МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

Методические указания

по выполнению лабораторных работ
по дисциплине
«Основы распознавания образов»
для направления подготовки **09.03.02 Информационные системы и технологии**
направленность (профиль) **Информационные системы и технологии обработки цифрового контента**

Пятигорск
2023

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1

Основы байесовских методов классификации

Цель и содержание работы: познакомить студента с основными аспектами байесовских методов классификации.

Теоретическое обоснование

Байесовский подход является классическим в теории распознавания образов и лежит в основе многих методов. Он опирается на теорему о том, что если плотности распределения классов известны, то алгоритм классификации, имеющий минимальную вероятность ошибок, можно выписать в явном виде. Для оценивания плотностей классов по выборке применяются различные подходы.

Рассмотрим пример параметрического подхода, но сначала определим вероятностную постановку задачи классификации.

Пусть X – множество объектов, Y – конечное множество имён классов, множестве $X \times Y$ является вероятностным пространством с плотностью распределения p(x, y) = P(y)p(x|y). Вероятности появления объектов каждого из классов P_y = P(y) называются априорными вероятностями классов. Плотности распределения $p_y(x)$ = p(x|y) называются функциями правдоподобия классов. Вероятностная постановка задачи классификации разделяется на две независимые подзадачи.

1. Имеется простая выборка $X^l = (x_i, y_i)_{i=1}^l$ (l – размер выборки) из неизвестного распределения p(x, y) = $P_y p_y(x)$. Требуется построить эмпирические оценки априорных вероятностей $\hat{P}_y$ и функций правдоподобия $\hat{p}_y(x)$ для каждого из классов $y \in Y$.
2. По известным плотностям распределения $p_y(x)$ и априорным вероятностям P_y всех классов $y \in Y$ построить алгоритм a(x), минимизирующий вероятность ошибочной классификации.

Первая задача имеет множество решений, поскольку многие распределения p(x, y) могли бы породить одну и ту же выборку X^l. Приходится привлекать различные предположения о плотностях, что и приводит к большому разнообразию байесовских методов.

В параметрическом подходе предполагается, что плотность распределения выборки известна.

Нормальный дискриминантный анализ – это специальный случай байесовской классификации, когда предполагается, что плотности всех классов $p_y(x)$, $y \in Y$ являются многомерными нормальными.

Приведём формулу n-мерного (гауссова) распределения:

$$p(x) = \frac{1}{(2\pi)^{n/2}|S|^{1/2}} \exp\left(-\frac{1}{2}(x-m)^T S^{-1}(x-m)\right), \qquad (1.1)$$

где n – количество числовых признаков, m = E[x] – математическое ожидание (центр), S – ковариационная матрица ($S = E[(x-m)(x-m)^T]$), |S| - детерминант S.

Пример 1. Вычислим, используя Matlab, значение гауссова распределения для $x_1 = [0.2, 1.3]^T$ и $x_2 = [2.2, -1.3]^T$, где $m = [0, 1]^T$, а $S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

```
close('all');
clear;
m=[0 1]'; S=eye(2);
x1=[0.2 1.3]'; x2=[2.2 -1.3]';
pg1=comp_gauss_dens_val(m,S,x1)
pg2=comp_gauss_dens_val(m,S,x2)
```

Где comp_gauss_dens_val(·) определим как

```
function [z]=comp_gauss_dens_val(m,S,x)
[l,c]=size(m);
z=(1/( (2*pi)^(l/2)*det(S)^0.5) )*exp(-0.5*(x-m)'*inv(S)*(x-m));
```

Функцию нужно создать в отдельном файле Matlab (New → Function). Таким образом будет два файла Matlab, которые могут располагаться в одной папке. Чтобы Matlab понимал откуда ему загружать необходимую функцию, можно нажать в окне Current Folder на нужную папку правой мышкой и установить (Add to Path → Selected Folders and Subfolders), рисунок 1.1.

Для того, чтобы выбрать нужную папку в окне Current Folder необходимо нажать кнопку «Browse for folder».

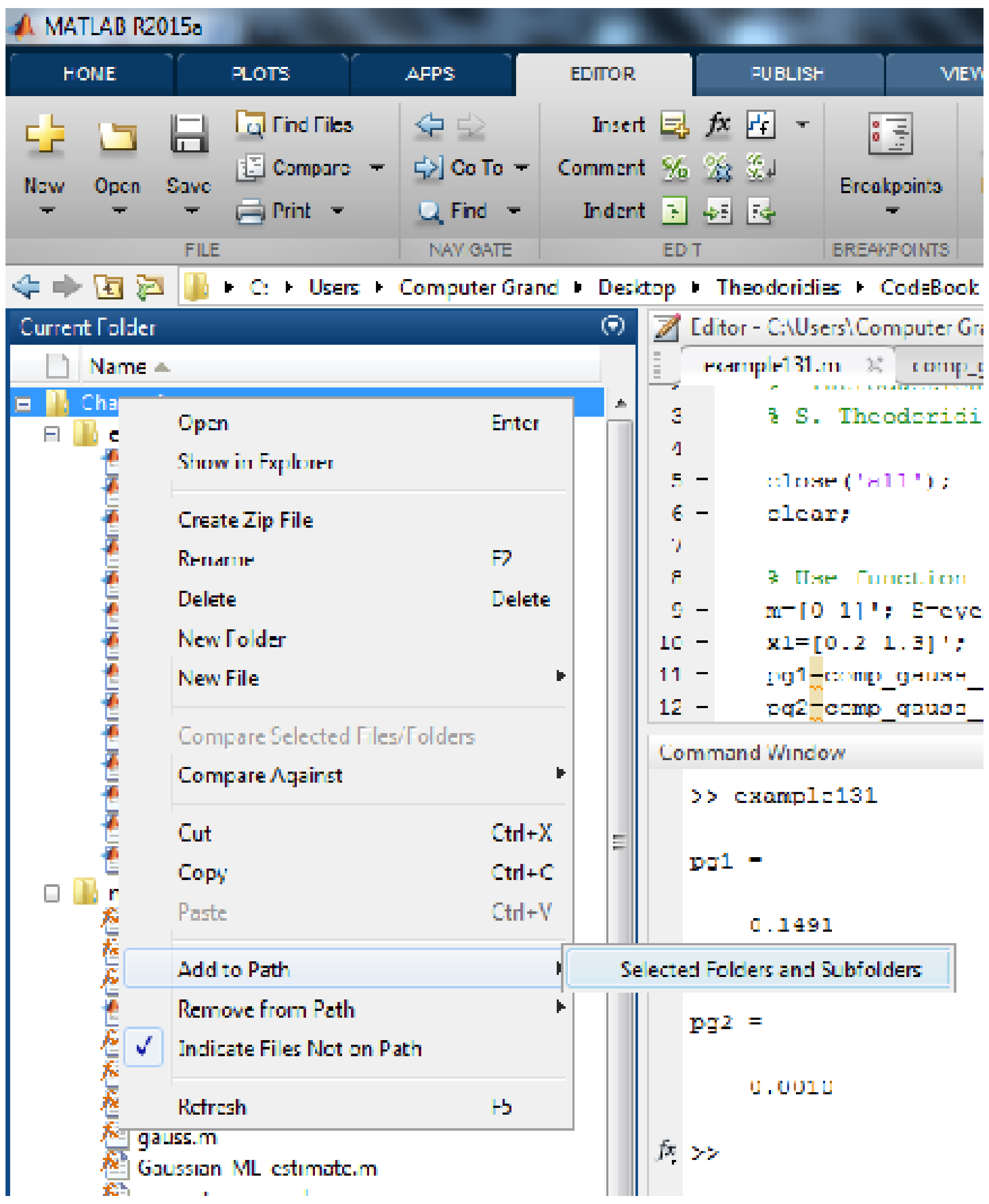

Рисунок 1.1 – Установка папки, откуда Matlab будет брать необходимые ему файлы

Функция **eye()** – создаёт единичную матрицу, запятая после квадратных скобок означает транспонирование вектора (**[]'**). Для получения справки по неизвестному объекту языка matlab, достаточно установить на него курсор и нажать F1 или написать команду help с именем объекта, допустим, **help eye**.

В результате выполнения кода получится, что pg1 = 0.1491, pg2 = 0.001.

На основании этого кода можно написать код байесовского классификатора, который определит к какому из двух классов относится входной вектор.

Есть два класса w_1 и w_2, имеющих гауссовы распределения $N(m_1, S_1)$, $N(m_2, S_2)$. $m_1 = [1, 1]T$, $m_2 = [3, 3]T$, $S_1 = S_2 = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$. Примем во внимание, что вероятность появления первого и второго классов подсчитаны заранее и равны $P(w_1) = P(w_2) = ½$. Требуется узнать к какому классу принадлежит вход $x = [1.8, 1.8]^T$.

Пример 2. Напишем код байесовского классификатора, который решит эту задачу:

```
close('all');
clear;
P1=0.5;
P2=0.5;
m1=[1 1]';
m2=[3 3]';
S=eye(2);
x=[1.8 1.8]';
p1=P1*comp_gauss_dens_val(m1,S,x)
p2=P2*comp_gauss_dens_val(m2,S,x)
```

Получается, что p1 = 0.042, p2 = 0.0189, т.е. p1 > p2 , соответственно двумерный вектор принадлежит первому классу.

Далее, проведём геометрический анализ нормальной плотности, её зависимость от матрицы ковариации.

Пример 3. Рассмотрим код, который генерирует 500 двумерных точек, распределенных в соответствие с двумерным нормальным распределением с центром m = $[0, 0]^T$ и матрицей ковариации $S = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}$, где существует 8 типов этой матрицы:

1. $\sigma_1^2 = \sigma_2^2 = 1, \sigma_{12} = 0$;
2. $\sigma_1^2 = \sigma_2^2 = 0.2, \sigma_{12} = 0$;
3. $\sigma_1^2 = \sigma_2^2 = 2, \sigma_{12} = 0$;
4. $\sigma_1^2 = 0.2, \sigma_2^2 = 2, \sigma_{12} = 0$;
5. $\sigma_1^2 = 2, \sigma_2^2 = 0.2, \sigma_{12} = 0$;
6. $\sigma_1^2 = \sigma_2^2 = 1, \sigma_{12} = 0.5$;
7. $\sigma_1^2 = 0.3, \sigma_2^2 = 2, \sigma_{12} = 0.5$;
8. $\sigma_1^2 = 0.3, \sigma_2^2 = 2, \sigma_{12} = -0.5$.

```
close('all');
clear;
% Генерируем точки для первого случая
randn('seed',0); % используем старый вариант вызова генератора
% случайных чисел, слово 'seed' заменяет ряд параметров для такого генератора
m=[0 0]'; % центр
S=[1 0;0 1]; % матрица ковариации
N=500; % количество точек
X = mvnrnd(m,S,N)'; % стандартная функция по генерации многомерного
% нормального случайного распределения
% Рисуем точки для первого варианта
figure(1), plot(X(1,:),X(2,:),'.');
figure(1), axis equal % устанавливаем тип стиля для осей
figure(1), axis([-7 7 -7 7])
% Рисуем точки для второго варианта
m=[0 0]';
S=[0.2 0;0 0.2];
N=500;
X = mvnrnd(m,S,N)';
figure(2), plot(X(1,:),X(2,:),'.');
figure(2), axis equal
figure(2), axis([-7 7 -7 7])
% Рисуем точки для третьего варианта
```

```
m=[0 0]';
S=[2 0;0 2];
```

```
N=500;
X = mvnrnd(m,S,N)';
figure(3), plot(X(1,:),X(2,:),'.');
figure(3), axis equal
figure(3), axis([-7 7 -7 7])
% Рисуем точки для четвертого варианта
m=[0 0]';
S=[0.2 0;0 2];
N=500;
X = mvnrnd(m,S,N)';
figure(4), plot(X(1,:),X(2,:),'.');
figure(4), axis equal
figure(4), axis([-7 7 -7 7])
% Рисуем точки для пятого варианта
m=[0 0]';
S=[2 0;0 0.2];
N=500;
X = mvnrnd(m,S,N)';
figure(5), plot(X(1,:),X(2,:),'.');
figure(5), axis equal
figure(5), axis([-7 7 -7 7])
% Рисуем точки для шестого варианта
m=[0 0]';
S=[1 0.5;0.5 1];
N=500;
X = mvnrnd(m,S,N)';
figure(6), plot(X(1,:),X(2,:),'.');
figure(6), axis equal
figure(6), axis([-7 7 -7 7])
% Рисуем точки для седьмого варианта
m=[0 0]';
S=[.3 0.5;0.5 2];
N=500;
X = mvnrnd(m,S,N)';
figure(7), plot(X(1,:),X(2,:),'.');
figure(7), axis equal
figure(7), axis([-7 7 -7 7])
% Рисуем точки для восьмого варианта
m=[0 0]';
S=[.3 -0.5;-0.5 2];
N=500;
X = mvnrnd(m,S,N)';
figure(8), plot(X(1,:),X(2,:),'.');
figure(8), axis equal
figure(8), axis([-7 7 -7 7])
```

На рисунках 1.2 – 1.9 показано сгенерированное множество точек.

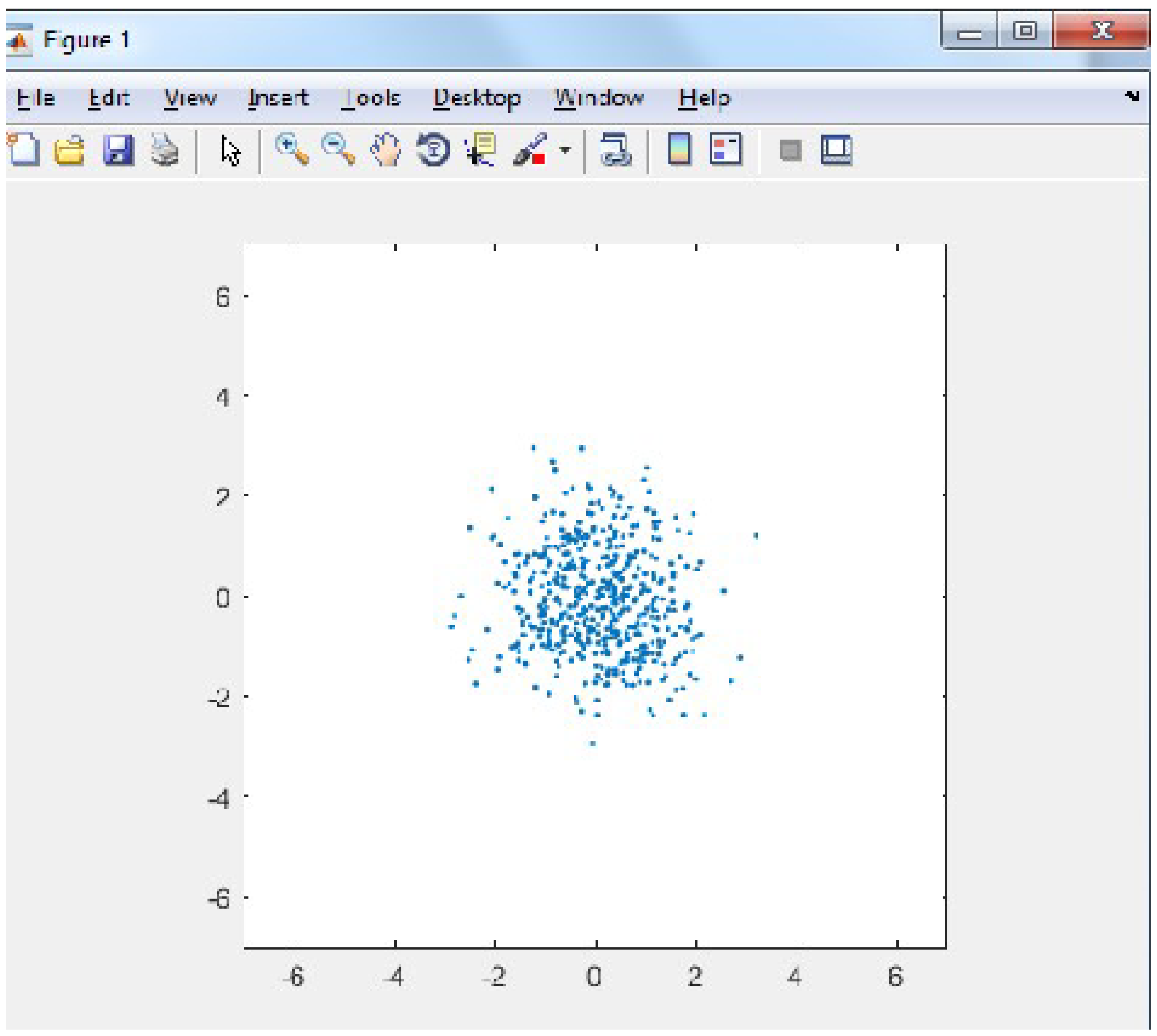

Рисунок 1.2 – Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 1, \rho_{12} = 0$

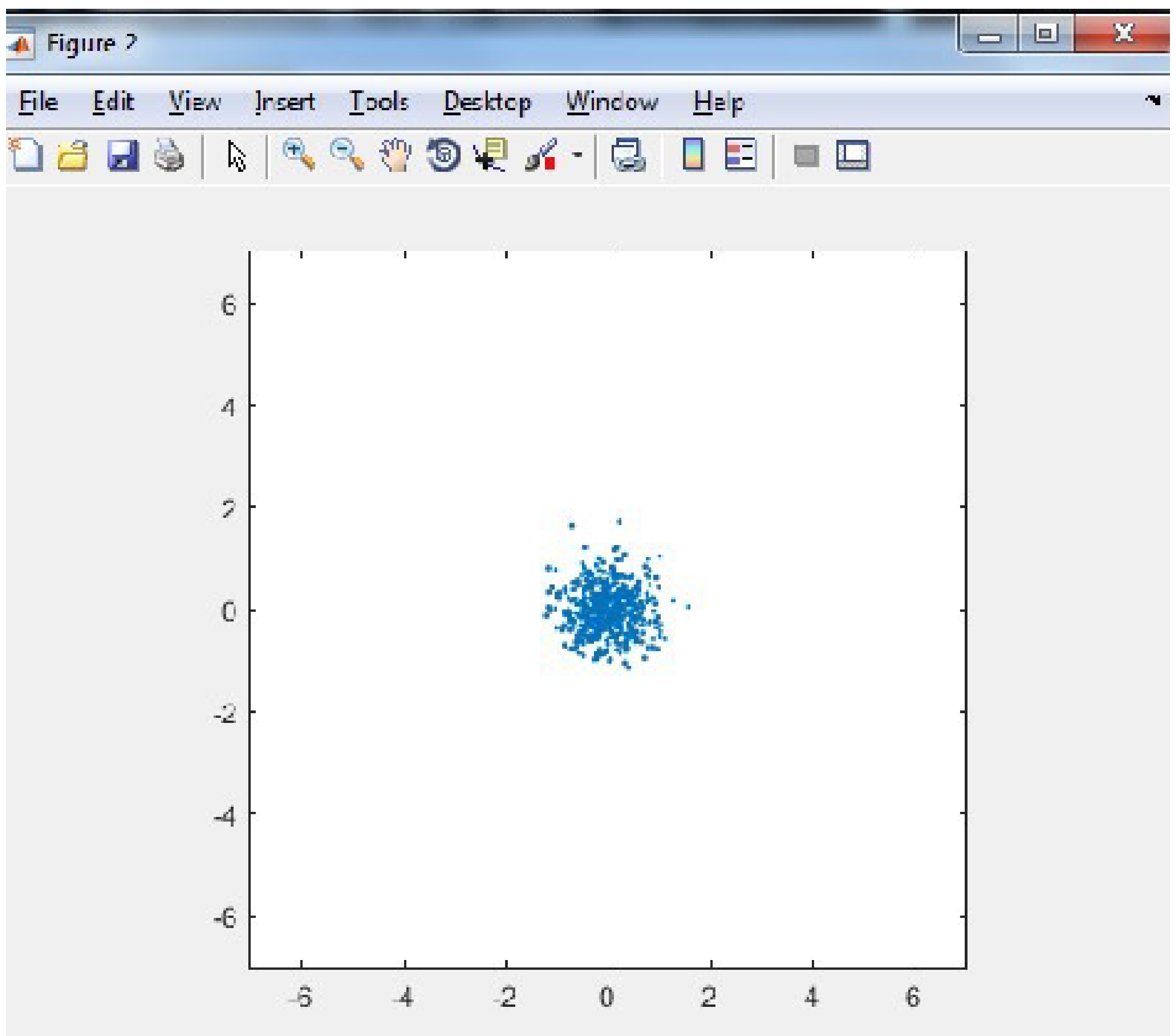

Рисунок 1.3 – Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 0.2, \rho_{12} = 0$

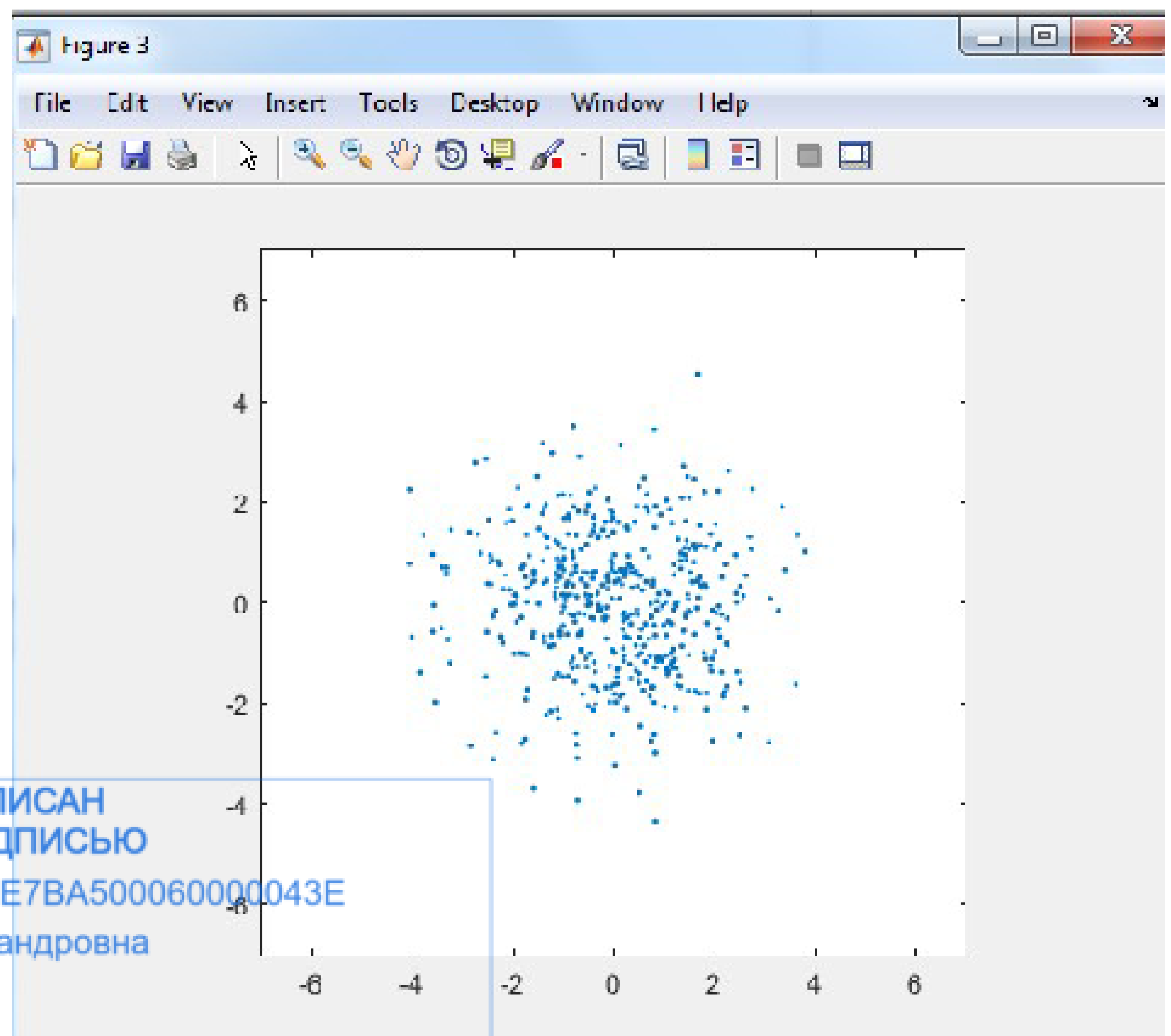

Рисунок 1.4 – Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 2$, $\sigma_{12} = 0$

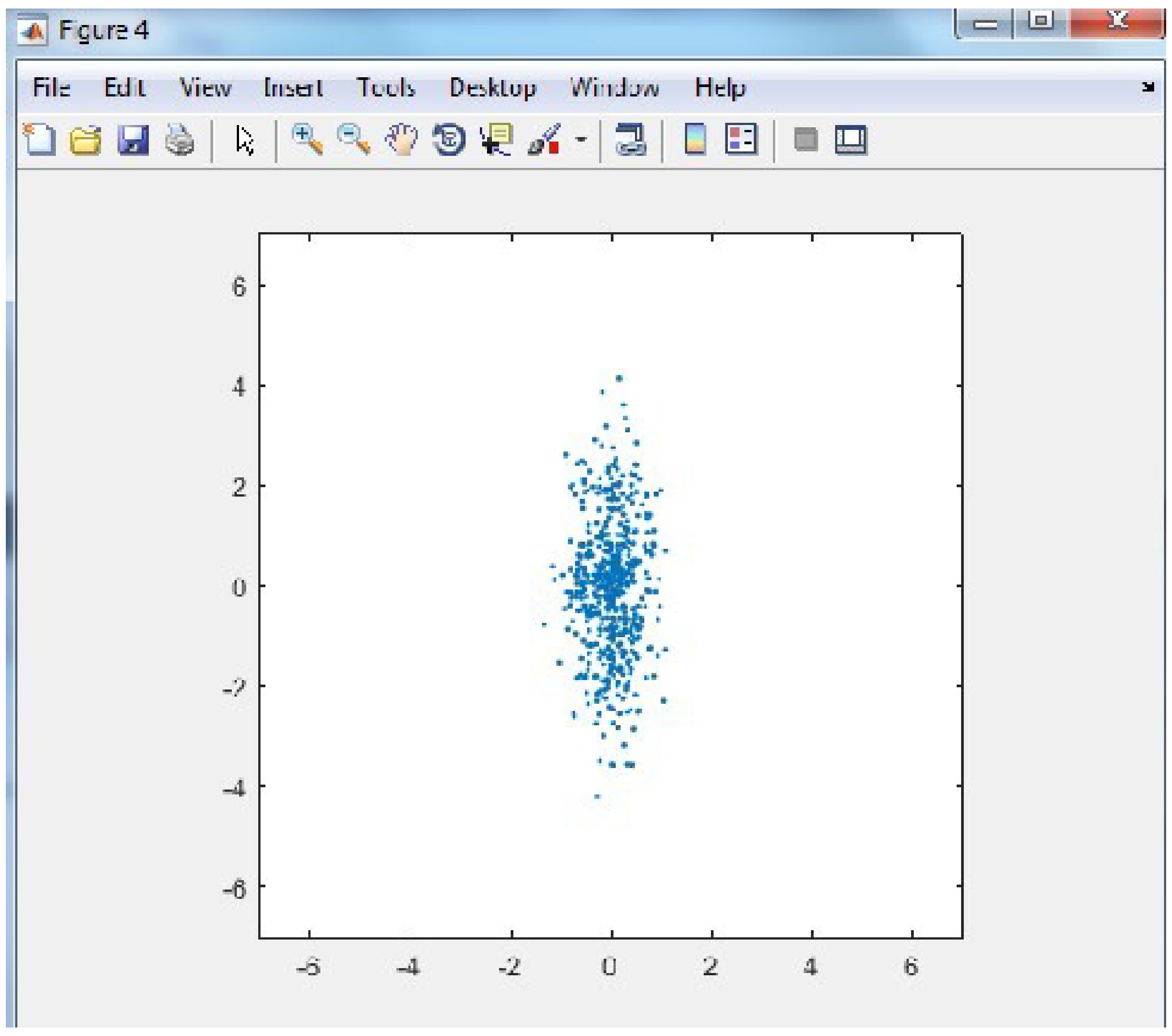

Рисунок 1.5 – Сгенерированное множество точек для случая $\sigma_1^2 = 0.2$, $\sigma_2^2 = 2$, $\sigma_{12} = 0$

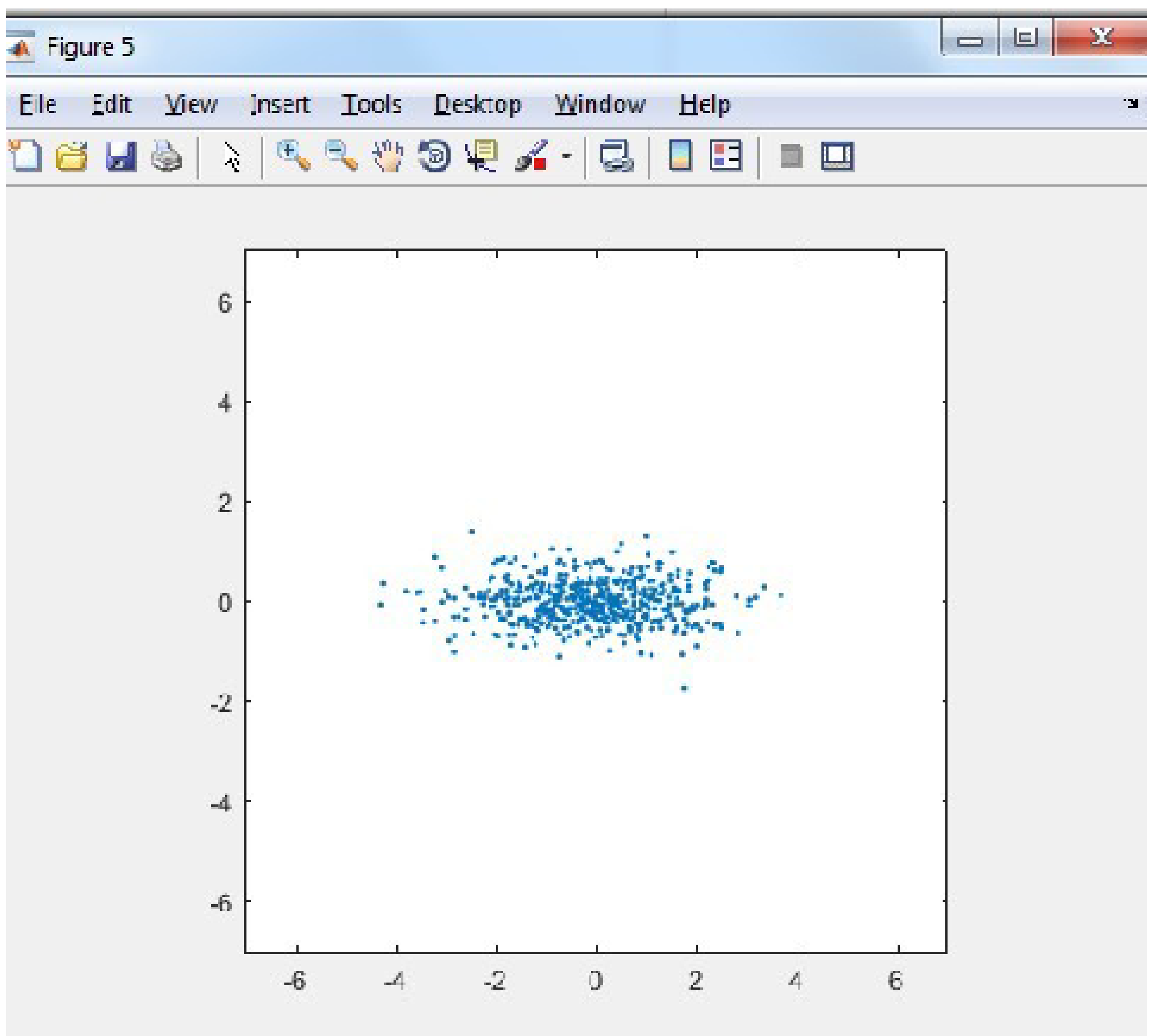

Рисунок 1.6 – Сгенерированное множество точек для случая $\sigma_1^2 = 2$, $\sigma_2^2 = 0.2$, $\sigma_{12} = 0$

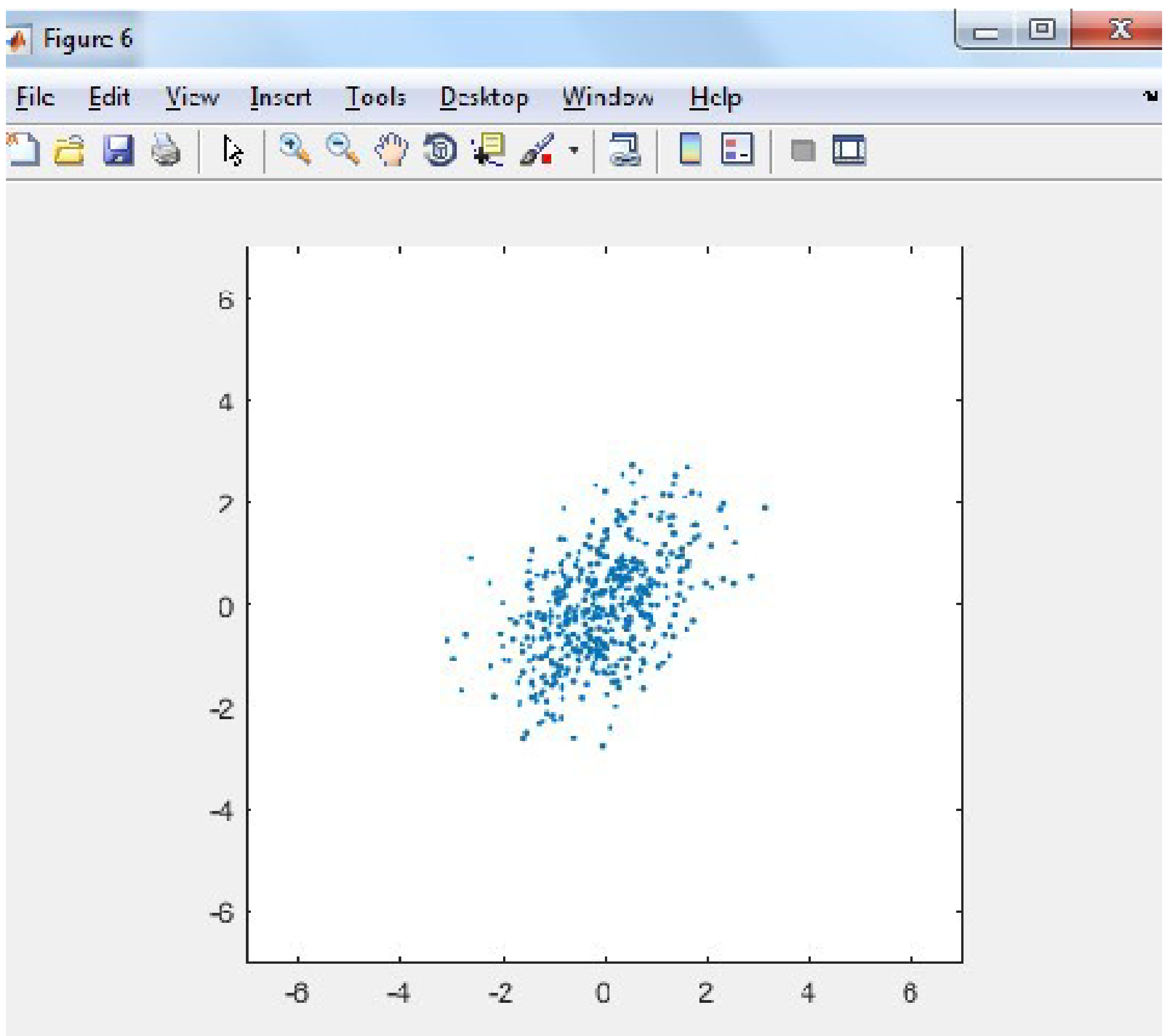

Рисунок 1.7 – Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 1$, $\sigma_{12} = 0.5$

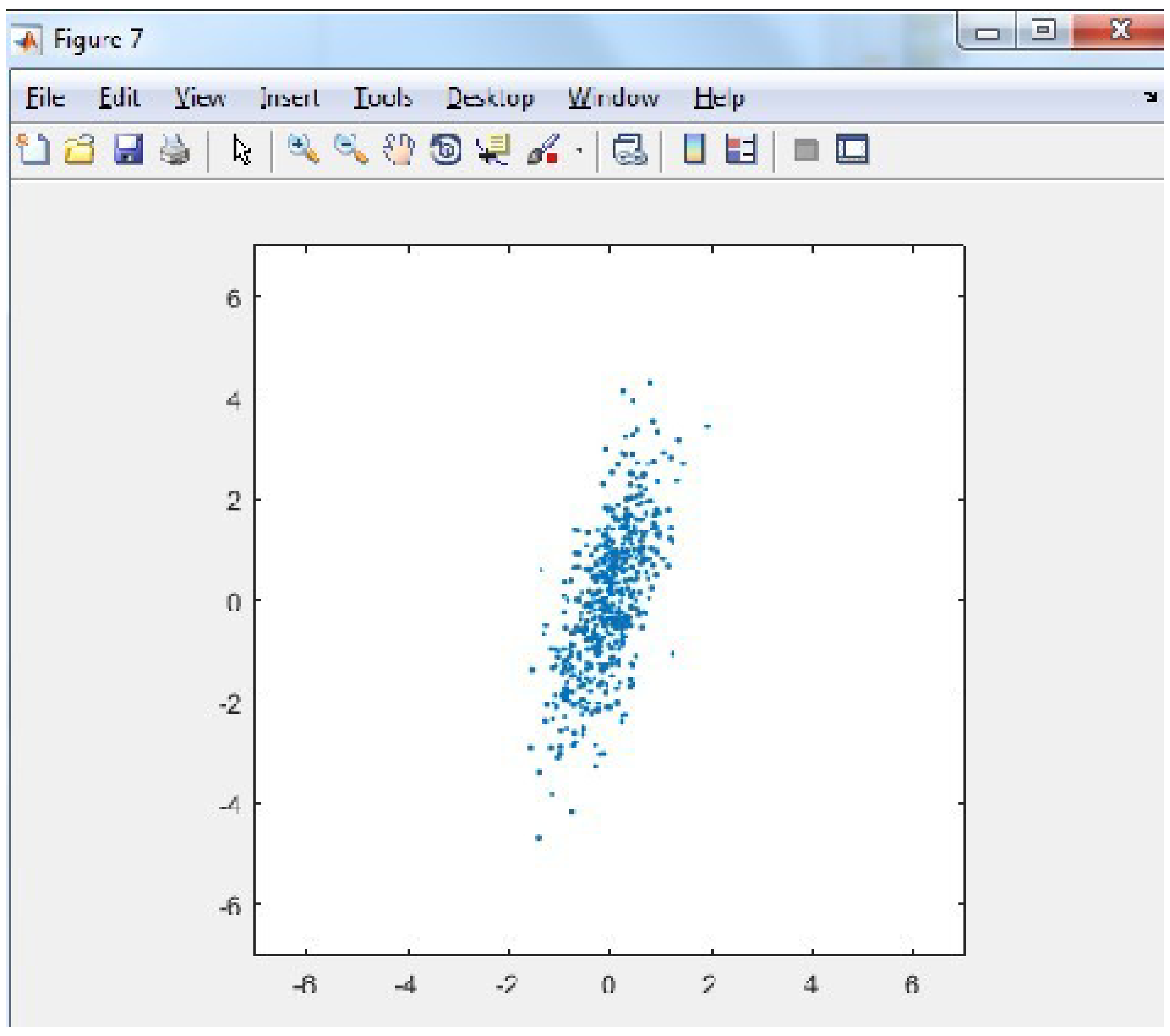

Рисунок 1.8 – Сгенерированное множество точек для случая $\sigma_1^2 = 0.3$, $\sigma_2^2 = 2$, $\sigma_{12} = 0.5$

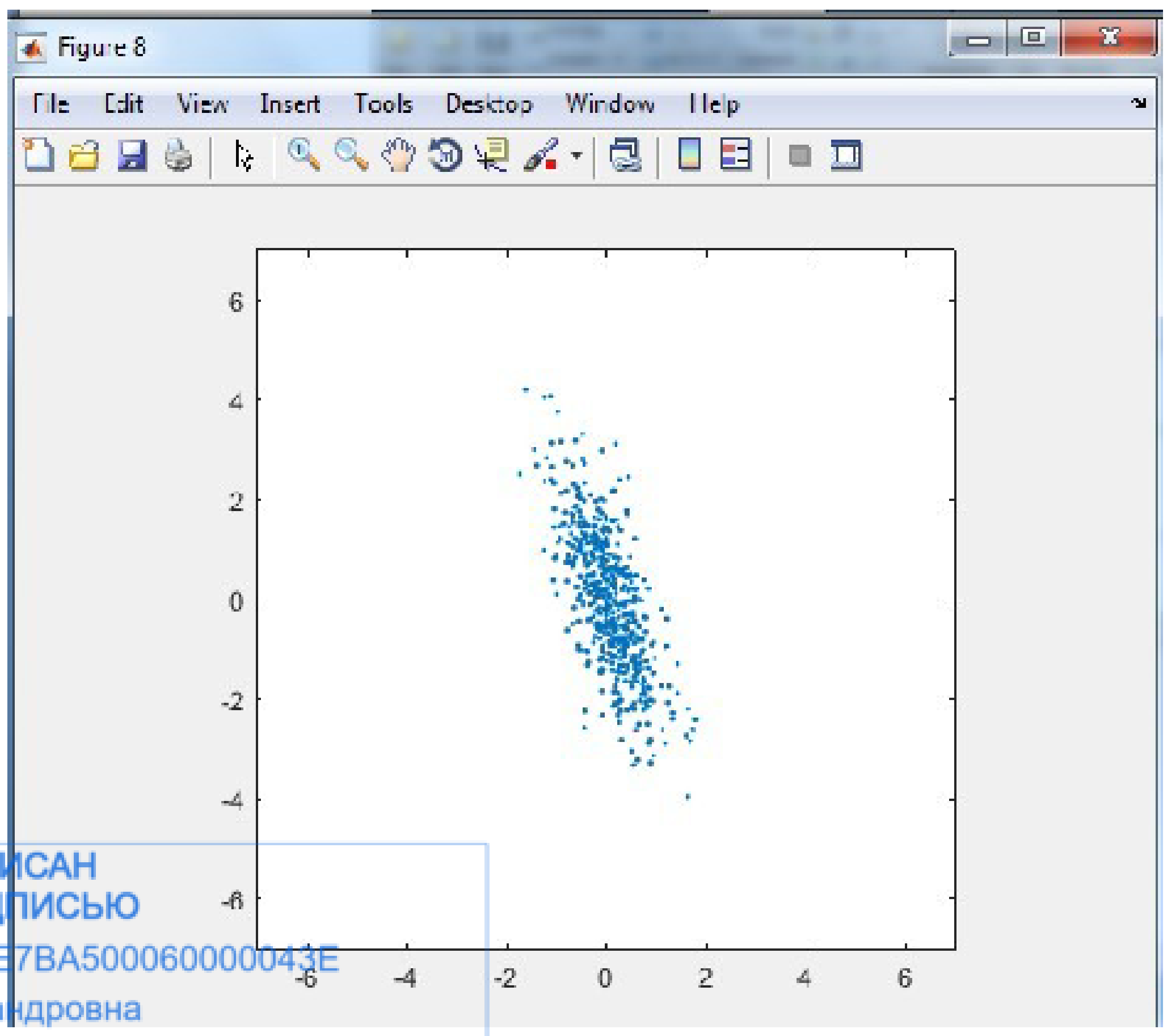

Рисунок 1.9 – Сгенерированное множество точек для случая $\sigma_1^2 = 0.3$, $\sigma_2^2 = 2$, $\sigma_{12} = -0.5$

Из этих графиков можно сделать следующий вывод, если признаки некоррелированы, $S = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2)$, то линии уровня плотности распределения имеют форму эллипсоидов с центром m и осями, параллельными линиям координат (рисунок 1.5 и 1.6). Если признаки имеют одинаковые дисперсии, то эллипсоиды являются сферами (рисунок 1.2–1.4). Если признаки коррелированны, то матрица S не диагональная и линии уровня имеют форму эллипсоидов, оси которых повернуты относительно исходной системы координат (рисунок 1.7–1.9).

Итак, в коде байесовского классификатора мы определяли класс по формуле из теории вероятности: умножали вероятность соответствующего класса на плотность распределения, причем предполагали, что плотность – нормальная, а вероятности известны.

Но можно решать задачу классификации и путём вычисления расстояний. Допустим, у нас есть некая точка в виде n-мерного вектора, и есть выборка, которую тоже можно отразить в виде набора точек в n-мерном пространстве. Тогда, если мы сначала получим некие характеристики выборки, допустим, узнаем её центр-масс, то можно определить расстояние от точки до центра масс. Если выборок много, то какое расстояние меньше, тому классу и принадлежит точка.

Пример 4. Сначала используем формулу из школьного курса: евклидово расстояние.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}, \qquad (1.2)$$

где p и q n-мерные вектора.

```
close('all');
clear;
% x – входной вектор
x=[0.1 0.5 0.1]';
% m1 и m2 играют роль оценки двух неизвестных множеств: их центры масс
m1=[0 0 0]';
m2=[0.5 0.5 0.5]';
m=[m1 m2];
% передача информации в функцию
z=euclidean_classifier(m,x)


% функия в отдельном файле
function [z]=euclidean_classifier(m,X)
% вернёт индекс класса 1 или 2 к которому принадлежит точка X
% X – это матрица 1x3
% m – это матрица 3x2
% size() – возвращает координаты матрицы
[l,c]=size(m); % (l, c) = (3, 2)
[l,N]=size(X); % (l, N) = (3, 1)

for i=1:N % цикл по строкам X от 1 до 3
```

```
   for j=1:c % цикл по столбцам m от 1 до 2
      % высчитываем два евклидовых расстояния
      de(j)=sqrt((X(:,i)-m(:,j))'*(X(:,i)-m(:,j)));
   end
   % de(1) = 0.5196
   % de(2) = 0.5657
   % в num будет содержаться минимальное значение, т.е. de(1)
   % z(i) – будет содержать его индекс, т.е. 1
   [num,z(i)]=min(de);
end
```

Однако можно использовать и более сложную формулу расстояния: расстояние Махаланобиса, которое использует не только центр масс, но и матрицу корреляции.

Если мы оценили множества и нашли его среднее значение m = [m_1, m_2, …, m_n]T, а также матрицу ковариации S, то расстояние определится следующим образом от точки x = [x_1, x_2, …, x_n]T до m.

$$D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} \qquad (1.3)$$

Тогда точка будет принадлежать тому классу для которого выполняется условие:

$$\sqrt{(x-\mu_i)^T \Sigma^{-1} (x-\mu_i)} < \sqrt{(x-m_j)^T S^{-1} (x-m_j)},\ j \neq i, \qquad (1.4)$$

т.е. для которого расстояние Махаланобиса минимально.

Пример 5. Напишем код для соответствующей классификации.

```
close('all');
clear;
x=[0.1 0.5 0.1]';
m1=[0 0 0]';
m2=[0.5 0.5 0.5]';
m=[m1 m2];
% для двух множеств одна и таже матрица ковариации S
S=[0.8 0.01 0.01;
   0.01 0.2 0.01;
   0.01 0.01 0.2];
z=mahalanobis_classifier(m,S,x)

% функция для расстояния Махаланобиса
function z=mahalanobis_classifier(m,S,X)
[l,c]=size(m);
[l,N]=size(X);
for i=1:N
   for j=1:c
      dm(j)=sqrt((X(:,i)-m(:,j))'*inv(S)*(X(:,i)-m(:,j)));
   end
   % dm(1) = 1.1334
   % dm(2) = 0.9918
   [num,z(i)]=min(dm);
   end
```

% z = 2

Видим, что в этом случае точка принадлежит классу 2. Получается противоречие: евклидово расстояние показывает принадлежность к классу 1, а расстояние Махаланобиса – наоборот. В таком случае нужно принять ответ классификатора Махаланобиса, т.к. за счёт матрицы ковариации расстояние Махаланобиса учитывает ещё и форму распределения точек вокруг центра масс. По сути расстояние Махаланобиса – это просто расстояние между заданной точкой и центром масс, делённое на ширину эллипсойда в направлении заданной точки.

Рассмотрим далее принцип максимума правдоподобия, который составляет основу параметрического подхода. Пусть задано множество объектов $X^m = \{x_1,...,x_m\}$, выбранных независимо друг от друга из вероятностного распределения с плотностью φ(x;θ). Функцией правдоподобия называется совместная плотность распределения всех объектов выборки:

$$p(X^m;\theta) = p(x_1,...,x_m;\theta) = \prod_{i=1}^{m} \varphi(x_i;\theta). \qquad (1.5)$$

Значение параметра θ, при котором выборка максимально правдоподобна, то есть функция $p(X^m;\theta)$ принимает максимальное значение, называется оценкой макисмума правдоподобия.

В случае гауссовской плотности с параметрами θ = (m, S) задача максимизации правдоподобия имеет аналитическое решение:

$$m_{ML} = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad (1.6)$$

$$S_{ML} = \frac{1}{N}\sum_{i=1}^{N} (x_i - m_{ML})(x_i - m_{ML})^T. \qquad (1.7)$$

Пример 6. Рассмотрим на примере как работает алгоритм максимума правдоподобия. Сгенерируем 50 точек, имеющих по две координаты каждая, используя гауссово распределение с центром $m = [2, -2]^T$, $S = \begin{bmatrix} 0.9 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$. Потом оценим параметры этого распределения по формулам (1.6) и (1.7), сравним результаты.

```
close('all');
clear;
% Генерируем 50 точек с заданным распределением гаусса
randn('seed',0);
m = [2 -2]; S = [0.9 0.2; 0.2 .3];
X = mvnrnd(m,S,50)';
% Оцениваем параметры распределения по выборке
[m_hat, S_hat]=Gaussian_ML_estimate(X)
```

Функцию можно создать в отдельном файле Matlab.

```
function [m_hat,S_hat]=Gaussian_ML_estimate(X)
% Входной параметр:
%  X:      lxN матрица, чьи колонки есть наши данные.
% Выходной аргумент:
%  m_hat:  l-размерная оценка среднего вектора распределения.
%  S_hat:  lxl оценка ковариационной матрицы распределения.
 [l,N]=size(X); % l = 2, N = 50
m_hat=(1/N)*sum(X')'; % получаем среднеарифметический центр масс
% создаём и инициализируем 0 квадратную матрицу 2x2
S_hat=zeros(l);
for k=1:N
   S_hat=S_hat+(X(:,k)-m_hat)*(X(:,k)-m_hat)';
end
% Высчитываем среднее
S_hat=(1/N)*S_hat;
```

По результатам получился вектор m_hat = $[2.0495, \ -1.9418]^T$,

$S\_hat = \begin{bmatrix} 0.8082 & 0.0885 \\ 0.0885 & \end{bmatrix}$.

Как видно, получившиеся оценки близки к оригиналу, однако, нужно учесть, что они проводились на выборке, состоящей из 50 элементов, что очень мало, поэтому эти оценки не надёжны. Для увеличения надёжности оценок нужно увеличить выборку.

Пример 7. Теперь рассмотрим последний пример в котором обобщим весь предыдущий материал. Сгенерируем два множества X и X_1 каждое содержит 999 трехмерных точек, X – обучающее множество, X_1 – тестовое. Каждое множество включает три равновероятных класса: w_1, w_2, w_3; классы создаются с помощью распределения Гаусса со средними $m_1 = [0, 0, 0]T$, $m_2 = [1, 2, 2]T$, $m_3 = [3, 3, 4]T$ и матрицами ковариации

$$S_1 = S_2 = S_3 = \begin{bmatrix} 0.8 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.8 \end{bmatrix} = \sigma^2 I.$$

Используя X с помощью принципа максимального правдоподобия оценим такие параметры как среднее m и матрицы ковариации S_1, S_2, S_3, т.к. у нас 103 точек, то эти оценки будут надёжными. Далее, используя эти оценки проведём классификацию с помощью расстояния Махаланобиса, Евклида, а также байесовским классификатором. Для каждого способа вычислим ошибку вероятности и сравним результаты.

Сначала сгенерируем обучающее множество X.

```
close('all');
clear;
% Матрица m 3x3
```

```
m=[0 0 0; 1 2 2; 3 3 4]';
% Создаём матрицу S1 3x3 (единичную матрицу умножаем на 0.8)
S1=0.8*eye(3);
% Как переменная m содержит три вектора-центра масс, так и тут
% трехмерный массив S содержит три одинаковых матрицы ковариации
S(:,:,1)=S1;S(:,:,2)=S1;S(:,:,3)=S1;
% вероятность появления каждого из трех классов
P=[1/3 1/3 1/3]';
% количество элементов в обучающем множестве
N=1000;
% устанавливаем датчик случайных чисел с параметрами seed и 0.
randn('seed',0);
% Генерируем множество X[3;999] – сами точки, y[1, 999] – их метки
% от 1 до 333 – метка 1, от 334 до 666 – метка 2, от 667 до 999 – метка 3.
[X,y]=generate_gauss_classes(m,S,P,N);

% Функцию пишем в отдельном файле
function [X,y]=generate_gauss_classes(m,S,P,N)
[l,c]=size(m); % (l, c) = (3, 3)
X=[]; % Инициализация множеств
y=[];
for j=1:c % цикл от 1 до 3
% Генерируем трехмерные точки со средним m(:, j), где ":" – для всех строк,
% матрицей ковариации S(:, :, j) и количеством 1/3*1000, ограниченным снизу
  t=mvnrnd(m(:,j),S(:,:,j),fix(P(j)*N))';
  % сгенерированные точки присваиваем вектору X причем так, чтобы
  % они добавлялись в конец уже существующего вектора X. Сгенерировали
  % точки для j = 1, добавили их к пустому вектору, затем сгенерировали
  % точки для j = 2, добавили их в конец вектора, который уже содержит
  % точки для j = 1 и т.д.
  X=[X t]; % генерируем метки
  y=[y ones(1,fix(P(j)*N))*j];
end
```

Далее аналогично сгенерируем множество X1, но уже с другими параметрами для **randn()**.

```
% Генерируем тестовую выборку X1
randn('seed',100);
[X1,y1]=generate_gauss_classes(m,S,P,N);
```

Теперь по принципу максимума правдоподобия вычислим оценки для распределения, которое использовалось при создании множества X.

```
% извлекаем из X только те точки, которые имеют метку 1
class1_data=X(:,find(y==1));
% вычисляем центр масс и ковариационную матрицу
% функция для вычисления была разобрана ранее
[m1_hat, S1_hat]=Gaussian_ML_estimate(class1_data);
% Аналогично
class2_data=X(:,find(y==2));
[m2_hat, S2_hat]=Gaussian_ML_estimate(class2_data);
```

```
% Аналогично
class3_data=X(:,find(y==3));
[m3_hat, S3_hat]=Gaussian_ML_estimate(class3_data);
% Получаем общую оценку единой ковариационной матрицы,
% как среднеарифметическое трех уже вычисленных матриц
S_hat=(1/3)*(S1_hat+S2_hat+S3_hat);
m_hat=[m1_hat m2_hat m3_hat];
```

В итоге получаем $S\_hat = \begin{bmatrix} 0.8602 & 0.0212 & -0.0259 \\ 0.0212 & 0.8070 & -0.0134 \\ -0.0259 & -0.0134 & 0.8031 \end{bmatrix}$ и $m\_hat = \begin{bmatrix} 0.0512 & 0.9468 & 3.0138 \\ 0.0150 & 2.0470 & 3.0130 \\ -0.0698 & 1.9889 & 3.9398 \end{bmatrix}$. Видно, что получившиеся оценки близки к тем, которые мы задавали изначально. В реальности S и m из условия задачи нам нужны были только для того, чтобы создать выборки. Предполагается, что выборки X и X_1 мы получаем откуда-то со стороны и делаем предположение, что они сформированы посредством нормального распределения. Такое предположение мы можем сделать, т.к. рисунки 1.2–1.9 демонстрируют, что гауссово распределение может моделировать широкий класс данных.

Теперь, зная характеристики выборки, точнее трёх классов, входящих в неё, можно провести классификацию точек из тестового множества X_1.

```
% высчитываем расстояние от точки X1[i], i=1..999 до точки-центра
% соответствующего класса, z_euclidean будет содержать 999 индексов классов
% с самым минимальным расстоянием
z_euclidean=euclidean_classifier(m_hat,X1);
% Аналогично
z_mahalanobis=mahalanobis_classifier(m_hat,S_hat,X1);
% Используем байесовский классификатор, однако с математической
% точки зрения, чтобы он работал корректно, мы должны подавать ему
% не вычисленные оценки множества X, а данные в условии, т.е. точные
z_bayesian=bayes_classifier(m,S,P,X1);
% Находим процент несовпадений по классам
err_euclidean = (1-length(find(y1==z_euclidean))/length(y1))
err_mahalanobis = (1-length(find(y1==z_mahalanobis))/length(y1))
err_bayesian = (1-length(find(y1==z_bayesian))/length(y1))

% Функция для классификатора байеса
function [z]=bayes_classifier(m,S,P,X)
[l,c]=size(m);
[l,N]=size(X);

for i=1:N % цикл от 1 до 999
   for j=1:c % цикл от 1 до 3
      t(j)=P(j)*comp_gauss_dens_val(m(:,j),S(:,:,j),X(:,i));
```

```
   end
   [num,z(i)]=max(t);
end
```

Получается, что во всех трех случаях ошибки похожи: 7.61%, 7.71%, 7.61%. Это не случайно, если выполняются 4 следующих условия, то все три метода классификации эквивалентны:

1. Все классы равновероятны;
2. Данные во всех классах имеют гауссовы распределения;
3. Матрицы ковариации одни и те же;
4. Ковариационные матрицы не просто равны, но и диагональные, причем все элементы на главной диагонали равны.

Аппаратура и материалы. 64-разрядный (x64) персональный компьютер, процессор с тактовой частотой 1 ГГц и выше, оперативная память 1 Гб и выше, свободное дисковое пространство не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Matlab (R2013) и выше.

Указание по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения; ремонт персонального компьютера. Соблюдать правила технической эксплуатации и техники безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Задание для всех вариантов.

1. Вычислите как в примере 2 значения p1 и p2, но для следующих двух случаев: ($P(w_1)=1/6$, $P(w_2)=5/6$), ($P(w_1)=5/6$, $P(w_2)=1/6$). Дайте объяснения зависимости результатов классификации от априорных вероятностей.
2. Повторите пример 6, но для N = 500 и N = 5000 точек. Прокомментируйте результат.

В таблице 1.1 приведены индивидуальные задания.

Таблица 1.1 – Индивидуальные варианты

1	Повторите пример 7, но где $S_1 = S_2 = S_3 = \begin{pmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.8 \end{pmatrix} \neq \sigma^2 I$. Напишите вывод по получившемуся ответу.
2	Повторите пример 7, но где для генерации X и X_1 будут использованы следующие вероятности классов $P_1 = 1/2$, $P_2 = P_3 = 1/4$. Для этого случая байесовский классификатор должен показать лучший результат. Почему?
3	Повторите пример 7, но где P(w1) = P(w2) = P(w3) = 1/3 и $S_1 = \begin{pmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.8 \end{pmatrix}$, $S_2 = \begin{pmatrix} 0.6 & 0.01 & 0.01 \\ 0.01 & 0.8 & 0.01 \\ 0.01 & 0.01 & 0.6 \end{pmatrix}$, $S_3 = \begin{pmatrix} 0.6 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.6 \end{pmatrix}$. Объясните результат.
4	Повторите пример 7, но где $S_1 = S_2 = S_3 = \begin{pmatrix} 0.8 & 0.3 & 0.1 \\ 0.3 & 0.8 & 0.3 \\ 0.1 & 0.3 & 0.8 \end{pmatrix} \neq \sigma^2 I$. Напишите вывод по получившемуся ответу.
5	Повторите пример 7, но где для генерации X и X_1 будут использованы следующие вероятности классов $P_1 = 0.8$, $P_2 = 0.15$, $P_3 = 0.05$. Прокомментируйте результат.
6	Повторите пример 7, но где P(w1) = P(w2) = P(w3) = 1/3 и $S_1 = \begin{pmatrix} 0.9 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.9 \end{pmatrix}$, $S_2 = \begin{pmatrix} 0.6 & 0.01 & 0.01 \\ 0.01 & 0.8 & 0.01 \\ 0.01 & 0.01 & 0.6 \end{pmatrix}$, $S_3 = \begin{pmatrix} 0.6 & 0.2 & 0.1 \\ 0.1 & 0.6 & 0.1 \\ 0.1 & 0.2 & 0.6 \end{pmatrix}$. Объясните результат.
7	Повторите пример 7, но где $S_1 = S_2 = S_3 = \begin{pmatrix} 0.7 & 0.2 & 0.4 \\ 0.2 & 0.7 & 0.2 \\ 0.4 & 0.2 & 0.7 \end{pmatrix} \neq \sigma^2 I$. Напишите вывод по получившемуся ответу.
8	Повторите пример 7, но где для генерации X и X_1 будут использованы следующие вероятности классов $P_1 = 0.6$, $P_2 = P_3 = 0.2$ Для этого случая байесовский классификатор должен показать лучший результат. Почему?
9	Повторите пример 7, но где P(w1) = P(w2) = 0.2, P(w3) = 0.6 и $S_1 = \begin{pmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \end{pmatrix}$,

	$_1$ $0.1 \quad 0.2$ 0.8 $S_2 = \begin{pmatrix} 0.6 & 0.01 & 0.01 \\ 0.01 & 0.8 & 0.01 \\ 0.01 & 0.01 & 0.6 \end{pmatrix}$, $S_3 = \begin{pmatrix} 0.6 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.6 \end{pmatrix}$. Объясните результат.
10	Повторите пример 7, но где для генерации X и X_1 будут использованы следующие вероятности классов $P_1 = 0.4$, $P_2 = P_3 = 0.3$. Для этого случая байесовский классификатор должен показать лучший результат. Почему?

Содержание отчета и его форма

Отчёт по лабораторной работе должен содержать следующую информацию:

1) Название лабораторной работы и её номер.
2) ФИО и группу студанта.
3) Формулировка индивидуального задания и номер варианта.
4) Ответ на задание в виде кода и результатов работы Matlab.
5) Ответы на контрольные вопросы.

Вопросы для защиты работы

1. Что такое матрица ковариации?
2. Что такое дисперсия и математическое ожидание?
3. Чем отличается классификация на основе расстояния Махаланобиса от классификации на основе расстояния Евклида?
4. Что такое байесовский классификатор?
5. Что такое принцип максимума правдоподобия?

Лабораторная работа № 2

Классификаторы, основанные на оценки функции оптимизации

Цель и содержание работы: познакомить студента с персептроном.

Задачи:

1) Разобрать принцип действия и алгоритм работы персептрона;
2) Разобрать алгоритм минимизации среднеквадратической ошибки (LMS).

Теоретическое обоснование

Техника получения оптимального байесовского классификатора основывается на оценке функции распределения Гаусса для обучающих данных. Однако, это сложная задача особенно для многомерных данных. Альтернативное решение заключается в отделении классов в многомерном пространстве с помощью разделяющих гиперплоскостей.

Разделяющая гиперплоскость может быть записана в виде

$$w^T x + w_0 = 0. \quad (2.1)$$

Также (2.1) можно переписать в виде

$$w'^T x' = [w^T, w_0] \begin{bmatrix} x \\ 1 \end{bmatrix} = 0, \quad (2.2)$$

где **w** – вектор настраиваемых параметров, w_0 – свободный член, **x** – входной вектор.

С помощью таких гиперплоскостей можно отделить линейно отделимые классы.

Алгоритм адаптации вектора весовых коэффициентов элементарного персептрона можно сформулировать следующим образом.

Если n-ый элемент x(n) обучающего множества корректно классифицирован с помощью весовых коэффициентов w(n), вычисленных на n-ом шаге алгоритма, то вектор весов не корректируется, т.е. действует следующее правило:

$$\begin{aligned} w(n+1) &= w(n), \text{ если } w^T x(n) > 0 \text{ и } x(n) \in C_1 \\ w(n+1) &= w(n), \text{ если } w^T x(n) \le 0 \text{ и } x(n) \in C_2 \end{aligned} \quad (2.3)$$

В противном случае вектор весов персептрона подвергается коррекции в соответствии со следующим правилом:

$$\begin{aligned} w(n+1) &= w(n) - \eta(n)\, x(n), \text{ если } w^T(n) x(n) > 0 \text{ и } x(n) \in C_2 \\ w(n+1) &= w(n) + \eta(n)\, x(n), \text{ если } w^T(n) x(n) \le 0 \text{ и } x(n) \in C_1 \end{aligned}, \quad (2.4)$$

где C_1 и C_2 – линейно разделимые классы, а $\eta(n)$ – параметр скорости обучения.

Создадим функцию, которая будет обучать наш линейный нейрон (настраиваемую гиперплоскость). Сигнатура функции будет следующей: **[w, iter, mis_clas] = perce(X, y, w_ini. rho)**, где X – это матрица размером (l+1)xN (l – размер входного вектора, N – количество паттернов для обучения), y – вектор меток для класса C_1 (+1) и C_2 (-1), w_ini – вектор начальных параметров, которые нужно настроить в процессе обучения так, чтобы гиперплоскость отделяла C_1 от C_2, rho – [redacted]=const, т.е. скорость обучения, w – вектор, вычисленный алгоритмом, iter – количество итераций за которые был вычислен w, mis_clas – количество неправильно классифицированных векторов.

Рассмотрим пример.

Сгенерируем 4 множества X_i, каждое – это набор точек в двумерном пространстве. Каждая точка из X_i имеет метку -1, точки случайно разбросаны в квадрате [3, 5]x[3, 5], [2, 4]x[2, 4], [0, 2]x[2, 4], [1, 3]x[1, 3]. Точки, распложенные в квадрате [0, 2]x[0,2] имеют метку класса +1. Требуется визуально отобразить классы, обучить персептрон для [redacted]=0.01 и [redacted]=0.05, и начальном векторе параметров $[1, 1, -0.5]^T$.

Листинг функции, реализующий персептрон, приведён ниже.

```
function [w,iter,mis_clas]=perce(X,y,w_ini,rho)

[l,N]=size(X);
max_iter=20000; % Максимальное количество итераций,
% которые будет работать персептрон, после этого будет
% считаться, что решение не найдено.

w=w_ini;       % Инициализируем вектор параметров
iter=0;        % Счётчик итераций

mis_clas=N;    % Инициализируем количество неправильно
% классифицированных паттернов

% цикл while по двум условиям
while(mis_clas>0)&&(iter<max_iter)
   iter=iter+1;
   mis_clas=0;

   gradi=zeros(l,1); % Вычисление корректирующего компонента для вектора
   for i=1:N
      if((X(:,i)'*w)*y(i)<0)
         mis_clas=mis_clas+1;
         gradi=gradi+rho*(-y(i)*X(:,i));
      end
   end

   if(iter==1)
      fprintf('\n First Iteration: # Misclassified points = %g \n',mis_clas);
   end

   w=w-rho*gradi; % Обновление вектора параметров
```

```
end
```

Листинг основного кода приведён ниже.

```
close('all');
clear

rand('seed',0);

% Генерируем класс Xi
N=[100 100]; % 100 векторов для каждого класса
l=2; % Размерность входа каждого вектора

% для генерирования паттернов из X2, X3, X4 нужно вместо
% нижней строчки подставить одну из закомментированных строк
x=[3 3]';
% x=[2 2]'; для X2
% x=[0 2]'; для X3
% x=[1 1]'; для X4

% получаем 200 точек для Xi и для точек квадрата [0, 2]x[0, 2]
X1=[2*rand(l,N(1)) 2*rand(l,N(2))+x*ones(1,N(2))];
X1=[X1; ones(1,sum(N))];
y1=[-ones(1,N(1)) ones(1,N(2))]; % получаем метки

% 1. Выводим множество Xi
figure(1), plot(X1(1,y1==1),X1(2,y1==1),'bo',...
X1(1,y1==-1),X1(2,y1==-1),'r.')
figure(1), axis equal

% 2. Запускаем алгоритм обучения персептрна для эта=0.01
rho=0.01; % Скорость обучения
w_ini=[1 1 -0.5]'; % начальные веса
[w,iter,mis_clas]=perce(X1,y1,w_ini,rho)
```

Имеем 4 ситуации X_i, $i = 1..4$, которые нужно отделить с помощью персептрона (рисунок 2.1 – 2.4).

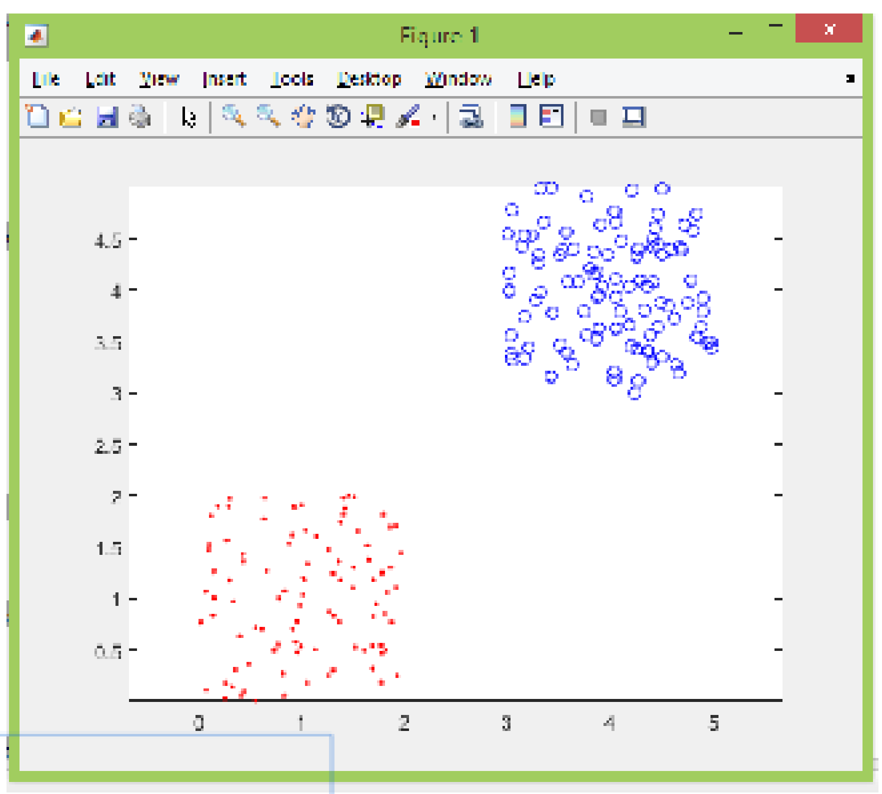

Рисунок 2.1 – Два класса для X_1

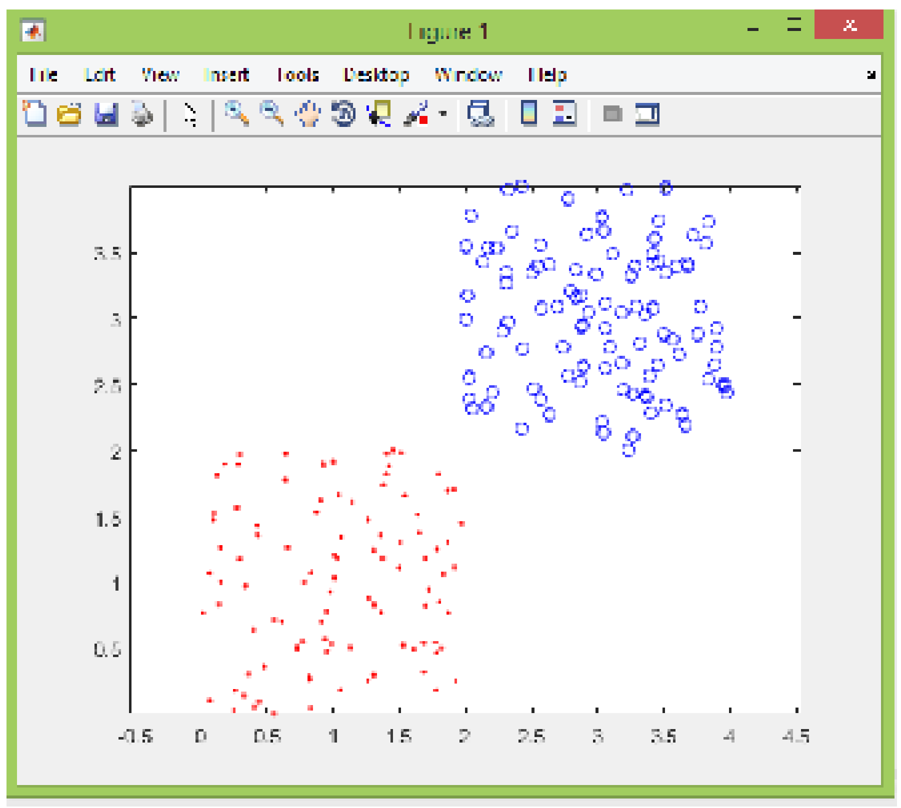

Рисунок 2.2 – Два класса для X_2

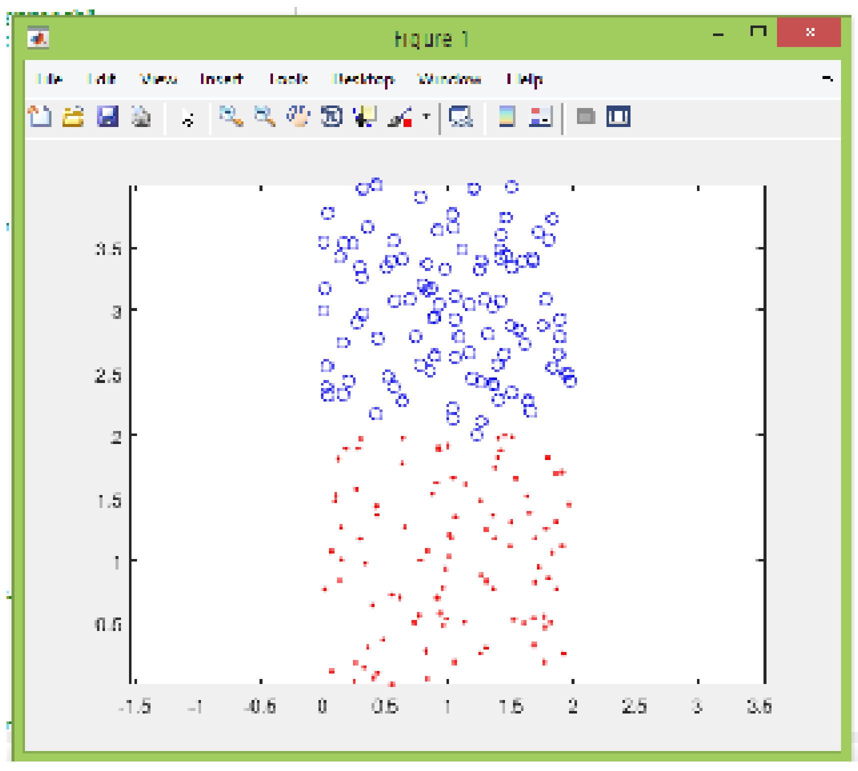

Рисунок 2.3 – Два класса для X_3

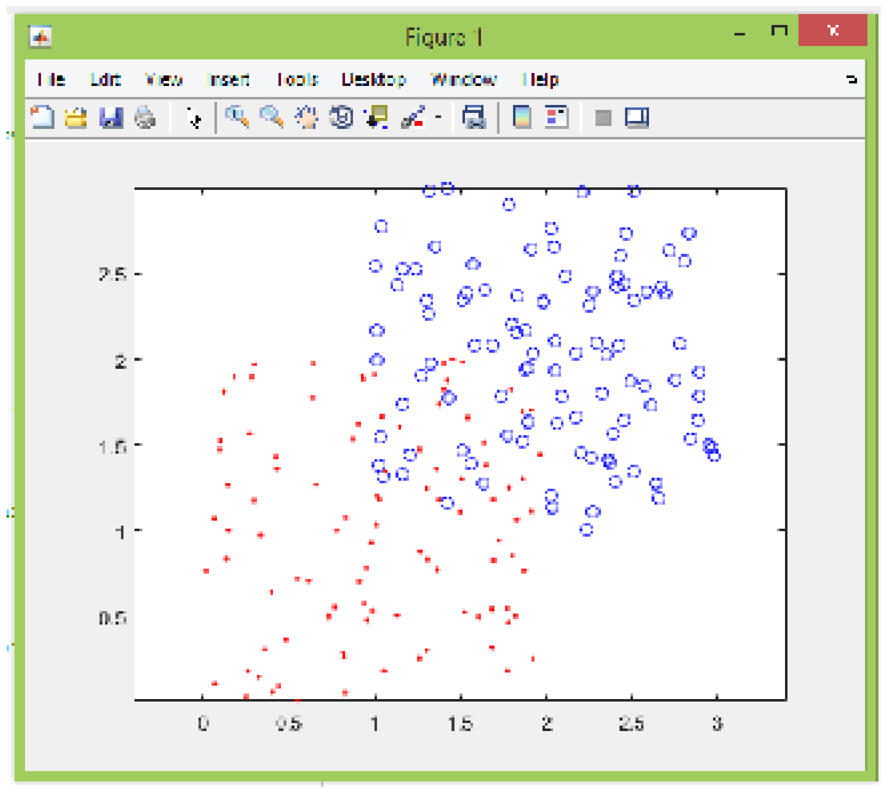

Рисунок 2.4 – Два класса для X_4

Видно, что в случае X_4 (рисунок 2.4) классы линейно не разделимы.

Результаты обучения приведены в таблице 2.1, 2.2.

Таблица 2.1 – Количество итераций, за которые персептрон находит решение в зависимости от [redacted]

	X_1	X_2	X_3	X_4
[redacted] = 0.01	134	134	5441	Оптимального решения не найдено

▬ = 0.05	5	5	252	Оптимального решения не найдено

Таблица 2.2 – Количество неправильно распознанных паттернов

	X_1	X_2	X_3	X_4
▬ = 0.01, ▬ = 0.05	0	0	0	25

Видно, что при увеличении ▬ увеличивается скорость обучения персептрона. В последнем случае оптимального решения персептрон не находит, поэтому проведённая прямая отделяет не все паттерны класса.

Функция **perce(X,y,w_ini,rho)** запрограммирована для пакетного режима, т.е. вектор **w** корректируется один раз после вычисления поправок для каждого примера. Ниже приведён листинг функции, где персептрон обучается в онлайн-режиме, где корректировки вычисляются для каждого примера, но далее, они не накапливаются, а сразу применяются к вектору весов.

```
function [w,iter,mis_clas]=perce_online(X,y,w_ini,rho)

[l,N]=size(X);
max_iter=10000000; % максимальное количество итераций
% требуется, если классы расположены близко друг к другу

w=w_ini;
iter=0;

mis_clas=N;

while(mis_clas>0)&&(iter<max_iter)
   mis_clas=0;

   for i=1:N
      if((X(:,i)'*w)*y(i)<0)
         mis_clas=mis_clas+1;
         w=w+rho*y(i)*X(:,i);  % Обновляем вектор весов
      end
      iter=iter+1;
   end

   if(iter==1)
      mis_clas
   end
end
```


Вызвать эту функцию можно с помощью следующей сигнатуры:

[w,iter,mis_clas]=perce_online(X1,y1,w_ini,rho).

Если поставить предыдущий эксперимент, то получим следующие результаты, таблица 2.3, 2.4.

Таблица 2.3 – Количество итераций, за которые персептрон находит решение в зависимости от ■

	X_1	X_2	X_3	X_4
■ = 0.01	600	600	6589400	Оптимального решения не найдено
■ = 0.05	400	400	7729200	Оптимального решения не найдено

Таблица 2.4 – Количество неправильно распознанных паттернов

	X_1	X_2	X_3	X_4
■ = 0.01, ■ = 0.05	0	0	0	6

Требуется значительно больше итераций, онлайн-обучение хуже сходится, зато в практическом плане более экономное, т.к. не требует специальных структур для хранения и накопления поправок.

Аппаратура и материалы. 64-разрядный (x64) персональный компьютер, процессор с тактовой частотой 1 ГГц и выше, оперативная память 1 Гб и выше, свободное дисковое пространство не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Matlab (R2013) и выше.

Указание по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения; ремонт персонального компьютера. Соблюдать правила технической эксплуатации и техники безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Создать два частично пересекающихся класса точек (100 точек для каждого класса) как показано в таблице 2.5. Разделить их насколько это возможно с помощью персептрона. Тип обучения персептрона онлайн или пакетный указан в таблице. Нарисовать найденное решение.

Таблица 2.5 – Индивидуальные варианты

1	A Онлайн обучение. 0 0 B
2	0 Пакетное обучение. 0 A B
3	0 Пакетное обучение. 0 B A
4	A Онлайн обучение. 0 0 B
5	0 Пакетное обучение. 0 B A
6	0 Пакетное обучение. 0 A B
7	A Онлайн обучение. 0 0 B
8	0 Онлайн обучение. 0 A B
9	A Пакетное обучение. 0 0 B
10	0 Онлайн обучение. 0 B A

Содержание отчета и его форма

Отчёт по лабораторной работе должен содержать следующую информацию:

1) Название лабораторной работы и её номер.
2) ФИО и группу студанта.

3) Формулировка индивидуального задания и номер варианта.
4) Ответ на задание в виде кода и результатов работы Matlab.

Вопросы для защиты работы

1. Что такое персептрон?
2. Чем персептрон отличается от сети прямого распространения?
3. Какие задачи можно решать с помощью персептрона?

Лабораторная работа № 3

Создание и обучение нейронной сети на языке высокого уровня среды Matlab

Цель и содержание работы: создать и обучить несколько нейронных сетей, решающих задачи линейного и нелинейного разделения классов, используя встроенный язык программирования Matlab.

Задачи:

- освоить базовые команды для создания и обучения простых сетей в Matlab;
- научиться создавать и обучать сети для линейного и нелинейного разделения классов;
- научиться строить графики, отражающие результаты работы сетей.

Теоретическое обоснование

Самый гибкий способ по работе с ИНС в Matlab – это использовать встроенный язык программирования для создания и обучения сетей. В данной лабораторной работе на примере линейного разделения классов и задачи XOR будет рассмотрена техника создания и обучения разных нейронных сетей.

Код Matlab будем выделять жирным шрифтом. Комментарий к коду записывается после символа «%» и идёт до конца строчки. При желании, можно получить более подробную справку о любой функции, установив курсор на нужной и нажав клавишу «F1».

Прежде, чем программировать персептрон, рассмотрим пример построения поверхности отклика для обычного нелинейного нейрона с функцией активации – гиперболический тангенс.

```
% Готовим Matlab к работе
close all, clear all, clc, format compact;
% Задаём веса нейрона
w = [4 -2];
% Задаём смещение
b = -3;
% Задаём функцию, которую хотим построить, гипербол. тангенс
func = 'tansig';
% Задаём входной вектор
p = [2 3];
% Вычисляем взвешенную сумму входа и весов
activation_potential = p*w'+b;
% Вычисляем выход нелинейной части нейрона
```

```
neuron_output = feval(func, activation_potential);
% Задаём входной вектор для диапазона от -10 до +10 с шагом 0.25
[p1, p2] = meshgrid(-10:.25:10);
% Вычисляем вектор выхода нейрона
z = feval(func, [p1(:) p2(:)]*w'+b);
% Пересчёт вещественных чисел в воксели для 3D-графика
z = reshape(z, length(p1), length(p2));
% Строим график
plot3(p1, p2, z);
% Включаем сетку
grid on;
% Подписываем оси графика
xlabel('Input 1');
ylabel('Input 2');
zlabel('Neuron output');
```

Разберём код. Команда **close all** закрывает все вспомогательные окна, которые могли быть открыты (окна различных мастеров, вроде nntool, не закрываются). **Clear all** удаляет все переменные из области workspace, **clc** – очищает область окна Command window, где будет набираться код. **Format compact** устанавливает формат отображения для чисел, так для вещественных числе будет отображаться 4 точки после запятой.

Веса задаются обычным вектором **w = [4 -2]**, как видно, значения отделяются друг от друга пробелом, указывать тип не обязательно. Функция активации задаётся строкой **'tansig'**. Далее идёт вычисление взвешенной суммы: скалярное произведение входа на веса и прибавка смещения. Вектора **p** и **w** нельзя просто так взять и перемножить, т.к. по правилам линейной алгебры один из них должен быть транспонирован, что и делается с вектором **w** с помощью **w'**. Можно посмотреть значение переменной, оно должно быть -1. Далее вычисляем функцию активации при входе равном -1. Делаем это через **feval()**. Как понятно из названия, функция вычисляет значение некоторой функции (первый параметр) от вектора входов (второй параметр). Причём, как и многие другие фунции Matlab, она перегружена, т.е. её вызов может происходить при разных способах записи второго параметра. В первом случае вектор редуцируется к скалярной величине, выход равен -0.7616.

Далее присваиваем переменным **p1** и **p2** значения от -10 до +10 с шагом 0.25. Делаем это через **meshgrid()**, которая создаёт прямоугольную сетку в специальном формате пригодном для построения графиков. Но для построения самого графика нам нужно знать значения не только **p1** и **p2**, но и выхода нейрона. Теперь вычисляем эти значения для вектора **z** с помощью **feval()**, видно, что теперь второй параметр напрямую записывается в виде скалярного произведения двух векторов плюс смещение. Двоеточие означает перевод значения в формат double (используется для научных и инженерных вычислений. Никогда не используйте float для этих задач в таких языках как C/C++, чтобы избежать переполнения переменной).

Функция **reshape()** пересчитывает числовые значения в воксели для отображения на графике. **Plot3()** строит трёхмерный график, рисунок 3.1. **Grid on** включает сетку на этом графике, рисунок 3.2, 3.3. **Xlabel()**, **ylabel()**, **zlabel()** – подписывают оси у графика.

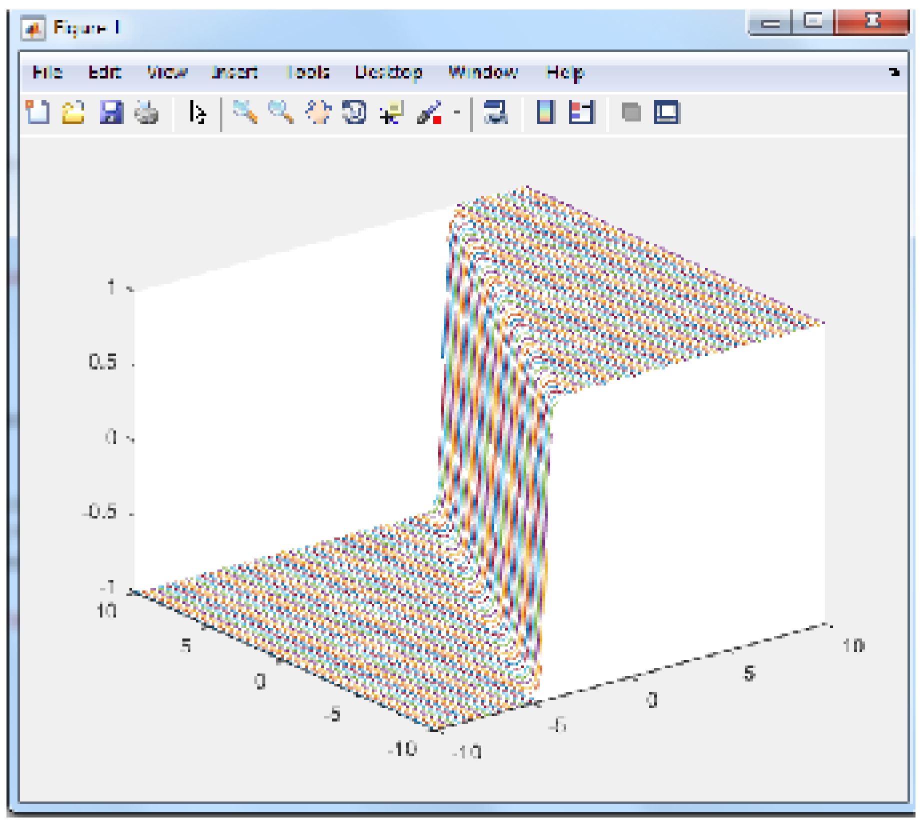

Рисунок 3.1 – Получившийся 3D-гарфик без сетки и подписей осей

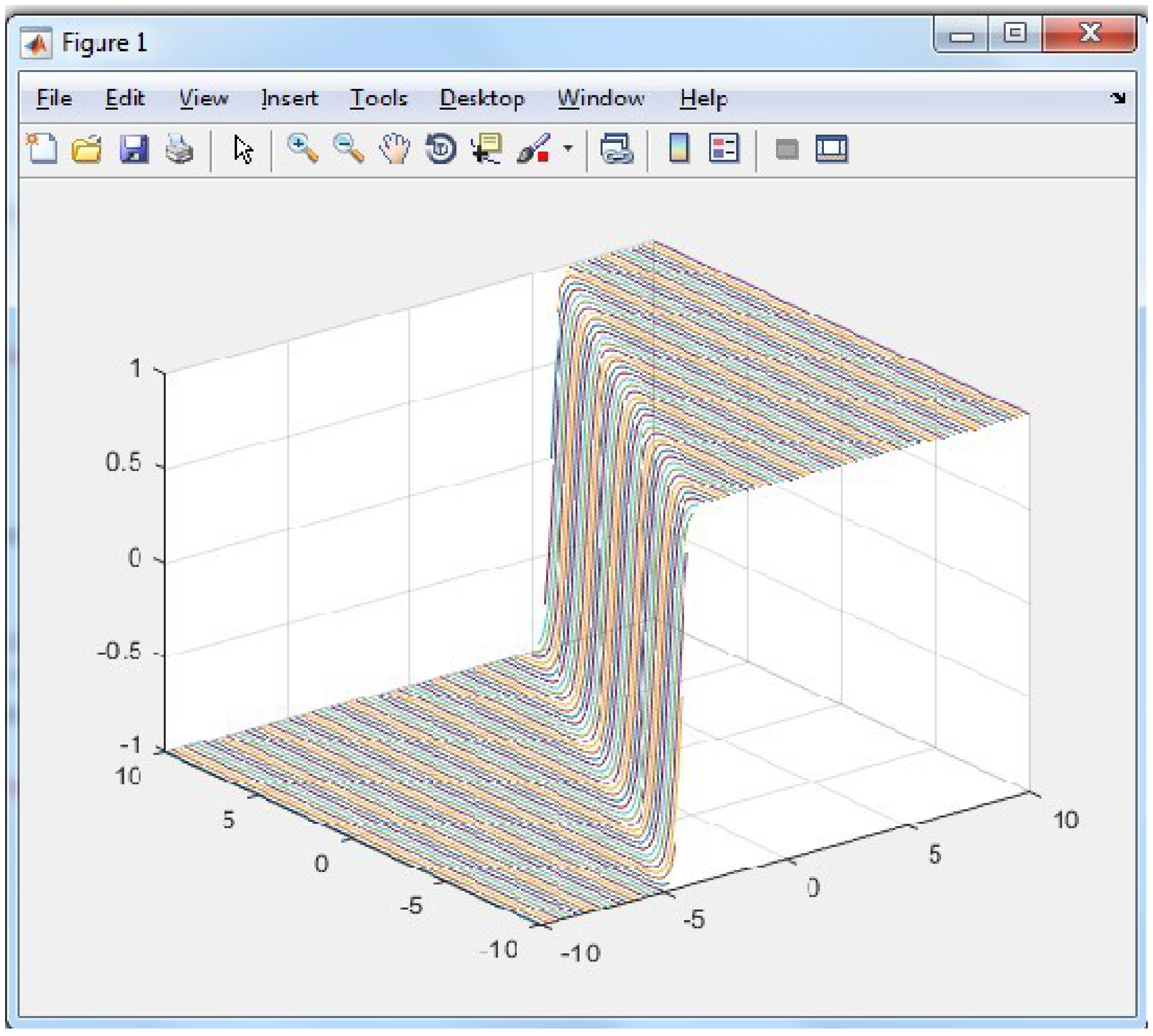

Рисунок 3.2 – Добавили сетку на фон

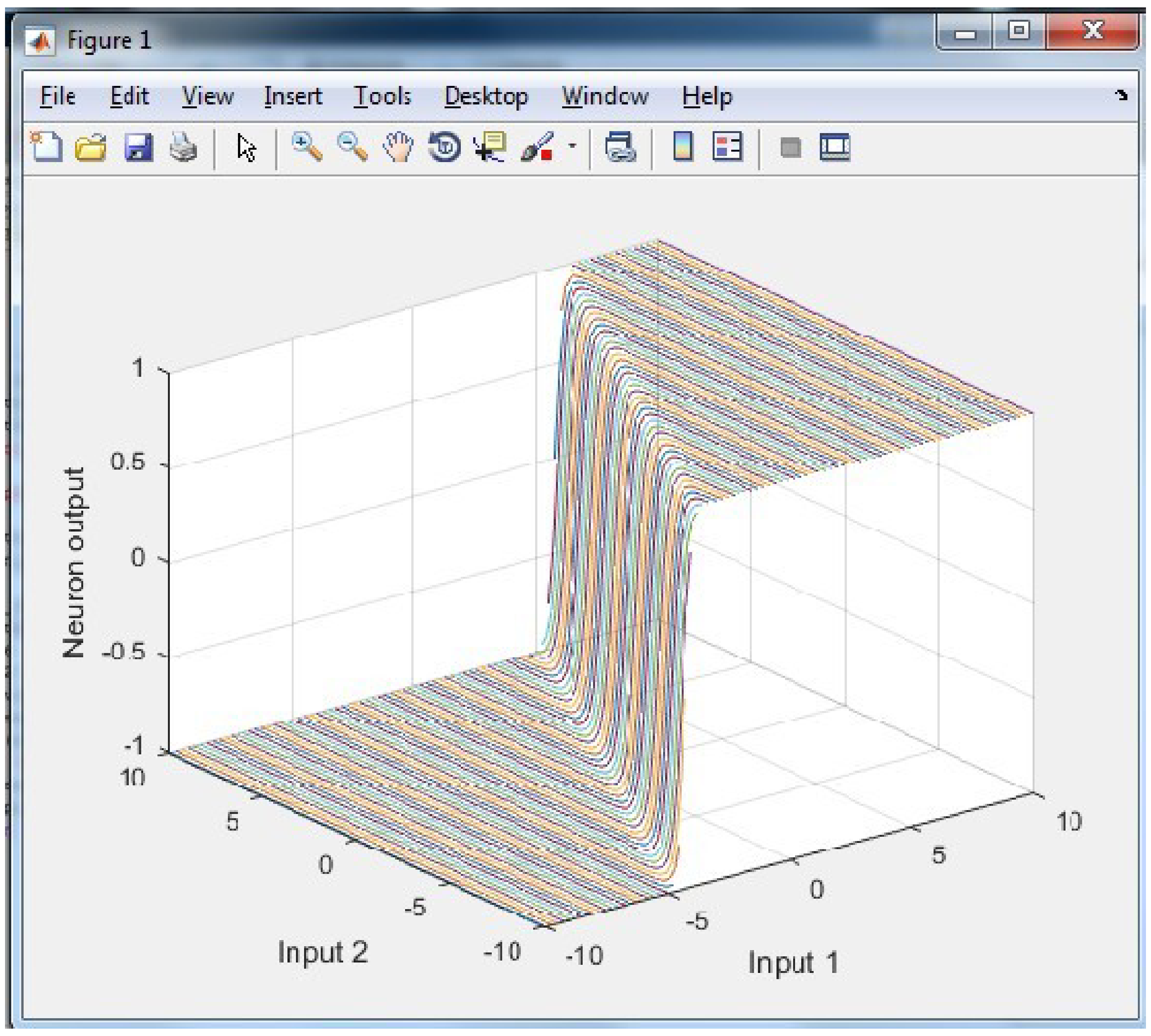

Рисунок 3.3 – Добавили подписи для осей x, y, z

Теперь создадим и обучим двухслойную сеть на одном примере.

```
close all, clear all, clc, format compact
% Входной транспонированный вектор
inputs = [1:6]'
% Выходной вектор, который должна промоделировать сеть
outputs = [1 2]';
% Задаём общую структуру сети
net = network(1, 2, [1; 0], [1; 0], [0 0; 1 0], [0 1]);
% Показываем её
view(net);
% Задаём количество промежуточных слоёв
net.layers{1}.size = 5;
% Задаём функции активации на них
net.layers{1}.transferFcn = 'logsig';
% Ещё раз отображаем сеть
view(net);
% Устанавливаем размерность входа и выхода
net = configure(net, inputs, outputs);
% Смотрим окончательный вариант сети
view(net);
% Получаем изначальный выход сети без обучения
initial_output = net(inputs);
% Устанавливаем метод обучения
net.trainFcn = 'trainlm';
% Устанавливаем функцию ошибки
net.performFcn = 'mse';
% Обучаем сеть на одном примере
net = train(net, inputs, outputs);
% Опять подаём вход, но уже на обученную сеть,
% сеть должна промоделировать выход [1 2]
final_output = net(inputs);
```

Входной вектор будет равен массиву из шести элементов от 1 до 6, т.к. диапазон записывается через «:». Функция **network()** задаёт общую структуру сети. Рассмотрим более подробно, что означает каждый параметр.

Первый параметр означает количество входов (не в смысле размерность входного вектора, а в смысле, – сколько векторов будет подано на вход сети. Допустим, в сверточных сетях обычное дело, что на вход подаётся не один, а два или три массива, каждый из которых как-то связан с дальнейшим слоем). У нас этот параметр равен 1. Второй параметр означает количество слоёв в сети. В данном случае имеем двухслойную сеть. Третий параметр – булев вектор, который означает какие нейроны будут иметь смещение, а какие – нет. Т.к. вектор [1; 0], то очевидно, что все нейроны первого слоя будут иметь смещение, а нейроны второго слоя – нет. Четвёртый параметр – это также булев вектор, который означает, – с какими слоями связан вход. В данном случае используется классическая схема, где вход связна только с последующим слоем, а следовательно, вектор [1; 0]. Пятый параметр – это матрица связей между слоями, которая означает, какой слой с каким связан. Она записывается в виде вектора. Имеем [0 0; 1 0], если записать в виде матрицы, то получится $\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$. Тогда понятно, что первый слой связан со вторым слоем, т.к. единица располагается в позиции (1, 2), где 1 – это номер столбца, 2 – номер строки. Шестой параметр [0 1] – булев вектор, который показывает, с какими слоями связан выход. При таких значениях выход связан с предпоследним слоем.

При такой конфигурации получим структуру сети как на рисунке 3.4.

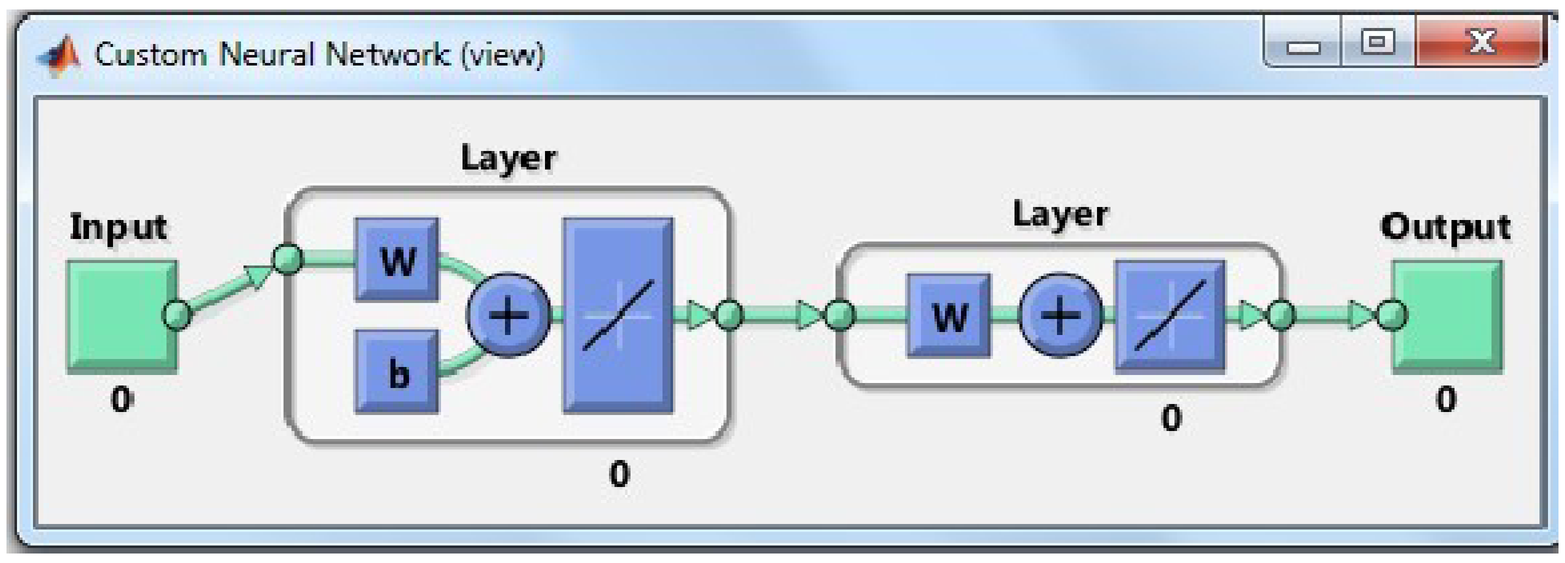

Рисунок 3.4 – Изначальная структура сети

Изменим третий параметр с [1; 0] на [1; 1], получим структуру как на рисунке 3.5.

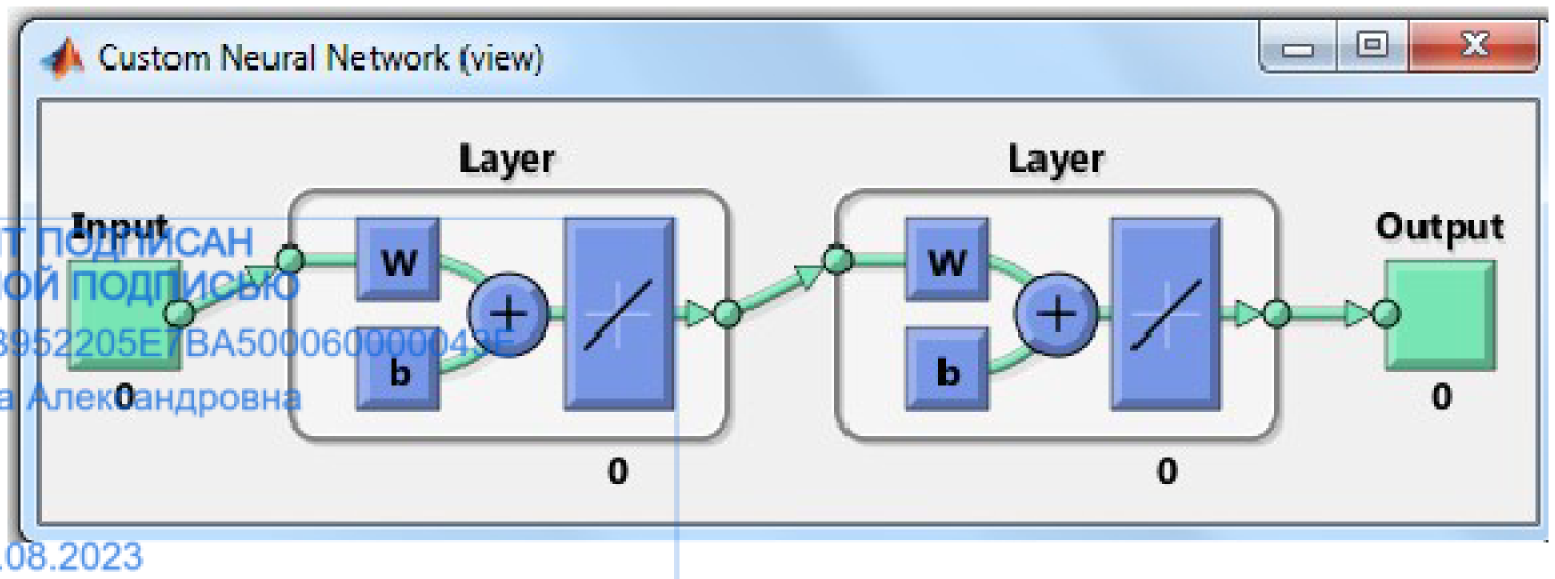

Рисунок 3.5 – Видно, что у второго слоя тоже появилось смещение

Изменим теперь и четвёртый параметр на [1; 1], получим структуру сети как на рисунке 3.6.

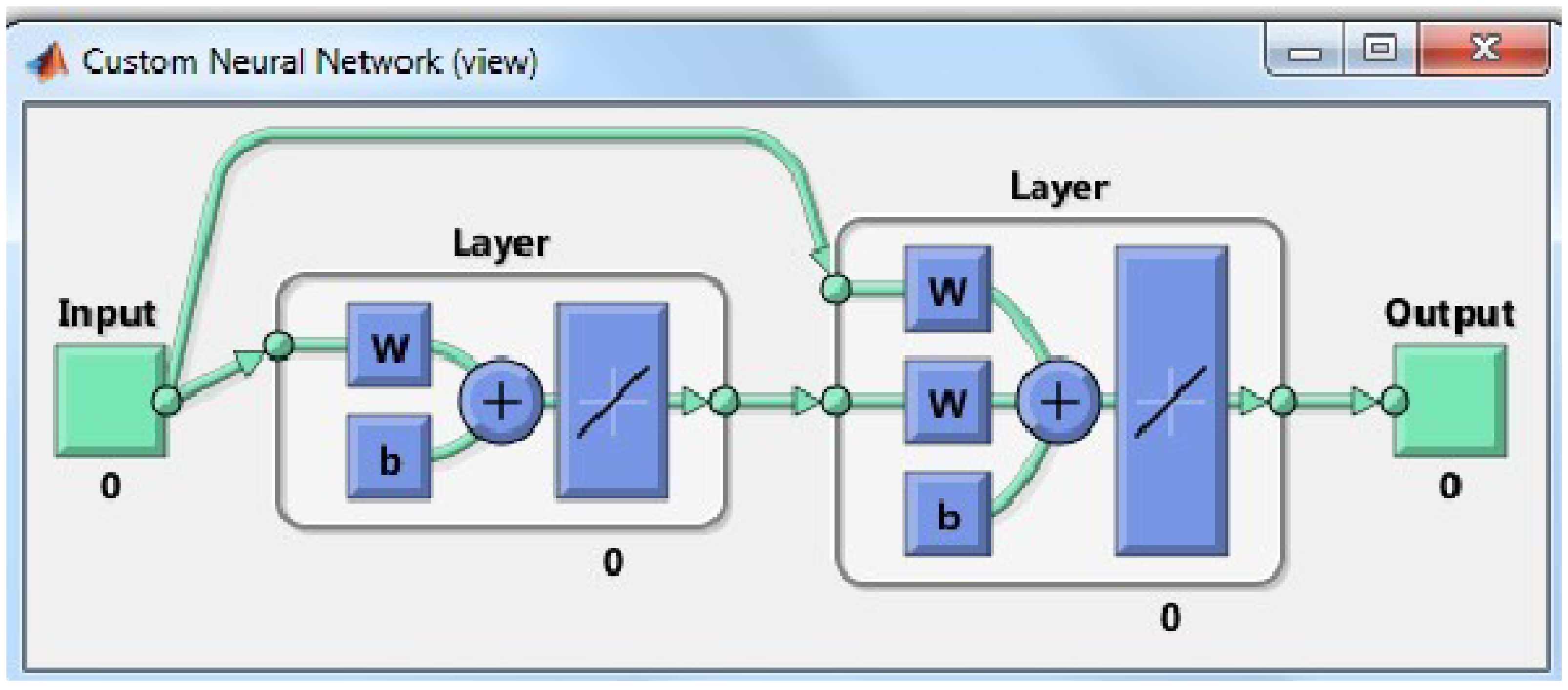

Рисунок 3.6 – Добавилась связь входа со вторым слоем

Изменим остальные два вектора на единичные, получим структуру как на рисунке 3.7.

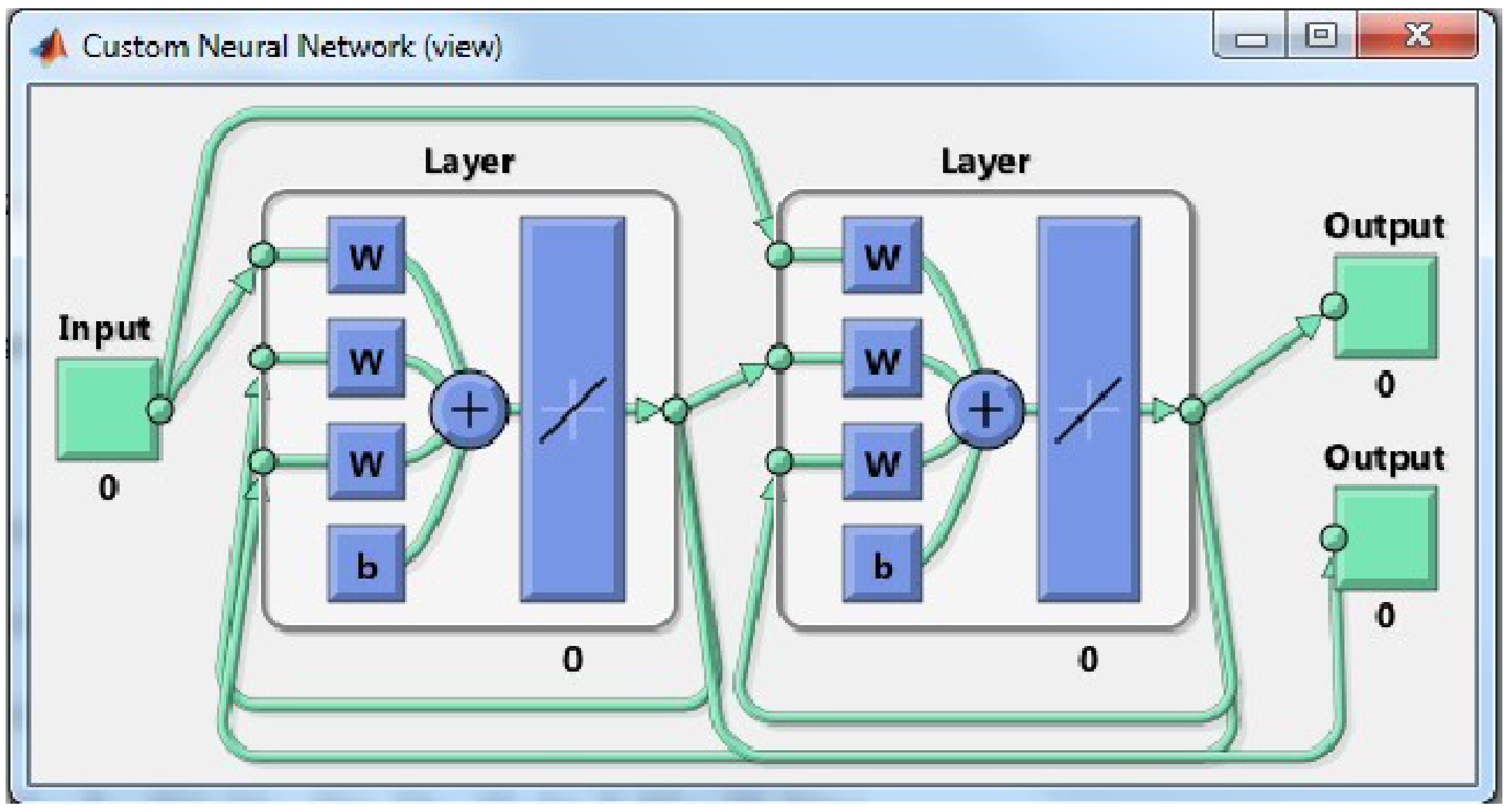

Рисунок 3.7 – Появился второй выход, связанный с первым слоем, а также каждый слой связан с самим собой и второй слой связан с первым (подаёт свои сигналы на вход первого слоя)

Net.layers{1}.size = 5 – присваиваем количество нейронов первому слою сети (индекс записывается в фигурных скобках). Видно, что обращение к данным параметрам осуществляется классическим образом для объектов: через точку. Т.е. сеть в Matlab – это объект (по терминологии объектно-ориентированного программирования (ООП)).

Небольшое примечание. Концепция ООП напрашивается для описания ИНС, т.к. вполне логично, что объект «сеть» включает в себя объекты – «слой», а те – «нейрон». Однако, нужно помнить, что при таком подходе оперативная память выделяется и группируется под конкретные объекты и в случае попыток быстрой реализации ИНС, возможно, с использованием ассемблера, не получится быстро реализовывать матричные операции. Короче, если требуется быстрая реализация сети, то от ООП-описания придётся отказаться.

Net.layers{1}.transferFcn = 'logsig' – присваивает слоям первого слоя функцию активации сигмойд (без отрицательных значений), рисунок 3.8.

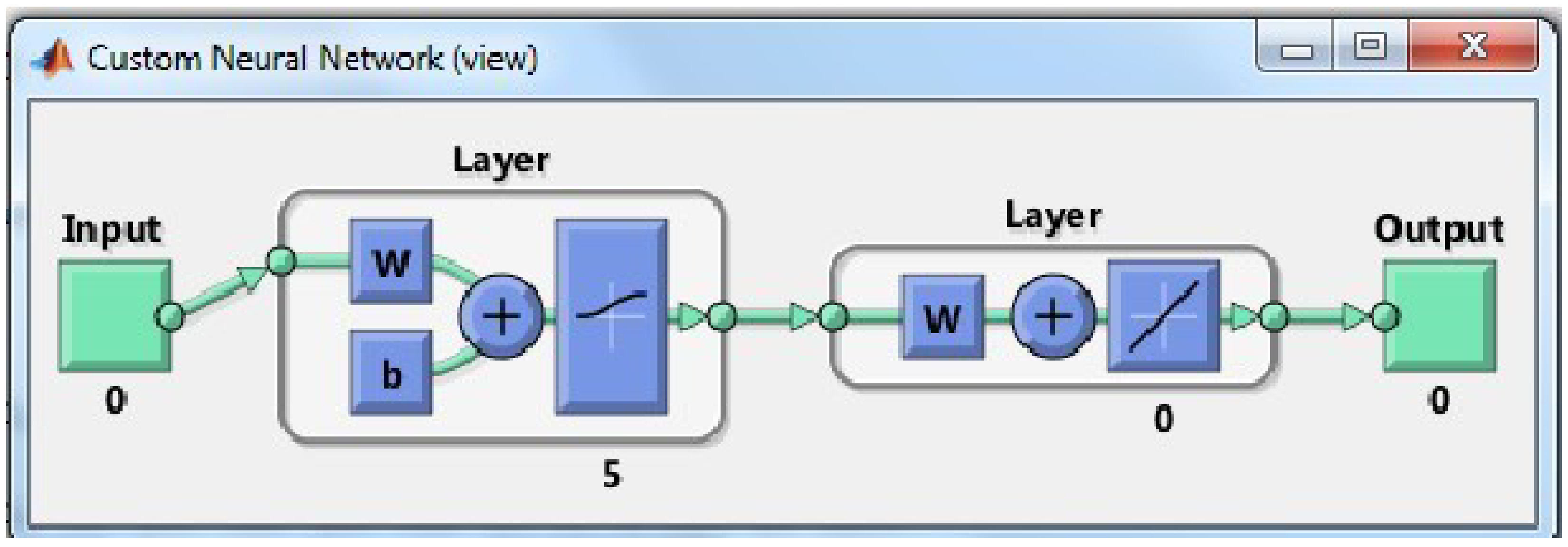

Рисунок 3.8 – Меняем функцию активации у нейронов первого слоя

Для установки входного и выходного слоя можно сконфигурировать сеть по отношению к вектору входа и выхода: **net = configure(net, inputs, outputs)**, рисунок 3.9.

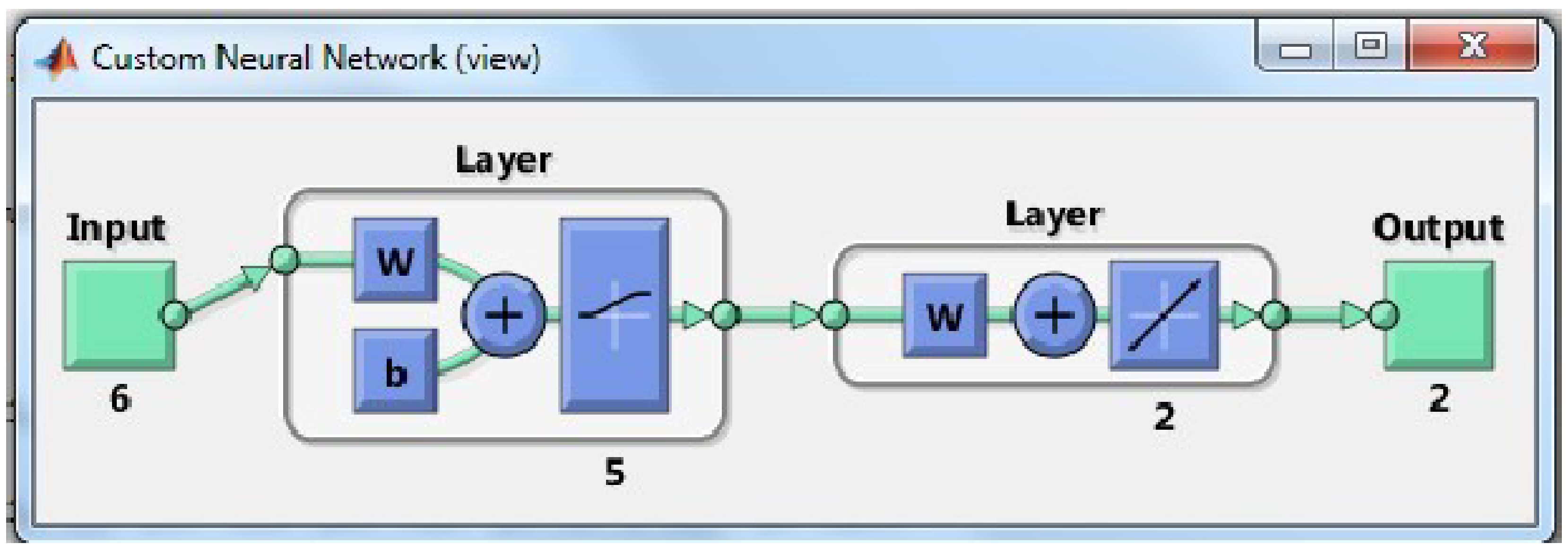

Рисунок 3.9 – Окончательно получаем общую структуру сети, где размер входа – 6, а выхода – 2

Вектор **initial_output** получит значения $[0\ 0]^T$, т.к. веса не инициализированы, т.е. сеть ничего делать не умеет. Для начала обучения нужно как минимум задать метод обучения и функцию ошибки, что и делается с помощью **net.trainFcn = 'trainlm'** и **net.performFcn = 'mse'**. **Trainlm** – это обучение по методу Левенберга-Марквардта, а **MSE** (mean square error) – это среднеквадратическая ошибка. **Train()** – запускает обучение сети и возвращает объект – обученную сеть. Процесс обучения показан на рисунке 3.10.

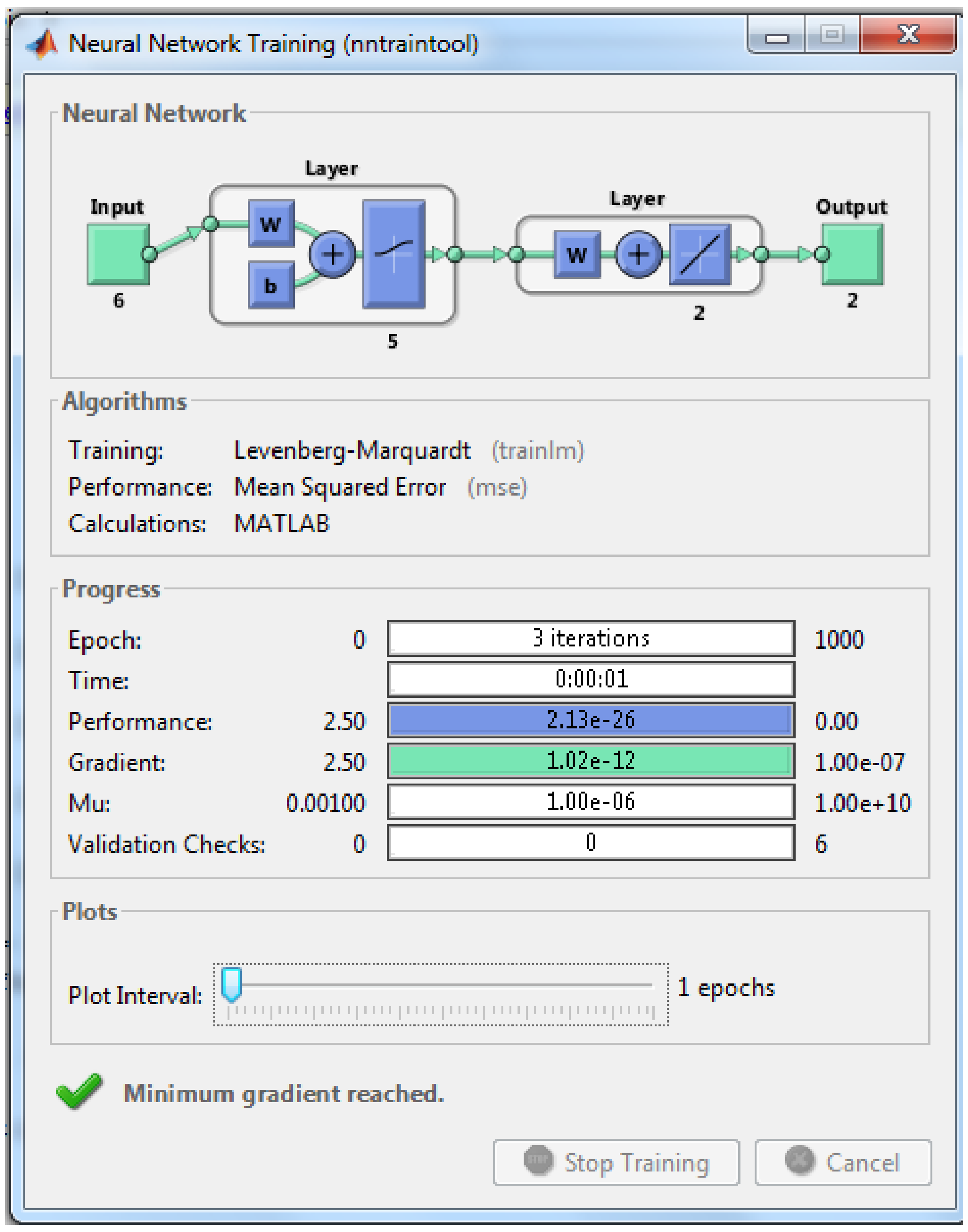

Рисунок 3.10 – Обучение сети

Позже формируем вектор **final_output**, который содержит ответ сети на вход, т.е. 1.0000 и 2.0000 (помним про формат **compact**).

Теперь решим задачу о разделении классов с помощью персептрона, подобную той, которая была решена в лабораторной работе 3.

```
close all, clear all, clc, format compact
% Задаём количество паттернов каждого класса
N = 20;
% Задаём смещение
offset = 5;
% Создаём входные значения каждого класса
x = [randn(2, N) randn(2, N)+offset];
% Создаём выходные значения для этих классов
y = [zeros(1, N) ones(1, N)];
% Открываем окно для рисования
figure(1);
% Наносим точки (паттерны)
plotpv(x, y);
% Создаём объект-сеть типа Персептрон
net = perceptron;
% Обучаем персептрон
net = train(net, x, y);
% Рисуем общую структуру сети
view(net);
% Возвращаемся в окно для рисования и рисуем прямую линию,
% которая разделит два класса
figure(1);
```

plotpc(net.IW{1}, net.b{1});

rand(2, N) – создаёт массив размером 2xN, числа которого являются случайные величины, распределённые по нормальному закону с математическим ожиданием 0 и среднеквадратическим отклонением 1. Таким образом, будут созданы две матрицы 2xN, содержащие координаты точек-паттернов в двумерном пространстве, причём координаты второй матрицы будут смещены на 5 позиций, т.е. заранее гарантируется, что классы будут линейно разделимые. Обратим внимание, что эти две матрицы на самом деле записаны в одну матрицу размером 2x2*N. **Zeros(1, N)** и **ones(1, N)** создаёт массив нулей и единиц размером N, это метки для входных паттернов.

Figure(1) – вызывает окно, содержащее канву для рисования, рисунок 3.11.

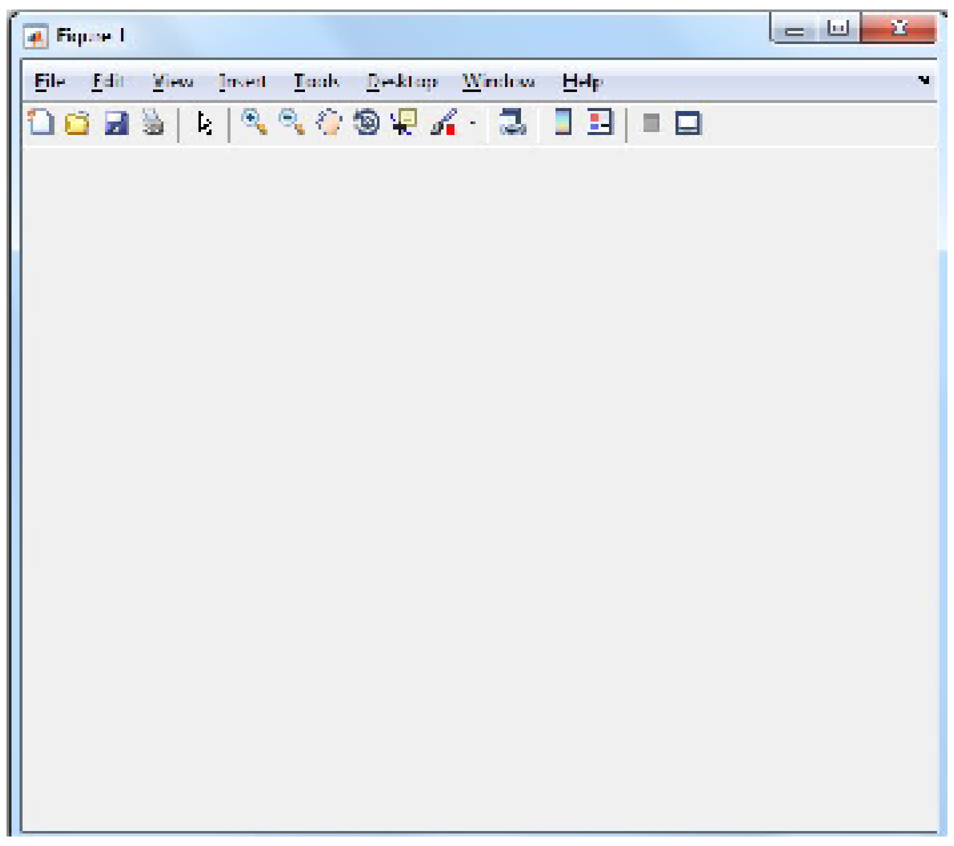

Рисунок 3.11 – Окно для рисования

Plotpv(x, y) – наносит точки на канву с координатами x и метками y, рисунок 3.12. Причём нули отображаются кружками, а единицы плюсами.

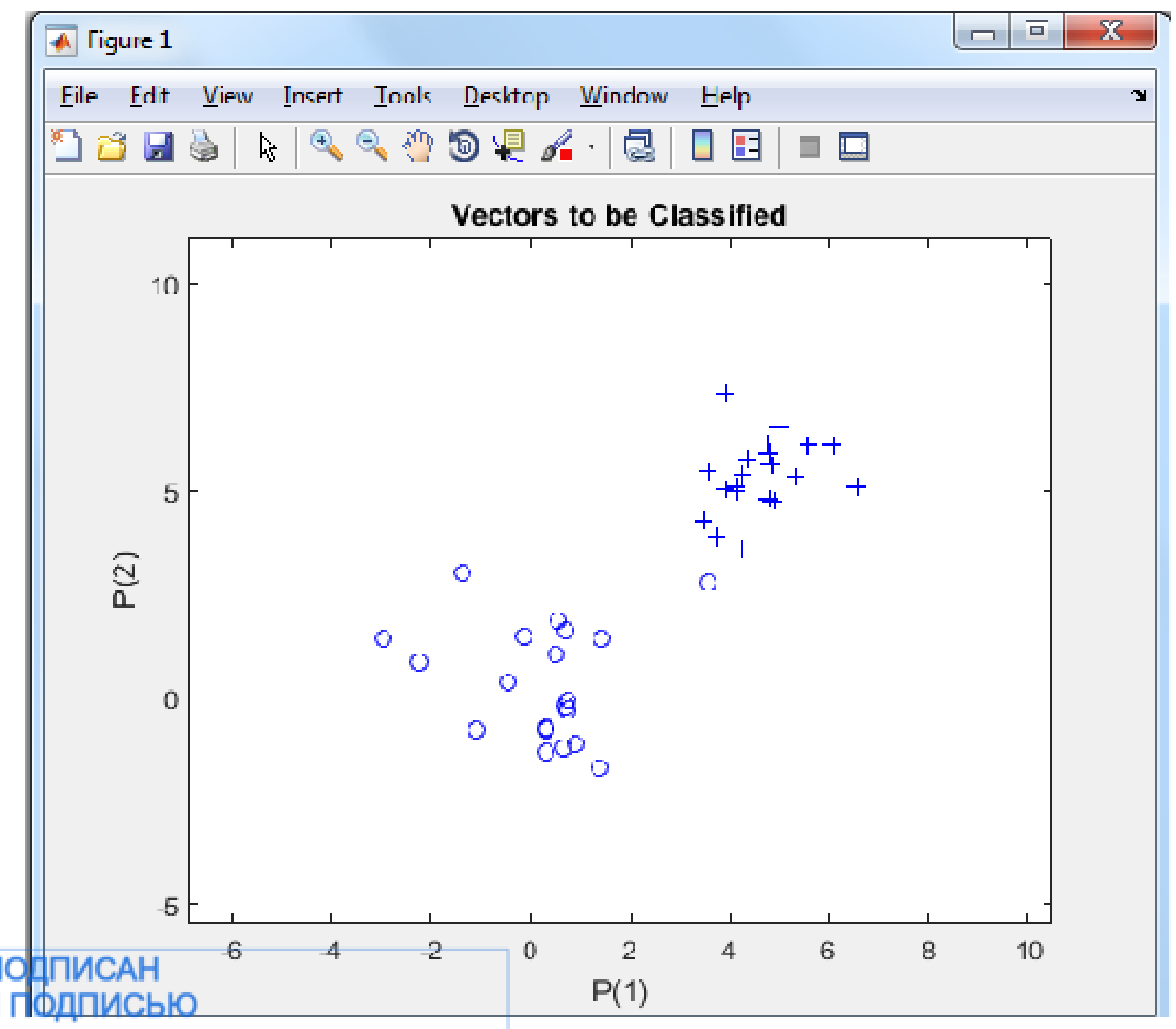

Рисунок 3.12 – Два созданных нами класса, видно, что они линейно разделимы

Net = perceptron – создаём сеть типа Персептрон, если после записи команды не ставить точку с запятой, то среда развернёт все установленные по умолчанию параметры, рисунок 3.13.

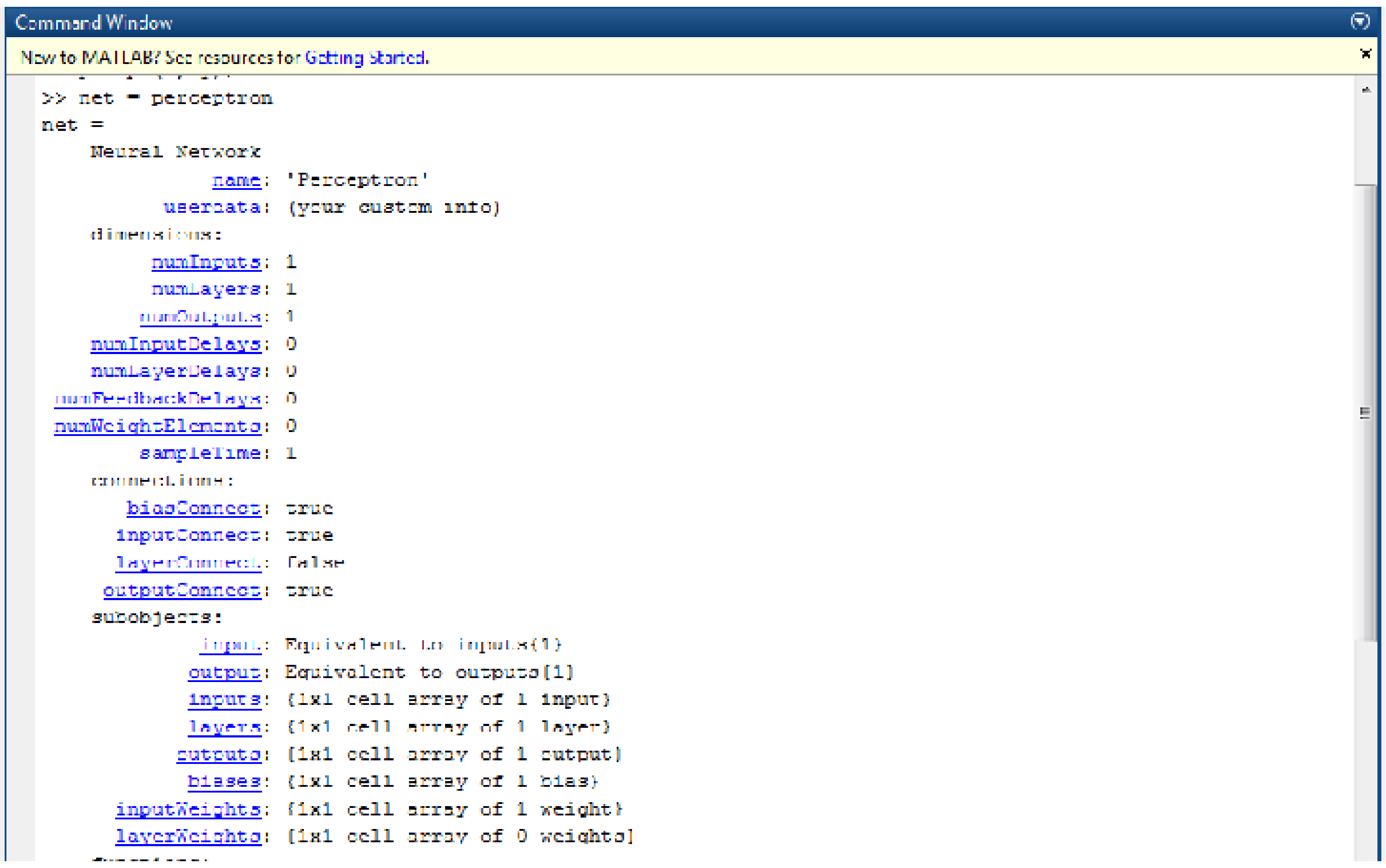

Рисунок 3.13 – Методы и свойства объекта perceptron

Далее происходит обучение сети. Общая структура сети представлена на рисунке 3.14.

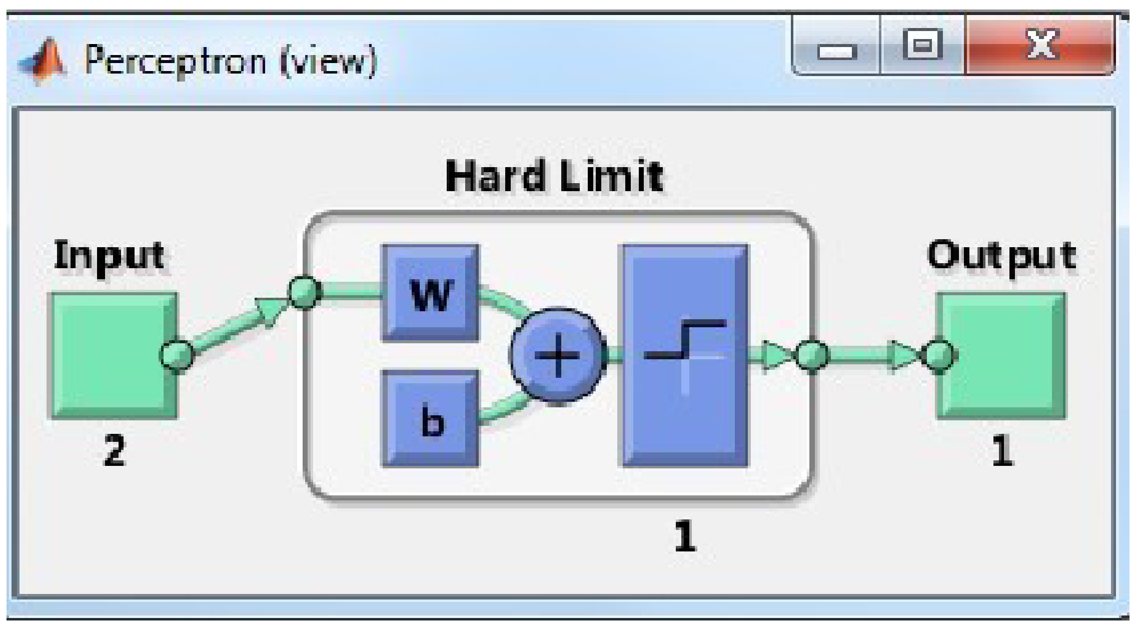

Рисунок 3.14 – Общая структура сети

Результаты обучения представлены на рисунке 3.15. Видно, что потребовалось 29 итераций.

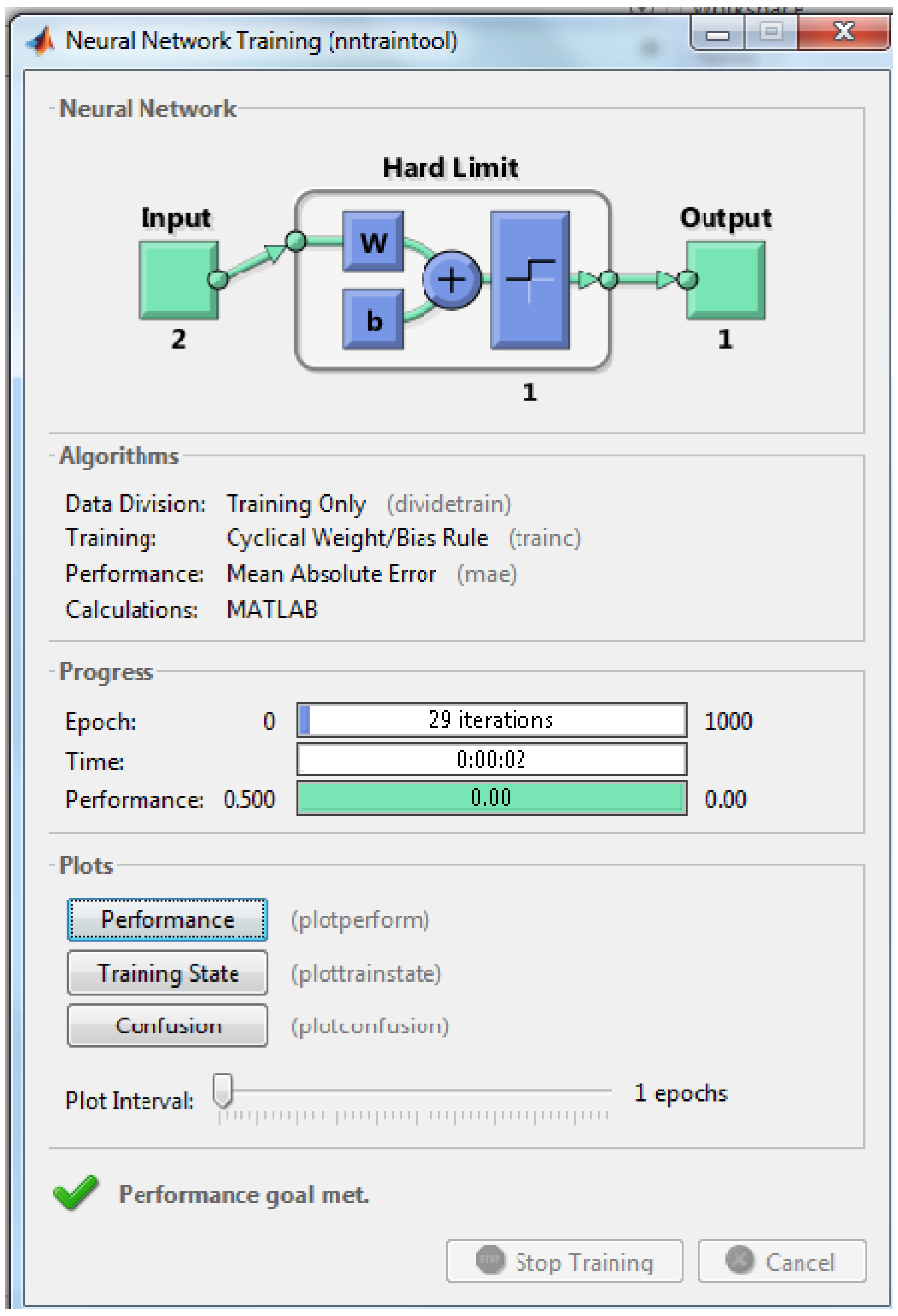

Рисунок 3.15 – Результаты обучения персептрона

На рисунке 3.16 представлен график настройки персептрона. График носит характерную зубчатую форму. После каждой эпохи высчитывается MSE между предполагаемыми ответами и реальными. Напомним, что при адаптации не используется спуск по поверхности ошибки, поэтому ждать постепенного снижения ошибки обучения не следует. Обучение идёт до тех пор, пока не будет найдена первая прямая, отделяющая один класс от другого.

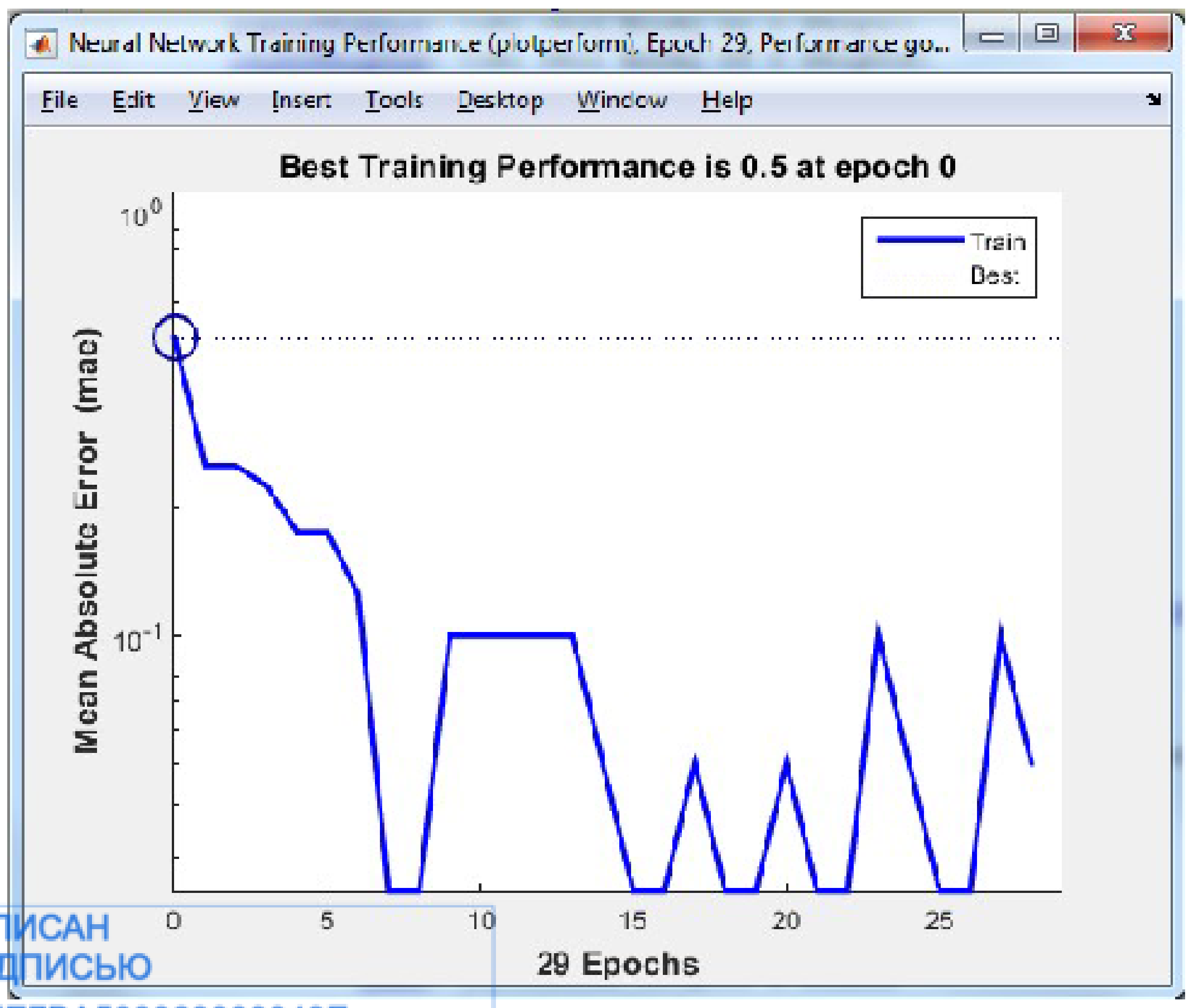

Рисунок 3.16 – График настройки весовых коэффициентов персептрона

Plotpc(net.IW{1}, net.b{1}) – строит прямую линию с соответствующими коэффициентами. **IW** – обозначает слой коэффициентов между входом и первым слоем, **b{1}** – смещение первого слоя. Итоговая прямая показана на рисунку 3.17.

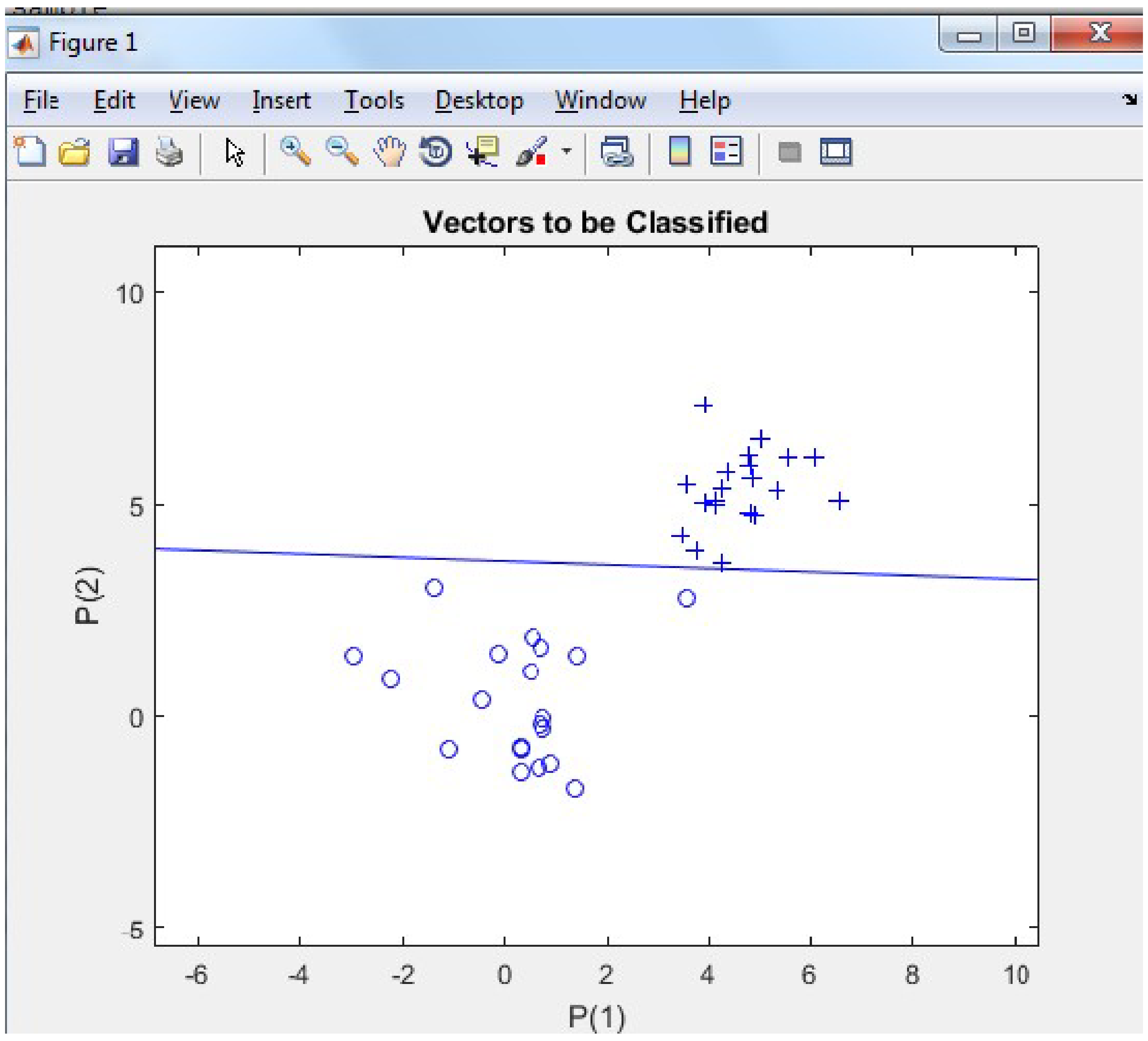

Рисунок 3.17 – Полученное решение по разделению двух классов

Теперь рассмотрим решение задачи разделения на 4 класса с помощью персептрона и обычной многослойной сети прямого распространения. Расположение классов оставим таким же, как и в задаче XOR (по углам «квадрата»), но теперь у нас не два класса, а 4, а также каждый класс представлен 30 паттернами.

Решение этой задачи с помощью персептрона.

```
close all, clear all, clc, format compact
% Каждый класс имеет 30 паттернов
K = 30;
% Смещение для классов 0.6
q = .6;
% Задаём 4 класса в виде векторов A, B, C, D (2 строки и 30 столбцов)
A = [rand(1, K)-q; rand(1, K)+q];
B = [rand(1, K)+q; rand(1, K)+q];
C = [rand(1, K)+q; rand(1, K)-q];
D = [rand(1, K)-q; rand(1, K)-q];
% Рисуем на канве точки A определённого типа (третий параметр)
plot(A(1, :), A(2, :), 'bs');
% Фиксируем канву
hold on;
grid on;
% На той же канве рисуем остальные классы
plot(B(1, :), B(2, :), 'r+');
plot(C(1, :), C(2, :), 'go');
plot(D(1, :), D(2, :), 'm*');
```

```
text(.5-q, .5+2*q, 'Class A');
text(.5+q, .5+2*q, 'Class B');
text(.5+q, .5-2*q, 'Class C');
text(.5-q, .5-2*q, 'Class D');
% Задаём кодировку классов (метки)
a = [0 1]';
b = [1 1]';
c = [1 0]';
d = [0 0]';
% Получаем общий вектор входных значений
P = [A B C D];
% Получаем общий вектор меток
T = [repmat(a, 1, length(A)) repmat(b, 1, length(B)) repmat(c, 1, length(B)) repmat(d, 1, length(D))];
% Сеть типа «персептрон»
net = perceptron;
% Пусть среднеквадратическая ошибка не равна 0
E = 1;
% Установка параметра о том, что будет минимум один прогон
net.adaptParam.passes = 1;
% Построили изначально ответ персептрона, до обучения
linehandle = plotpc(net.IW{1}, net.b{1});
% Счётчик по циклам
n = 0;
% Цикл будет идти пока среднеквадратическая ошибка не
% станет равной нулю или пока не достигнем значения итераций в 1000
while (sse(E) & n<1000)
    n = n + 1;
    % Обучить персептрон
    [net, Y, E] = adapt(net, P,
    T);
    % Нарисовать линии
    linehandle = plotpc(net.IW{1}, net.b{1}, linehandle);
    drawnow;
end
view(net);
% Проверка работы персептрона на входном векторе
p = [0.7; 1.2];
% Выдаст класс (1; 1)
y = net(p);
```

rand(1, K) – генерирует числа из равномерного распределения в виде вектора (первый параметр) размером K, в итоге **A = [rand(1, K)-q; rand(1, K)+q]** – генерируется матрица 2xK, что соответствует координатам точек.

Plot(A(1, :), A(2, :), 'bs') – рисует на канве точки из матрицы A, переводя их в вещественный формат. Параметр **'bs'** – это цвет и форма точек: b – синий и s – квадрат. **'r+'** – красный (r) крест (+), **'go'** – зелёный (g) кружок (o), **'m*'** – пурпурная (m) звездочка (*).

Text(.5-q, .5+2*q, 'Class A') – наносит текст на канву с координатами, записанными как два первых параметра.

Общий вид отображения классов представлен на рисунке 3.18.

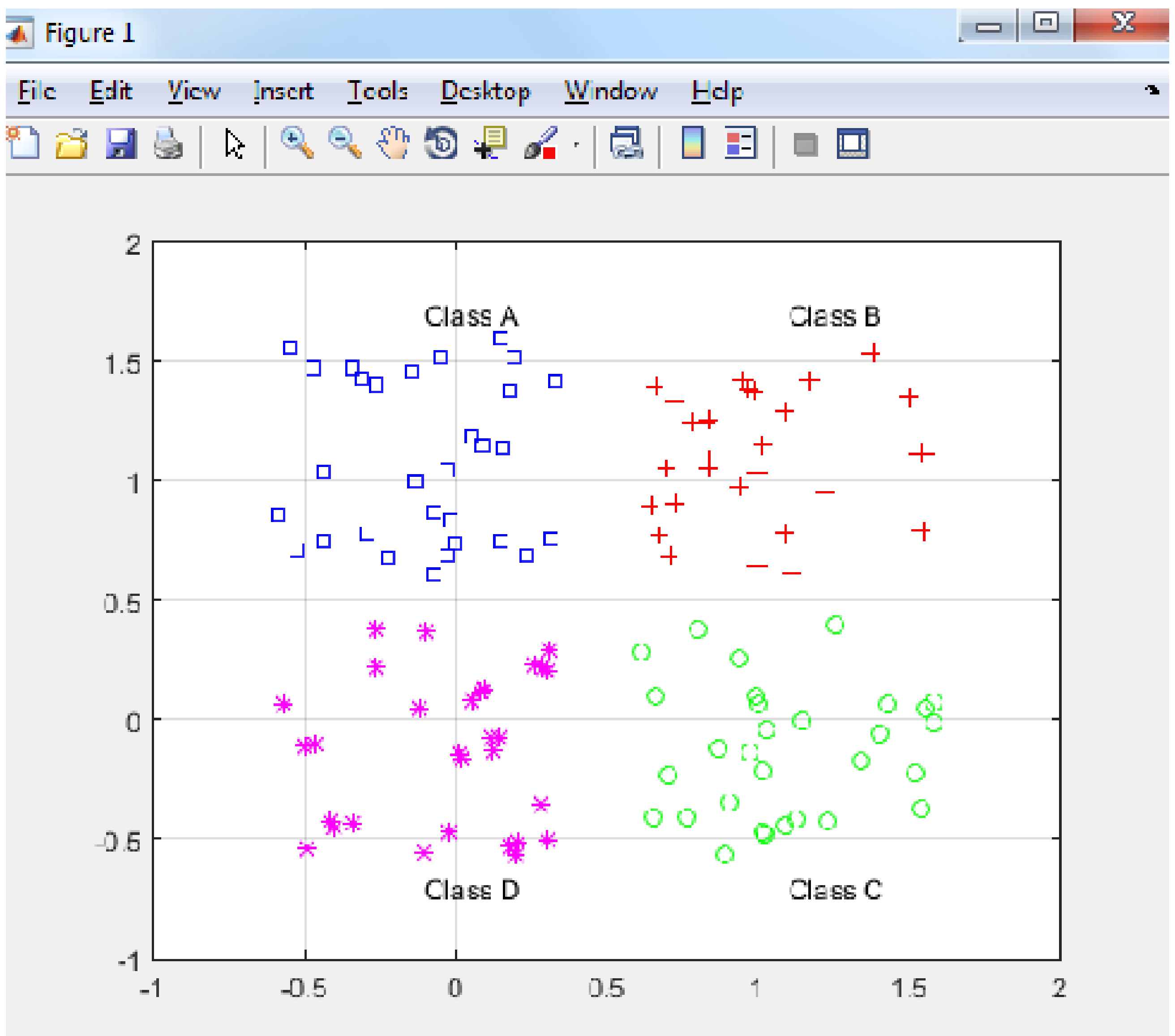

Рисунок 3.18 – Классы, которые предстоит разделить

Для получения вектора T необходимо раскопировать транспонированные вектора **a**, **b**, **c**, **d**. Для этой цели используется **repmat()**: **repmat(что копируем, копий по строкам, копий по столбцам)**. В результате вектор **T** примет вид как на рисунке 3.19: матрица из двух строк и 30 * 4 = 120 колонок.

```
>>  T = [repmat(a, 1, length(A)) repmat(b, 1, length(B)) repmat(c, 1, length(C)) repmat(d, 1, length(D))]
T =
  Columns 1 through 17
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
  Columns 18 through 34
     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
  Columns 35 through 51
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
  Columns 52 through 68
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     0     0     0     0     0     0     0     0
  Columns 69 through 85
     1     1     1     1     1     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
  Columns 86 through 102
     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
  Columns 103 through 119
     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
  Column 120
     0
     0
```

Рисунок 3.19 – Общий вид вектора меток T

Цикл **while()** – это цикл с предусловие, поэтому он может не выполниться ни разу, а следовательно, перед ним необходимо построить возможное решение (т.к. при создании сети **net = perceptron**) веса получают случайные значения.

[net, Y, E] = adapt(net, P, T) – процесс адаптации весов. **Net** – получит обученную сеть, **Y** – вектор ответов сети для входа **P**, **E** – ошибки сети для меток **T** и входов **P**. **Sse(E)** –

сумма квадратов этих ошибок.

На рисунке 3.20 показана структура сети, видно, что единственный слой сети содержит два нейрона, что соответствует двум прямым линиям для разделения 4-х классов.

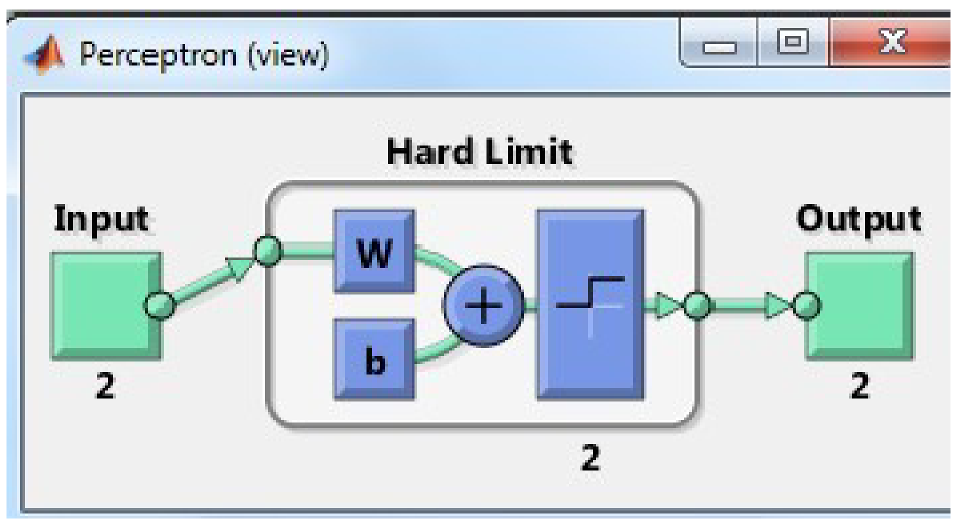

Рисунок 3.20 – Общая структура персептрона

Возможный ответ сети приведён на рисунке 3.21.

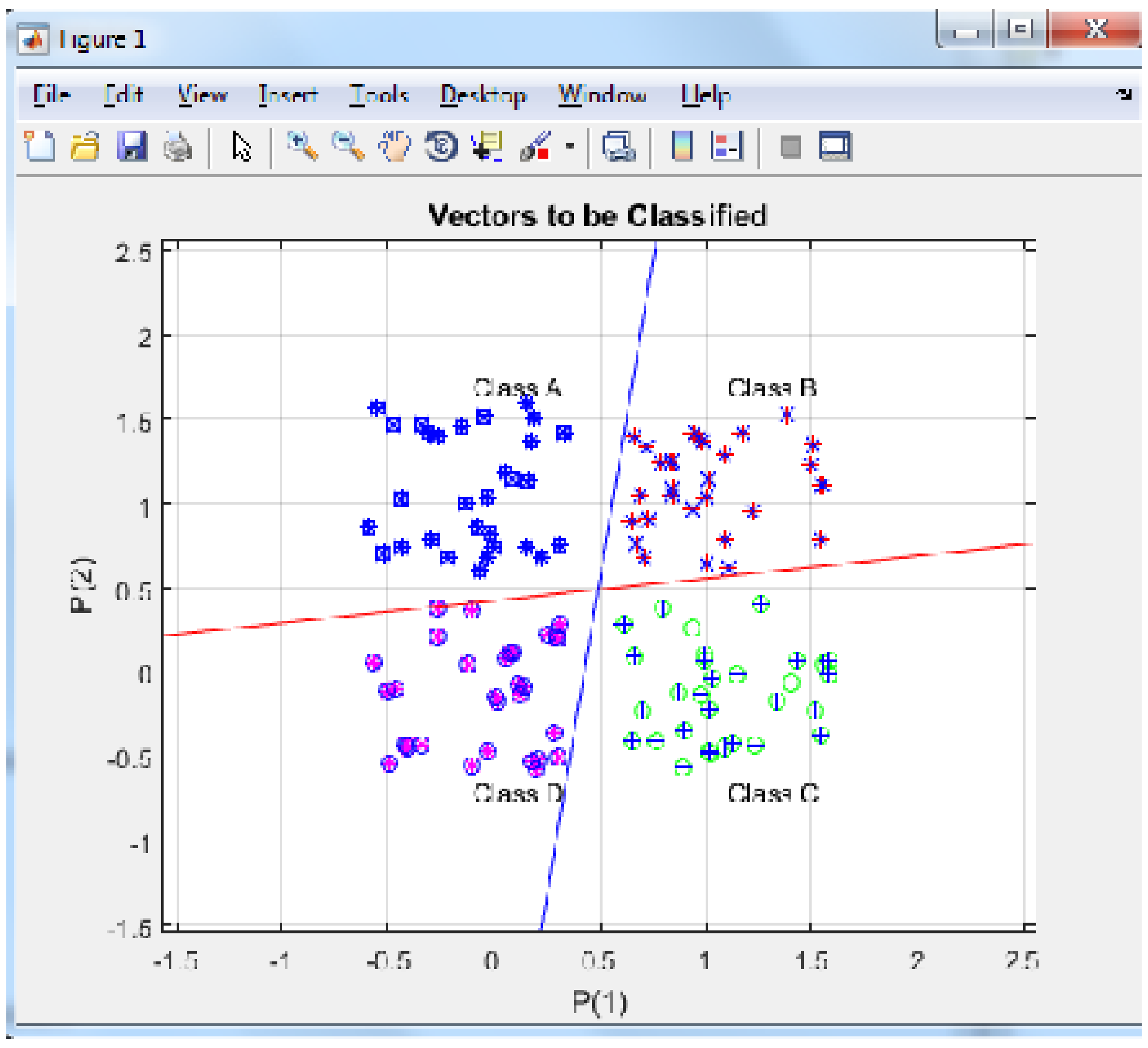

Рисунок 3.21 – Решения задачи разбития на 4 класса

Рассмотрим решение XOR-проблемы с помощью многослойной сети прямого распространения, но расширим представление каждого класса до 100 паттернов.

```
close all, clear all, clc, format compact
K = 100;
q = .6;
%Данные для обучения
%Обратите внимание на то, что randn() отделяются с помощью
%";". Если опустить этот символ, то A будет не 2x100, а 1x200.
A = [rand(1, K)-q; rand(1, K)+q];
B = [rand(1, K)+q; rand(1, K)+q];
C = [rand(1, K)+q; rand(1, K)-q];
D = [rand(1, K)-q; rand(1, K)-q];
figure(1);
% Наносим точки на канву
plot(A(1, :), A(2, :), 'k+');
hold on;
grid on;
plot(B(1, :), B(2, :), 'bd');
```

```
plot(C(1, :), C(2, :), 'k+');
plot(D(1, :), D(2, :), 'bd');
% Наносим названия классов
text(.5-q, .5+2*q, 'Class A');
text(.5+q, .5+2*q, 'Class B');
text(.5+q, .5-2*q, 'Class A');
text(.5-q, .5-2*q, 'Class B');
% Определяем метки для классов
a = -1, c = -1, b = 1, d = 1;
% Формируем входной вектор
P = [A B C D];
% Формируем вектор ответов
T = [repmat(a, 1, length(A)) repmat(b, 1, length(B)) repmat(c, 1, length(C)) repmat(d, 1,
length(D))];
% Определяем сеть прямого распространения
net = feedforwardnet([5 3]);
% Вся выборка под обучающие примеры
net.divideParam.trainRatio = 1;
% Нет валидационной выборки
net.divideParam.valRatio = 0;
% Нет тестовой выборки
net.divideParam.testRatio = 0;
% Обучаем сеть
[net, tr, Y, E] = train(net, P, T);
view(net);
figure(2);
% График ожидаемых ответов сети
% график получается зубчатым, т.к. метки для классов
%определены как -1 и +1
plot(T', 'linewidth', 2);
hold on;
% График реальных ответов сети
plot(Y', 'r--');
grid on;
% Рисуем легенду к графикам
legend('Targets', 'Network response', 'location', 'best');
% Определяем видимый диапазон по оси y
ylim([-1.25 1.25]);
% Создаём  набор 601 числа
span = -1:.005:2;
% P1 и P2 станут матрицами размером 601x601
[P1, P2] = meshgrid(span, span);
% Транспонируем вектор резмером 601*601 = 361201
pp = [P1(:) P2(:)]';
% Получаем выходы сети
aa = net(pp);
% Возвращаемся к первой канве
figure(1);
% Рисуем трехмерную сетку
mesh(P1, P2, reshape(aa, length(span), length(span))-5);
% Подбираем лучшее сочетание цветов
colormap cool;
```

На рисунке 3.22 показаны паттерны входных классов.

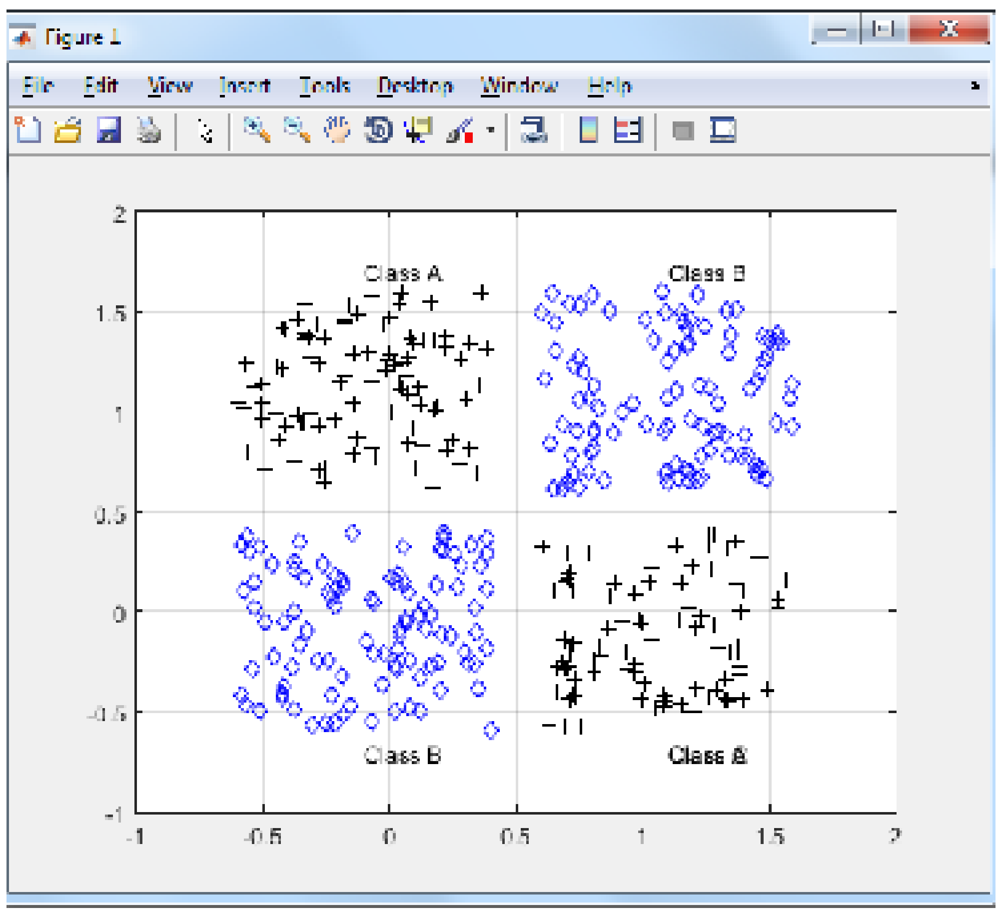

Рисунок 3.22 – Входные данные для двух классов, которые нужно разделить

Net = feedforwardnet([5 3]) – создаём сеть прямого распространения с двумя слоями, в первом будет 5 нейронов, во втором – 3. Структура сети показана на рисунке 3.23.

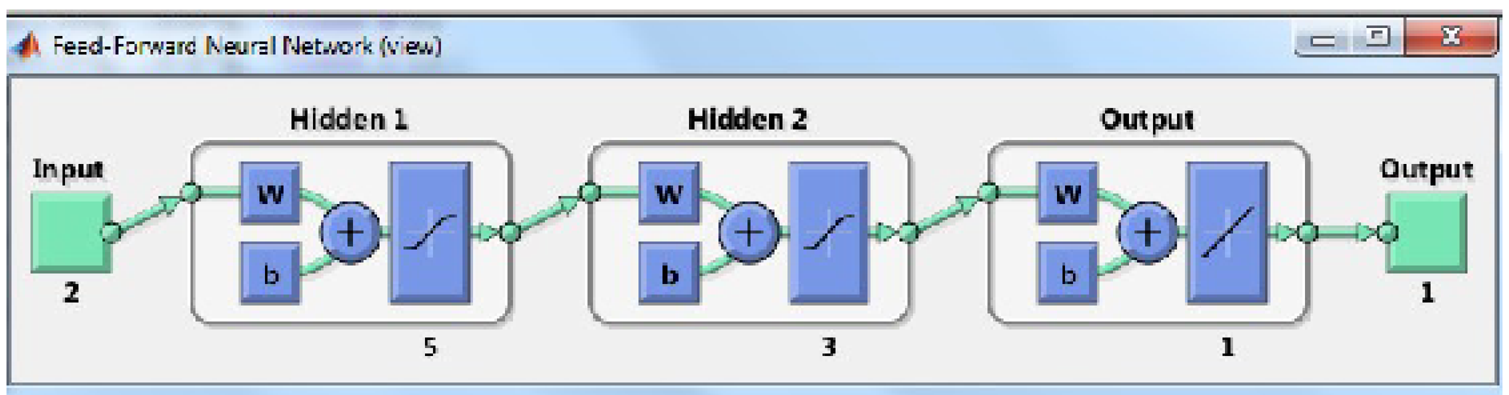

Рисунок 3.23 – Общая структура сети

Обратим внимание, что выходной нейрон имеет линейную функцию активации.

Все паттерны войдут в обучающую выборку, поэтому **net.divideParam.trainRatio = 1**, валидационную и тестовую выборку установим в 0.

Обучение сети осуществляется через **[net, tr, Y, E] = train(net, P, T)** – где **tr** – переменная, содержащая информацию об обучении (в среде Matlab её можно открыть и посмотреть на входящие в неё поля и их значения). Процесс обучения показан на рисунке 3.24, кривая обучения на рисунке 3.25.

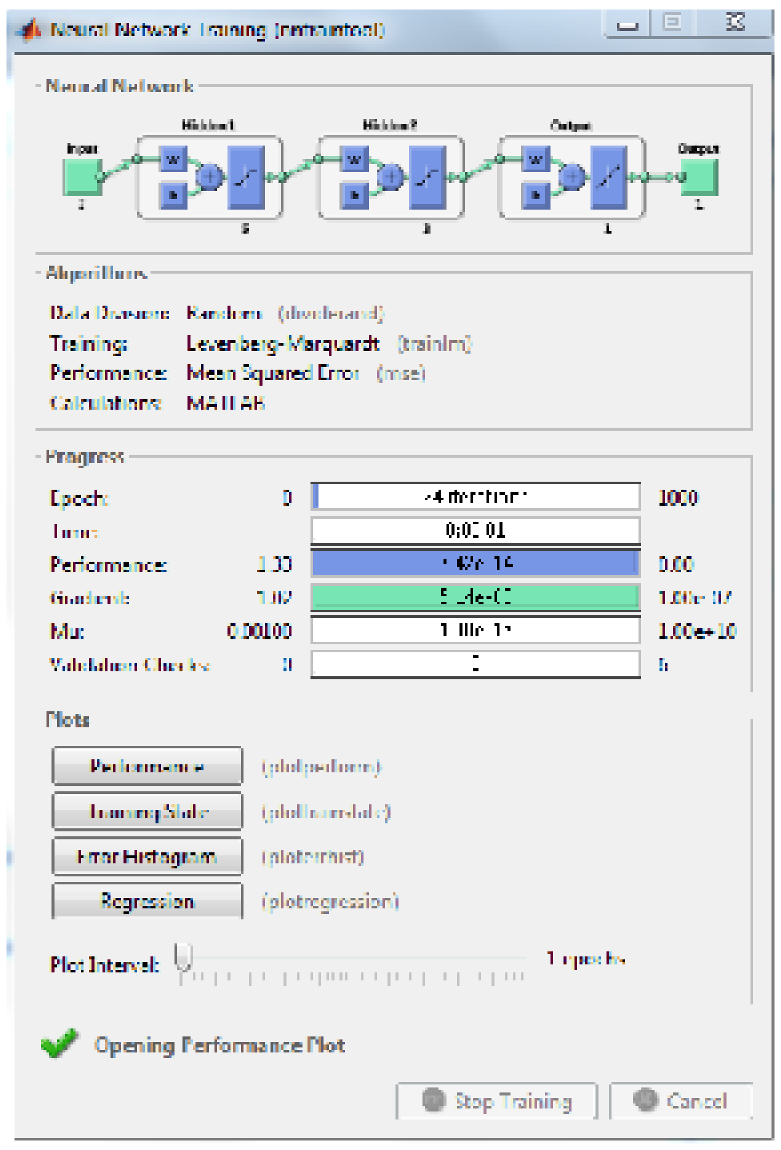

Рисунок 3.24 – Процесс обучения сети, потребовалось 24 итерации

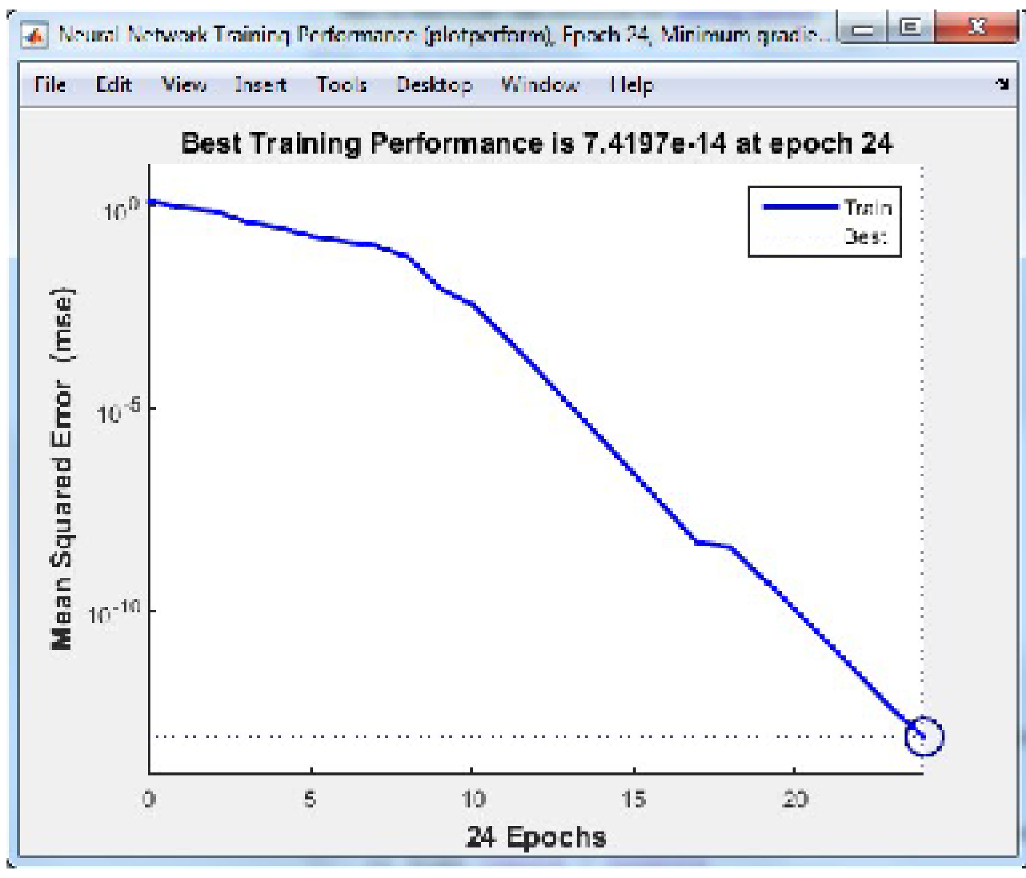

Рисунок 3.25 – Кривая обучения (MSE), лучший показатель меньше 10^{-10}

Далее отображаем на новой канве ожидаемый ответ сети, рисунок 3.26.

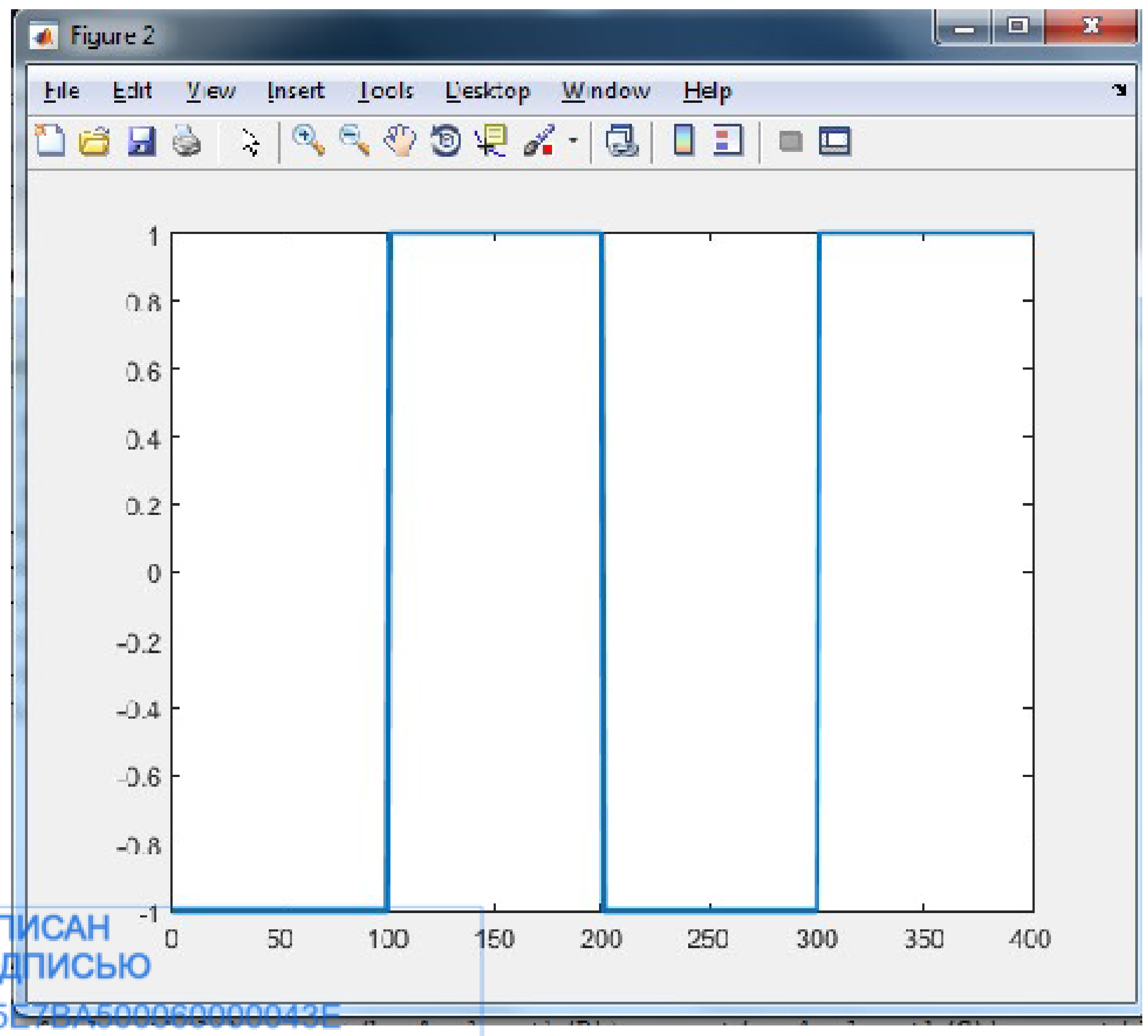

Рисунок 3.26 – Ожидаемый ответ сети (классы кодировались ±1)

На той же канве отображаем реальный ответ сети, рисунок 3.27. Видно, что графики совпадают.

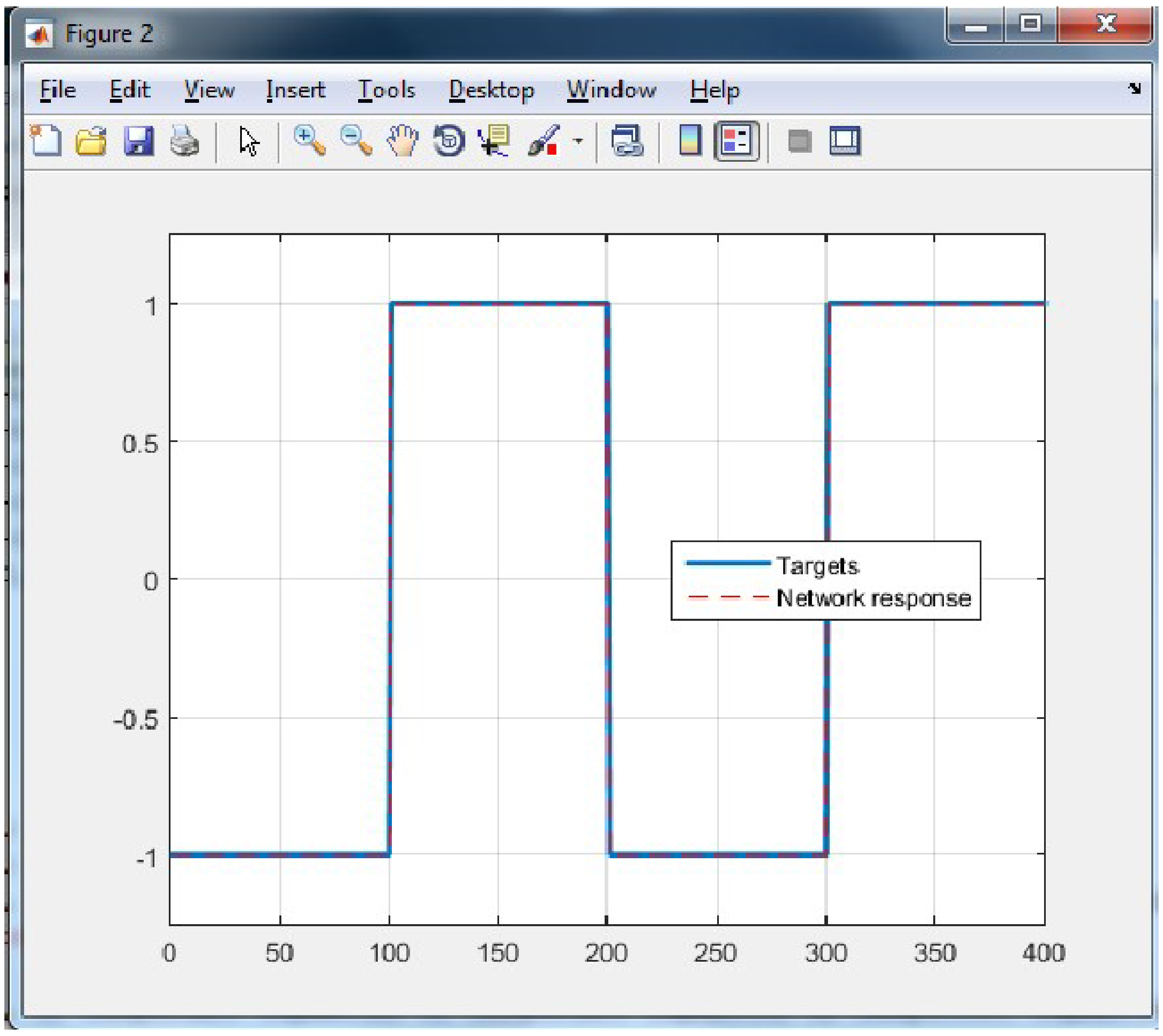

Рисунок 3.27 – Реальный ответ сети, графики совпадают

Ylim([-1.25 1.25]) – ограничивает ось OY данными значениями.

Чтобы отобразить получившееся решение уже нельзя просто построить прямые линии (как в случае с персептроном). Для этого нужно поверх первой канвы отобразить значения выходов сети на очень мелкой сетке (входные данные) и закрасить получившиеся ответы. Граница решения для разбиения двух классов окажется видна, рисунок 3.28.

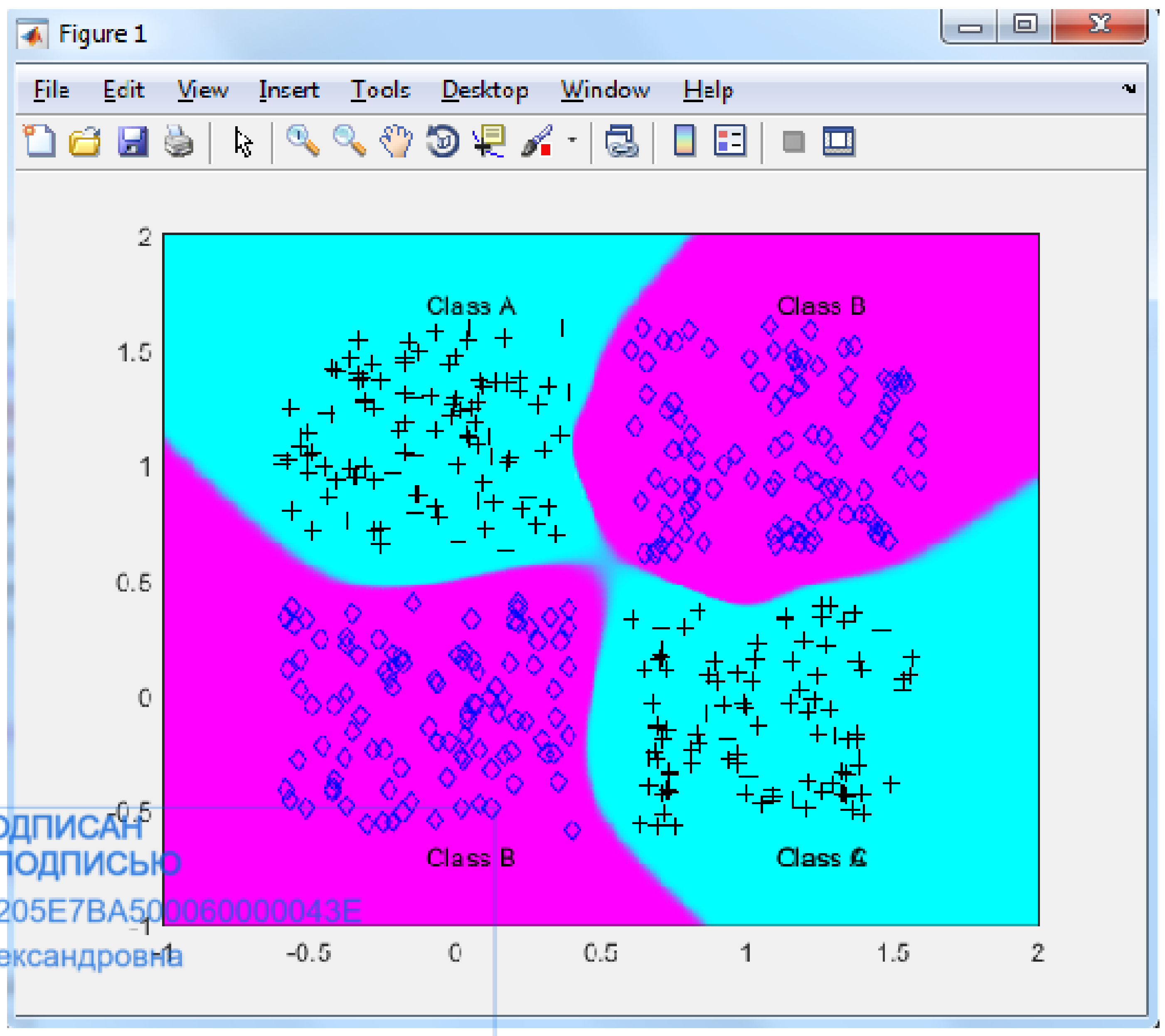

Рисунок 3.28 – Получившаяся граница решения для задачи XOR

Примечание. Сеть со структурой 2-5-3-1 явно избыточна для решения данной задачи. Можно переопределить структуру сети, как было показано в начале данной лабораторной работы:

```
net.layers{1}.size = 5;
% Получаем двухслойную сеть, где скрытый слой имеет 5 нейронов
net = configure(net, P, T);
% Меняем функцию активацию на выходном нейроне на нелинейную
net.layers{2}.transferFcn = 'tansig';
net.divideParam.trainRatio = 1;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0;
[net, tr, Y, E] = train(net, P, T);
```

Структура такой сети показана на рисунке 3.29.

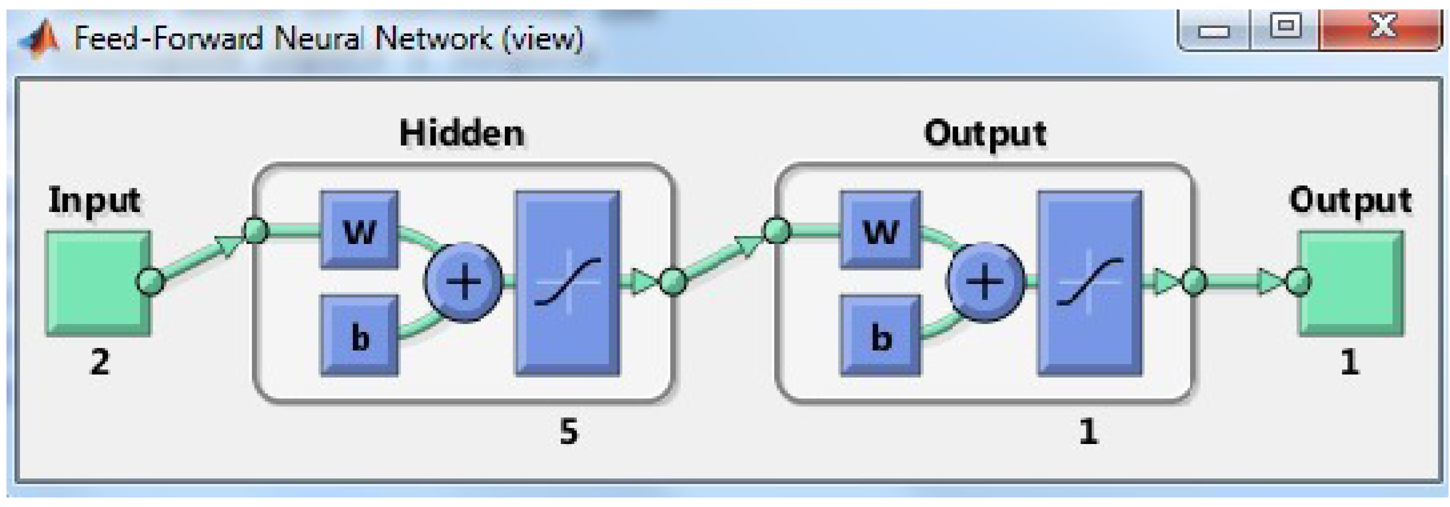

Рисунок 3.29 – Данная сеть тоже справится с задачей XOR

Однако, как видно из рисунка 3.30, ошибка обучения опустится в район 10^{-9}, что больше, чем у первой сети, что в целом хуже.

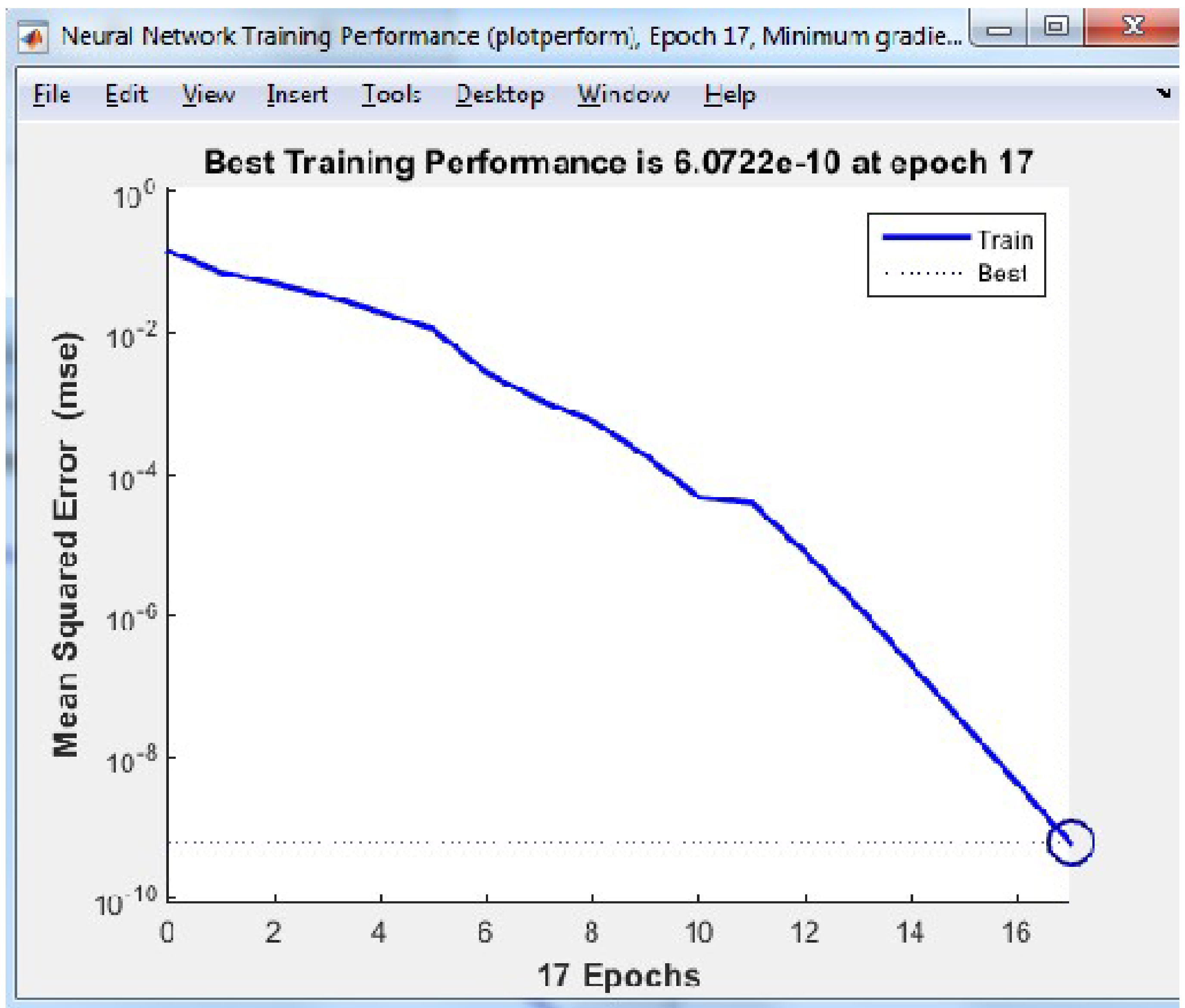

Рисунок 3.30 – Ошибка обучения для сети 2-5-1

Граница решений для этой сети показана на рисунке 3.31.

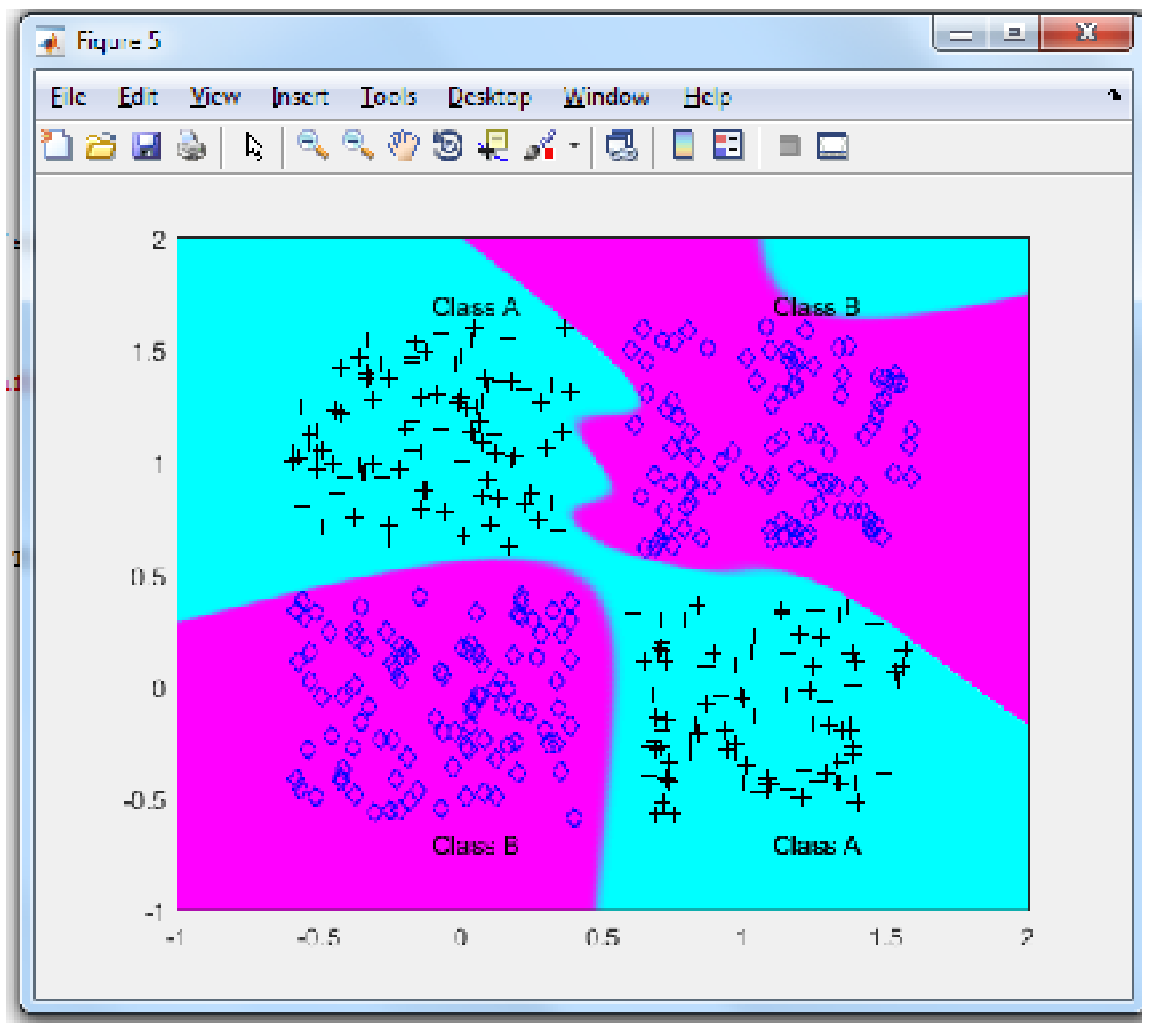

Рисунок 3.31 – Граница решений для сети с пятью скрытыми нейронами

Сеть со структурой 2-3-1 даст ошибку, показанную на рисунке 3.32.

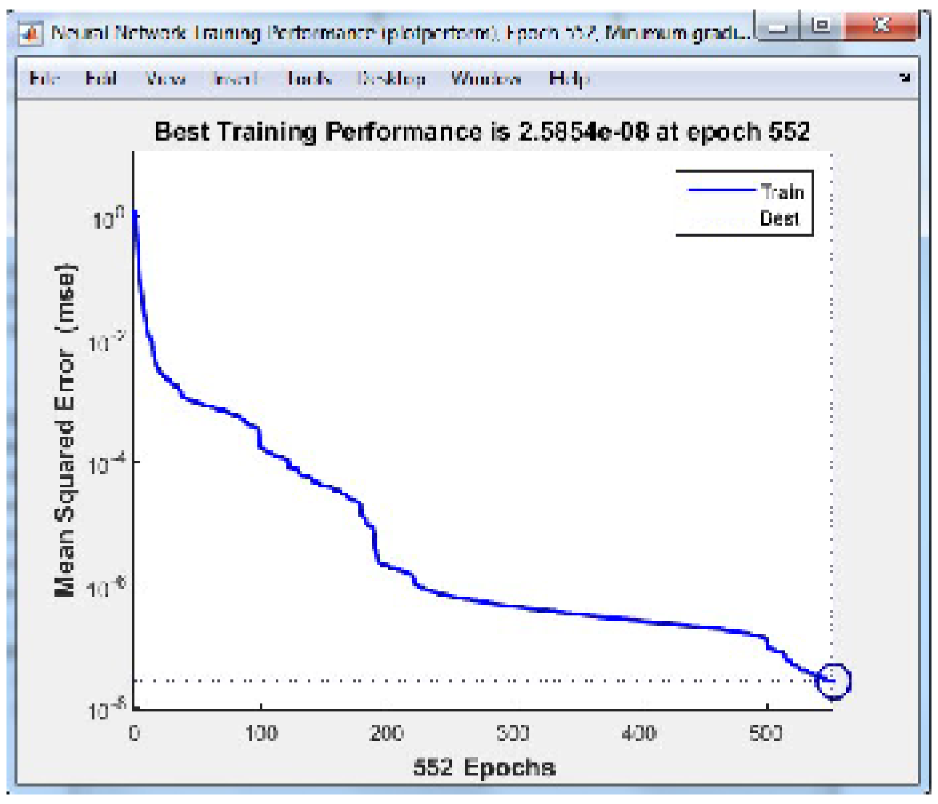

Рисунок 3.32 – Ошибка обучения для сети 2-3-1

Получившаяся граница решений для сети с тремя скрытыми нейронами показана на рисунке 3.33.

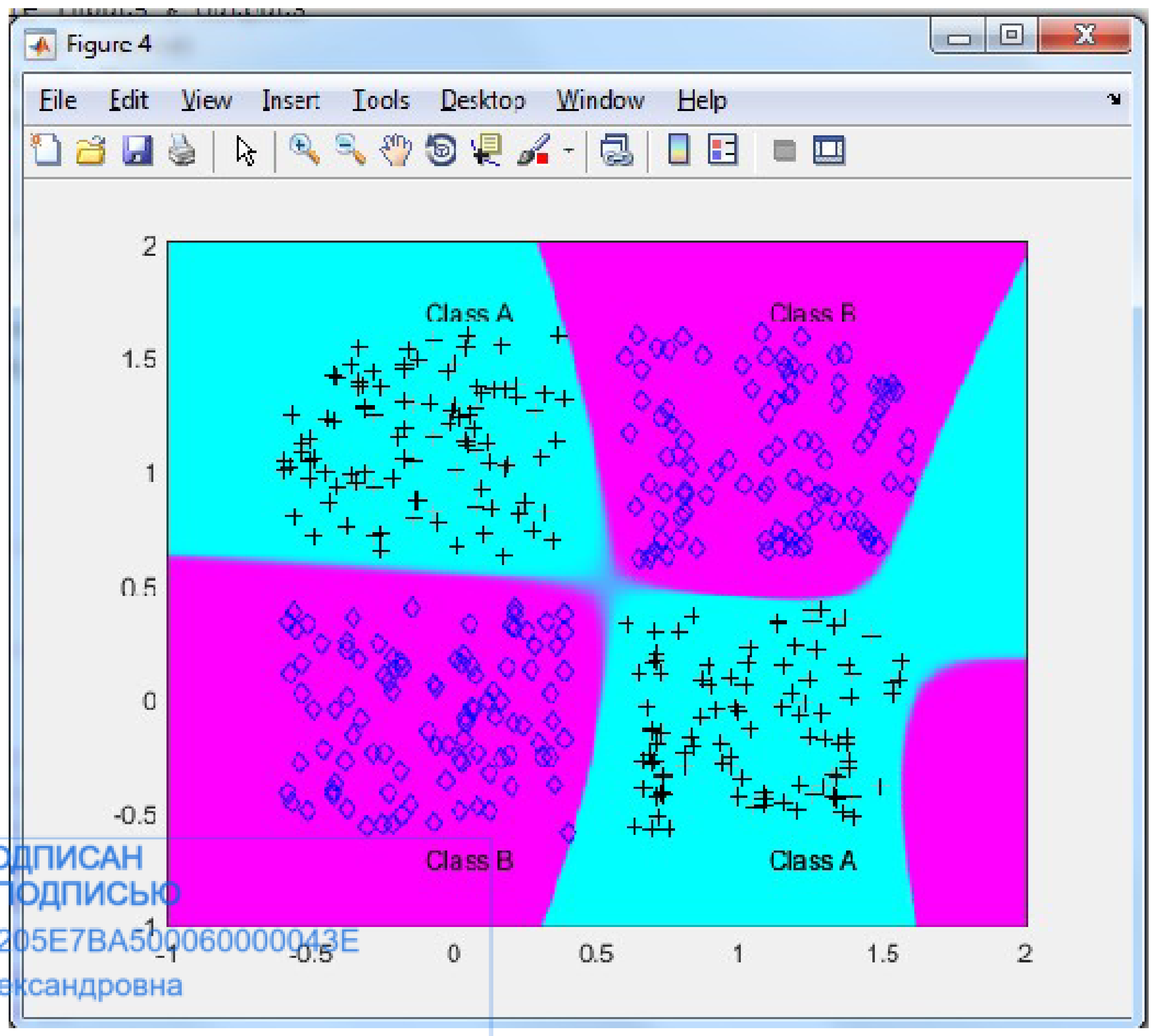

Рисунок 3.33 – Граница решений для сети из трёх нейронов в скрытом слое

Можно сделать вывод, что чем меньше количество нейронов в скрытом слое, тем грубее будут разделены классы и граница решений может в некоторых местах принимать хаотический вид (рисунок 3.31).

Конечно, такой разброс в решениях получается вследствие лёгкой разделимости классов.

Теперь применим сеть прямого распространения к разделению 4-х классов (ранее решали эту задачу с помощью персептрона).

```
close all, clear all, clc, format compact
K = 100;
q = .6;
A = [rand(1, K)-q; rand(1, K)+q];
B = [rand(1, K)+q; rand(1, K)+q];
C = [rand(1, K)+q; rand(1, K)-q];
D = [rand(1, K)-q; rand(1, K)-q];
figure(1);
plot(A(1, :), A(2, :), 'k+');
hold on;
grid on;
plot(B(1, :), B(2, :), 'b*');
plot(C(1, :), C(2, :), 'kx');
plot(D(1, :), D(2, :), 'bd');
text(.5-q, .5+2*q, 'Class A');
text(.5+q, .5+2*q, 'Class B');
text(.5+q, .5-2*q, 'Class C');
text(.5-q, .5-2*q, 'Class D');
% Кодируем классы
a = [-1 -1 -1 +1]';
b = [-1 -1 +1 -1]';
c = [-1 +1 -1 -1]';
d = [+1 -1 -1 -1]';
P = [A B C D];
T = [repmat(a, 1, length(A)) repmat(b, 1, length(B)) repmat(c, 1, length(C)) repmat(d, 1, length(D))];
net =feedforwardnet([4 3]);
net.divideParam.trainRatio = 1;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0;

[net, tr, Y, E] = train(net, P, T);
view(net);
% Получаем в “m” все максимальные элементы по столбцам, а в “i”
% их строковые индексы
[m, i] = max(T);
% Получаем в j индексы строк максимальных элементов
[m, j] = max(Y);
N = length(Y);
k = 0;
```

```
% Сравниваем эти индексы, они должны совпасть, если они не
% совпадают, то увеличиваем k на 1, т.е. k будет содержать количество
% ответов ИНС, которые не совпадают с метками
if find(i - j),
        k = length(find(i-j));
end
% Высчитываем это количество в процентах
fprintf('Correct classified samples: %.if%% samples\n', 100*(N-k)/N);
figure;
% Строим верхний график на канве
subplot(211);
plot(T');
title('Targets');
ylim([-2 2]);
grid on;
% Строим нижний график на канве
subplot(212);
plot(Y');
title('Network response');
xlabel('# sample');
ylim([-2 2]);
grid on;
span = -1:.01:2;
[P1, P2] = meshgrid(span, span);
pp = [P1(:) P2(:)]';
aa = net(pp);
figure(1);
% Наносим на сетку все ответы первого нейрона для
% всей выборки, они должны занять ¼ от площади всех классов
m = mesh(P1, P2, reshape(aa(1, :), length(span), length(span))-5);
set(m, 'facecolor', [1 0.2 .7], 'linestyle', 'none');
hold on;
% Тоже самое делаем для других нейронов
m = mesh(P1, P2, reshape(aa(2, :), length(span), length(span))-5);
set(m, 'facecolor', [1 1.0 0.5], 'linestyle', 'none');
m = mesh(P1, P2, reshape(aa(3, :), length(span), length(span))-5);
set(m, 'facecolor', [.4 1.0 0.9], 'linestyle', 'none');
m = mesh(P1, P2, reshape(aa(4, :), length(span), length(span))-5);
set(m, 'facecolor', [.3 .4 0.5], 'linestyle', 'none');
view(2);
```

Создаём 4 класса, каждый представлен 100 паттернами, рисунок 3.34.

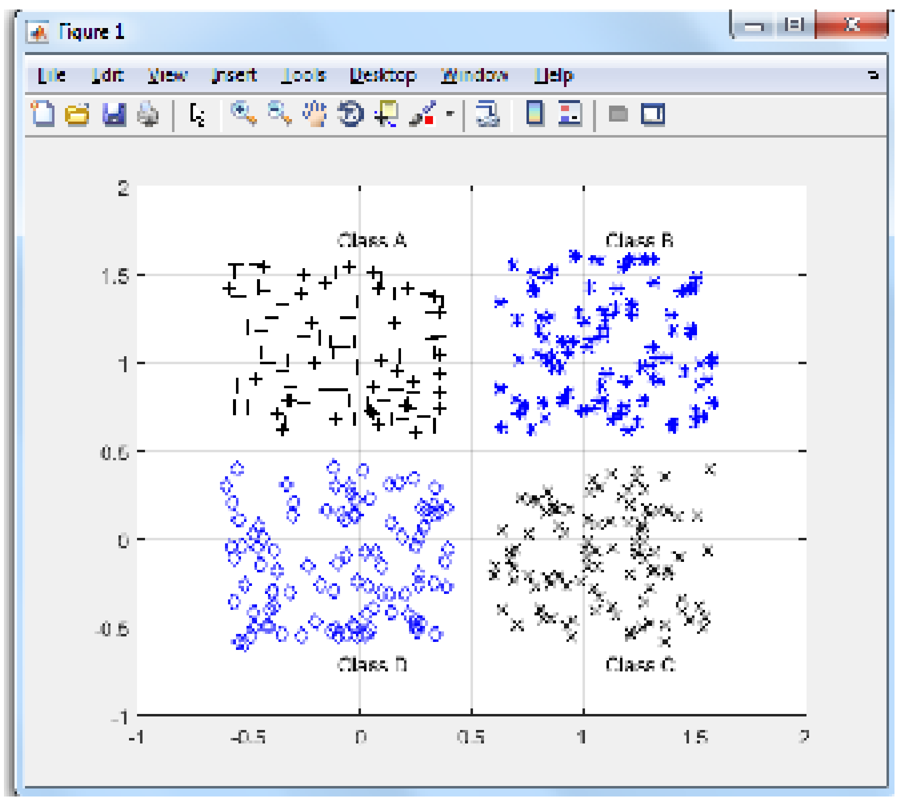

Рисунок 3.34 – 4 класса, которые нужно разделить

Далее создаём сеть со структурой 2-4-3-4 и обучаем её, рисунки 3.35, 3.36, 3.37.

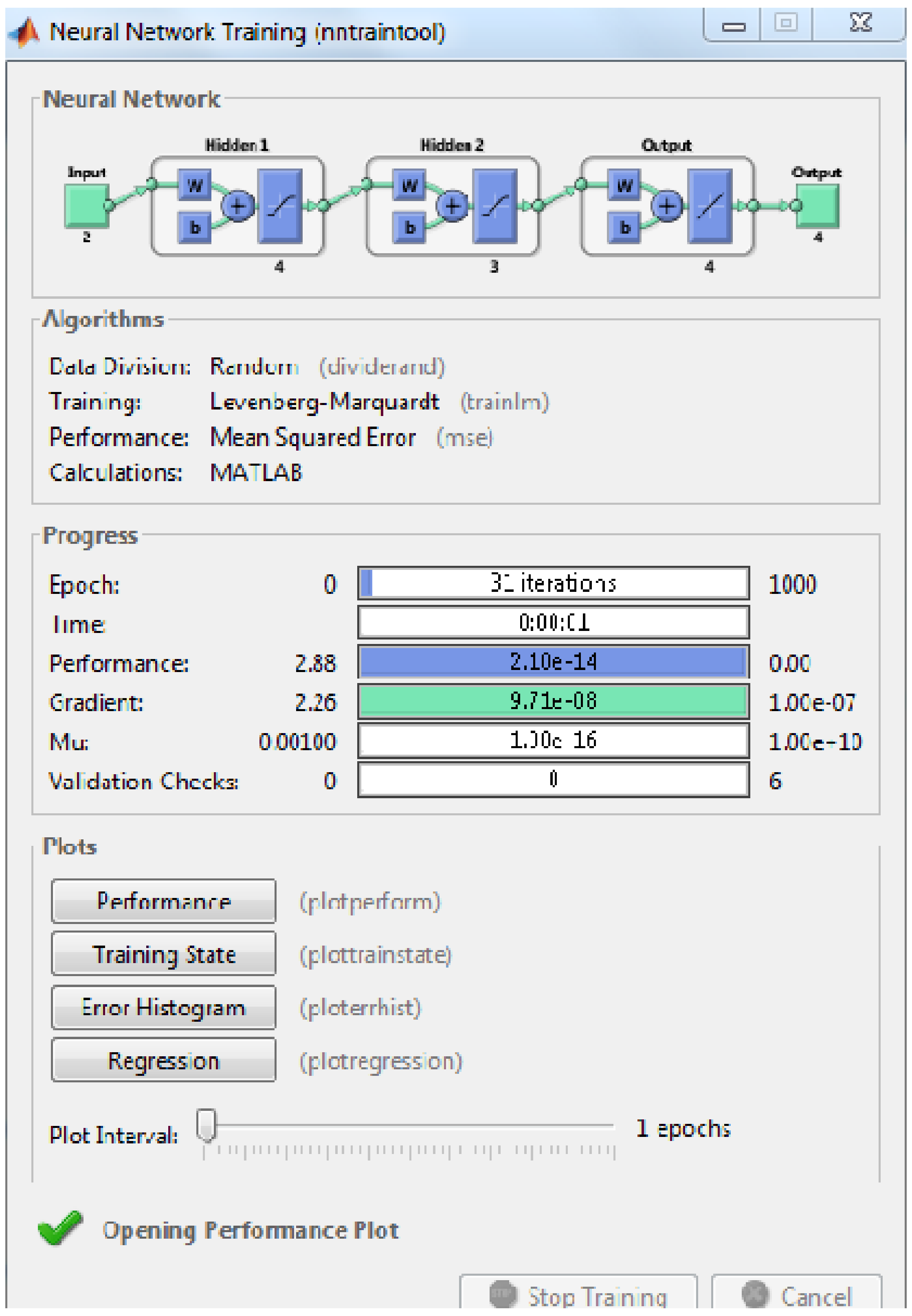

Рисунок 3.35 – Обучение сети, потребовалось 31 итерация

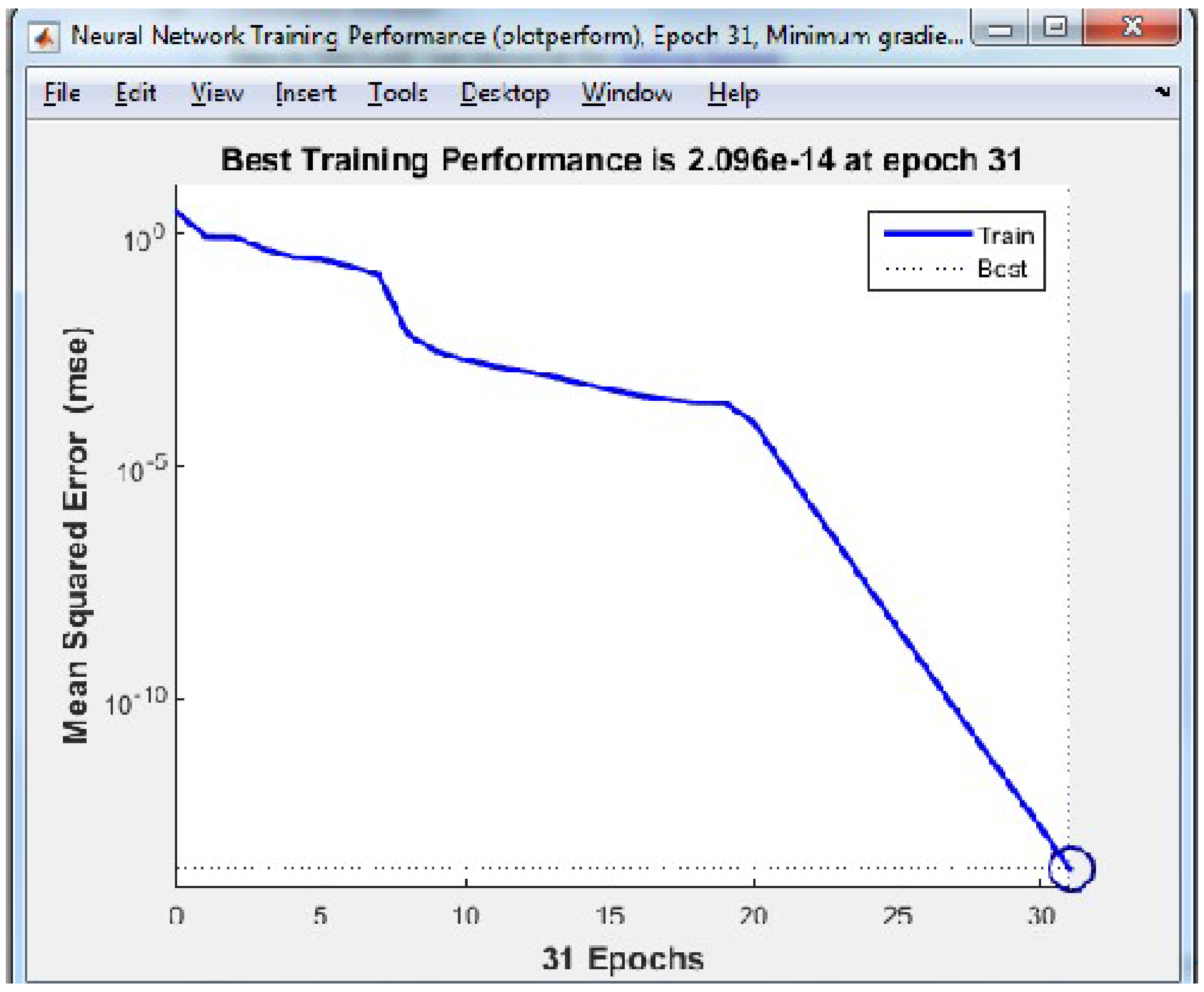

Рисунок 3.36 – Ошибка обучения очень хорошая, ушла ниже 10^{-10}

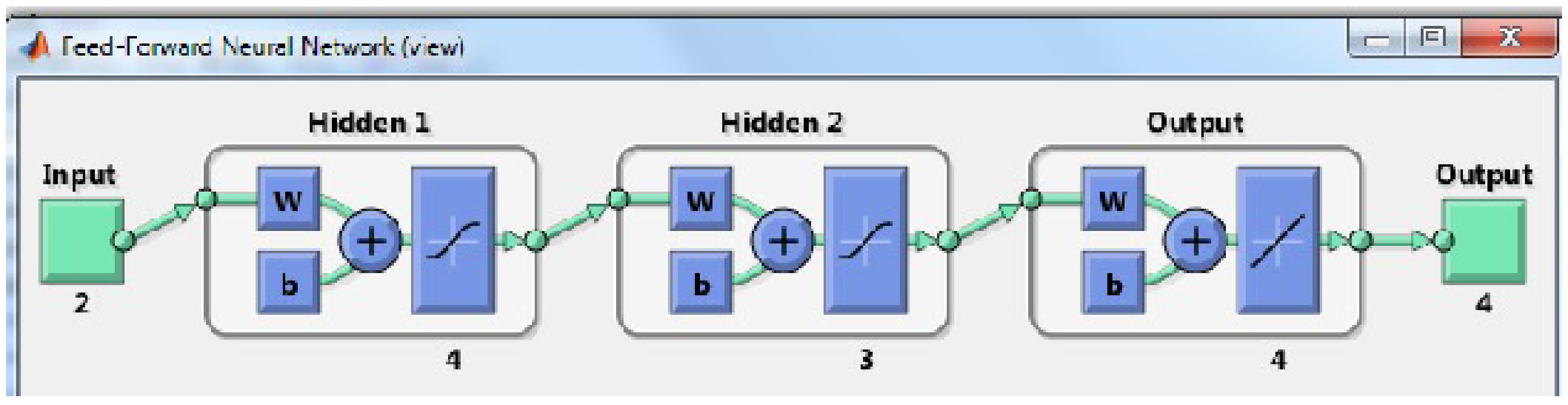

Рисунок 3.37 – Общая структура сети, которая использовалась

Разумеется, предложенная структура не единственная, на самом деле хватило бы сети с одним скрытым слоем и четырьмя нелинейными нейронами на выходном слое.

Метки и ответы сети изображены на рисунке 3.38, видно, что графики полностью совпадают, т.е. сеть точно моделирует все реакции на входные значения.

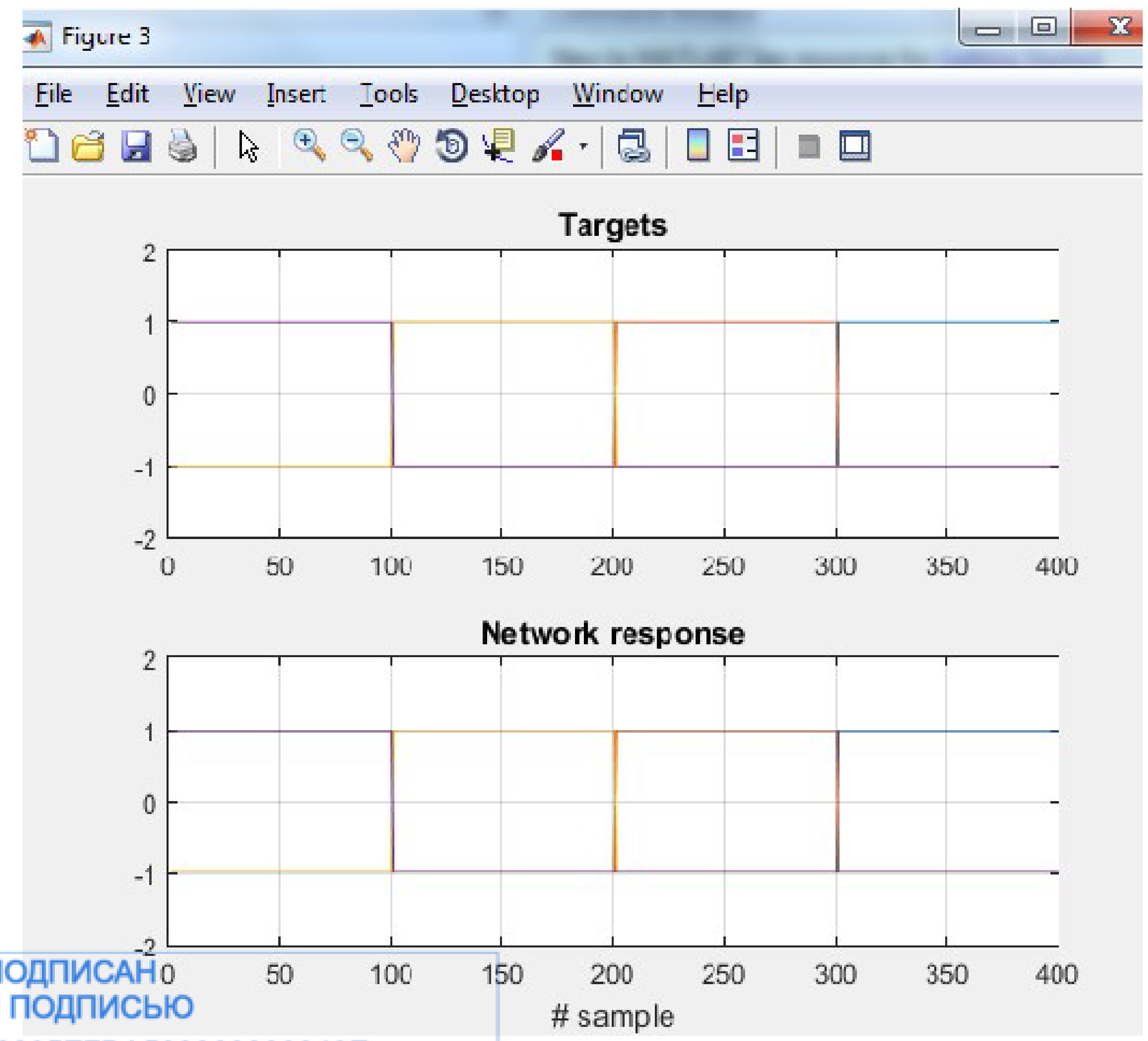

Рисунок 3.38 – График ожидаемых и реальных ответов ИНС

После построения границ решения можно видеть, что каждый класс отделён от другого, рисунок 3.39.

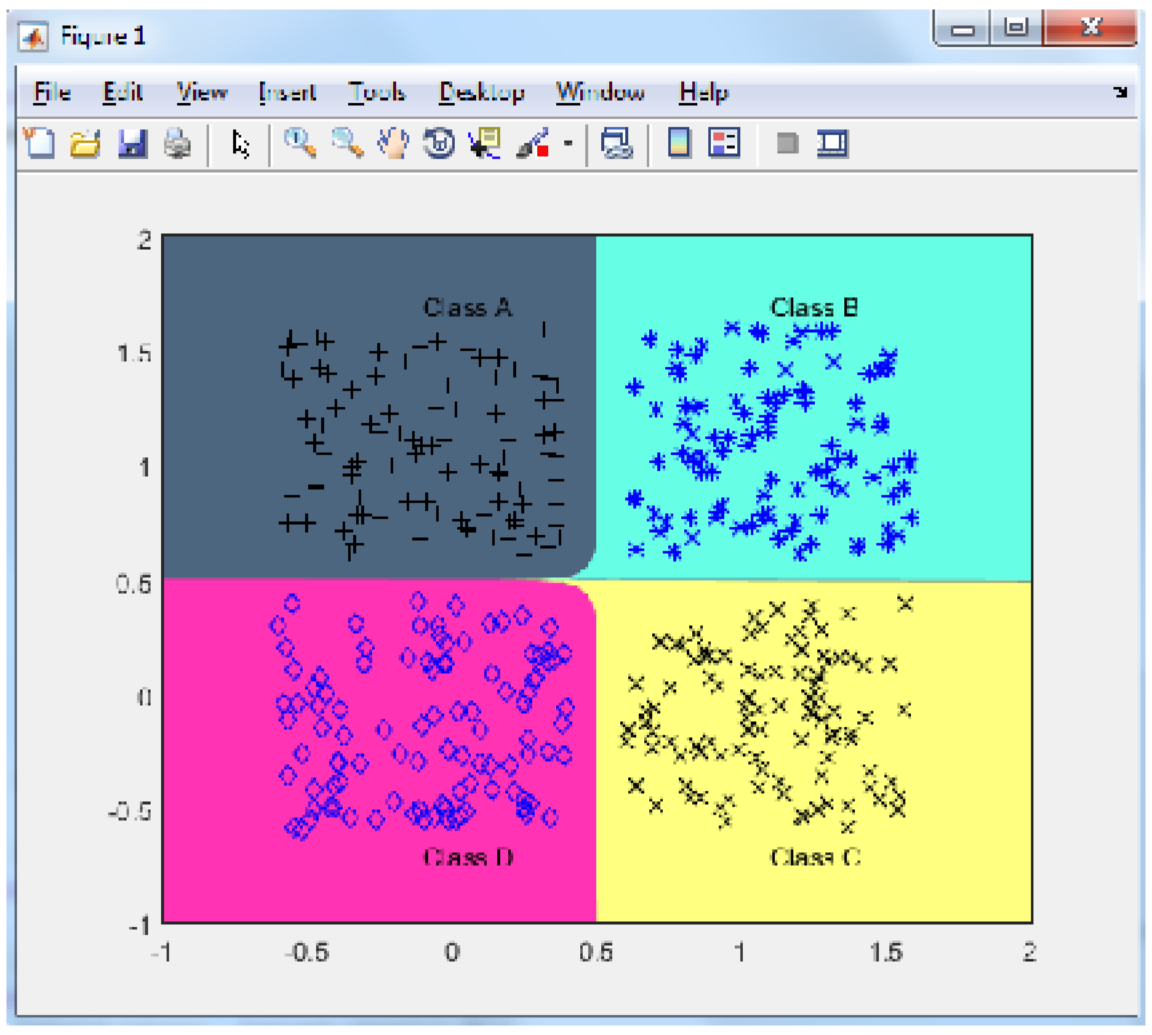

Рисунок 3.39 – Границы решения для задачи по разделению четырёх классов с помощью сети прямого распространения

Если сравнить это решение с тем, которое было достигнуто с помощью персептрона, то станет очевидным, что возможности по отделению одних классов от других у ИНС прямого распространения значительно выше, чем у персептрона, т.к. такие сети позволяют строить более сложные границы.

Аппаратура и материалы. 64-разрядный (x64) персональный компьютер, процессор с тактовой частотой 1 ГГц и выше, оперативная память 1 Гб и выше, свободное дисковое пространство не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Matlab (R2013) и выше.

Указание по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения; ремонт персонального компьютера. Соблюдать правила технической эксплуатации и техники безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

В индивидуальном варианте (таблица 3.1) дано расположение двух классов (A и B) по «углам» квадрата (по типу как на рисунке 3.39). Каждый «угол» представлен 200 точками. Требуется отделить класс A от B с помощью персептрона и полносвязанной многослойной сети прямого распространения.

Таблица 3.1 – Варианты заданий

Номер варианта	Условие задачи
1	A B B B
2	B A B B
3	B B A B
4	B B B A
5	A A B B
6	B A A B
7	B B A A
8	A B B A
9	B A

	B *A*
10	*A* *B* *B* *A*
11	*A* *A* *A* *B*
12	*A* *B* *A* *A*
13	*B* *A* *A* *A*
14	*A* *A* *B* *A*

Содержание отчета и его форма

Отчёт по лабораторной работе должен содержать следующую информацию:

1. Название лабораторной работы и её номер.
2. ФИО студента и группу.
3. Формулировка индивидуального задания.

4. Документ отчёта с Print Prtscr диалоговых окон по шагам для своего варианта по подобию того, что описано в теоретической части. Обязательно должна присутствовать раскрашенная форма с финальным разделением для сети и цветные линии, отделяющие один класс от другого для персептрона. Должен присутствовать ступенчатый график, показывающий качество работы сети.
5. Ответы на контрольные вопросы.

Вопросы для защиты работы

1) Как кодируются классы для персептронов или сетей прямого распространения?

2) Опишите последовательность действий для нанесения входных значений на канву, чтобы визуально было видно расположение классов относительно друг друга.

3) Опишите последовательность действий для построения на канве границ решения задачи классификации.

4) Приведите пример сети прямого распространения с одним скрытым слоем, с помощью которой можно решить задачу о разделении 4-х классов в рассмотренной лабораторной работе.

5) Почему с помощью сетей прямого распространения можно лучше справляться с задачами классификации, чем с помощью персептронов?

Лабораторная работа № 4

Классификаторы, основанные на оценки функции оптимизации.

Машины опорных векторов и нейронные сети прямого распространения

Цель и содержание работы: познакомить студента с машинами опорных векторов и нейронной сетью прямого распространения.

Задачи:

1) Разобрать принцип действия и алгоритм работы машины опорных векторов;
2) Разобрать принцип действия нейронной сети прямого распространения.

Теоретическое обоснование

Машина опорных векторов (**S**upport **V**ector **M**achine) основывается на понятии разницы. Рассмотрим линейный классификатор (4.1)

$$w^T x \;\; w_0 \;\; 0 \qquad (4.1)$$

,

который, по сути представляет собой разделяющую гиперплоскость (4.2).

$$\begin{matrix} w^T x \;\; w \;\; w^T & & & \\ & 0, \text{ тогда} & & 1 \\ x \;\; w^0 & \mathrm{d}_i & & 1 \\ {}^0 & & & \\ & 0, \text{ тогда} & & \\ & \mathrm{d}_i & & \end{matrix} \qquad (4.2)$$

Для данного вектора весов **w** и порога w_0 расстояние между гиперплоскостью, задаваемой уравнением (4.1), и ближайшей точкой из набора данных называется границей разделения и обозначается как p. Основной целью SVM является поиск конкретной гиперплоскости, для которой границе разделения будет максимальной. При этом условии поверхность решения называется оптимальной гиперплоскостью.

Общую идею можно понять из рисунка 4.1.

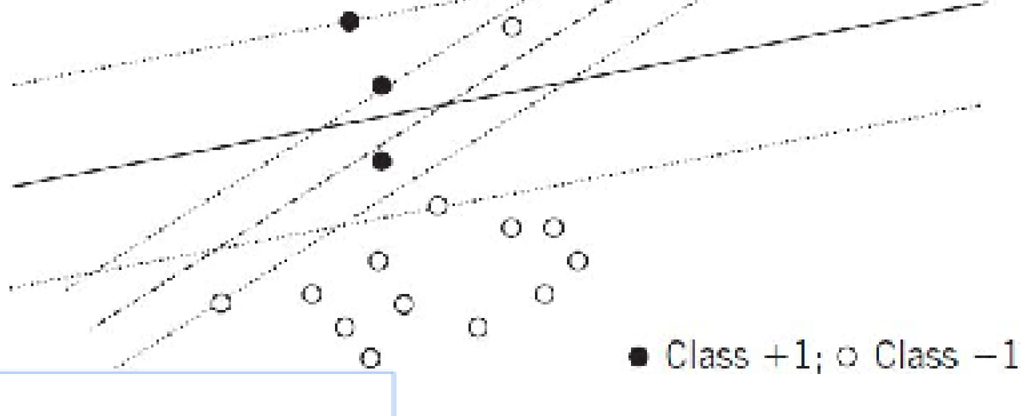

Рисунок 4.1 – Два способа отделения двух линейно разделимых классов

На рисунке 4.1 изображены два линейно отделимые класса: черные точки и белые точки. Представим, что черные и белые точки – это дома в деревне. Через деревню нужно проложить дорогу так, чтобы разрушить минимальное количество домов и при этом, чтобы она была максимально широкая. На рисунке видны два возможных варианта прокладки дорог: дорога с разделительной пунктирной линией, идущая из левого нижнего угла в верхний правый, и дорога со сплошной разделительной линией. Обе дороги ломают по 5 домов, но очевидно, что более широкая дорога выгоднее, т.к. более удобная, чем узкая, поэтому нужно строить именно широкую дорогу. Средняя сплошная разделительная линия и будет оптимальной разделяющей гиперплоскостью.

Почему это лучше для задач распознавания образов? Дело в том, что обычные нейронные сети просто удовлетворяются от нахождения любой разделяющей гиперплоскости, они не выискивают оптимальные гиперплоскости, а просто находят первую попавшуюся в процессе обучения. Но потом, когда сеть начинает функционировать не на обучающих данных, а на тестовых, которые не участвовали в процессе обучения, то оказывается, что найденная гиперплоскость может давать много ошибок (узкую дорогу легко «перейти» и в результате паттерн из одного класса может оказаться на стороне, где распложены паттерны другого класса). Широкая дорога, т.е. оптимальная гиперплоскость, увеличивает вероятность, что сеть справится с задачей лучше.

На рисунке 4.2 представлена оптимальная гиперплоскость для линейно разделимых образов.

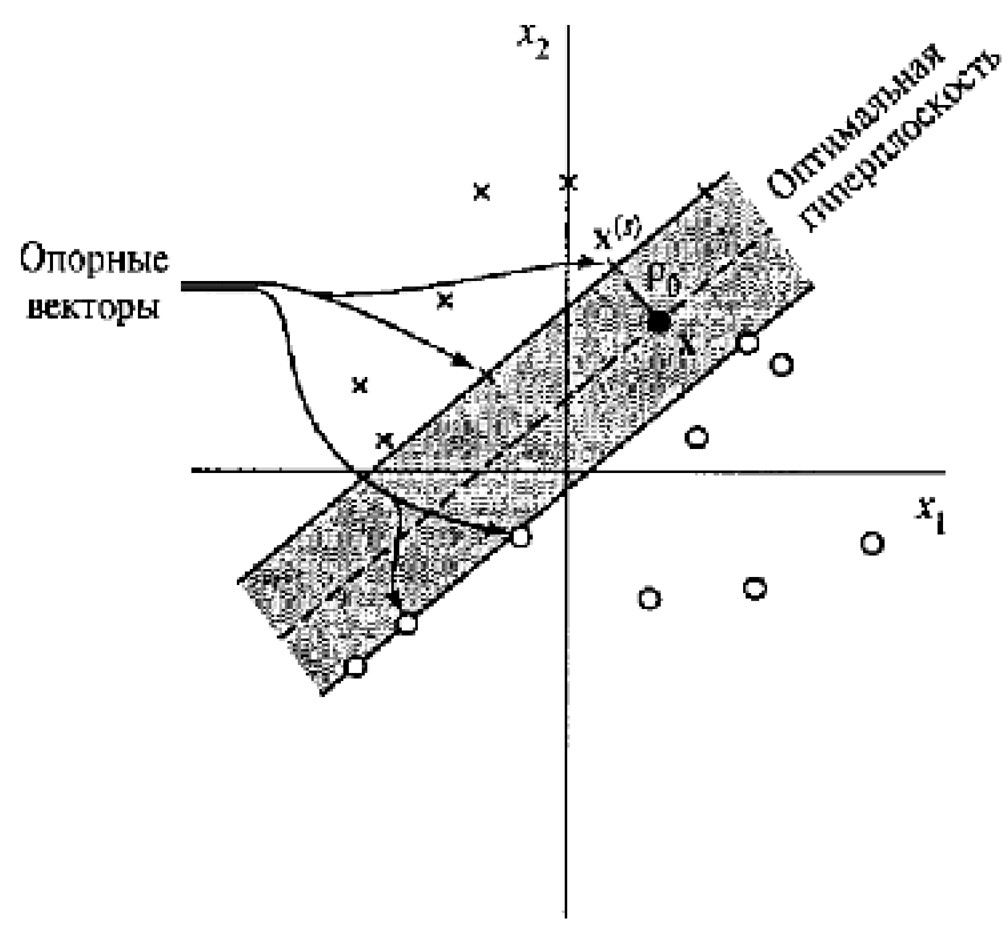

Рисунок 4.2 – Оптимальная гиперплоскость для линейно разделимых образов

Данное решение ведёт к следующей математической формулировки. Возьмем множество обучающих точек x_i с соответствующими им метками $y_i = \{\pm1\}$, $i = 1..N$ для задачи классификации двух классов. Вычисление оптимальной разделяющей гиперплоскости обозначим как минимизацию J():

$$J(\mathrm{w}, \mathrm{w}_0, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i, \qquad (4.3)$$

$$
\begin{aligned}
w^T x_i + w_0 \geq 1 - \xi_i, &\ \text{if } x_i \in \omega_1 \\
w^T x_i + w_0 \leq -1 + \xi_i, &\ \text{if } x_i \in \omega_2 \\
\xi_i \geq 0 &
\end{aligned}
\tag{4.4}
$$

где разница между областями равна 2 / ||w||, величины ошибок ξ_i не отрицательные, они равны 0 для точек, лежащих за пределами разделяющей области и на правильной стороне, и положителы для точек, лежащих внутри области или вне области и на неправильной стороне. С – это константа, определяемая пользователем.

Решение будет представлено в следующем виде:

$$
w = \sum_{i=1}^{N} \lambda_i y_i x_i , \tag{4.5}
$$

где коэффициент λ_i – это множители Лагранжа для оптимизационной задачи, и они равны 0 для всех точек, лежащих за пределами разделяющей области и на правильной стороне классификатора. Эти точки не предоставляют информацию для постройки оптимальной разделяющей гиперплоскости, остальные точки называются опорными точками или опорными векторами.

Для создания линейной SVM будет использован функция SMO2. Сигнатура данной функции имеет следующий вид: **[alpha, w0, w, evals, stp, glob] = SMO2(X′, y′, kernel, kpar2, C, tol, steps, eps, method)**.

Список формальных параметров: матрица X' включает точки данных, каждая строка – это координаты точки в многомерном пространстве, метки содержатся в y′, тип ядерной функции – линейный. Два параметра ядра kra1 и kra2 в линейном случае устанавливаются всегда как 0. Пользовательский параметр C, параметр tol, максимальное число итераций алгоритма, порог eps (очень маленькое число обычно в районе 10^{-10}) используется для сравнения двух чисел (если их разница меньше, чем порог, то они считаются равными). И последний параметр – тип оптимизационного метода, который мы используем (0 – метод Платта, 1 – модификация Кири 1, 2 – модификация Кири 2).

Список возвращаемых значений: alpha – вектор, содержащий множители Лагранжа, w_0 – порог, w – вектор, содержащий параметры гиперплоскости.

Пример 1.

В двумерной плоскости есть два равновероятных класса, каждый представлен распределением Гаусса со средним m_1 = [0, 0]T и m_2 = [1.2, 1.2]T и ковариационными матрицами $S_1 = S_2$ = 0.2I, где I – это единичная матрица 2x2.

1. Сгенерировать и нарисовать обучающее множество X_1, содержащее 200 точек для каждого класса (всего 400 точек). Сгенерировать другое множество X_2, которое будет тестовым (также содержит по 200 точек для каждого класса).

2. Используя X_1 и метод Платта сгенерировать 6 SVM, которые бы разделяли два класса, используя пользовательскую константу со значением C = 0.1, 0.2, 0.5, 1, 2, 20; константа tol = 0.001. Вычислить ошибку обучения и обобщения, подсчитать количество опрных векторов, вычислить разницу между двумя областями (2 / ||w||), нарисовать решение.

Листинг задачи приведён ниже, полный листинг функции SMO2, svcplot_book и CalcKernel приведён в приложении 1.

```
close('all');
clear;

% Генерируем и рисуем X1
randn('seed',50)
m=[0 0; 1.2 1.2]'; % среднее векторов
S=0.2*eye(2); % матрица ковариации
points_per_class=[200 200];
X1=mvnrnd(m(:,1),S,points_per_class(1))';
X1=[X1 mvnrnd(m(:,2),S,points_per_class(2))'];
y1=[ones(1,points_per_class(1)) -ones(1,points_per_class(2))];

figure(1), plot(X1(1,y1==1),X1(2,y1==1),'r.', X1(1,y1==-1),X1(2,y1==-1),'bo')

% Генерируем X2
randn('seed',100)
X2=mvnrnd(m(:,1),S,points_per_class(1))';
X2=[X2 mvnrnd(m(:,2),S,points_per_class(2))'];
y2=[ones(1,points_per_class(1)) -ones(1,points_per_class(2))];

% Генерируем
SVM
kernel='linear';
kpar1=0;
kpar2=0;
C=0.1;
% Тут можно выбирать нужную константу
% C=0.2;
% C= 0.5;
% C=1;
% C=2;
% C=20;
tol=0.001;
steps=100000;
eps=10^(-10);
method=0;
[alpha, w0, w, evals, stp, glob] = SMO2(X1', y1',kernel, kpar1, kpar2, C, tol, steps, eps,
method);
```

```
% Вычисляем ошибку обучения
Pe_tr=sum((2*(w*X1-w0>0)-1).*y1<0)/length(y1)

% Вычисляем ошибку обобщения
Pe_te=sum((2*(w*X2-w0>0)-1).*y2<0)/length(y2)

% Рисуем оптимальную разделяющую гиперплоскость
global figt4
figt4=2;
svcplot_book(X1',y1',kernel,kpar1,kpar2,alpha,-w0)

% Вычисляем количество опорных векторов
sup_vec=sum(alpha>0)

% Вычисляем разницу для разделительной области
marg=2/sqrt(sum(w.^2))
```

На рисунке 4.3 представлены исходные данные для обучения.

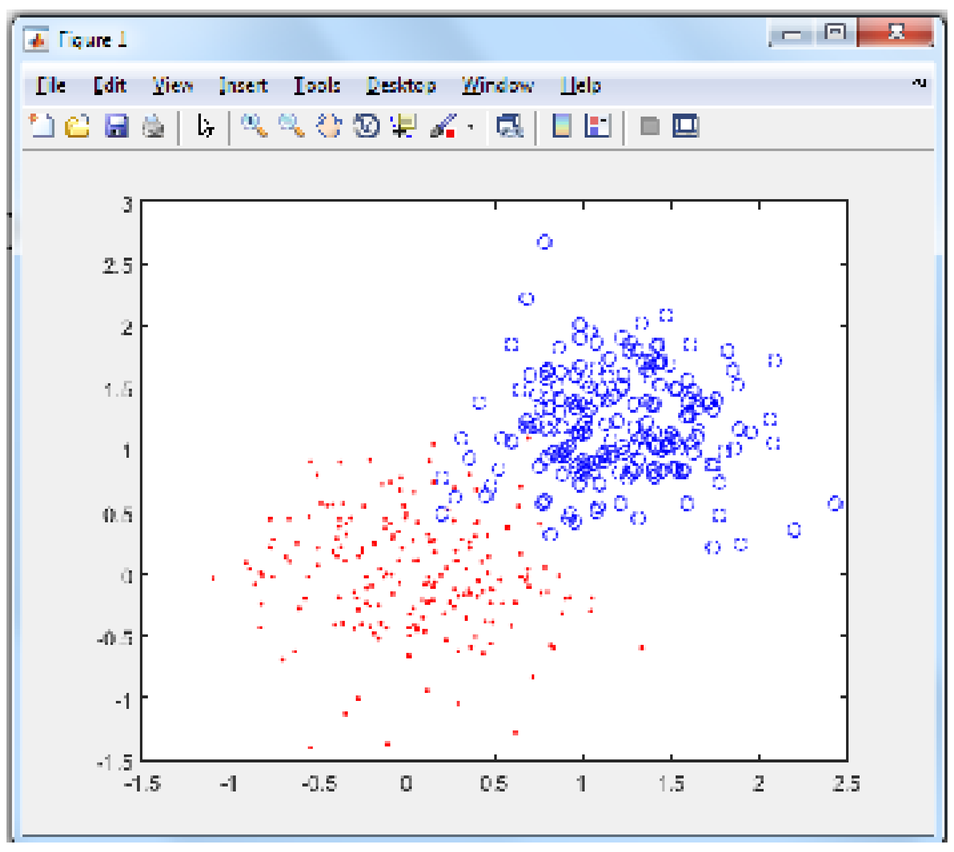

Рисунок 4.3 – Исходные данные для обучения

На рисунке 4.4. представлена нарисованная оптимальная гиперплоскость, которая отделяет один класс от другого

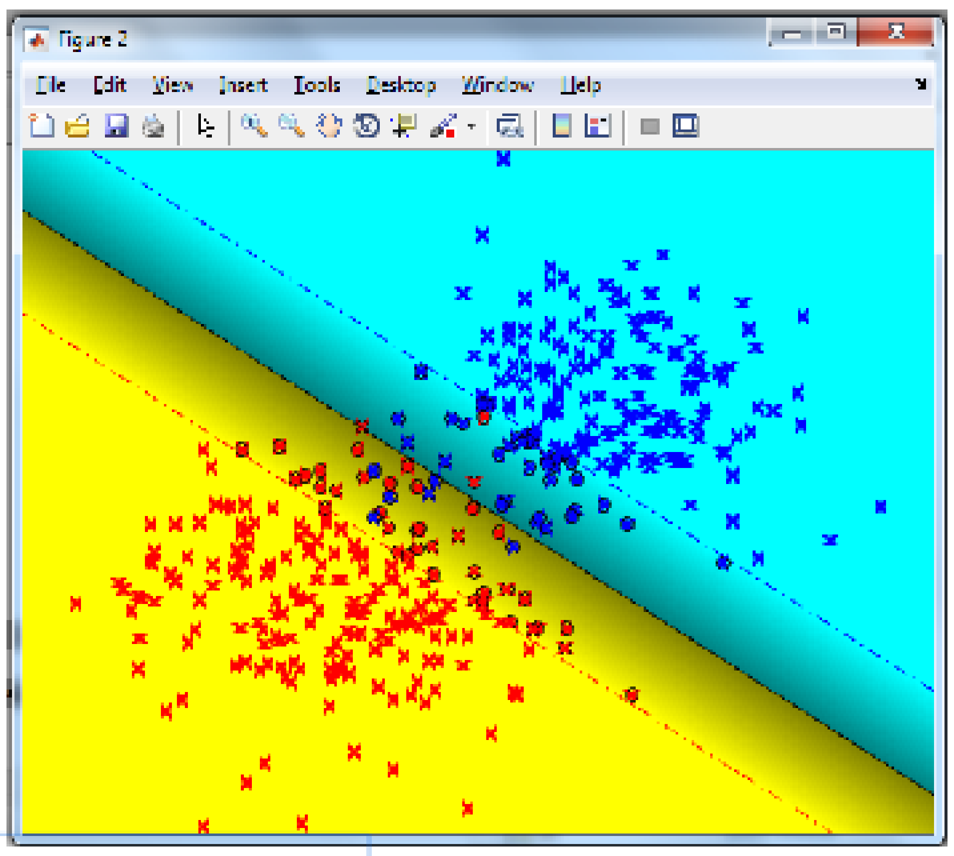

Рисунок 4.4 – Решение на обучающей выборке

В таблице 4.1 представлены результаты работы алгоритма для различных пользовательских констант.

Таблица 4.1 – Сравнительные результаты для различных пользовательских констант С

	C=0.1	C=0.2	C=0.5	C=1	C=2	C=20
Количество поддерживающих векторов	82	61	44	37	31	25
Ошибка обучения	2.25%	2.00%	2.00%	2.25%	3.25%	2.50%
Ошибка на тестовом множестве	3.25%	3.00%	3.25%	3.25%	3.50%	3.50%
Величина зазора (margin)	0.9410	0.8219	0.7085	0.6319	0.6047	0.3573

Из таблицы видно, что ширина разделяющей области увеличивается, когда уменьшается константа С. Это естественно, потому что уменьшение С делает последнее слагаемое в (4.3) более важным. Лучшее решение, т.е. минимальная ошибка обобщения получена для С = 0.2.

Далее рассмотрим многослойную сеть прямого распространения. Общий вид такой сети представлен на рисунке 4.5.

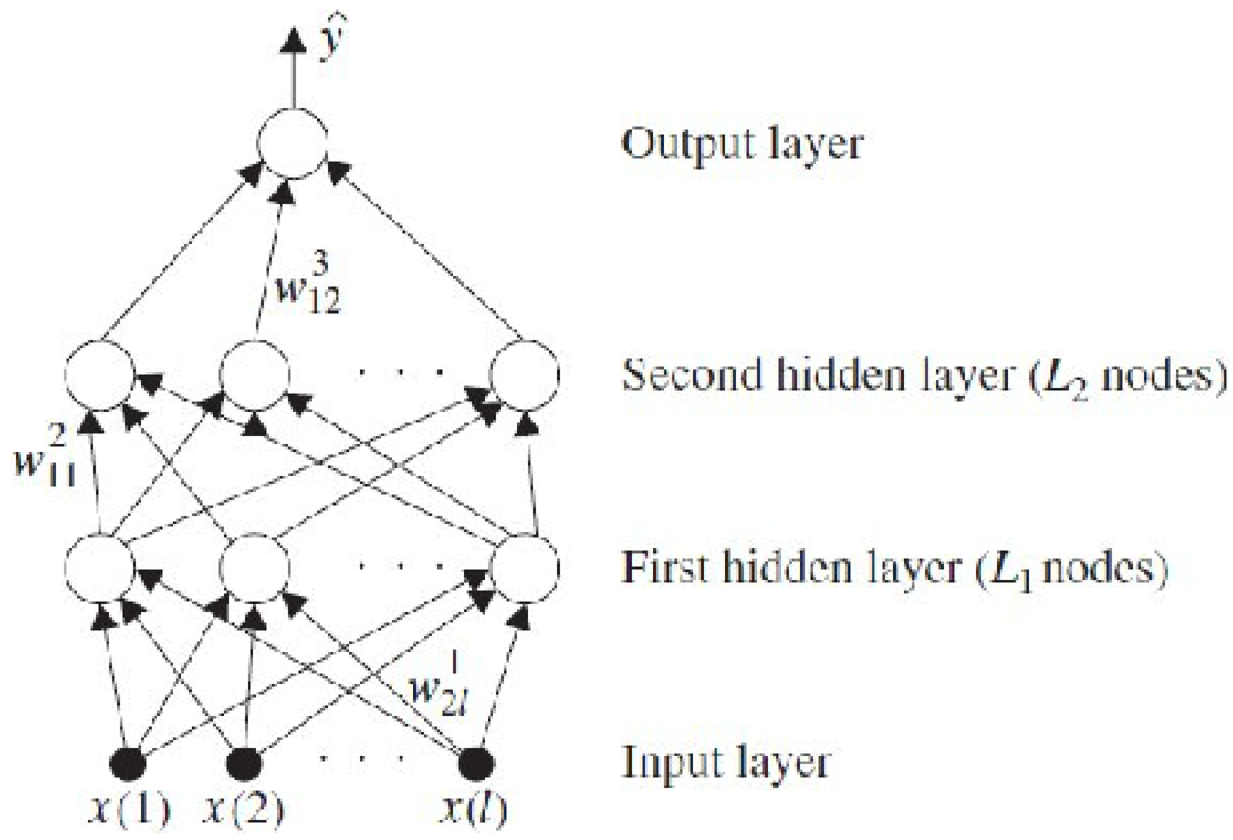

Рисунок 4.5 – Общий вид полносвязанной сети прямого распространения

Общая ошибка обучения вычисляется как

$$J = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \quad (4.6)$$

,

где y_i – предполагаемый ответ сети, а $\hat{y}_i$ – фактический ответ сети. Поиск минимума по поверхности ошибок функции J и есть обучение нейронной сети.

Веса сети обновляются по формуле (4.7):

$$w(new) = w(old) + \Delta w \qquad (4.7)$$

Δw находим исходя из (4.8):

$$\Delta w = -\eta \frac{\partial J}{\partial w}, \qquad (4.8)$$

где η - скорость обучения. Формулу (4.8) можно немного модифицировать, чтобы получилось обучение с инерцией:

$$\Delta w(new) = \alpha \Delta w(old) - \eta \frac{\partial J}{\partial w}, \qquad (4.9)$$

где α - коэффициент инерции, также как и η вводится пользователем и принимает значения от 0 до 1.

Для обучения нейронной сети будем использовать функцию NN_training со следующей сигнатурой: **[net, tr] = NN_training(X, y, k, code, iter, par_vec)**.

Список формальных параметров: X содержит обучающие вектора по колонкам, y – содержит метки для соответствующих классов, code – параметр для определения типа алгоритма обучения (1 – стандартный алгоритм обратного распространения, 2 – обратное распространение с моментом (с инерцией), 3 – обратное распространение с адаптивной скоростью обучения), iter – максимально допустимое количество итераций, par_vec – 5 размерный вектор, содержащий в себе скорость обучения, момент, три значения для третьего алгоритма обучения с адаптивной скоростью обучения.

Список возвращаемых значений: net – структура сети, tr – структура, используемая MATLAB, куда помещаются различные параметры сети.

Пример 2.

Рассмотрим задачу классификации двух классов на плоскости. Точки первого (второго) класса имеют метки +1 (-1) соответственно. Точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-5, 5]^T$, $[5, -5]^T$, $[10, 0]^T$ ($[-5, -5]^T$, $[0, 0]^T$, $[5, 5]^T$, $[15, -5]^T$). Ковариационная матрица для каждого распредления равна $\sigma^2 I$, где $\sigma^2 = 1$, а I – это единичная матрица размером 2x2.

1. Сгенерировать и нарисовать множество X_1 (обучающее множество), содержащее 60 точек для класса +1 (приблизительно по 20 точек для каждого распределения) и 80 точек для класса -1 (также по 20 точек для каждого распределения). Аналогичные предпосылки для формирования X_2 (тестового множества).

2. На X_1 обучить две двухслойный сети прямого распространения с двумя и четырьмя нейронами в скрытом слое. Для всех нейронах на скрытых слоях использовать функцию активации гиперболический тангенс, на выходном слое – линейную функцию

активации. Запустить стандартный алгоритм обратного распространения ошибки для 9000 итераций со скоростью обучения 0.01. Вычислить ошибку обучения и обобщения, и нарисовать регионы решений.

3. Повторить шаг 2, используя скорость обучения 0.0001 для стандартного алгоритма обратного распространения ошибки.

4. Повторить шаг 2, применив адаптивный алгоритм (тип 3) с 6000 итераций, со скоростью обучения 0.0001 и $r_i = 1.05$, $r_d = 0.7$, $c = 1.04$.

Листинг программы рассмотрен ниже.

```
close('all');
clear;

randn('seed',0);
% 1. Генерируем X1
l=2; % Размерность
m1=[-5 5; 5 -5; 10 0]'; % Центроиды
m2=[-5 -5; 0 0; 5 5; 15 -5]';
[l,c1]=size(m1); % Номер гауссиана для каждого класса
[l,c2]=size(m2);

P1=ones(1,c1)/c1; % Веса для смешенной модели
P2=ones(1,c2)/c2;
s=1; % разница

% Генерируем данные для первого класса
N1=60; % Количество точек для первого класса
for i=1:c1
   S1(:,:,i)=s*eye(l);
end
sed=0; % Генератор случайных чисел, параметр для него
[class1_X,class1_y]=mixt_model(m1,S1,P1,N1,sed);

% Генерируем точки для второго класса
N2=80; % Количество точек для второго класса
for i=1:c2
   S2(:,:,i)=s*eye(l);
end
sed=0;
[class2_X,class2_y]=mixt_model(m2,S2,P2,N2,sed);

% Форма X1
X1=[class1_X class2_X]; % Вектор данных
y1=[ones(1,N1) -ones(1,N2)]; % Вектор меток
figure(1), hold on
figure(1), plot(X1(1,y1==1),X1(2,y1==1),'r.',X1(1,y1==-1),X1(2,y1==-1),'bx')

% Генерируем тестовое множество X2

   % Данные первого класса
```

```
sed=100; % Используем генератор случайных чисел
[class1_X,class1_y]=mixt_model(m1,S1,P1,N1,sed);

%Данные для второго класса
sed=100; % Используем генератор случайных чисел
[class2_X,class2_y]=mixt_model(m2,S2,P2,N2,sed);

% Объединяем данные в единый вектор и с метками также
X2=[class1_X class2_X]; %Для данных
y2=[ones(1,N1) -ones(1,N2)]; %Для меток


% 2. Шаг 2

rand('seed',100)
randn('seed',100)
iter=9000; % Количество итераций
code=1; % Выбираем тип алгоритма для обучения нейронной сети
k=4; % число нейронов в скрытом слое
lr=.01; % скорость обучения
par_vec=[lr 0 0 0 0];
[net,tr]=NN_training(X1,y1,k,code,iter,par_vec);

% Вычисляем ошибку обучения и обобщения
pe_train=NN_evaluation(net,X1,y1)
pe_test=NN_evaluation(net,X2,y2)

% Рисуем данные
maxi=max(max([X1'; X2']));
mini=min(min([X1'; X2']));
bou=[mini maxi];
fig_num=2;
resolu=(bou(2)-bou(1))/100; % разрешения рисунка
plot_NN_reg(net,bou,resolu,fig_num); % рисуем регион решения
figure(fig_num), hold on % рисуем обучающее множество
figure(fig_num), plot(X1(1,y1==1),X1(2,y1==1),'r.', X1(1,y1==-1),X1(2,y1==-1),'bx')

figure(3), plot(tr.perf)
pause

% Шаг 4
iter=6000; % Количество итераций
code=3; % Выбираем алгоритм обучения
k=4; % количество нейронов в скрытом слое
lr=.0001; % скорость обучения
par_vec=[lr 0 1.05 0.7 1.04]; % вектор параметров для алгоритма code = 3
[net,tr]=NN_training(X1,y1,k,code,iter,par_vec);

% Вычисляем ошибку обучения и обобщения
pe_train=NN_evaluation(net,X1,y1)
pe_test=NN_evaluation(net,X2,y2)
```

Первоначальные данные представлены на рисунке 4.6.

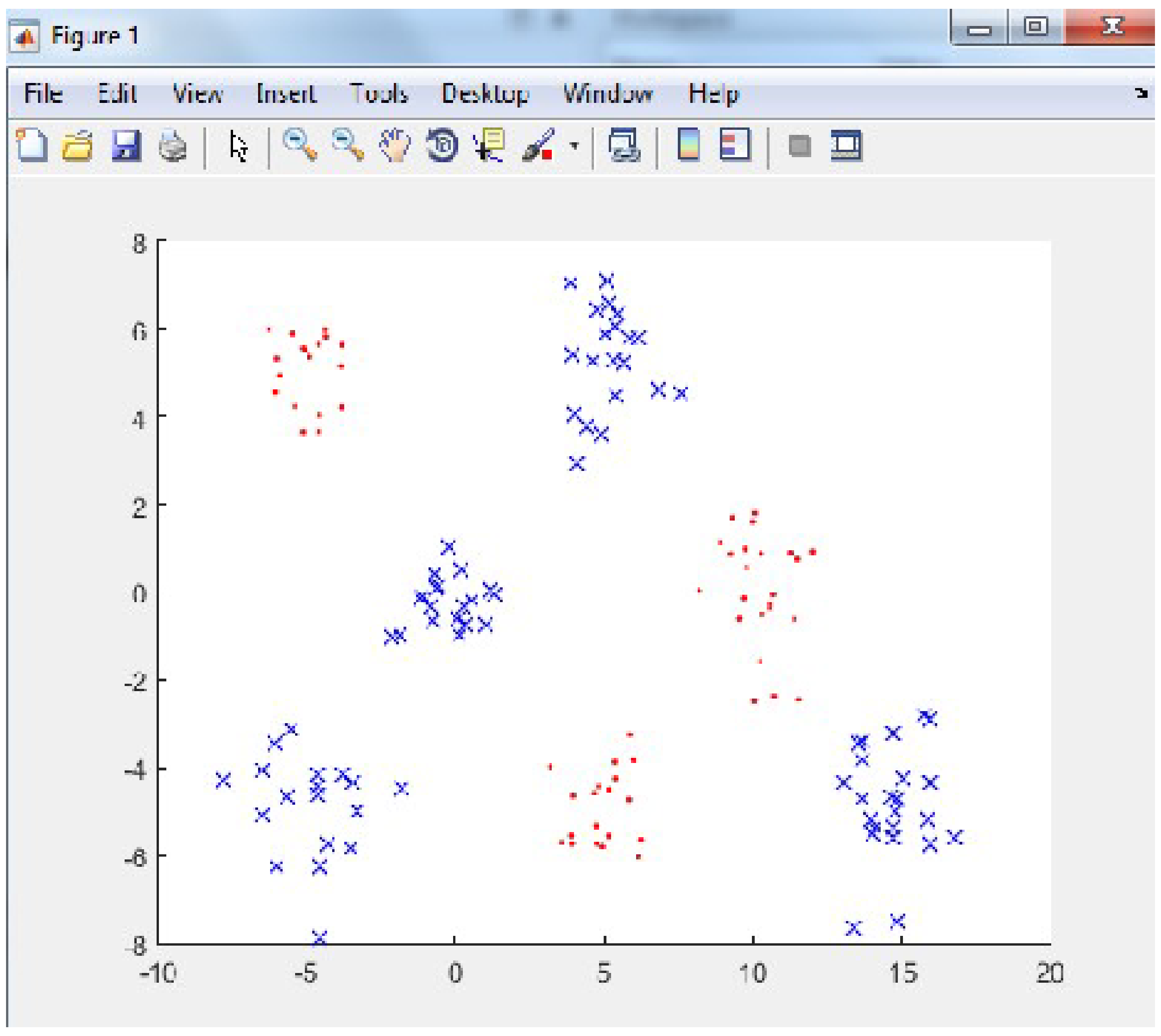

Рисунок 4.6 – Данные, которые необходимо разделить

Получившееся решение представлено на рисунке 4.7, видно, что сеть проведя разделяющие границы успешно отделяет один класс от другого.

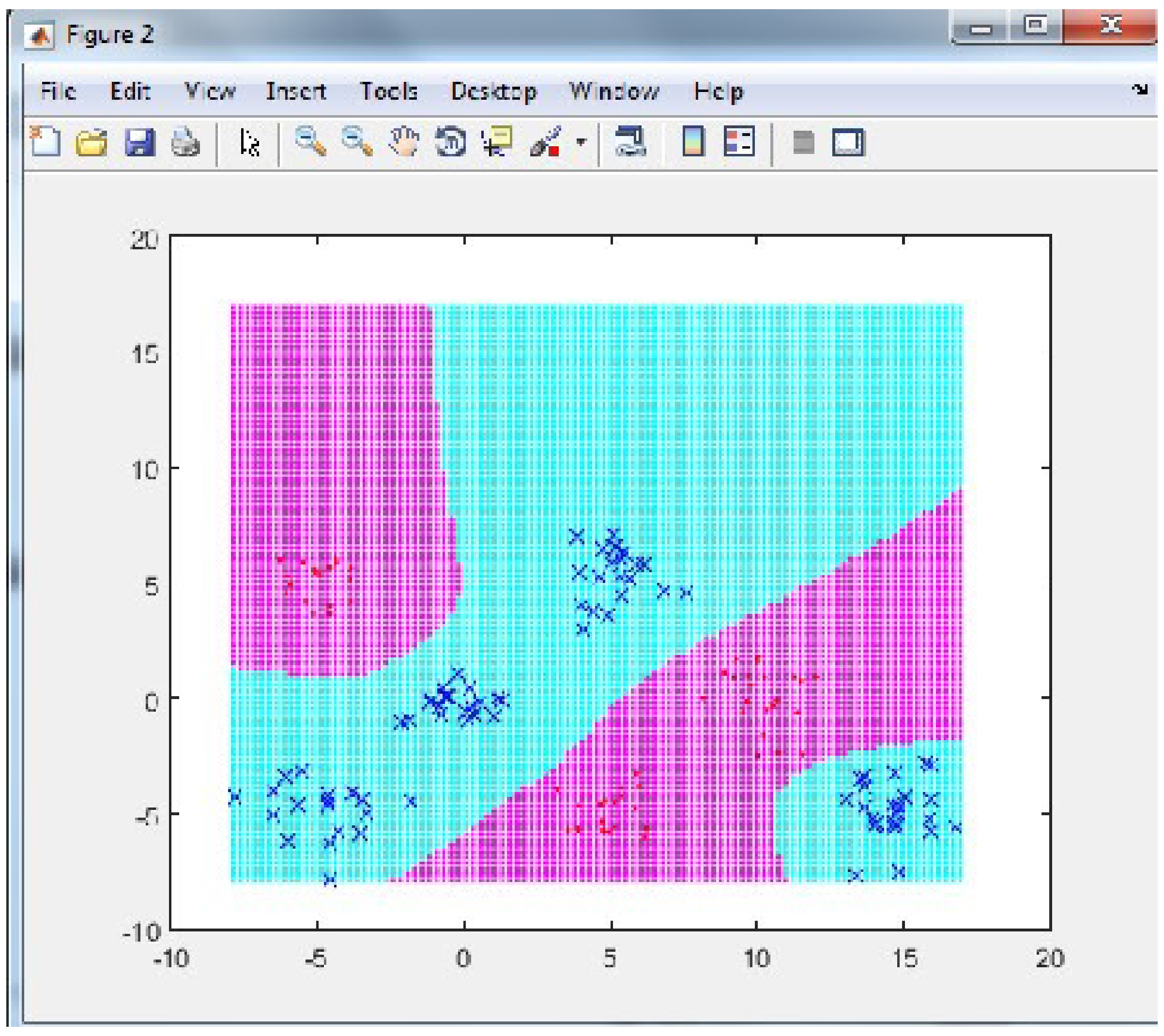

Рисунок 4.7 – Получившееся решение

Зависимость ошибки обучения от эпохи приведена на рисунке 4.8.

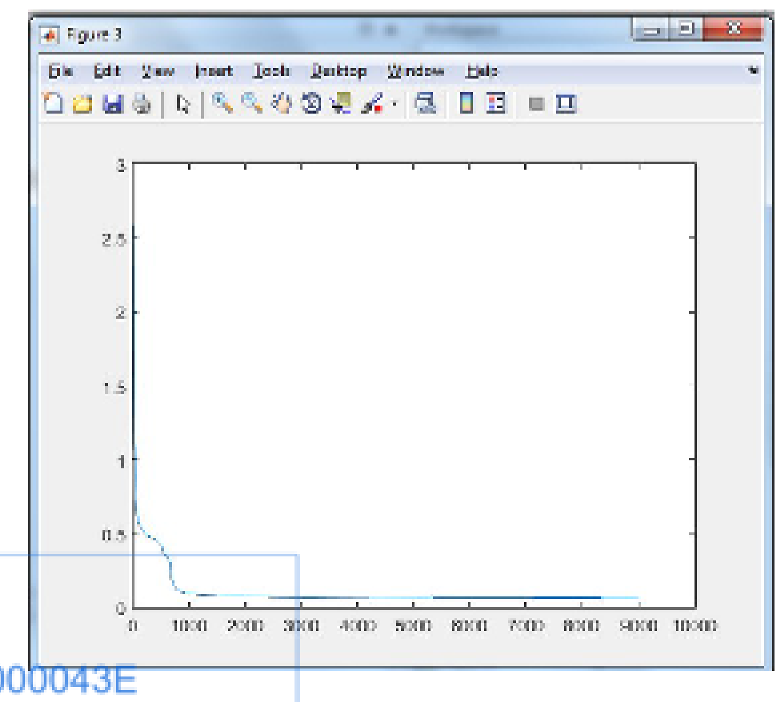

Рисунок 4.8 – Зависимость ошибки обучения от эпохи

После 2000-ой эпохи обучение практически останавливается.

Общие параметры обучения приведены на рисунке 4.9.

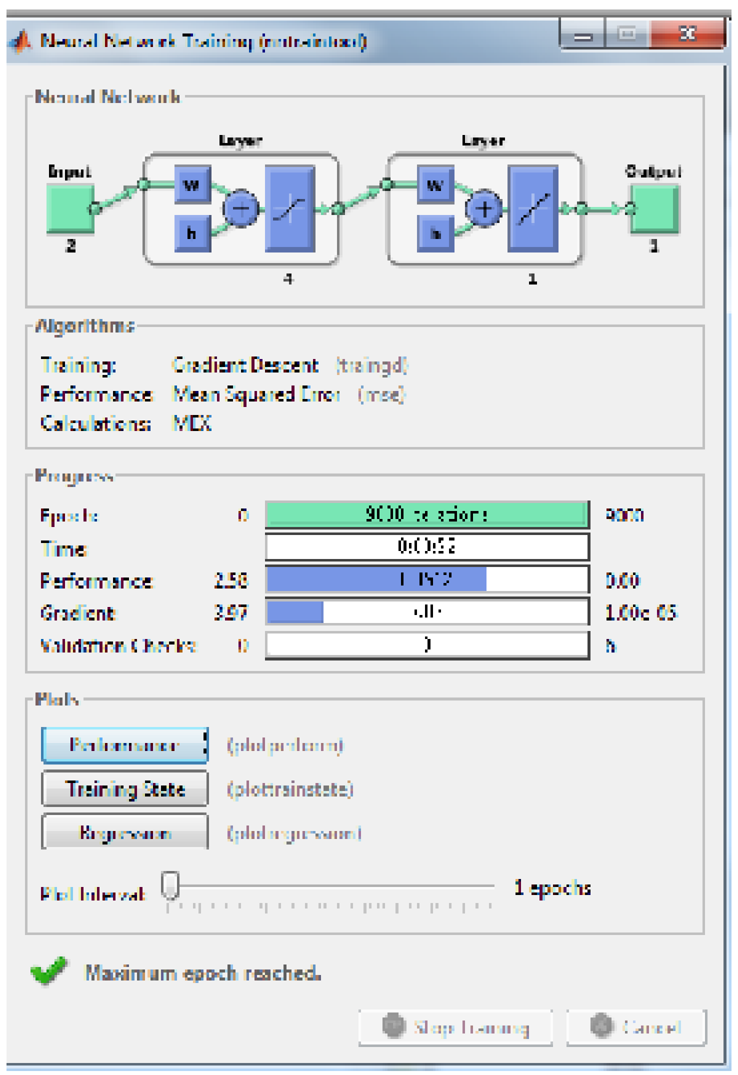

Рисунок 4.9 – Окно среды Matlab, отражающее процесс обучения нейронной сети

В таблице 4.2 приведены результаты обучения сети для стандартного алгоритма обратного распространения ошибки со скоростью обучения 0.01 и 9000 итераций. В таблице 4.2 приведены результаты для адаптивного алгоритма обратного распространения ошибки, скорости обучения 0.0001 и 6000 итераций.

Таблица 4.2 – Применение стандартного алгоритма обратного распространения ошибки

	Два узла	Четыре узла
Ошибка обучения	29.29%	0
Ошибка обобщения	30.71%	0

Таблица 4.3 – Применение адаптивного алгоритма обратного распространения ошибки

	Два узла	Четыре узла
Ошибка обучения	29.29%	0
Ошибка обобщения	32.14%	0

Функции, использующиеся в листинге, приведены ниже.

Первая функция.

```
function [net,tr]=NN_training(X,y,k,code,iter,par_vec)

rand('seed',0);  %Датчик случайных чисел
randn('seed',0);

%Список методов обучения для ИНС, они встроены в Matlab
methods_list={'traingd'; 'traingdm'; 'traingda';'trainlm'; 'traingdx'};
```

```
% Ограничение для области, где лежат данные
limit=[min(X(:,1)) max(X(:,1)); min(X(:,2)) max(X(:,2))];
% Определение нейронной сети
net=newff(limit,[k 1],{'tansig','purelin'}, methods_list{code,1}); %  'traingda')

% Инициализация нейронной сети
net=init(net);

% Установка параметров
net.trainParam.epochs=iter;
net.trainParam.lr=par_vec(1);
if(code==2)
   net.trainParam.mc=par_vec(2);
elseif(code==3)
   net.trainParam.lr_inc=par_vec(3);
   net.trainParam.lr_dec=par_vec(4);
   net.trainParam.max_perf_inc=par_vec(5);
end

% Обучение нейронной сети
[net,tr]=train(net,X,y);
```

Вторая функция.

```
function [X,y]=mixt_model(m,S,P,N,sed)

rand('seed',sed);
[l,c]=size(m);

P_acc=P(1);
for i=2:c
   t=P_acc(i-1)+P(i);
   P_acc=[P_acc t];
end

X=[];
y=[];
for i=1:N
   t=rand;
   ind=sum(t>P_acc)+1;
   X=[X; mvnrnd(m(:,ind)',S(:,:,ind),1)];
   y=[y ind];
end
X=X';
```

Третья функция.

```
function pe=NN_evaluation(net,X,y)

y1=sim(net,X);

pe=sum(y.*y1<0)/length(y);
```

Четвертая функция.

```
function plot_NN_reg(net,bou,resolu,fig_num)

t=round(((bou(2)-bou(1))/resolu)+1);
t_vec=bou(1):resolu:bou(2);
t_vec=t_vec';
X1=[];
for i=bou(1):resolu:bou(2)
   X1=[X1; i*ones(t,1) t_vec];
end
X1=X1';
out_vec=2*(sim(net,X1)>0)-1;
figure(fig_num),
plot(X1(1,out_vec>0),X1(2,out_vec>0),'m.',X1(1,out_vec<0),X1(2,out_vec<0),'c.')
```

Аппаратура и материалы. 64-разрядный (x64) персональный компьютер, процессор с тактовой частотой 1 ГГц и выше, оперативная память 1 Гб и выше, свободное дисковое пространство не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Matlab (R2013) и выше.

Указание по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения; ремонт персонального компьютера. Соблюдать правила технической эксплуатации и техники безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Решить собственный вариант по образцу примера 1 и 2. Индивидуальные варианты представлены в таблице 4.4.

Таблица 4.4 – Индивидуальные варианты

1	Первая задача. Рассмотреть задачу из примера 1 с параметрами: $m_1 = [0, 0]T$ и $m_2 = [1, 1]T$ и ковариационными матрицами $S_1 = S_2 = 0.5I$, количество паттернов 400, а не 200. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят

	от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-3, 3]^T$, $[2, -3]^T$, $[7, 1]^T$ ($[-7, -7]^T$, $[0, 0]^T$, $[10, 10]^T$, $[10, -7]^T$).
2	Первая задача. Рассмотреть задачу из примера 1 с параметрами: m_1 = [0, 0]T и m_2 = [1.7, 1.5]T и ковариационными матрицами $S_1 = S_2$ = 0.1I, количество паттернов 200. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-1.5, 2.5]^T$, $[3.5, -4.5]^T$, $[10, -3]^T$ ($[-6, -4]^T$, $[0, 0]^T$, $[6, 6]^T$, $[17, -8]^T$).
3	Первая задача. Рассмотреть задачу из примера 1 с параметрами: m_1 = [-1, -1]T и m_2 = [1.5, 1.5]T и ковариационными матрицами $S_1 = S_2$ = 0.7I, количество паттернов 200. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-5, 5]^T$, $[7, -5]^T$, $[10, 1]^T$ ($[-5, -6]^T$, $[1, 1.5]^T$, $[5.6, 5.9]^T$, $[15, -10]^T$).
4	Первая задача. Рассмотреть задачу из примера 1 с параметрами: m_1 = [3, 3]T и m_2 = [1.5, 1.5]T и ковариационными матрицами $S_1 = S_2$ = 0.5I, количество паттернов 400. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-10, 5]^T$, $[5, -10]^T$, $[20, 0]^T$ ($[-10, -10]^T$, $[0, 0]^T$, $[7, 5]^T$, $[15, -6]^T$).
5	Первая задача. Рассмотреть задачу из примера 1 с параметрами: m_1 = [0, 0]T и m_2 = [1.8, 1.8]T и ковариационными матрицами $S_1 = S_2$ = 0.1I, количество паттернов 400. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-2, 2]^T$, $[2, -2]^T$, $[7, 0]^T$ ($[-5, -5]^T$, $[1, 1.5]^T$, $[6, 6.5]^T$, $[19, -5.5]^T$).
6	Первая задача. Рассмотреть задачу из примера 1 с параметрами: m_1 = [0, 0]T и m_2 = [1.5, 1.4]T и ковариационными матрицами $S_1 = S_2$ = 0.3I, количество паттернов 200. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-5.5, 5]^T$, $[6.7, -4.5]^T$, $[10, 0]^T$ ($[-4, -4]^T$, $[1.1, 1.5]^T$, $[5, 5]^T$, $[20, -5]^T$).
7	Первая задача. Рассмотреть задачу из примера 1 с параметрами: m_1 = [1.1, 0]T и m_2 = [1.5, 1.9]T и ковариационными матрицами $S_1 = S_2$ = 0.2I, количество паттернов 200. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-5, 5]^T$, $[5, -7]^T$, $[10, 0]^T$ ($[-5, -5]^T$, $[1, 1]^T$, $[8, 8]^T$, $[15, -9]^T$).
8	Первая задача. Рассмотреть задачу из примера 1 с параметрами: m_1 = [0, 0]T и m_2 = [1.6, 1.6]T и ковариационными матрицами $S_1 = S_2$ = 0.4I, количество паттернов 400. Вторая задача. Рассмотреть задачу из примера 2 с параметрами: точки происходят от трех (четырех) Гауссовых распределений с одинаковыми вероятностями: $[-5, 6]^T$, $[6, -5]^T$, $[12, 2]^T$ ($[-6, -6]^T$, $[1, 1.4]^T$, $[5, 5]^T$, $[19, -5]^T$).

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать следующую информацию:

1) Название лабораторной работы и её номер.
2) ФИО и группу студанта.
3) Формулировка индивидуального задания и номер варианта.

4) Ответ на задание в виде кода и результатов работы Matlab.

Вопросы для защиты работы

1. Чем SVM отличается от сети прямого распространения?
2. Чем SVM отличается от персептрона?
3. Что такое оптимальная разделяющая гиперплоскость?

Лабораторная работа № 5

Пример создания и обучения нейронных сетей для задач классификации в среде Matlab. Часть 1

Цель и содержание работы: создать и обучить несколько нейронных сетей, решающих задачи классификации на примере задачи ирисов Фишера и двух спиралей.

Задачи:

- разобрать решение задачи об ирисах Фишера;
- научиться обучать сети с разной архитектурой и отбирать из них наилучшие на примере задачи об ирисах;
- разобрать решение задачи двух спиралей;
- научиться подбирать ряд параметров для архитектуры нейронных сетей на основе задаче о двух спиралях;
- научиться создавать сети с помощью разных функций Matlab и обучать их с помощью разных алгоритмов на основе задачи о двух спиралях;
- разобрать код обратного распространения ошибки для задачи о двух спиралях.

Теоретическое обоснование

В предыдущей лабораторной работе рассматривались очень лёгкие задачи классификации, причём тестовое и валидационное множество отсутствовали. В этой лабораторной работе будут рассмотрены два реальных примера задачи классификации.

Первая задача – это задача ирисов Фишера. Ирисы Фишера состоят из данных о 150 экземплярах ириса, по 50 экземпляров из трёх видов – Ирис щетинистый (Iris setosa), Ирис виргинский (Iris virginica), Ирис разноцветный (Iris versicolor), рисунок 5.1.

Рисунок 5.1 – Слева направо: Iris versicolor, Iris virginica, Iris setosa

Каждый экземпляр того или иного ириса характеризовался четырьмя числами, которые выражали определённые характеристики цветков: длина наружной доли околоцветника (sepal length), ширина наружной доли околоцветника (sepal width), длина внутренней доли околоцветника (petal length), ширина внутренней доли околоцветника (petal width); всё в сантиметрах. На основании этого набора данных требуется построить правило классификации, определяющее вид растения по данным измерений. Это задача многоклассовой классификации, т.к. имеются три вида ириса.

Все данные для данной задачи приведены в таблице 5.1. Именно в подобном виде записывается большинство задач классификации. Стоит обратить внимание, что данные приведены в вещественном формате, где разделителем служит запятая, а не точка. Это сделано для того, чтобы Excel правильно интерпретировал вещественные числа, т.к. в этой среде разделитель – запятая, а точка обычно используется для записи даты.

Таблица 5.1 – Данные для задачи ирисов Фишера

№	**Sep**	**Sepa**	**Petal**	**Petal**	**Species**	**№**	**Sepa**	**Sepa**	**Petal**	**Petal**	**Species**
1	5,1	3,5	1,4	0,2	*I, setosa*	76	6,6	3,0	4,4	1,4	*I. versicol*
2	4,9	3,0	1,4	0,2	*I, setosa*	77	6,8	2,8	4,8	1,4	*I. versicol*
3	4,7	3,2	1,3	0,2	*I, setosa*	78	6,7	3,0	5,0	1,7	*I. versicol*
4	4,6	3,1	1,5	0,2	*I, setosa*	79	6,0	2,9	4,5	1,5	*I. versicol*
5	5,0	3,6	1,4	0,2	*I, setosa*	80	5,7	2,6	3,5	1,0	*I. versicol*
6	5,4	3,9	1,7	0,4	*I, setosa*	81	5,5	2,4	3,8	1,1	*I. versicol*
7	4,6	3,4	1,4	0,3	*I, setosa*	82	5,5	2,4	3,7	1,0	*I. versicol*
8	5,0	3,4	1,5	0,2	*I, setosa*	83	5,8	2,7	3,9	1,2	*I. versicol*
9	4,4	2,9	1,4	0,2	*I, setosa*	84	6,0	2,7	5,1	1,6	*I. versicol*
10	4,9	3,1	1,5	0,1	*I, setosa*	85	5,4	3,0	4,5	1,5	*I. versicol*
11	5,4	3,7	1,5	0,2	*I, setosa*	86	6,0	3,4	4,5	1,6	*I. versicol*
12	4,8	3,4	1,6	0,2	*I, setosa*	87	6,7	3,1	4,7	1,5	*I. versicol*
13	4,8	3,0	1,4	0,1	*I, setosa*	88	6,3	2,3	4,4	1,3	*I. versicol*
14	4,3	3,0	1,1	0,1	*I, setosa*	89	5,6	3,0	4,1	1,3	*I. versicol*
15	5,8	4,0	1,2	0,2	*I, setosa*	90	5,5	2,5	4,0	1,3	*I. versicol*
16	5,7	4,4	1,5	0,4	*I, setosa*	91	5,5	2,6	4,4	1,2	*I. versicol*
17	5,4	3,9	1,3	0,4	*I, setosa*	92	6,1	3,0	4,6	1,4	*I. versicol*
18	5,1	3,5	1,4	0,3	*I, setosa*	93	5,8	2,6	4,0	1,2	*I. versicol*
19	5,7	3,8	1,7	0,3	*I, setosa*	94	5,0	2,3	3,3	1,0	*I. versicol*
20	5,1	3,8	1,5	0,3	*I, setosa*	95	5,6	2,7	4,2	1,3	*I. versicol*
2	5,	3,4	1,7	0,2	*I,*	9	5,7	3,0	4,2	1,2	*I.*

1	4				*setosa*	6					*versicol*
22	5,1	3,7	1,5	0,4	*I, setosa*	97	5,7	2,9	4,2	1,3	*I. versicol*
23	4,6	3,6	1,0	0,2	*I, setosa*	98	6,2	2,9	4,3	1,3	*I. versicol*
24	5,1	3,3	1,7	0,5	*I, setosa*	99	5,1	2,5	3,0	1,1	*I. versicol*
25	4,8	3,4	1,9	0,2	*I, setosa*	100	5,7	2,8	4,1	1,3	*I. versicol*
26	5,0	3,0	1,6	0,2	*I, setosa*	101	6,3	3,3	6,0	2,5	*I. virginic*
27	5,0	3,4	1,6	0,4	*I, setosa*	102	5,8	2,7	5,1	1,9	*I. virginic*
28	5,2	3,5	1,5	0,2	*I, setosa*	103	7,1	3,0	5,9	2,1	*I. virginic*
29	5,2	3,4	1,4	0,2	*I, setosa*	104	6,3	2,9	5,6	1,8	*I. virginic*
30	4,7	3,2	1,6	0,2	*I, setosa*	105	6,5	3,0	5,8	2,2	*I. virginic*
31	4,8	3,1	1,6	0,2	*I, setosa*	106	7,6	3,0	6,6	2,1	*I. virginic*
32	5,4	3,4	1,5	0,4	*I, setosa*	107	4,9	2,5	4,5	1,7	*I. virginic*
33	5,2	4,1	1,5	0,1	*I, setosa*	108	7,3	2,9	6,3	1,8	*I. virginic*
34	5,5	4,2	1,4	0,2	*I, setosa*	109	6,7	2,5	5,8	1,8	*I. virginic*
35	4,9	3,1	1,5	0,2	*I, setosa*	110	7,2	3,6	6,1	2,5	*I. virginic*

36	5,0	3,2	1,2	0,2	*I, setosa*	111	6,5	3,2	5,1	2,0	*I. virginic*
37	5,5	3,5	1,3	0,2	*I, setosa*	112	6,4	2,7	5,3	1,9	*I. virginic*
38	4,9	3,6	1,4	0,1	*I, setosa*	113	6,8	3,0	5,5	2,1	*I. virginic*
39	4,4	3,0	1,3	0,2	*I, setosa*	114	5,7	2,5	5,0	2,0	*I. virginic*
40	5,1	3,4	1,5	0,2	*I, setosa*	115	5,8	2,8	5,1	2,4	*I. virginic*
41	5,0	3,5	1,3	0,3	*I, setosa*	116	6,4	3,2	5,3	2,3	*I. virginic*
42	4,5	2,3	1,3	0,3	*I, setosa*	117	6,5	3,0	5,5	1,8	*I. virginic*
43	4,4	3,2	1,3	0,2	*I, setosa*	118	7,7	3,8	6,7	2,2	*I. virginic*
44	5,0	3,5	1,6	0,6	*I, setosa*	119	7,7	2,6	6,9	2,3	*I. virginic*
45	5,1	3,8	1,9	0,4	*I, setosa*	120	6,0	2,2	5,0	1,5	*I. virginic*
46	4,8	3,0	1,4	0,3	*I, setosa*	121	6,9	3,2	5,7	2,3	*I. virginic*
47	5,1	3,8	1,6	0,2	*I, setosa*	122	5,6	2,8	4,9	2,0	*I. virginic*
48	4,6	3,2	1,4	0,2	*I, setosa*	123	7,7	2,8	6,7	2,0	*I. virginic*
49	5,3	3,7	1,5	0,2	*I, setosa*	124	6,3	2,7	4,9	1,8	*I. virginic*
50	5,0	3,3	1,4	0,2	*I, setosa*	125	6,7	3,3	5,7	2,1	*I. virginic*
51	7,0	3,2	4,7	1,4	*I, versicol*	126	7,2	3,2	6,0	1,8	*I. virginic*
52	6,4	3,2	4,5	1,5	*I, versicol*	127	6,2	2,8	4,8	1,8	*I. virginic*
53	6,9	3,1	4,9	1,5	*I, versicol*	128	6,1	3,0	4,9	1,8	*I. virginic*
54	5,5	2,3	4,0	1,3	*I, versicol*	129	6,4	2,8	5,6	2,1	*I. virginic*
55	6,	2,8	4,6	1,5	*I, versicol*	130	7,2	3,0	5,8	1,6	*I. virginic*

	5										
56	5,7	2,8	4,5	1,3	*I, versicol*	131	7,4	2,8	6,1	1,9	*I. virginic*
57	6,3	3,3	4,7	1,6	*I, versicol*	132	7,9	3,8	6,4	2,0	*I. virginic*
58	4,9	2,4	3,3	1,0	*I, versicol*	133	6,4	2,8	5,6	2,2	*I. virginic*
59	6,6	2,9	4,6	1,3	*I, versicol*	134	6,3	2,8	5,1	1,5	*I. virginic*
60	5,2	2,7	3,9	1,4	*I, versicol*	135	6,1	2,6	5,6	1,4	*I. virginic*
61	5,0	2,0	3,5	1,0	*I, versicol*	136	7,7	3,0	6,1	2,3	*I. virginic*
62	5,9	3,0	4,2	1,5	*I, versicol*	137	6,3	3,4	5,6	2,4	*I. virginic*
63	6,0	2,2	4,0	1,0	*I, versicol*	138	6,4	3,1	5,5	1,8	*I. virginic*
64	6,1	2,9	4,7	1,4	*I, versicol*	139	6,0	3,0	4,8	1,8	*I. virginic*
65	5,6	2,9	3,6	1,3	*I, versicol*	140	6,9	3,1	5,4	2,1	*I. virginic*
66	6,7	3,1	4,4	1,4	*I, versicol*	141	6,7	3,1	5,6	2,4	*I. virginic*
67	5,6	3,0	4,5	1,5	*I, versicol*	142	6,9	3,1	5,1	2,3	*I. virginic*
68	5,8	2,7	4,1	1,0	*I, versicol*	143	5,8	2,7	5,1	1,9	*I. virginic*
69	6,2	2,2	4,5	1,5	*I, versicol*	144	6,8	3,2	5,9	2,3	*I. virginic*
70	5,6	2,5	3,9	1,1	*I, versicol*	145	6,7	3,3	5,7	2,5	*I. virginic*
71	5,9	3,2	4,8	1,8	*I, versicol*	146	6,7	3,0	5,2	2,3	*I. virginic*
72	6,1	2,8	4,0	1,3	*I, versicol*	147	6,3	2,5	5,0	1,9	*I. virginic*
73	6,3	2,5	4,9	1,5	*I, versicol*	148	6,5	3,0	5,2	2,0	*I. virginic*
74	6,1	2,8	4,7	1,2	*I, versicol*	149	6,2	3,4	5,4	2,3	*I. virginic*
7	6	2,9	4,3	1,3	*I,*	15	5,	3,0	5,1	1,8	*I.*

5	, 4				*versicol*	0	9				*virginic*

Данная выборка по ирисам является одной из классических задач классификации. Чтобы оценить сложность разделимости классов, можно отобразить точки в пространстве признаков. Так как каждый цветок характеризуется четырьмя числами, то искомая точка лежит в 4-х мерном пространстве, отобразить нельзя, но можно отобразить от двух и от трёх параметров. На рисунке 5.2 приведено расположение классов в зависимости от двух параметров, на рисунке 5.3 – от трёх параметров.

Рисунок 5.2 – Расположение экземпляров классов ирисов Фишера в зависимости от двух параметров: petal width и petal length

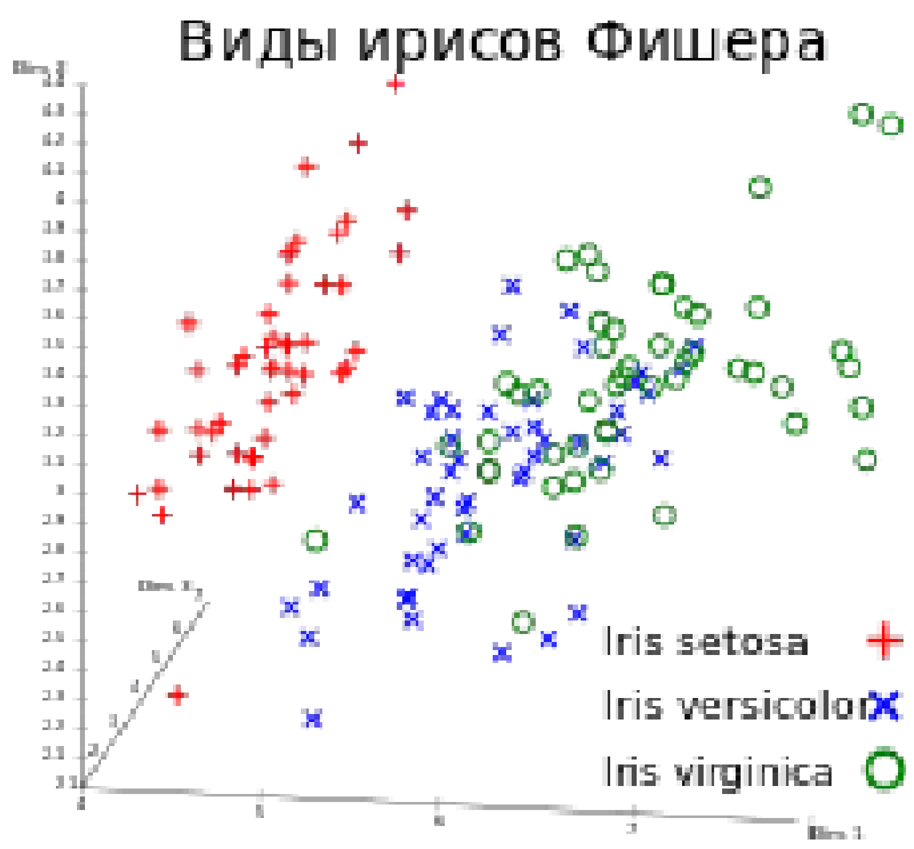

Рисунок 5.3 – Расположение экземпляров классов ирисов Фишера в зависимости от трёх параметров: petal width, petal length, sepal width

Из рисунков видно, что Iris setosa (на рисунке 5.2 – синие крестики, на рисунке 5.3 – красные крестики) линейно отделим от двух других классов, которые не слишком сильно смешаны между собой. Т.е. задача не слишком сложная, каждый класс имеет ярко выраженный собственный центр в пространстве признаков.

Вторая задача – это задача о спиралях, которые нужно разделить. Обычно имеется в виду два класса, экземпляр каждого класса – это точка в двумерном пространстве, каждый класс – это одна из непересекающихся спиралей, рисунок 5.4. Соответственно нужно отделить одну спираль от другой.

Рисунок 5.4 – Две непересекающиеся спирали

Эта задача – пример синтетических (сгенерированных человеком) данных. Есть разные варианты этой задачи: 3 спирали, 5 спиралей и т.д.

Пример пяти спиралей приведён на рисунке 5.5.

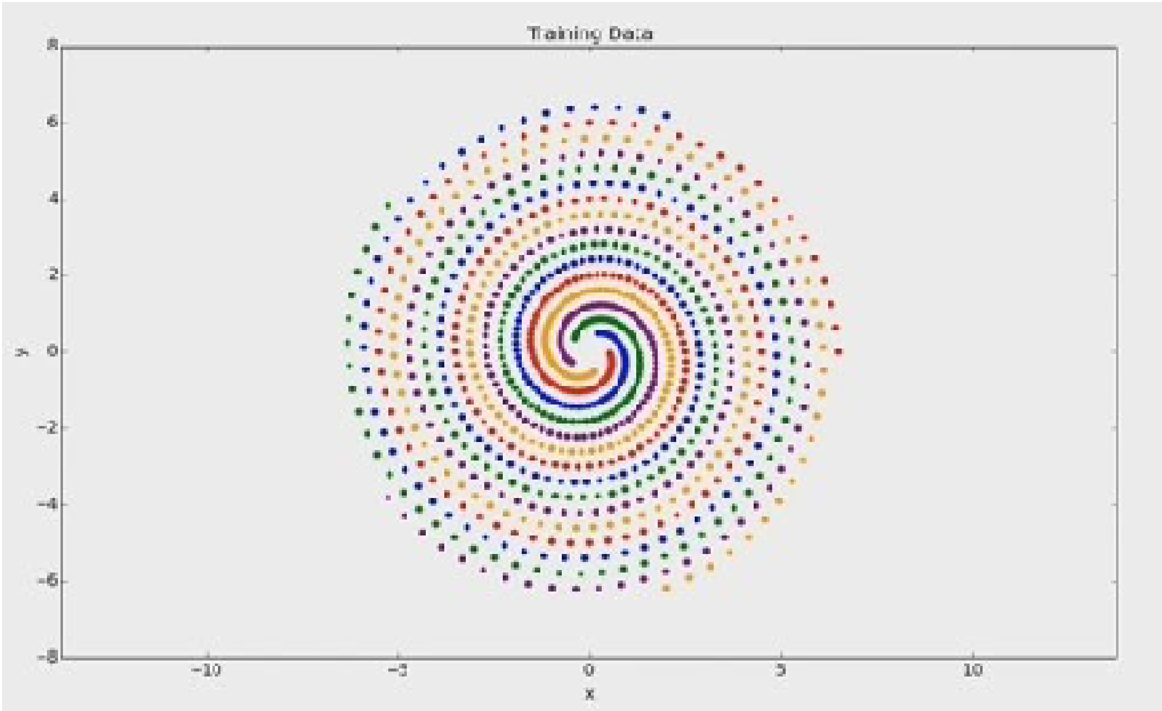

Рисунок 5.5 – Пять спиралей

Соответственно решение для подобных задач будет представлять из себя N спиралевидных поверхностей из сигмойд, на вершине каждой лежит нужный класс, а в ложбине все остальные классы. Пример такой поверхности для пяти спиралей показан на рисунке 5.6 (поверхность отображается в данном случае через ответы сети, покрашенные в разный цвет).

Рисунок 5.6 – Разделение пяти спиралей

На рисунке 5.7 показан пример поверхности отклика для двух спиралей, синий цвет – ложбина для первой спирали, красный цвет – возвышенность для второй спирали.

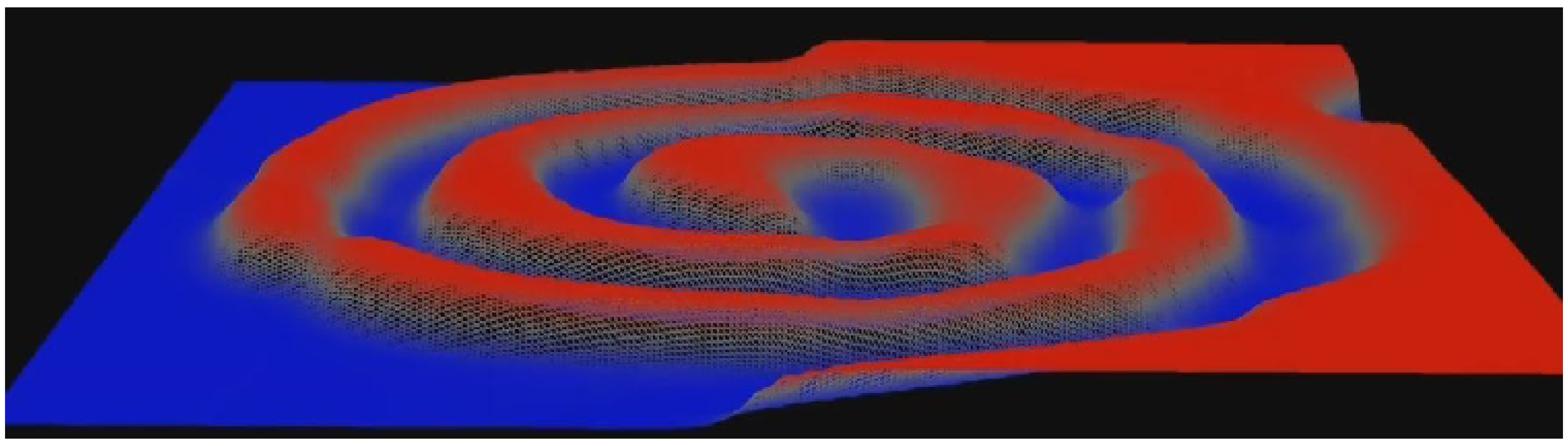

Рисунок 5.7 – Поверхность отклика для задачи о разделении двух спиралей

Для того, чтобы детально разобраться как происходит построение поверхности отклика смотрите приложение 1.

В отличие от задач первого класса, которые встречаются в реальной «природе», такие задачи для ИНС сложны и созданы главным образом для тестирования мощности той или иной сети (Так, для пяти спиралей требуется больше 16000 эпох обучения).

Задача о двух спиралях появилась впервые в [1]. В статье также приведён код на С по генерированию этого набора данных.

В таблице 5.2 приведены данные для задачи двух спиралей.

Таблица 5.2 – Данные для задачи о разделении двух спиралей

№	x	y	Class	№	x	y	Class
1	6,5	0	1	98	3,5	0,00008	-1
2	-6,5	0	-1	99	-3,37143	-0,6707	1
3	6,3138	1,2559	1	100	3,37143	0,6707	-1
4	-6,3138	-1,2559	-1	101	-3,11806	-1,29163	1
5	5,88973	2,4396	1	1	3,1180	1,2916	-1

		1		02	6	3	
6	-5,88973	-2,43961	-1	103	-2,7542	-1,84039	1
7	5,24865	3,50704	1	104	2,7542	1,84039	-1
8	-5,24865	-3,50704	-1	105	-2,29804	-2,29815	1

9	4,41941	4,41943	1	106	2,29804	2,29815	-1
10	-4,41941	-4,41943	-1	107	-1,77082	-2,65035	1
11	3,43758	5,14473	1	108	1,77082	2,65035	-1
12	-3,43758	-5,14473	-1	109	-1,19581	-2,88715	1
13	2,34392	5,65877	1	110	1,19581	2,88715	-1
14	-2,34392	-5,65877	-1	111	-0,59739	-3,00367	1
15	1,18272	5,94601	1	112	0,59739	3,00367	-1
16	-1,18272	-5,94601	-1	113	0,00008	-3	1
17	-0,00002	6	1	114	-0,00008	3	-1
18	0,00002	-6	-1	115	0,57315	-2,88104	1
19	-1,15837	5,82341	1	116	-0,57315	2,88104	-1
20	1,15837	-5,82341	-1	117	1,10029	-2,65612	1
21	-2,24829	5,42778	1	118	-1,10029	2,65612	-1
22	2,24829	-5,42778	-1	119	1,5626	-2,33847	1
23	-3,22928	4,8329	1	120	-1,5626	2,33847	-1
24	3,22928	-4,8329	-1	121	1,9446	-1,94449	1
25	-4,06589	4,06584	1	122	-1,9446	1,94449	-1
26	4,06589	-4,06584	-1	123	2,23462	-1,49303	1
27	-4,729	3,15978	1	124	-2,23462	1,49303	-1

28	4,729	-3,15978	-1	125	2,42521	-1,00447	1
29	-5,19684	2,15256	1	126	-2,42521	1,00447	-1
30	5,19684	-2,15256	-1	127	2,51328	-0,49985	1
31	-5,45563	1,08515	1	128	-2,51328	0,49985	-1
32	5,45563	-1,08515	-1	129	2,5	0,00007	1
33	-5,5	-0,00004	1	130	-2,5	-0,00007	-1
34	5,5	0,00004	-1	131	2,39065	0,4756	1
35	-5,33301	-1,06085	1	132	-2,39065	-0,4756	-1
36	5,33301	1,06085	-1	133	2,19419	0,90894	1
37	-4,96584	-2,05696	1	134	-2,19419	-0,90894	-1
38	4,96584	2,05696	-1	135	1,92273	1,28482	1
39	-4,41716	-2,95151	1	136	-1,92273	-1,28482	-1
40	4,41716	2,95151	-1	137	1,59094	1,59104	1
41	-3,71228	-3,71234	1	138	-1,59094	-1,59104	-1
42	3,71228	3,71234	-1	139	1,21525	1,81888	1
43	-2,88198	-4,31328	1	140	-1,21525	-1,81888	-1
44	2,88198	4,31328	-1	141	0,81314	1,96327	1
45	-1,9612	-4,7349	1	142	-0,81314	-1,96327	-1
46	1,9612	4,7349	-1	143	0,40231	2,02288	1
47	-0,98759	-4,9652	1	14	-0,4023	-2,02288	-1

		4		4	1		
48	0,98759	4,96524	-1	145	-0,00007	2	1
49	0,00006	-5	1	146	0,00007	-2	-1
50	-0,00006	5	-1	147	-0,37805	1,90026	1
51	0,96331	-4,84262	1	148	0,37805	-1,90026	-1
52	-0,96331	4,84262	-1	149	-0,71759	1,73225	1
53	1,86564	-4,50389	1	150	0,71759	-1,73225	-1
54	-1,86564	4,50389	-1	151	-1,00702	1,507	1
55	2,67373	-4,00141	1	152	1,00702	-1,507	-1
56	-2,67373	4,00141	-1	153	-1,23748	1,23739	1
57	3,3588	-3,35871	1	154	1,23748	-1,23739	-1
58	-3,3588	3,35871	-1	155	-1,40314	0,93748	1
59	3,89755	-2,60418	1	156	1,40314	-0,93748	-1

60	-3,89755	2,60418	-1	157	-1,50133	0,62181	1
61	4,27297	-1,76985	1	158	1,50133	-0,62181	-1
62	-4,27297	1,76985	-1	159	-1,53249	0,30477	1
63	4,47485	-0,89004	1	160	1,53249	-0,30477	-1
64	-4,47485	0,89004	-1	161	-1,5	-0,00006	1
65	4,5	0,00007	1	162	1,5	0,00006	-1
66	-4,5	-0,00007	-1	163	-1,40987	-0,28049	1
67	4,35222	0,86578	1	164	1,40987	0,28049	-1
68	-4,35222	-0,86578	-1	165	-1,27031	-0,52624	1
69	4,04195	1,6743	1	166	1,27031	0,52624	-1
70	-4,04195	-1,6743	-1	167	-1,09128	-0,72923	1
71	3,58567	2,39595	1	168	1,09128	0,72923	-1
72	-3,58567	-2,39595	-1	169	-0,88385	-0,88392	1
73	3,00515	3,00525	1	170	0,88385	0,88392	-1
74	-3,00515	-3,00525	-1	171	-0,6597	-0,9874	1
75	2,32639	3,48182	1	172	0,6597	0,9874	-1
76	-2,32639	-3,48182	-1	173	-0,43048	-1,03938	1
77	1,5785	3,81103	1	174	0,43048	1,03938	-1
78	-1,5785	-3,81103	-1	175	-0,20724	-1,04209	1

79	0,79248	3,98445	1	176	0,20724	1,04209	-1
80	-0,79248	-3,98445	-1	177	0,00004	-1	1
81	-0,00007	4	1	178	-0,00004	1	-1
82	0,00007	-4	-1	179	0,18293	-0,91948	1
83	-0,76824	3,86183	1	180	-0,18293	0,91948	-1
84	0,76824	-3,86183	-1	181	0,33488	-0,80838	1
85	-1,48297	3,58	1	182	-0,33488	0,80838	-1
86	1,48297	-3,58	-1	183	0,45143	-0,67555	1
87	-2,11817	3,16994	1	184	-0,45143	0,67555	-1
88	2,11817	-3,16994	-1	185	0,53035	-0,53031	1
89	-2,6517	2,6516	1	186	-0,53035	0,53031	-1
90	2,6517	-2,6516	-1	187	0,57165	-0,38193	1
91	-3,06609	2,0486	1	188	-0,57165	0,38193	-1
92	3,06609	-2,0486	-1	189	0,57744	-0,23915	1
93	-3,34909	1,38716	1	190	-0,57744	0,23915	-1
94	3,34909	-1,38716	-1	191	0,5517	-0,10971	1
95	-3,49406	0,69493	1	192	-0,5517	0,10971	-1
96	3,49406	-0,69493	-1	193	0,5	0,00002	1
97	-3,5	-0,00008	1	194	-0,5	-0,00002	-1

Рассмотрим на примере решения задачи ирисов Фишера общий алгоритм подбора нейронной сети к ранее неизвестной задаче.

Не существует надёжных математических формул, с помощью которых можно однозначно подобрать оптимальную структуру ИНС к задаче (а те, что есть – для простых сетей и не всегда дают качественный результат). Поэтому желательно отталкиваться от уже готовых решений этой или похожей задач. Допустим, получить начальную структуру сети для задачи об ирисах можно с помощью команды nnstart, и выбора данной выборки из стандартных примеров Matlab (в разделе классификации). Но предположим, что нам не удалось получить никаких идей о том, какая должна быть сеть. Тогда единственный путь методом перебора: обучить некое количество сетей и из них выбрать сеть с оптимальной структурой.

Общий план работы будет такой:

1) Разбиваем искомую выборку на три подмножества: обучающее, тестовое, валидационное (тот случай, когда это подмножество будет полезным).

2) Перебираем 25^2 сетей общей структуры 4-i-j-3, где i, j = 1..25. Два скрытых слоя – это конечно излишество для данной задачи, но они приведены для демонстрации более полного перебора. В переборе используем для контроля от переобучения только валидационную выборку. Запоминаем сеть с наилучшими i, j (таких сетей может быть несколько, поэтому запоминаем ещё и с наименьшими i и j). «Наилучшесть» сети определяется по количеству распознанных паттернов на валидационной выборке: чем больше распознали, тем лучше.

3) Далее объединяем обучающее и валидационное множество в новое обучающее, и учим нашу выбранную сеть ($4\text{-}i_{const}\text{-}j_{const}\text{-}3$). В качестве контроля используем тестовое множество, которое теперь формально будет новым валидационным.

Как видно, тестовое множество не используется в процессе подбора структуры сети, это сделано для того, чтобы это множество не стало артефактом обучения, а осталось объективной мерой оценки качества обучения. Если же структура сети известна сразу или количество возможных структур, из которых происходит выбор, будет небольшим, то можно валидационную выборку не использовать, а сразу бить все данные на обучающую и тестовую выборки.

Листинг программы с комментариями приведён ниже. Программа составлена без использования какой-либо модульности (процедур, функций, библиотек-модулей).

```
% загрузка выборки
load fisheriris;
% рисуем паттерны классов от sepal length и sepal width
gscatter(meas(:, 1), meas(:, 2), species, 'rgb', 'osd');
xlabel('Sepal length');
ylabel('Sepal width');

% бьём искомую выборку на три части: тестовое, валидационное и
% обучающее множество
collndx = 1:4;
trainSeto = meas(1:35, collndx);
trainVers = meas(51:85, collndx);
trainVirg = meas(101:135, collndx);
valSeto = meas(36:43, collndx);
valVers = meas(86:93, collndx);
valVirg = meas(136:143, collndx);
```

```
testSeto = meas(44:50, colIndx);
testVers = meas(94:100, colIndx);
testVirg = meas(144:150, colIndx);
% формируем ответы учителя
a = [-1 -1 +1]';
b = [-1 +1 -1]';
c = [+1 -1 -1]';
% Заново объединяем все множества в одно, но паттерны уже
% располагаются в нужном нам порядке и оно транспонировано
initSet = [trainSeto' trainVers' trainVirg' valSeto' valVers' valVirg' testSeto' testVers' testVirg'];
T = [repmat(a, 1, length(trainSeto)) repmat(b, 1, length(trainVers)) repmat(c, 1, length(trainVirg)) repmat(a, 1, length(valSeto)) repmat(b, 1, length(valVers)) repmat(c, 1, length(valVirg)) repmat(a, 1, length(testSeto)) repmat(b, 1, length(testVers)) repmat(c, 1, length(testVirg))];
% оформляем ответы и транспонируем три подмножества
trainSet = [trainSeto' trainVers' trainVirg'];
trainT = [repmat(a, 1, length(trainSeto)) repmat(b, 1, length(trainVers)) repmat(c, 1, length(trainVirg))];
valSet = [valSeto' valVers' valVirg'];
valT = [repmat(a, 1, length(valSeto)) repmat(b, 1, length(valVers)) repmat(c, 1, length(valVirg))];
testSet = [testSeto' testVers' testVirg'];
testT = [repmat(a, 1, length(testSeto)) repmat(b, 1, length(testVers)) repmat(c, 1, length(testVirg))];
% инициализируем некоторые начальные переменные перед обучением
size = 25;
trainCor = zeros(size, size);
valCor = zeros(size, size);
MAX = 0;
max_i = 0;
max_j = 0;
% перебираем возможные структуры сетей и проводим обучение
for i = 1:size,
    for j = 1:size,
        net = newff(initSet, T, [i j], {'tansig' 'tansig' 'tansig'}, 'trainbfg');
        net = init(net);
        % бить будем через диапазоны индексов
        net.divideFcn = 'divideind';
        % для обучающего множества
        net.divideParam.trainInd = 1:105;
        % для валидационного
        net.divideParam.valInd = 106:129;
        % максимальное количество ошибок на валидационном
        % множестве, имеется ввиду, что ошибка обучения падает,
        % а ошибка на валидационном множестве растёт
        net.trainParam.max_fail = 2;
        % обучаем сеть
        [net, tr] = train(net, initSet, T);
        % получаем ответы сети на обучающей части
        Y = sim(net, trainSet);
        % получаем ответы сети на валидационной части
```

```
        Z = sim(net, valSet);
         % высчитываем процент правильных ответов
        col = 0;
        for k = 1:length(trainSet)
         % процент правильных ответов определяется через евклидово
         % расстояние между ответом учителя и ответом сети,
         % величина расхождения – не более 0.01
          if norm(trainT(k)-Y(k)) < 0.01
            col = col + 1;
          end
        end
        trainCor(i, j) = 100 * col / length(trainT);

        col = 0;
        for k = 1:length(valT)
          if norm(valT(k)-Z(k)) < 0.01
            col = col + 1;
          end
        end
        valCor(i, j) = 100 * col / length(valT);
         % определяем максимальное количество ответов, причём
         % сохраняем сеть с минимальной архитектурой (в условии
         % используем “>”, а не “>=”
        if valCor(i, j) > MAX
          MAX   =   valCor(i,
          j); max_i = i;
          max_j = j;
        end
    end
end

figure
% рисуем получившиеся матрицы
surf(1:size, 1:size, trainCor);
% в 3-D
view(3)
% в 2-D
view(2)
figure
surf(1:size, 1:size, valCor);
view(2);
% учим выбранную сеть до тех пор, пока количество правильных
% ответов на всех ирисах не станет больше 90%
Q = 10;
while (Q < 90)
        net = newff(initSet, T, [max_i max_j], {'tansig' 'tansig' 'tansig'}, 'trainbfg');
        net = init(net);
        net.divideFcn = 'divideind';
         % объединяем обучающее и валидационное множество
        net.divideParam.trainInd = 1:129;
         % новое валидационное множество теперь то, что было
         % зарезервировано под тестовое
```

```
        net.divideParam.valInd = 130:150;
        net.trainParam.max_fail = 3;
        [net, tr] = train(net, initSet, T);
        Z1 = sim(net, initSet);
        col = 0;
        for k = 1:length(T)
           if norm(T(k)-Z1(k)) < 0.01
                col = col + 1;
           end
        end
        Q = 100 * col / length(T);
en
d
```

Сначала выведем распределение классов, результат показан на рисунке 5.8.

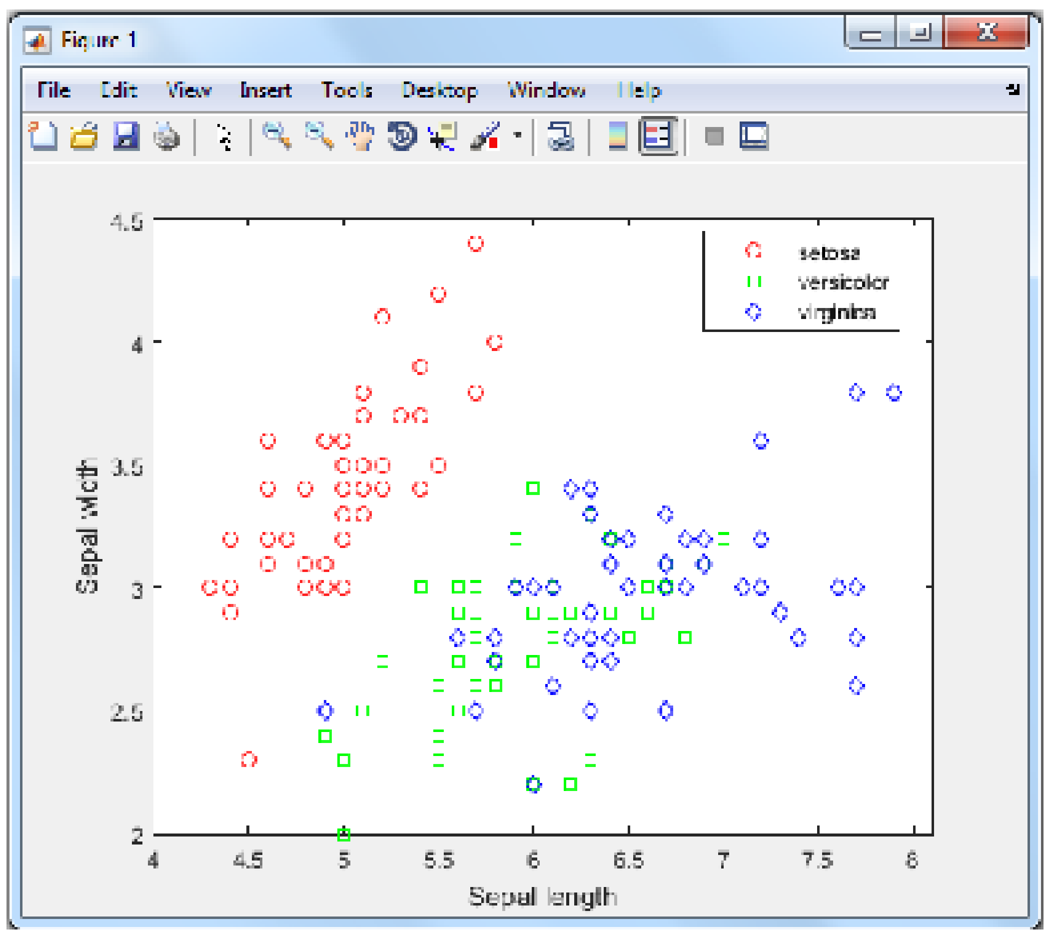

Рисунок 5.8 – Распределение паттернов классов в зависимости от двух параметров: sepal width и sepal length

Затем разобьём исходное множество на три выборки: валидационное, тестовое, обучающее. Каждый класс представлен 50 паттернами, 35 из них оставим для обучающей выборки, 8 – для валидационной и 7 – для тестовой.

Стоит отметить, что разбитие выборки на три множества – серьёзное дело. В учебной задаче можно обойтись простыми диапазонами не углубляясь в то, какие данные попадут в эти диапазоны, в реальных же задачах так бить нельзя. В одни диапазоны могут попасть крупные выбросы (нетипичные значения), в другие – усреднённые данные. В результате может получиться, что, допустим, валидационная и тестовая выборки будут составлены некорректно.

Когда мы загрузили выборку (**load fisheriris**), то массив species стал содержать названия ирисов, но ИНС работает только с числами, поэтому нам необходимо создать ответы учителя. Классов три, поэтому и выходов нейронов – три. Правильный нейрон кодируется +1, неправильный -1.

При создании сети с помощью **newff()** необходимо обозначить, что выходной нейрон будет обладать нелинейной функцией активации, иначе по умолчанию, среда создаст сеть с нейронами, имеющими на выходе функцию **purelin**, рисунок 5.9.

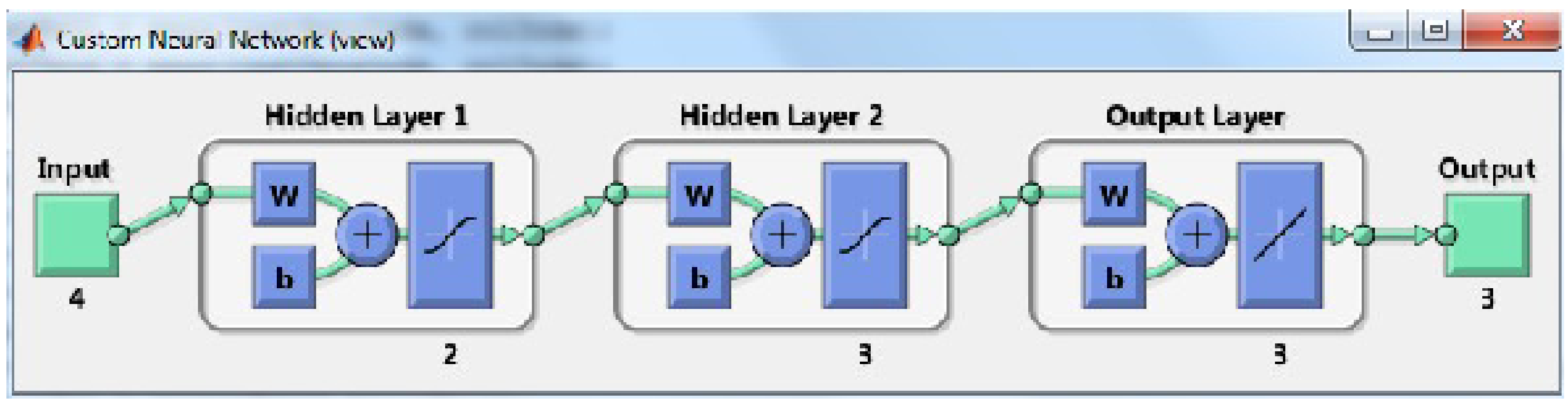

Рисунок 5.9 – Сеть, которая создастся по умолчанию, если в **newff()** прямо не указать, что нужны функции **tansig**

Процент правильно распознанных паттернов из обучающей выборки приведён на рисунке 5.10 (массив trainCor).

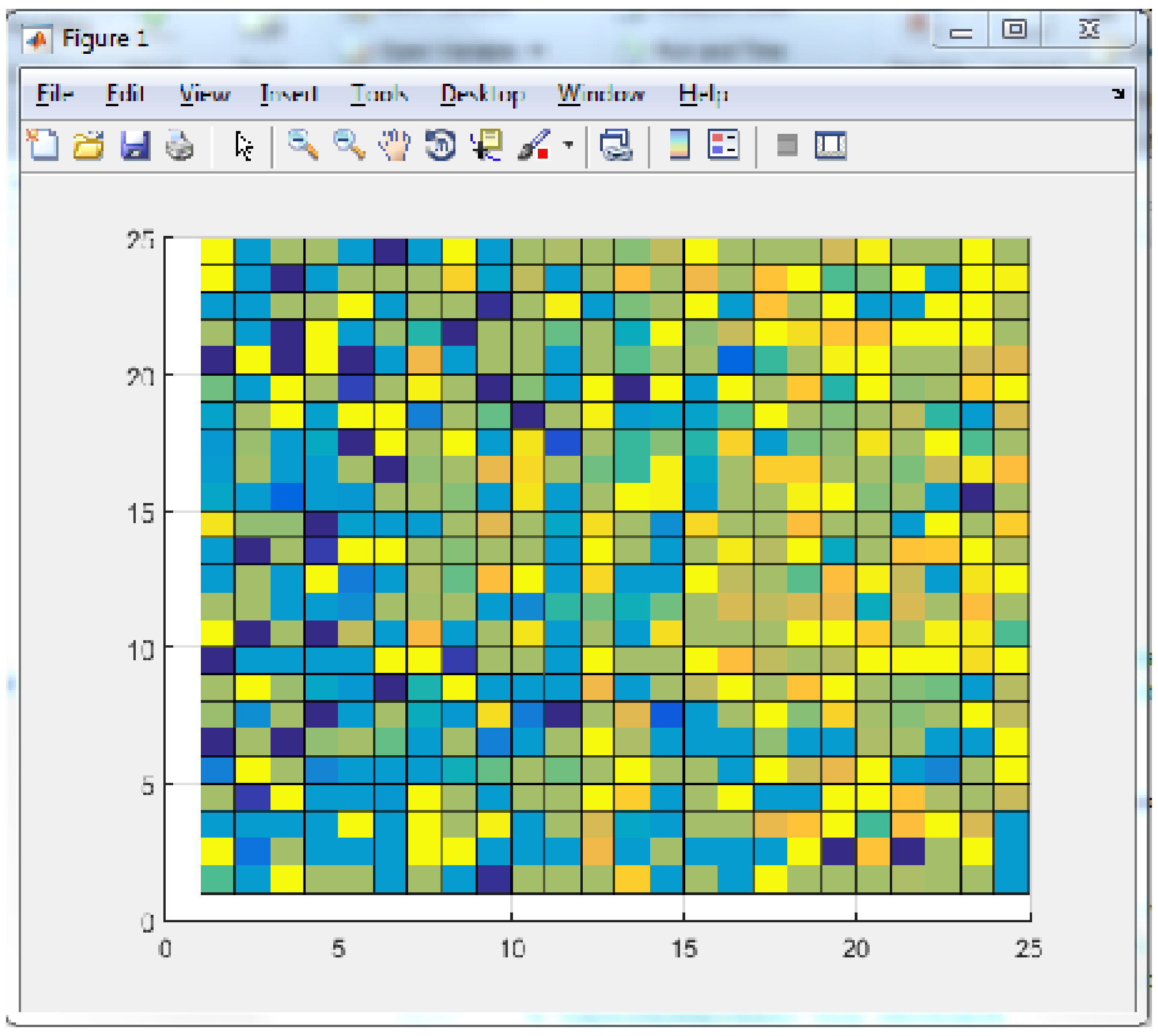

Рисунок 5.10 – Массив trainCor, желтые цвета соответствуют высокому проценту распознанных паттернов, синие - низкому

На рисунке 5.11 приведён массив valCor, именно по нему определяется первый максимально правильный ответ. В данном случае max = 100, max_i = 1, max_j = 3.

Рисунок 5.11 – Массив valCor

Можно наблюдать высокую корреляцию между двумя массивами, что говорит о хорошем разбиении искомого множества на валидационную и обучающую выборки. Также видно, что правый верхний угол матрицы значительно желтее, чем нижний левый, что говорит о том, что с ростом скрытых слоёв качество решения задачи улучшается.

Пример обучения сети со структурой 4-25-25-3 приведён на рисунке 5.12.

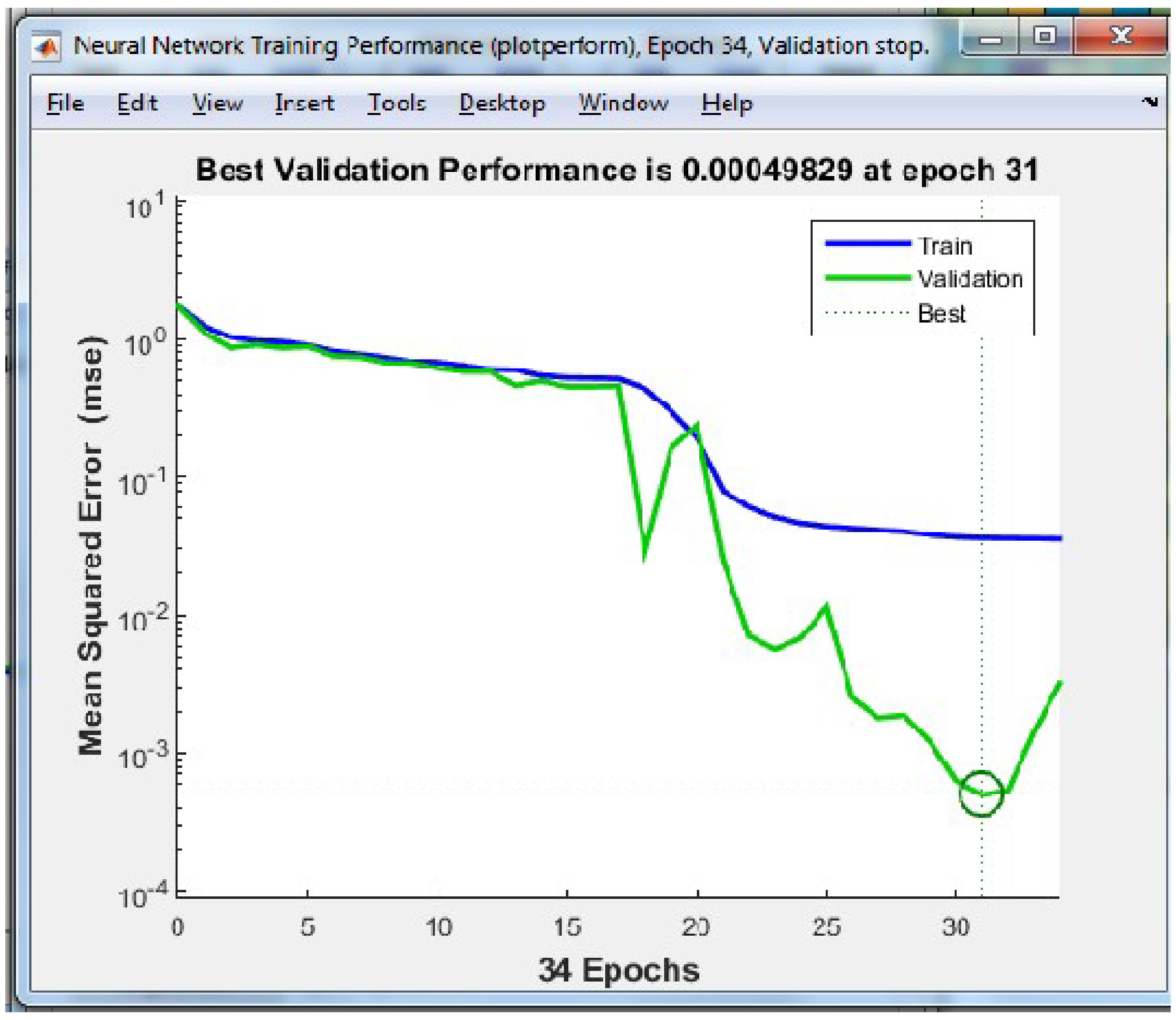

Рисунок 5.12 – Видно, что на 31 эпохе на валидационной выборки был достигнут лучший результат, потом произошёл рост ошибки, а так как значение max_fail установлено в 2, то очень быстро обучение было прервано, чтобы сеть не переобучилась и не превратилась в память

После выбора нужной структуры (4-1-3-3) необходимо обучить данную сеть. В коде использовался критерий оценки количества правильных результатов на всей