P3=[-0.5 -0.5  0.4  -0.2;
      -0.5  0.5 -0.4   1.0];
%целевой  (маркирующий) вектор
T3=[1 1 0 0];
plotpv(P3,T3);
```

В результате получим (рисунок 6):

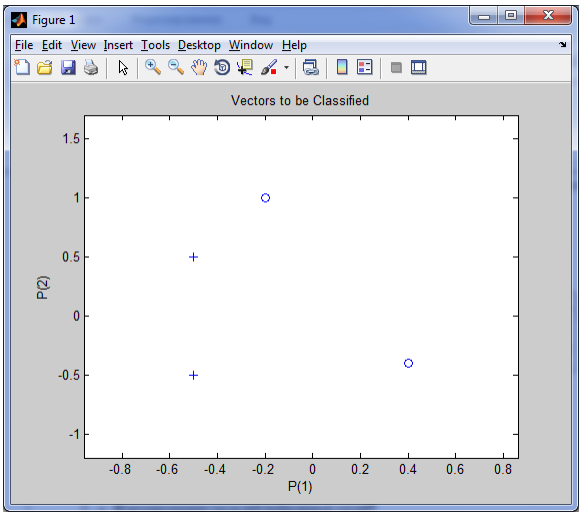


Рисунок 6 – Графическое представление координат входных векторов, отображаемых двумя классами маркеров («крестик» и «кружок»)

Таким образом, на рисунке 6 отображены два класса данных, в каждом из которых содержатся два входных вектора. Принадлежность входных векторов отображается маркерами («крестик» или «кружок») и определяется вектором T3, соответственно класс «крестиков» определяется значением 1 и класс «кружков» определяется значением 0.

Для построения линейной разделяющей линии необходимо обучить нейронную сеть с заданными входными и целевыми векторами. Далее на основе полученных матрицы весов net.IW{1} и вектора смещения net.b{1} строится линейная разделяющая линия, при помощи функции plotpc.

```
net=newp([-1 1;-1 1],1);
%обучение нейронной сети
net.adaptParam.passes=20;
net=adapt(net,P3,T3);
%получение управляющей структуры linehanle для изображения разделяющей
 %границы, формируемой нейронной сетью после обучения
linehandle=plotpc(net.IW{1},net.b{1});
%изображение разделяющей линии
plotpc(net.IW{1},net.b{1});
```

В результате получим графическое отображение разделяющей линии для классификации двух классов данных (рисунок 7).

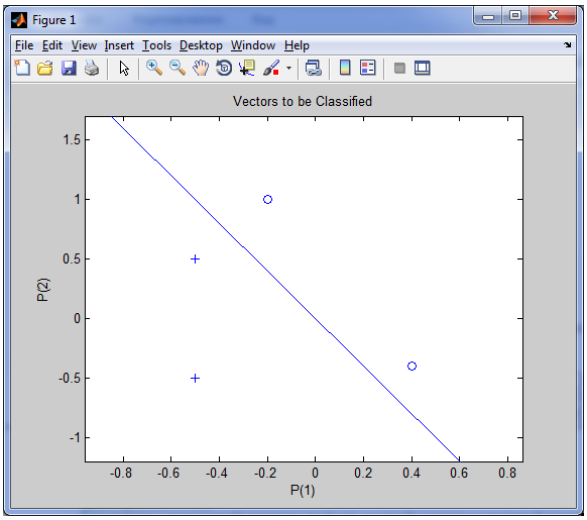


Рисунок 7 – Графическое отображение линейной разделяющей границы

Для тестирования нейросетевого классификатора определим новый вектор P, и определим к какому классу («крестиков» или «кружков») относится новый вектор.

```
%классификация нового вектора
P=[0.6;1.1];
a=sim(net,P);
plotpv(P,a);
ThePoint=findobj(gca,'type','line');%зададим цвет для нового маркера
set(ThePoint, 'Color','red');
hold on;%включение режима добавления графиков в графическом окне
plotpv(P3,T3);
plotpc(net.IW{1},net.b{1});
hold off;%отключение режима добавления графиков в графическом окне
```

В результате получим (рисунок 8):

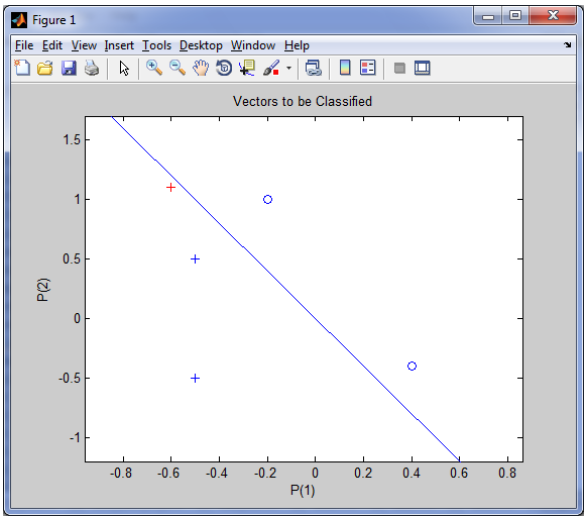


Рисунок 8 – Графическое отображение вектора, классифицируемого нейронной сетью

На рисунке 8 видно, что новый вектор с заданными координатами выделен красным цветом и правильно отнесен к классу «крестиков».

2. Обучение нейросетевого классификатора для разделения гауссовских случайных векторов. Решим аналогичную задачу для классификации многомерных гауссовских величин. Сформируем массив координат входных обучающих случайных векторов, значения которых получены при помощи функции randn (таблица 5).

Таблица 5 – Формирование массивов гауссовских случайных величин

Функция	Синтаксис	Назначение
randn	**X = randn(n)** , где n – размерность формируемого массива	Формирует массив размера n x n, элементами которого являются случайные величины, распределенные по гауссовскому закону с математическим ожиданием, равным 0, и дисперсией, равной 1.
randn	**X = randn(m, n)** , где m, n – размерность формируемого массива	Формирует массив размера m x n, элементами которого являются случайные величины, распределенные по гауссовскому закону с математическим ожиданием, равным 0, и дисперсией, равной 1.
randn	**X = randn(size(A))** , где size(A) – размерность формируемого массива	Формирует массив соразмерный с матрицей A, элементами которого являются случайные величины, распределенные по по гауссовскому закону с математическим ожиданием, равным 0, и дисперсией, равной 1.

```
n=20;%количество входных двумерных векторов в каждом классе образов
P41=randn(2,n);
%локализация  математического ожидания для векторов второго класса
A(1,1:n)=2;%задаем смещение относительно оси OX
A(2,1:n)=3;%задаем смещение относительно оси OY
P42=randn(2,n);
P4=[P41 P42+A];%матрица входных векторов
T41(1:n)=1;
T42(1:n)=0;
T4=[T41 T42];%матрица выходных (целевых) векторов
%графически отображаем координаты векторов первого и второго классов
plotpv(P4,T4)
```

В результате получим (рисунок 9):

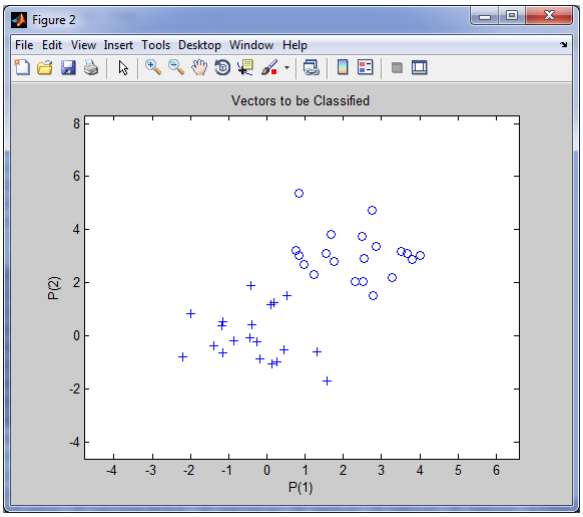


Рисунок 9 – Графическое представление входных векторов двух классов с использованием маркеров («крестик» и «кружок»)

Для иллюстрации процесса настройки и обучения нейросетевого классификатора будем обучать нейронную сеть пошагово, выполняя настройки нейронной сети на каждом шаге при помощи функции обучения adapt.

```
net=newp([-3 3;-5 5],1);
%получение управляющей структуры linehanle для отображения
%разделяющей границы
linehandle=plotpc(net.IW{1},net.b{1});
%изображение разделяющей линии
plotpc(net.IW{1},net.b{1});
cla;%очистка координатных осей
plotpv(P4,T4);
linehandle=plotpc(net.IW{1},net.b{1});
```

```
E=1;% задаем пороговое значение ошибки
net=init(net);
linehandle=plotpc(net.IW{1},net.b{1});
%органиация цикла пока E не равно 0
while(mse(E))',
    [net,Y,E]=adapt(net,P4,T4);
    linehandle=plotpc(net.IW{1},net.b{1},linehandle);
    plotpc(net.IW{1},net.b{1});
    drawnow;%очистка окна графиков
end;
```

В результате получим (рисунок 10):

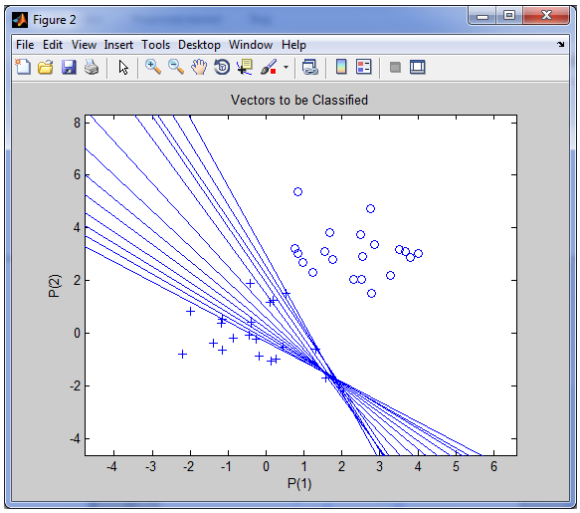


Рисунок 10 – Графическое представление пошагового построения линейной разделяющей границы

Для тестирования классификатора зададим новый вектор P, и при помощи обученной нейронной сети определим к какому классу («крестиков» или «кружков») относится новый вектор.

```
P=[0.6;1.1];
a=sim(net,P);
plotpv(P,a);
ThePoint=findobj(gca,'type','line');%зададим цвет для маркера нового вектора
set(ThePoint, 'Color','red');
hold on;%включение режима добавления графиков в графическом окне
plotpv(P4,T4);
plotpc(net.IW{1},net.b{1});
hold off;%отключение режима добавления графиков в графическом окне
```

В результате получим (рисунок 11):

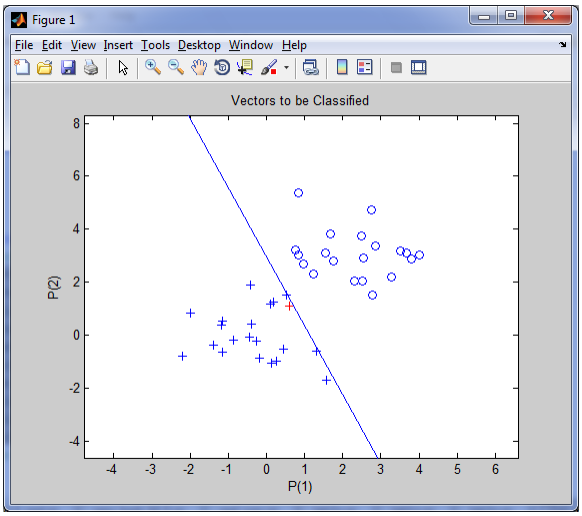


Рисунок 11 – Тестирование нейросетевого классификатора

На рисунке 11 видно, что новый вектор с заданными координатами выделен красным цветом и правильно отнесен к классу «крестиков».

Дополнительно проведем аналогичные исследования для количества новых входных векторов каждого класса, предъявляемых для тестирования, n=100 и вычислим ошибки первого и второго рода (ОС1 и ОС2), получаемые в процессе классификации новых векторов ранее созданной нейронной сетью.

Выдвинутая гипотеза о принадлежности нового вектора к тому или иному классу может быть правильной или неправильной, поэтому возникает необходимость ее проверки. В итоге статистической проверки гипотезы в двух случаях может быть принято неправильное решение, т. е. могут быть допущены ошибки двух родов.

Ошибка первого рода состоит в том, что будет отвергнута правильная гипотеза.

Ошибка второго рода состоит в том, что будет принята неправильная гипотеза.

```
n1=100;
P=randn(2,n1)
a=sim(net,P);
figure;%новое графическое окно
plotpv(P,a);
ThePoint=findobj(gca,'type','line');
set(ThePoint, 'Color','red');
hold on;%включение режима добавления графиков
%в графическом окне
plotpv(P4,T4);
```

```
plotpc(net.IW{1},net.b{1});
hold off;%отключение режима добавления графиков
%в графическом окне
OC1=1-sum(a)/n1;
disp(OC1);
```

В результате получим (рисунок 12), иллюстрирующий процесс возникновения ошибок первого рода.

```
OC1 =
    0.0300
```

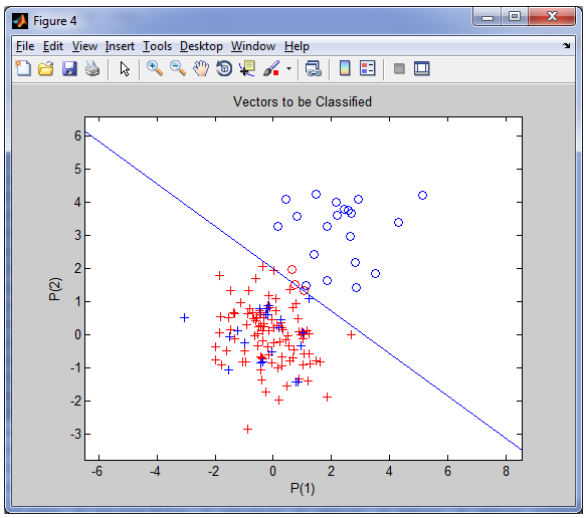


Рисунок 12 – Тестирование нейросетевого классификатора для n=100

На рисунке 12 видно, что новые входные вектора с заданными координатами выделены красным цветом, часть из которых правильно отнесена к классу «крестиков», а другая часть неправильно отнесена к классу «кружочков»), чем и иллюстрируется процесс возникновения ошибок первого рода.

Проведем аналогичное тестирование относительно класса «кружочков».

```
A1(1,1:n1)=3;
A1(2,1:n1)=3;
P1=randn(2,n1)+A1;
a11=sim(net,P1);
%включение режима добавления графиков
figure;
```

```
hold on;%включение режима добавления графиков
plotpv(P1,a11);
ThePoint=findobj(gca,'type','line');
set(ThePoint, 'Color','red');
hold on;
%в графическом окне
plotpv(P4,T4);
plotpc(net.IW{1},net.b{1});
hold off;%отключение режима добавления графиков
%в графическом окне
OC2=sum(a11)/n1;
disp(OC2);
```

В результате получим (рисунок 13), иллюстрирующий процесс возникновения ошибок второго рода.

```
OC2 =
     0
```

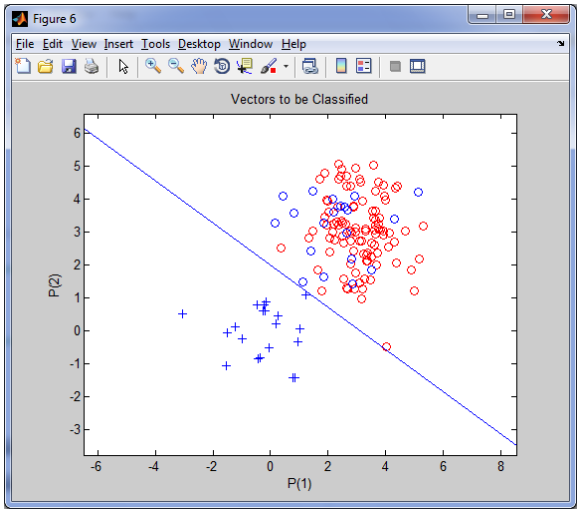
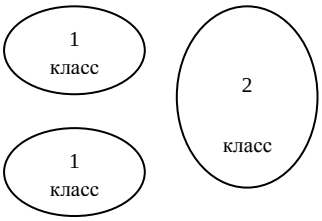


Рисунок 13 – Тестирование нейросетевого классификатора для n=100

На рисунке 13 видно, что новые входные вектора с заданными координатами выделены красным цветом правильно отнесена к классу «кружочков», при этом ни один из векторов не отнесен к классу «крестиков», то есть ошибка второго рода равна нулю.

Задания для самостоятельной работы

1. Провести обучение нейросетевого классификатора для разделения гауссовских случайных векторов, представленных следующим образом



2. Оценить ошибки первого и второго рода, и проиллюстрировать процесс их возникновения.

3. Построить график зависимости суммарной ошибки первого и второго рода от величины смещения центров кластеров.

4. Построить график зависимости суммарной ошибки первого и второго рода от объема обучающей выборки.

Лабораторная работа №3

«Нейросетевой классификатор данный с нелинейной разделяющей границей»

Цель: изучить возможности нейронных сетей для классификации гауссовских величин нелинейной разделяющей линией.

1. Построение нелинейной разделяющей линии. В лабораторная работе №2 мы рассмотрели построение линейной разделяющей линии, при помощи нейронной сети. Рассмотрим задачу классификации данных, в том случае когда области векторов двух классов располагается таким образом, что невозможно становится невозможным использование линейной разделяющей линии (рисунок 1).

```
clear all;
n=100;
ds=4;%смещение относительно осей координат
P41=randn(2,n);
P42=randn(2,n)+ds;
a43(1,1:n)=ds;
a43(2,1:n)=0;
P43=randn(2,n)+a43;
a44(1,1:n)=0;
a44(2,1:n)=ds;
P44=randn(2,n)+a44;
P4=[P41 P42 P43 P44];%входной вектор
T41(1:2*n)=1;
T42(1:2*n)=0;
T4=[T41 T42]; %целевой вектор
figure;%графическое окно
plotpv(P4,T4);%отображение маркеров
```

В результате получим (рисунок 14):

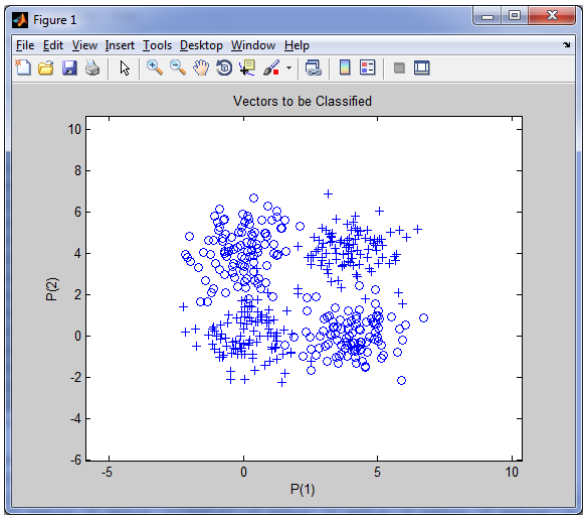


Рисунок 14 – Графическое отображение входного вектора

Для разделения гауссовских величин при помощи нелинейного классификатора необходимо использовать двухслойную нейронную сеть. Для создания двухслойной нейронной сети используется функция newff.

Таблица 6 – Создание многослойного персептрона

Функция	Синтаксис	Назначение
newff	**net = newff(PR, [S1 S2...SNl], {TF1 TF2...TFNl}, BTF, LF, PF)**, где PR - матрица минимальных и максимальных значений строк входной матрицы с размерностью. Для получения матрицы PR можно использовать функцию minmax;S_i – количество нейронов в i – м слое, N1 – количество слоев;TF_i - функция активации i - го слоя, по умолчанию = 'tansig';BTF – обучающая функция обратного распространения, по умолчанию='trainlm';BLF – алгоритм подстройки весов и смещений (обучающий алгоритм), по умолчанию = 'learngdm';PF - функция оценки функционирования сети, по умолчанию = 'mse'.	Формирует многослойную нейронную сеть.

```
toutn=0.1+(0.9-0.1)*T4;%целевой выход
bpn=newff(minmax(P4),[5 1],{'logsig' 'logsig'},'trainrp');
bpn=init(bpn);
bpn.trainParam.epochs=1000;
bpn=train(bpn,P4,toutn)
```

В результате получим

```
bpn =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

       numOutputs: 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
            biases: {2x1 cell} containing 2 biases
      inputWeights: {2x1 cell} containing 1 input weight
      layerWeights: {2x2 cell} containing 1 layer weight

    functions:

          adaptFcn: 'trains'
         divideFcn: (none)
       gradientFcn: 'gdefaults'
           initFcn: 'initlay'
        performFcn: 'mse'
          plotFcns: {'plotperform','plottrainstate','plotregression'}
          trainFcn: 'trainrp'

    parameters:
```

```
      adaptParam: .passes
     divideParam: (none)
   gradientParam: (none)
       initParam: (none)
    performParam: (none)
      trainParam: .show, .showWindow, .showCommandLine, .epochs,
                  .time, .goal, .max_fail, .min_grad,
                  .delt_inc, .delt_dec, .delta0, .deltamax

  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:

            name: '
```

Для тестирования нейросетевого классификатора определим новый вектор P, и определим к какому классу («крестиков» или «ноликов») он относится. Кроме того посчитаем ошибки первого и второго рода для данной задачи, аналогичным образом, что и в лабораторной работе №2.

```
cla;
n1=100;
P1=randn(2,n1);
P2=randn(2,n1)+ds;
P=[P1 P2];%входной вектор
a=sim(bpn,P);%моделирование нейронной сети
for i=1:2*n1,
   if a(i)>=0.5 a(i)=1; else a(i)=0; end; % округление полученных результатов
end;
figure;
plotpv(P,a);%отображение маркеров
OC1=1-sum(a)/(2*n1) ;
disp(OC1);
```

В результате получим

```
OC1 =
    0.0700
```

В результате получим (рисунок 16), иллюстрирующий процесс возникновения ошибок первого рода.

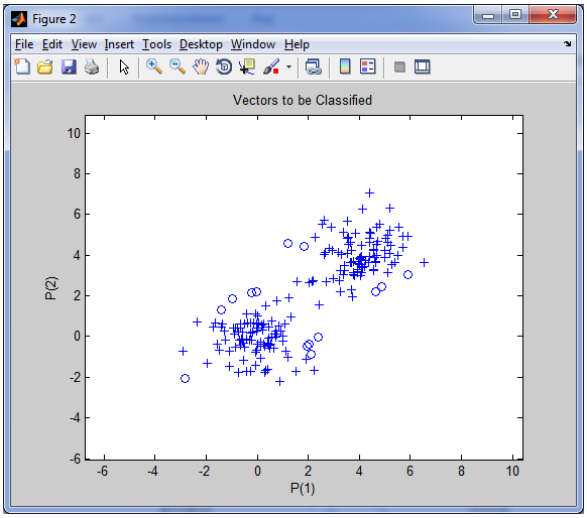


Рисунок 16 – Тестирование нейросетевого классификатора с нелинейной разделяющей линией

На рисунке 16 на основе классификатора, каждая из точек входного вектора правильно отнесена к классу «крестиков», и ошибочно отнесена к классу «кружочков», что иллюстрирует возникновение ошибки первого рода.

```
a3(1,1:n1)=ds;
a3(2,1:n1)=0;
P3=randn(2,n1)+a3;
a4(1,1:n1)=0;
a4(2,1:n1)=ds;
P4=randn(2,n1)+a4;
P=[P3 P4];
b=sim(bpn,P);
for i=1:2*n1,
   if b(i)>0.5 b(i)=1; else b(i)=0; end;
end;
figure;
plotpv(P,b);
OC2=sum(b)/(2*n1);
disp(OC2);
```

В результате получим (рисунок 17):

```
OC2 =

    0.0400
```

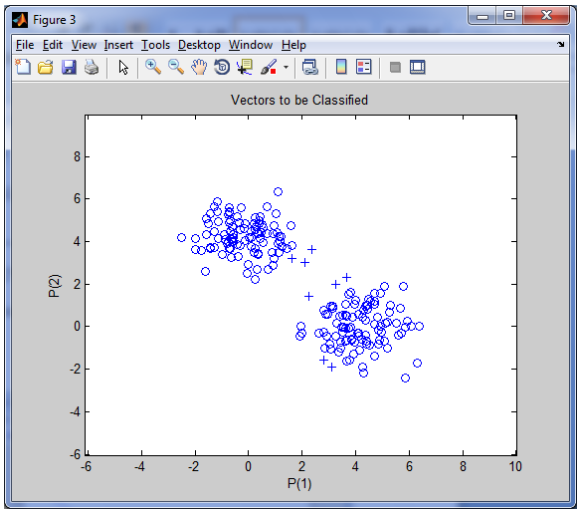
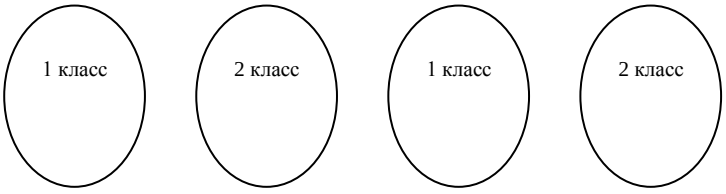


Рисунок 17 – Тестирование нейросетевого классификатора

На рисунке 17 на основе классификатора, каждая из точек входного вектора правильно отнесена к классу «кружочков», и ошибочно отнесена к классу «крестиков», что иллюстрирует возникновение ошибки второго рода.

Задания для самостоятельной работы

1. Провести обучение нейросетевого классификатора для разделения гауссовских случайных векторов, представленных следующим образом.



2. Оценить ошибки первого и второго рода, и проиллюстрировать процесс их возникновения.
3. Построить график зависимости суммарной ошибки первого и второго рода от величины смещения центров кластеров.

4. Построить график зависимости суммарной ошибки первого и второго рода от объема обучающей выборки.

Лабораторная работа №4

«Решения алгебраических уравнений с использование многослойного персептрона»

Цель: изучить возможности нейронных сетей для решения алгебраических уравнений на основе теоремы Виета.

1. **Теорема Виета.** Если $x_1, x_2, \ldots, x_n$ — корни многочлена $x^n + a_1 x^{n-1} + a_2 x^{n-1} + \ldots + a_n$, то коэффициенты $a_1, a_2, \ldots, a_n$ выражаются в виде симметрических многочленов от корней

$$\begin{cases} a_1 = -(x_1 + x_2 + \ldots + x_n) \\ a_2 = x_1x_2 + x_1x_3 + \ldots + x_1x_n + x_2x_3 + \ldots + x_{n-1}x_n \\ a_3 = -(x_1x_2x_3 + x_1x_2x_4 + \ldots + x_{n-2}x_{n-1}x_n) \\ \quad \ldots \\ a_{n-1} = (-1)^{n-1}(x_1x_2 \ldots x_{n-1} + x_1x_2 \ldots x_{n-2}x_n + \ldots + x_2x_3 \ldots x_n) \\ a_{n-1} = (-1)^n x_1x_2 \ldots x_n \end{cases}$$

Запишем теорему Виетта для частного случая кубического уравнения. Если x_1, x_2, x_3 — корни кубического уравнения $p(x) = ax^3 + bx^2 + cx + d = 0$, то

$$\begin{cases} x_1 + x_2 + x_3 = -\dfrac{b}{a} \\ x_1x_2 + x_1x_3 + x_2x_3 = \dfrac{c}{a} \\ x_1x_2x_3 = -\dfrac{d}{a} \end{cases}$$

2. Решение кубического уравнения использованием многослойного персептрона. Сформируем обучающие массив коэффициентов A, B, C и массив корней x_1, x_2, x_3 кубического уравнения. Для формирования массива корней x_1, x_2, x_3 кубического уравнения используем фиксированный интервал с заданным шагом дискретизации, и для каждой комбинации корней формируем массив коэффициентов многочлена A, B, C.

```
clear all;
mi=-10; ma=10; %диапазон значений
delt=5;%шаг дискретизации
intrvl=mi:delt:ma;
cnt=1;
for x1=intrvl,
```

```
    for x2=mi:delt:ma,
      for x3=mi:delt:ma,
        A(cnt)=-(x1+x2+x3);
        B(cnt)=x2*x3+x1*x3+x2*x1;
        C(cnt)=-x1*x2*x3;
        X(:,cnt)=[x1; x2; x3];
        cnt=cnt+1;
      end;
    end;
end;
P=[A;B;C];
```

Следует отметить, что при таком способе формирования массива корней будут повторяющиеся наборы корней, что может привести к возникновению неоднозначных ситуаций при обучении нейронной сети. К примеру, набор корней уравнения $x_1 = 5, x_2 = 10, x_3 = -5$ и $x_1 = 10, x_2 = 5, x_3 = -5$ являются идентичными. Поэтому следует изменить схему заполнений массива корней для выполнения следующего условия $x_1 \le x_2 \le x_3$.

```
clear all;
mi=-10; ma=10; %диапазон значений
delt=5;%шаг дискретизации
intrvl=mi:delt:ma;
cnt=1;
for x1=intrvl,
    for x2=x1:delt:ma,
      for x3=x2:delt:ma,
        A(cnt)=-(x1+x2+x3);
        B(cnt)=x2*x3+x1*x3+x2*x1;
        C(cnt)=-x1*x2*x3;
        X(:,cnt)=[x1; x2; x3];%массив корней
        cnt=cnt+1;
      end;
    end;
end;
P=[A;B;C];%массив коэффициентов
```

Для работы с нейронной сети необходимо нормировать массив корней и массив коэффициентов при помощи стандартных функций Matlab (таблица 7).

```
Xn=(X-mi)*0.8/(ma-mi)+0.1;
[Pn,Pmin,Pmax]=premnmx(P);
```

Таблица 7—Функции нормировки векторов

Функция	Синтаксис	Назначение
premnmx	**[pn,minp,maxp] = premnmx(p)**, где P - матрица входных векторов; MINP - вектор минимальных значений для каждого P; MAXP - вектор максимальных значений для каждого P	Выполняет предварительную обработку данных для тренировки сети посредством нормирования входов и эталонов так, чтобы они находились в диапазоне от –1 до +1
tramnmx	**[pn] = tramnmx(p,minp,maxp),** где P -матрица входных векторов; MINP - вектор минимумов для каждого входа; MAXP - вектор максимумов для каждого входа.	Преобразует входы с использованием максимумов и минимумов, которые были предварительно вычислены с помощью функции premnmx. Использование этой функции необходимо в случае, когда тренировка сети производилась с помощью данных, нормированных с использованием функции PREMNMX. Необходимо, чтобы все последующие входные значения сети были преобразованы с использованием той же нормировки.

Создадим и обучим нейронную сеть для решения кубического уравнения.

```
bpn=newff(minmax(Pn),[200 3],{'logsig' 'logsig'},'trainrp');
bpn.trainParam.epochs=500;
bpn=train(bpn,Pn,Xn);
```

Для тестирования нейронной сети сформируем случайным образом заполненный массив корней.

```
kk=10;
for cnt=1:kk,
```

```
x1=mi+rand*(ma-mi);
x2=x1+rand*(ma-x1);
x3=x2+rand*(ma-x2);
A1(cnt)=-(x1+x2+x3);
B1(cnt)=x2*x3+x1*x3+x2*x1;
C1(cnt)=-x1*x2*x3;
X1(:,cnt)=[x1; x2; x3];
end;
P1=[A1;B1;C1];
P1n=tramnmx(P1,Pmin,Pmax);
X1n=(X1-mi)*0.8/(ma-mi)+0.1;
y=sim(bpn,X1n);
mse(X1n-y)
```

В результате получим

```
ans =
   0.0203
```

Задания для самостоятельной работы

1. Провести тестирование нейронной сети с заданным смещением для диапазона значений корней полинома 3-й степени

```
mi=-10; ma=10; %диапазон значений
delt=5;%шаг дискретизации
smesh=5;
intrvl=mi+smesh:delt:ma+smesh;
```

2. Провести переобучение нейронной сети с уменьшенным числом нейронов в скрытом слое и проанализировать полученный результат.
3. Написать программу для решения полиномов 4-й степени.
4. Построить график зависимости среднеквадратичной ошибки от количества нейронов в скрытом слое

Лабораторная работа №5

«Сеть Хопфилда и ее использования как ассоциативной памяти»

Цель: Изучить возможности сети Хопфилда как структуры для создания ассоциативной памяти.

1. **Сеть Хопфилда.** Ассоциативная память это память с адресацией по содержанию, т.е. выборка и запись в ячейки такой памяти выполняется в зависимости от их содержимого. Ассоциативную память можно успешно реализовать как нейронную сеть, которая должна восстанавливать предъявленный слегка измененный образ, например, с наложенным шумом или содержащий лишь важную часть исходного образа. На основе таких систем можно создавать базы знаний, эффективный поиск в которых осуществляется не по ключу, а на основе предъявленного фрагмента требуемого образа. Ассоциативную память будем строить на основе сети Хопфилда, которая представляет собой однослойную нейронную сеть с обратными связями.

2. **Использование сети Хопфилда как ассоциативной памяти.** Для иллюстрации работы сети Хопфилда создадим бинарные изображения (буквы A,B,C) и отобразим при помощи функции imagesc, которая интерпретирует матрицу как прямоугольное изображение. Для создания сети Хопфилда используется функция newhop (таблица 8).

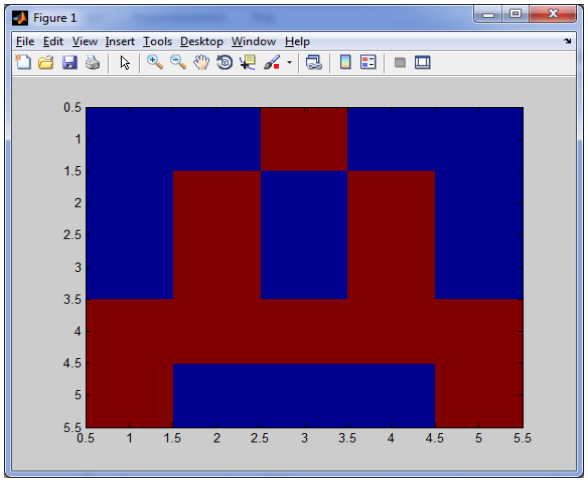
Таблица 8 - Функции создание сети Хопфилда и отображения входных векторов

Функция	Синтаксис	Назначение
imagesc	**imagesc(A)**, где A - матрица входных векторов	Графическое отображения матрицы
newhop	**newhop(T)**, где T – матрица входных векторов	Создание сети Хопфилда

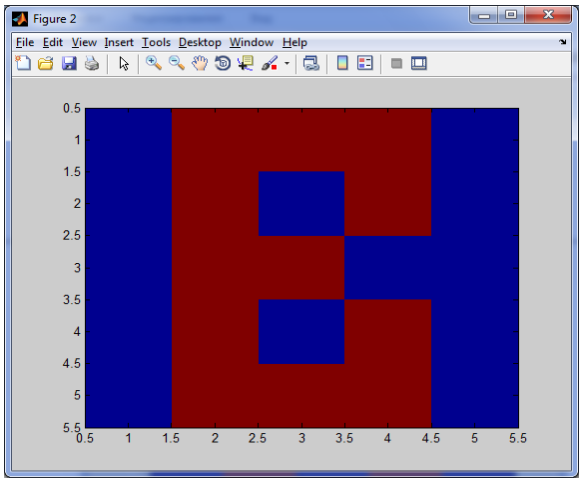
```
clear all;
Isize=5;%размер обрабатываемых изображений
IA=[0 0 1 0 0; %условное бинарное отображение буквы A
    0 1 0 1 0;
    0 1 0 1 0;
   1 1 1 1 1;
   1 0 0 0 1];
```

```
figure(1);
imagesc(IA);
IB=[0 1 1 1 0; %условное бинарное отображение буквы B
    0 1 0 1 0;
    0 1 1 0 0;
    0 1 0 1 0;
    0 1 1 1 0];
figure(2);
imagesc(IB);
IC=[0 1 1 1 0; %условное бинарное отображение буквы C
    0 1 0 0 0;
    0 1 0 0 0;
    0 1 0 0 0;
    0 1 1 1 0];
figure(3);
imagesc(IC);
```

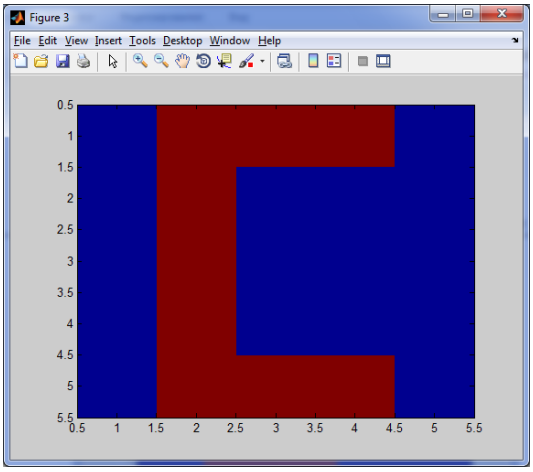
В результате получим следующие рисунки (рисунок 18):



а)



б)



с)

Рисунок 18 – Графическое отображение бинарных изображений

Для удобства работы с изображениями выполним следующее преобразование исходных матриц

```
for i=1:Isize,
    for j=1:Isize,
       if IA(i,j)==0 IA(i,j)=-1; end;
       if IB(i,j)==0 IB(i,j)=-1; end;
       if IC(i,j)==0 IC(i,j)=-1; end;
    end;
end;
```

Создадим сеть Хопфилда, настроенную на запоминание данных изображений.

```
%развертка матриц в векторы
cA=IA(:);
cB=IB(:);
cC=IC(:);
T=[cA cB cC]
net=newhop(T);%создание сети  Хопфилда
```

Для тестировании работы сети Хопфилда как ассоциативной памяти исказим исходное изображение буквы «А» с заданной вероятностью и отобразим полученный результат

```
po=0.1;%вероятность искажения единичного элемента (пикселя) изображения
IA1=IA;
for i=1:Isize,
    for j=1:Isize,
       x=rand;
       if x<po IA1(i,j)=-IA(i,j); end
    end;
end;
cA1=IA1(:);
Ai=cA1;
figure(4);
imagesc(reshape(Ai,[5 5]));
```

В результате получим (рисунок 19):

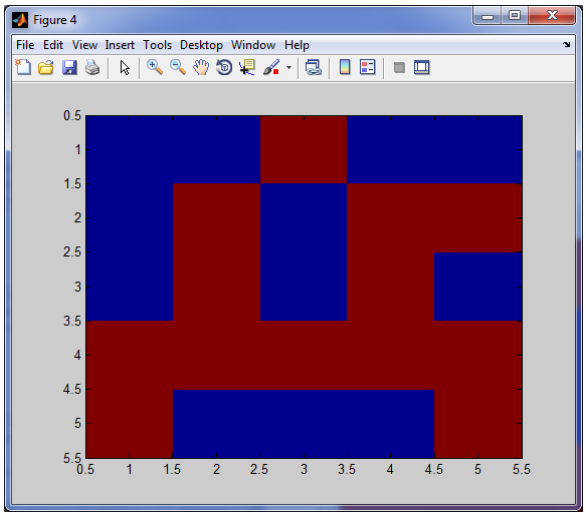


Рисунок 19– Искаженное изображение с заданной вероятностью

Теперь при помощи сети Хопфилда восстановим исходное изображение

```
[Y,Pf,Af]=sim(net, 1, [], Ai);
Yc=round(Y)
figure(8);
imagesc(reshape(Yc,[5 5]));
```

В результате получим (рисунок 20):

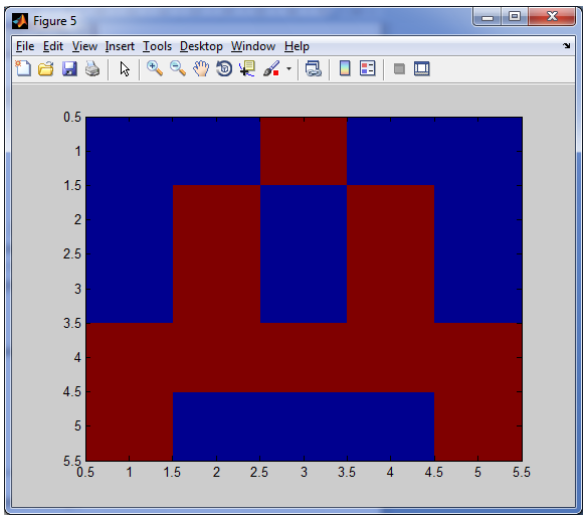


Рисунок 20 – Восстановленное изображение

Задания для самостоятельной работы

1. Выполнить тестирование для букв «B» и «C».

2. Провести исследование влияние вероятности искажения пикселя на качество восстановления изображения.

3. Провести моделирование сети Хопфилда и повторить п.п.1,2 для случая большей размерности изображений.

4. Провести исследование влияние вероятности искажения пикселя на качество восстановления изображенияв зависимости от размерности изображения.

Лабораторная работа №6

«Сеть Кохонена и ее применение»

Цель: изучить возможности сети Кохонена и рассмотреть возможности ее применения.

1. Сеть Кохонена. Нейронные сети Кохонена - класс нейронных сетей, основным элементом которых является *слой Кохонена*. Слой Кохонена состоит из адаптивных линейных сумматоров («линейных формальных нейронов»). Как правило, выходные сигналы слоя Кохонена обрабатываются по правилу «победитель забирает всё»: наибольший сигнал превращается в единичный, остальные обращаются в нуль. По способам настройки входных весов сумматоров и по решаемым задачам различают много разновидностей сетей Кохонена.

Рассмотрим самоорганизующиеся карты Кохонена. Самоорганизующаяся карта Кохонена (англ. *Self-organizing map* — SOM) —нейронная сеть с обучением без учителя на основе реализации конкурентных механизмов самоорганизации, выполняющая задачу визуализации и кластеризации данных.

Самоорганизующаяся карта состоит из компонент, называемых узлами или нейронами. Их количество задаётся аналитиком. Каждый из узлов описывается двумя векторами. Первый - вектор веса, имеющий такую же размерность, что и входные данные. Второй - вектор, представляющий собой координаты узла на карте. Обычно узлы располагают в вершинах регулярной решётки с квадратными или шестиугольными ячейками.

Изначально известна размерность входных данных, по ней некоторым образом строится первоначальный вариант карты. В процессе обучения векторы веса узлов приближаются к входным данным. Для каждого наблюдения (семпла) выбирается наиболее похожий по вектору веса узел, и значение его вектора веса приближается к наблюдению. Также к наблюдению приближаются векторы веса нескольких узлов, расположенных рядом, таким образом если в множестве входных данных два наблюдения были схожи, на карте им будут соответствовать близкие узлы. Циклический процесс обучения, перебирающий входные данные, заканчивается по достижении картой допустимой (заранее заданной аналитиком) погрешности, или по совершении заданного количества итераций.

2. Примеры применения сети Кохонена. Определим множество векторов для кластеризации с использованием датчика rands.

```
clear all;
P=rands(2,100);
figure;
plot(P(1,:),P(2,:),'*g');
```

В результате получим (рисунок 21):

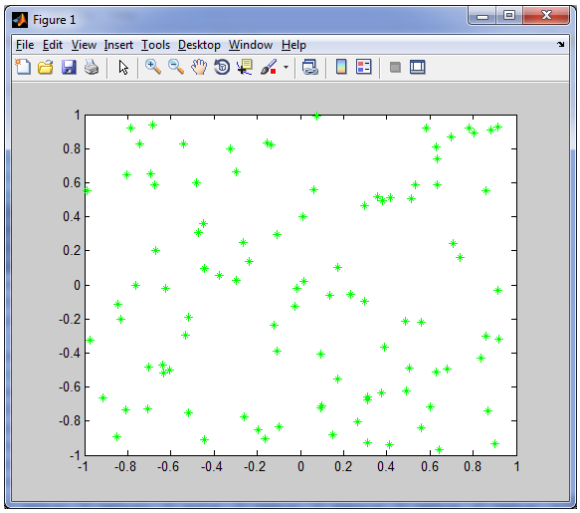


Рисунок 21 – Графическое отображение множества векторов, заданных случайным образом

Для создания сети Кохонена используем функцию newsom, для графического представления картины расположения нейронов слоя используем функцию plotsom.

Таблица 9 - Функции создание сети Кохонена и отображения картины расположения нейронов

Функция	Синтаксис	Назначение
newsom	**net = newsom(P,[D1,D2,...],TFCN,DFCN,STEPS,IN)**, Детальное описание синтаксиса описано в документации (см. doc newsom).	Создание самоорганизующейся карты
plotsom	**plotsom(W,d)** , где W – матрица весов ; d – матрица расстояний	Отображает картину расположения нейронов слоя

```
net=newsom([-1 1; -1 1],[5 6]);
figure;
plotsom(net.IW{1,1},net.layers{1}.distances);
```

В результате получим (рисунок 22):

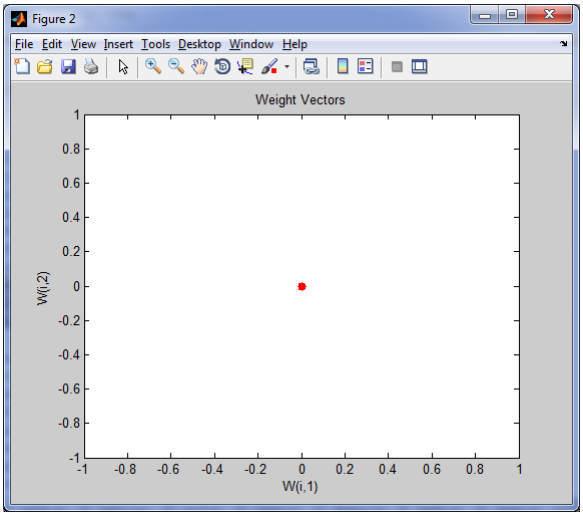


Рисунок 22 – Графическое представление картины расположения нейронов

На рисунке 22 представлена картина расположения весовых коэффициентов нейронов сети. Поскольку обучения не происходило они отображаются в виде одной точки, расположенной в начале координат, которая по сути является результатом наложения друг на друга точек с одними и теми же координатами.

Процесс разворачивания самоорганизующейся карты Кохонена в процессе самообучения и графическое представление расположения весов нейронов слоя реализуется следующим образом:

```
net.trainParam.epochs=2;
net=train(net,P);
figure;
plotsom(net.IW{1,1},net.layers{1}.distances);
hold on;
plot(P(1,:),P(2,:),'*g');
hold off;
```

В результате получим (рисунок 23):

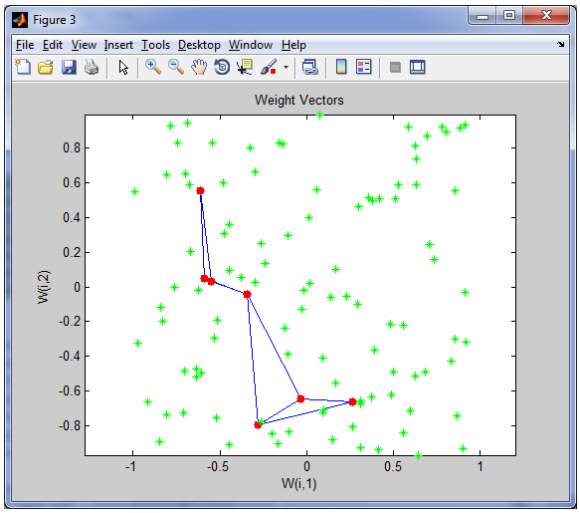


Рисунок 23 – Самоорганизующая карта Кохонена

```
net.trainParam.epochs=5;
 net=train(net,P);
 figure;
 plotsom(net.IW{1,1},net.layers{1}.distances);
 hold on;
 plot(P(1,:),P(2,:),'*g');
 hold off;
```

В результате получим (рисунок 24):

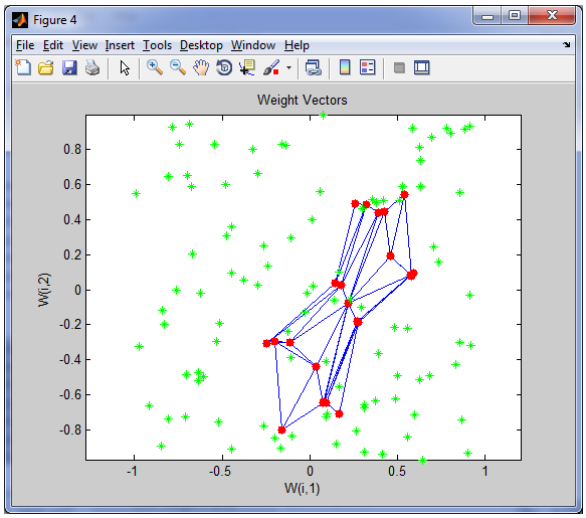


Рисунок 24 – Самоорганизующая карта Кохонена

```
net.trainParam.epochs=10;
net=train(net,P);
figure;
plotsom(net.IW{1,1},net.layers{1}.distances);
hold on;
```

```
 plot(P(1,:),P(2,:),'*g');
hold off;
```

В результате получим (рисунок 25):

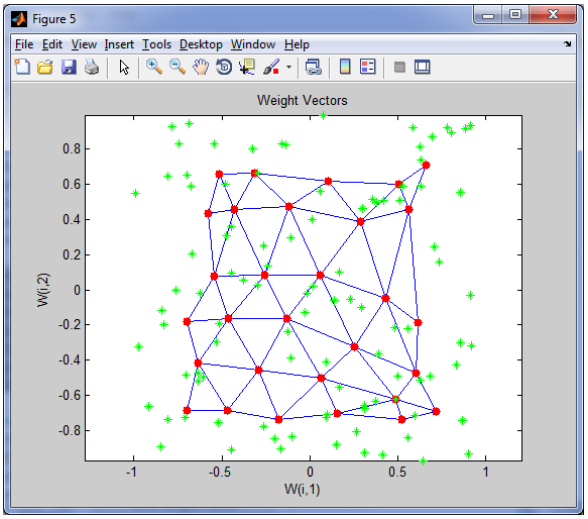


Рисунок 25 – Самоорганизующая карта Кохонена

Для тестирования сети Кохонена определим новый вектор

```
%тестирование сети
 Y=sim(net,[0.5;0.3]);
 disp(Y);
 plot(0.5,0.3,'*k');
 hold on;
 plot(P(1,:),P(2,:),'*g');
 hold off;
```

В результате получим (рисунок 26):

```
Y =
  (4,1)        1
```

Это означает, что новый вектор относится к кластеру номер 4.

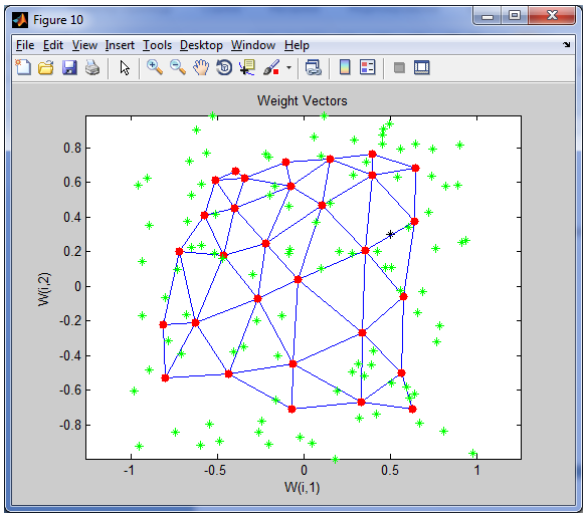


Рисунок 26 – Самоорганизующая карта Кохонена

Проведем анализ распределения данных по кластерам

```
a=sim(net,P);
 %анализ рапределения данных по кластерам
 figure;
 bar(sum(a'));
```

В результате получим (рисунок 27):

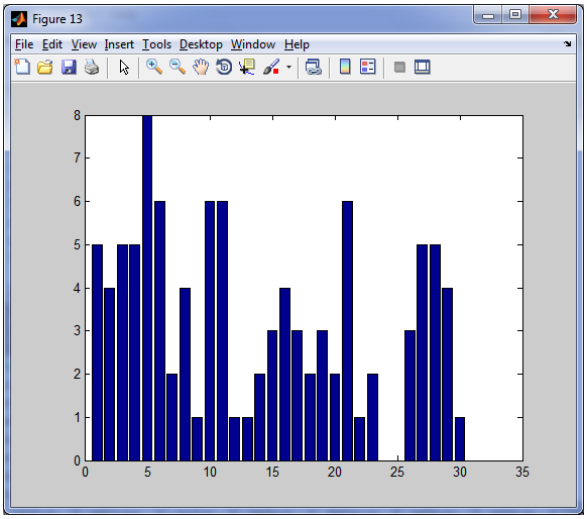


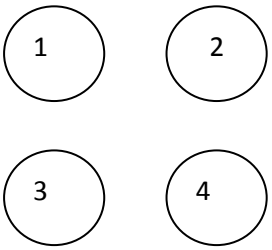
Рисунок 27 – Анализ распределения данных по кластерам

Задания для самостоятельной работы

1. Провести тестирование сети Кохонена для нового вектора и определить его принадлежностьк одному из кластеров.

2. Практически исследовать различные параметры функции newsom для создания сети Кохонена.

3. Решить задачу кластеризации множества векторов расположенных в виде четырех кластеров следующим образом



4. Построить график распределения данных по четырем кластерам в зависимости от центров кластеров.