МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

НЕЙРОКОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

Методические указания
к лабораторным работам по дисциплине
для студентов дневной и заочной форм обучения направлений
09.04.02 «Информационные системы и технологии»,
09.04.03 «Прикладная информатика»

Севастополь
СевГУ
2024

УДК 004.032.26 (076.5)
ББК 32.813я73
Н463

Р е ц е н з е н т ы :
В. С. Чернега – канд. техн. наук, доцент кафедры «Информационные системы»
А. А. Брюховецкий – канд. техн. наук, доцент кафедры
«Информационные технологии и компьютерные системы»

Ответственный за выпуск:
И. П. Шумейко – канд. физ.-мат. наук, доцент,
заведующий кафедрой «Информационные системы»

Составитель: В. Н. Бондарев

Н463 **Нейрокомпьютерные технологии** : методические указания к лабораторным работам по дисциплине для студентов дневной и заочной форм обучения направлений 09.04.02 «Информационные системы и технологии», 09.04.03 «Прикладная информатика» / Севастопольский государственный университет ; сост. В. Н. Бондарев. – Севастополь: СевГУ, 2024. – 120 с. – Текст : электронный.

Целью методических указаний является оказание помощи студентам в изучении процессов моделирования и обучения нейронных сетей, системы команд среды моделирования Scilab и пакета NeuralNetwork 3.0. Излагаются теоретические и практические сведения, необходимые для выполнения лабораторных работ, содержатся варианты заданий, программа исследований и требования к содержанию отчета.

УДК 004.032.26 (076.5)
ББК 32.813я73

Методические указания рассмотрены и утверждены на заседании кафедры «Информационные системы», протокол № 8 от 16 апреля 2024 г.

Методические указания рассмотрены и рекомендованы к изданию на заседании Учёного совета Института информационных технологий, протокол № 05 от 18 апреля 2024 г.

© Бондарев В. Н., сост., 2024
© ФГАОУ ВО «Севастопольский
государственный университет», 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ…… 4
Лабораторная работа № 1. ИССЛЕДОВАНИЕ ФУНКЦИЙ ПАКЕТА SCILAB ДЛЯ ОБРАБОТКИ И ВИЗУАЛИЗАЦИИ ДАННЫХ…… 5
Лабораторная работа № 2. ИССЛЕДОВАНИЕ АКТИВАЦИОННЫХ ФУНКЦИЙ НЕЙРОННЫХ ЭЛЕМЕНТОВ…….. …… 22
Лабораторная работа № 3. ИССЛЕДОВАНИЕ ОДНОСЛОЙНОГО ПЕРСЕПТРОНА…….. 33
Лабораторная работа № 4. ИССЛЕДОВАНИЕ АДАПТИВНОГО ЛИНЕЙНОГО ЭЛЕМЕНТА…… 45
Лабораторная работа № 5. ИССЛЕДОВАНИЕ МНОГОСЛОЙНОГО ПЕРСЕПТРОНА: АЛГОРИТМЫ ОБРАТНОГО РАСПРОСТРАНЕНИЯ С АДАПТИВНОЙ СКОРОСТЬЮ ОБУЧЕНИЯ И МОМЕНТОМ …… 67
Лабораторная работа № 6. ИССЛЕДОВАНИЕ МНОГОСЛОЙНОГО ПЕРСЕПТРОНА: АЛГОРИТМ ЛЕВЕНБЕРГА-МАРКВАРДТА…… 89
Лабораторная работа № 7. ИССЛЕДОВАНИЕ СОСТЯЗАТЕЛЬНЫХ СЕТЕЙ И СЕТЕЙ ВЕКТОРНОГО КВАНТОВАНИЯ…..…… 103
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ…….. 120

ВВЕДЕНИЕ

Дисциплина «Нейрокомпьютерные технологии» является дисциплиной по выбору, формируемой участниками образовательных отношений.

Целью преподавания дисциплины является формирование профессиональных компетенций в области проектирования базовых архитектур нейронных сетей и правил их обучения.

Основными задачами изучения дисциплины являются:

- обучение студентов основным понятиям нейронных сетей;
- обучение основным теоретическим принципам и алгоритмам поиска оптимальных значений параметров нейросетей;
- формирование компетенций в области машинного обучения базовых архитектур нейронных сетей: однослойного и многослойного персептронов, линейного адаптивного элемента, состязательных сетей, самоорганизующихся карт Кохонена, сетей векторного квантования, сетей на основе радиальных базисных функций;
- формирование практических умений по использованию инструментальных средств разработки нейронных сетей.

Лабораторный практикум по дисциплине включает семь лабораторных работ, для выполнения которых отводится 36 академических часов.

Лабораторные работы выполняются в среде численного моделирования Scilab с использованием плагина NeuralNetworks 3.0

ЛАБОРАТОРНАЯ РАБОТА № 1

ИССЛЕДОВАНИЕ ФУНКЦИЙ ПАКЕТА SCILAB ДЛЯ ОБРАБОТКИ И ВИЗУАЛИЗАЦИИ ДАННЫХ

1. Цель работы

Изучение среды численного моделирования Scilab и ее базовых функций, приобретение практических навыков моделирования в среде Scilab.

2. Основные теоретические положения

Scilab – это кроссплатформенный свободно распространяемый математический пакет, обладающий сходным с Matlab синтаксисом встроенного языка. Пакет программ Scilab предназначен для выполнения математических вычислений и численного моделирования широкого спектра информационных и управляющих систем.

Пакет можно скачать по адресу https://www.scilab.org/. После установки пакета Scilab для получения справочных сведений о его возможностях и командах необходимо выбрать пункт меню «Справка» или нажать кнопку F1. Откроется окно «Справочная система» (рисунок 2.1), в котором можно выполнять поиск необходимых сведений как с использованием разделов содержания справки, так и по ключевым словам.

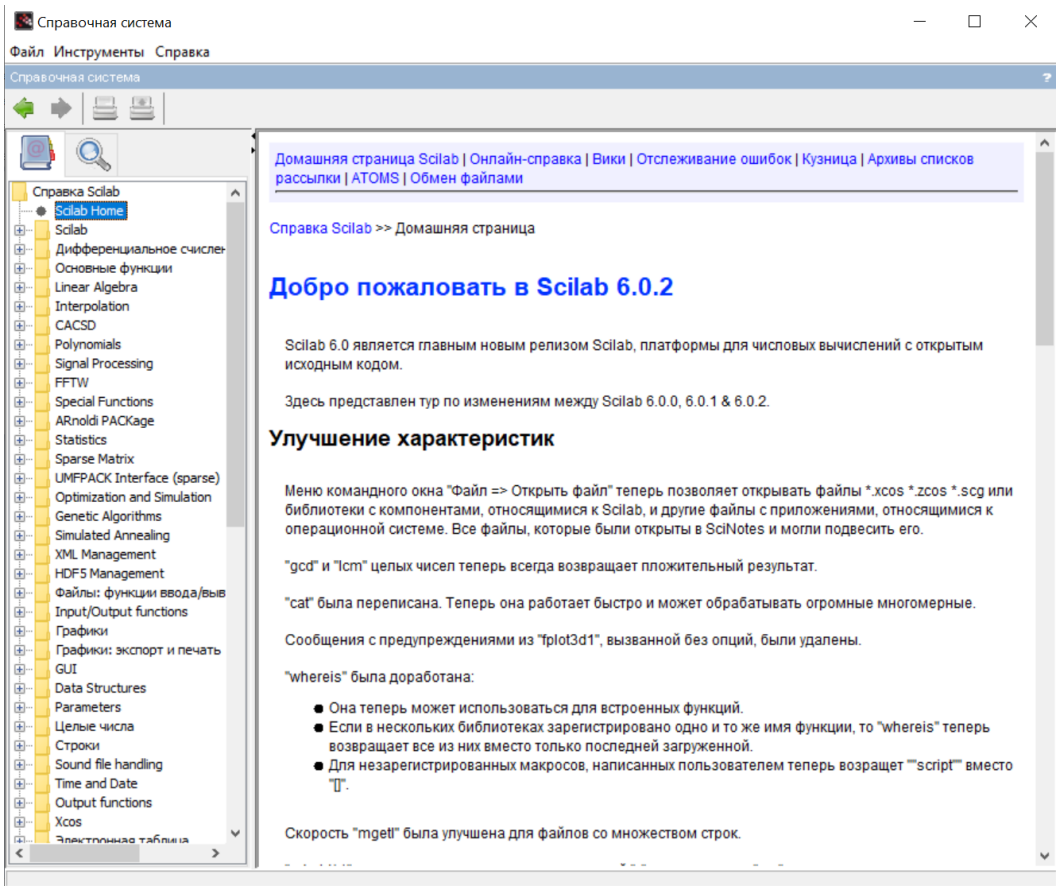


Рисунок 2.1 – Справочная система Scilab

Пакет Scilab позволяет выполнять команды в интерактивном режиме. Порядок ввода команд аналогичен пакету Матлаб. При этом используются следующие специфические обозначения:

// - начало однострочного комментария
% - начальный символ имени **системных переменных**, например:
%i _мнимая единица (√−1);
%pi _число π = 3.141592653589793;
%e _число e = 2.7182818;
%inf _машинный символ бесконечности (∞);
%NaN _неопределенный результат (0/0,∞/∞ и т. п.);
%eps _условный ноль %eps=2.220E-16;
%t, %f – логические константы True и False.

Системные переменные используются в качестве констант в выражениях и не могут быть изменены пользователем. Ниже представлен пример ввода выражения, содержащего системные переменные и встроенные функции, и вычисления его значения в командном окне Scilab:

```
--> y= sin(%pi/4)+%e^2
 y  =
   8.0961629
```

Scilab имеет расширенный перечень встроенных основных функций (см. п. меню «Справка»—> Основные функции). Клавиша Tab может использоваться для автозавершения ввода имени функции или переменной (рисунок 2.2).

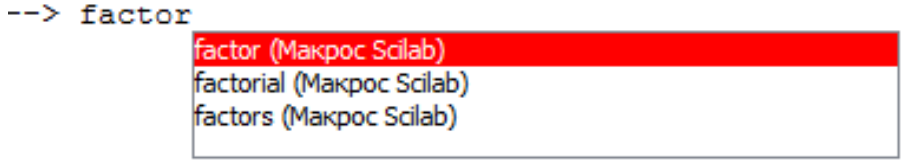


Рисунок 2.2 – Пример автозавершения имени функции

Ввод в программу исходных данных с клавиатуры выполняется вызовом функции x = input(message [, "string"]), где message – строка, которая отображается на экране перед выполнением ввода, необязательный параметр "string" указывается при вводе строки, x – вещественное число или строка (если указан параметр "string").

Для вывода на экран результатов вычислений может использоваться функция disp. Если требуется вывести комбинацию строковых значений и чисел, то требуется предварительно преобразовать числа в строки с помощью функции string:

```
--> t=19
```

```
--> disp("Средняя температура "+string(t)+" градусов")
 Средняя температура 19 градусов
```

Для форматированного вывода на экран значений переменных может использоваться функция, аналогичная функции printf языка Си. Например:

```
--> printf("Результат: exp(3) = %f",exp(3))
Результат: exp(3) = 20.085537
```

Ввод-вывод данных из файлов и в файлы выполняется в Scilab с использованием функций mopen, mfscanf, mfprintf, которые полностью аналогичны функциям языка Си: fopen, fscanf, fprintf.

При программировании в Scilab необходимо различать понятия **сценарий** и **функция, определяемая пользователем.**

Сценарием называют любую программу Scilab. Любой сценарий состоит из последовательности **инструкций** (команд), которые описывают конкретные действия с объектами Scilab. Сценарии создаются в окне текстового редактора SciNotes, который может быть вызван из пункта меню «Инструменты». Сценарии сохраняются в файлах с расширением *.sce. Для исполнения сценария используется команда exec, которую вводят в комендном окне Scilab. Пример вызова команды:

```
exec('C:\Users\HP\Documents\Scilab\lab3_2.sce', 2)
```

Здесь первый аргумент содержит путь доступа к файлу со сценарием, а цифра 2 – обозначает режим вызова, который блокирует в ходе исполнения отображение в командном окне команд сценария. По умолчанию при выполнении сценария из окна редактора SciNotes используется режим -1, который отображает команды сценария, но блокирует вывод значений вычисляемых переменных.

Функция, определяемая пользователем, также является сценарием, но характеризуется наличием имени, благодаря которому к ней можно обратиться из любого места сценария. Имеется два способа определения функций: function и deff. Определение функции с использованием конструкции function аналогично большинству языков программирования. Например:

```
function z=f(x,y)
        z=1/(exp(-y)+exp(-x))
endfunction
```

Обычно такое определение функции сохраняется в файле *.sce на диске (например, f.sce). Чтобы использовать функцию, необходимо загрузить её из файла в оперативную память. Для этого необходимо выполнить команду exec. Её аргументом является строка с указанием положения сохраненного файла в файловой системе. Например, если файл f.sce был сохранен в домашнем каталоге пользователя, то команда загрузки будет выглядеть следующим образом:

exec('~/f.sce'). После этого функция, описанная в файле, может быть вызвана в текущей сессии, например f(1,2).

Определение функции с использованием оператора deff реализуется в соответствии с шаблоном:

```
deff('[имя1,...,имяN]=имя_функции(переменная_1,...,переменная_M)',
                    'имя1=выражение1;...;имяN=выражениеN'),
```

где список [имя1,...,имяN] представляет имена выходных значений функции, выражение1,…, выражениеN – правила вычисления этих значений. Листинг ниже демонстрирует определение функции z=f(x,y) с помощью оператора deff и использование вызова f для вычисления значения функции:

```
--> deff('z=f(x,y)','z=1/(exp(-y)+exp(-x))');
--> a=1; b=2; c=f(a,b)
 c  =
   1.9872232
```

Scilab хранит установки, функции и переменные в рабочей области, называемой Workspace. Переменные рабочей области отображаются в окне *обозревателя переменных*. Для очистки рабочей области от всех пользовательских переменных используется команда clear. Если требуется удалить одну переменную, то применяется команда clear <имя переменной>.

Система Scilab предназначена для выполнения математических вычислений. Обозначение математических операций и управляющих конструкций (if, for, while и др.) такое же, как и в системе Matlab. Рассмотрим примеры некоторых операций с одномерными (векторы) и двумерными массивами (матрицы).

Вектор-строка задается своими элементами, которые записываются внутри квадратных скобок и разделяются пробелами или запятыми. Для задания вектора-столбца его элементы следует разделять точкой с запятой (;). Например:

```
--> xvec=[1 2 3] // вектор-строка
 xvec  =
   1.   2.   3.
--> yvec=[1;2;3] // вектор-столбец
 yvec  =
   1.
   2.
   3.
```

Векторы удобно создавать с помощью оператора “ : ”. Например, если требуется создать вектор из нечетных чисел от 1до 10, то можно применить оператор:

```
--> x=1:2:10  // Хнач:Шаг:Хкон
 x  =
```

```
 1.   3.   5.   7.   9.
```

Переменную заданную как массив можно использовать в арифметических выражениях в качестве аргумента стандартных математических функций. Результатом выполнения таких операторов являются массивы.

Для определения длины вектора используется функция length. Для получения i-го компонента вектора применяется команда x(i), а чтобы получить компоненты вектора со 2-го по 4-й, используется инструкция

```
--> x(2:4)
 ans  =
   3.   5.   7.
```

Матрица представляется в виде набора векторов-строк, разделяемых точкой с запятой:

```
--> m=[1 2 3;4 5 6]
 m  =
   1.   2.   3.
   4.   5.   6.
```

Размер матрицы определяется с помощью функции size

```
--> size(m)
 ans  =
   2.   3.
```

Для извлечения строки или столбца матрицы используют оператор “:”, например:

```
--> m(2,:)      #возвращает 2-ю строку матрицы
 ans  =
   4.   5.   6.
--> m(:,3)     #возвращает 3-й столбец матрицы
ans  =
   3.
   6.
```

Матрицы и векторы можно формировать, составляя их из ранее заданных матриц и векторов:

```
--> v1=[1 2 3]; v2=[4 5 6]; v3=[7 8 9];
--> //Горизонтальная конкатенация векторов–строк:
--> V=[v1 v2 v3]
  V = 1 2 3 4 5 6 7 8 9
-->//Вертикальная конкатенация векторов–строк,
-->//результат матрица:
--> V=[v1; v2; v3]
  V =
```

```
 1 2 3
 4 5 6
 7 8 9
-->//Горизонтальная конкатенация матриц:
--> M=[V V V]
 M =
 1 2 3 1 2 3 1 2 3
 4 5 6 4 5 6 4 5 6
 7 8 9 7 8 9 7 8 9
-->//Вертикальная конкатенация матриц:
--> M=[V;V]
 M =
 1 2 3
 4 5 6
 7 8 9
 1 2 3
 4 5 6
 7 8 9
```

Для работы с матрицами в Scilab применяются матричные операции: “+” — сложение; “–“ — вычитание; “ ‘ “— транспонирование; “ * “— матричное умножение; “^”— возведение в степень; “\”— левое деление; “/ “— правое деление; “.* ” — поэлементное умножение матриц; “.^”—_ поэлементное возведение в степень; “.\”— поэлементное левое деление; “./”— поэлементное правое деление. Эти операции аналогичны операциям Matlab.

Примеры действий над матрицами:

```
-->A=[1 2 0;-1 3 1;4 -2 5];
-->B=[-1 0 1;2 1 1;3 -1 -1];
-->//Вычислить (A^T+B)^2 - 2A(0.5B^T-A)
-->(A'+B)^2-2*A*(1/2*B'-A)
ans =
 10. 8. 24.
 11. 20. 35.
 63. - 30. 68.
--> //Решить матричные уравнения A·X=B и X·A=B.
-->A=[3 2;4 3];
-->B=[-1 7;3 5];
-->//Решение матричного уравнения AX=B: решение X=A^-1 B
-->X=A\B
X =
 - 9. 11.
 13. - 13.
-->//Решение матричного уравнения XA=B: решение X=B A^-1
-->X=B/A
X =
 - 31. 23.
 - 11. 9.
-->X*A-B // проверка
ans =
  0. 0.
```

0. 0.

Для работы с матрицами и векторами (массивами) в Scilab существуют специальные функции: ones(m,n) – создает матрицу из единиц; zeros(m,n) – создает матрицу из нулей; eye(m,n) - создает единичную матрицу. Функции для вычисления различных числовых характеристик массивов:

sum – сумма элементов массива;

prod – произведение элементов массива;

max, min – максимальное и минимальное значение массива;

mean, median – среднее и медианное значение массива;

det – определитель квадратной матрицы;

rank – ранг матрицы;

norm – норма квадратной матрицы;

cond – число обусловленности матрицы (произведение нормы исходной матрицы на норму обратной матрицы);

spec – возвращает *собственные значения* и *собственные векторы* квадратной матрицы;

inv – вычисляет обратную матрицу;

pinv – вычисляет псевдообратную матрицу;

linsolv – решает систему линейных алгебраических уравнений;

svd – выполняет сингулярное разложение матрицы и др.

Построение двумерного графика в отдельном графическом окне в координатах x,y осуществляется по команде plot(x,y). Команда позволяет также задавать цвет и стиль изображения точек на графике, например команда plot(1,2,"*r") отобразит звездочку красным цветом в позиции с координатами (1,2). Для отображения графика функции вида *y=f(x)* удобно значения *x* задавать в виде вектора, который формируется с помощью вызова функции x=linspace(a,b,n) , где параметры a и b задают интервал определения функции, а n – число точек разбиения этого интервала. На рисунке 2.3 приведен пример программы, определяющей функции *f(x)* и *g(x)* и отображающей их графики в одной системе координат, соответственно, красным и зеленым цветом с помощью вызова функции plot:

```
function y=f(x)
        y=(x^2+2*x)*exp(-x/2)
endfunction
function y=g(x)
        y=cos(2*x)*exp(-x/2)
endfunction
x=linspace(-2,10,100);
plot(x,f,"r",x,g,"g")
xgrid //отображение сетки
// отображение надписей
xtitle('Функции f(x) и g(x)',
        'Позиция','Ускорение')
//легенда: 4 – правый нижний угол, %t - рамка
legend('f(x)','g(x)',4,%t)
```

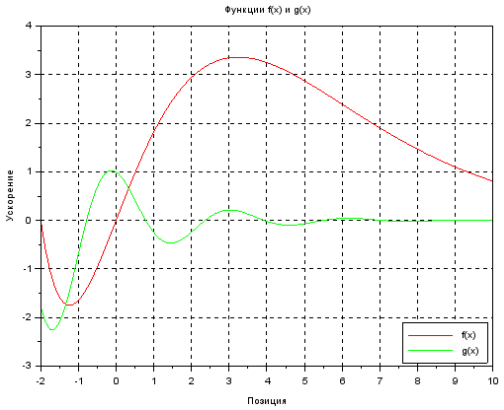


Рисунок 2.3 – Построение двумерных графиков с помощью plot

Функция subplot(m,n,p) разбивает графическое окно на m окон по вертикали и n окон по горизонтали, текущим окном становится окно с номером p. Применяется для отображения нескольких графиков – каждый в своем подокне.

Команда figure(i) делает активным i-ое окно. Для очистки текущего графического окна применяется функция clf. Функция xdel(i) закрывает графические окна, где i – вектор номеров окон.

Scilab позволяет отображать поверхности и кривые в 3-х мерном пространстве с помощью функции surf. Эта функция имеет три параметра: x, y и z. Векторы x и y, размерами m и n, определяют координатные точки по осям *Ox* и *Oy*. Матрица z, размером nxm, хранит высоты в координатных узлах. Для отображения поверхности, определяемой функцией z=f(x,y), необходимо предварительно определить z во всех координатных узлах с помощью вызова функции feval(x,y,f). Функция feval возвращает транспонированную матрицу, размером mxn. Поэтому при вызове функции surf матрицу z требуется транспонировать. Удобнее 3-d поверхности строить с использованием функций plot3d и plot3d1. Сравнительный пример построения 3-D поверхности с помощью указанных функций представлен на рисунке 2.4.

```
function z=f(x, y)
   z=3*x^2+y^2;
endfunction
x=linspace(-1,1,50);
y=linspace(-2,2,100);
z=feval(x,y,f);
clf
figure(1)
subplot(1,3,1)
surf(x,y,z')
subplot(1,3,2)
plot3d(x,y,z)
subplot(1,3,3)
plot3d1(x,y,z)
```

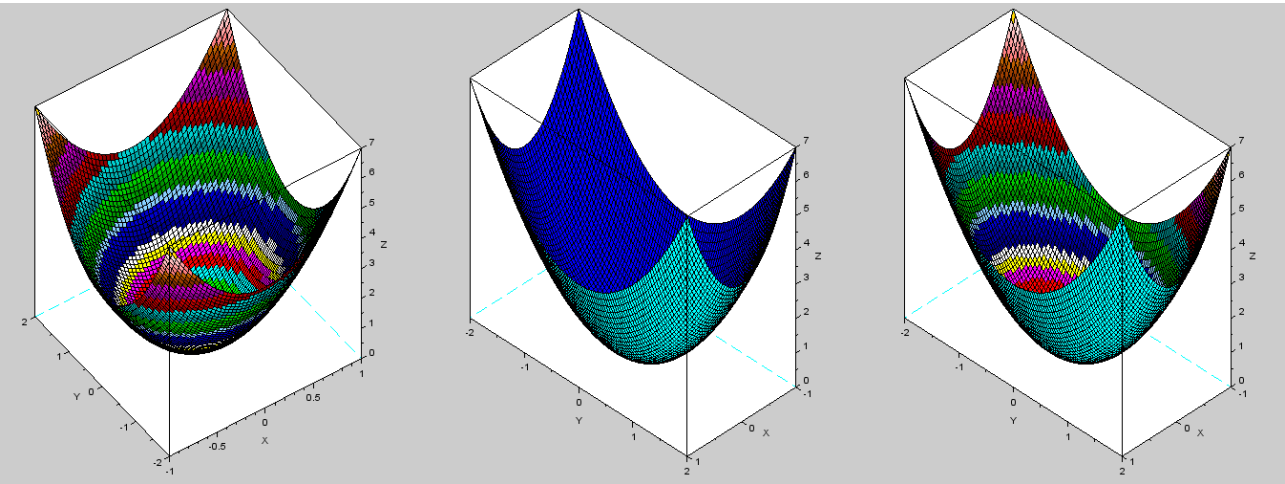


Рисунок 2.4 – 3-D поверхности, построенные функциями surf, plot3d, plot3d1

В Scilab имеются и другие функции для построения 2-х и 3-х мерных графиков, например: plot2d, mesh, contour, plot3d2, plot3d3, param3d и др.

Scilab позволяет выполнять статистическую обработку данных в массивах. Функция mean(M) вычисляет среднее всех элементов матрицы M, mean(M,'r') – вычисляет среднее по столбцам, а mean(M,'c')- среднее по строкам. Функция stdev(M) вычисляет стандартное отклонение, а функция variance(M) – дисперсию всех элементов массива M. Специальная функция tabul(M) вычисляет частоты появления значений в массиве M. Для генерации псевдослучайных чисел могут использоваться функции grand и rand. Функция rand(n1,n2,…nn[, p]) - формирует многомерную матрицу *случайных чисел*, необязательный параметр p - это символьная переменная, с помощью которой можно задать тип распределения случайной величины ('uniform' - равномерное , 'normal'- гауссовское). На рисунке 2.5 изображен листинг программы для генерации матрицы размером 10x10 из нормально распределенных случайных чисел с нулевым средним и единичной дисперсией, а также построена гистограмма распределения.

```
--> x=rand(10,10,'normal');
--> mean(x)
 ans  =
  -0.0218062
--> variance(x)
 ans  =
   1.1118125
--> histplot(12,x);
```

.

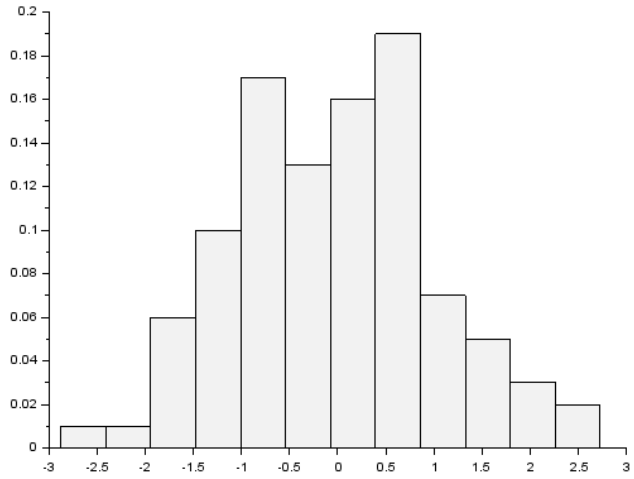


Рисунок 2.5 – Пример использования функций rand и histplot

В функции grand доступны улучшенные генераторы случайных чисел в том смысле, что они имеют более длинный период и более лучшие статистические свойства. Кроме этого, функция grand обладает большими функциональными возможностями, в частности, позволяет генерировать случайные числа с различными законами распределения. Ниже приведены некоторые примеры вызова функции grand:

```
x=grand(400,500,'nor',0,1); //нормальный распределение
x=grand(1,1000,'unf',-1,1); // равномерное распред., числа в диапазоне [-1,1]
x=grand(1,1000,'exp',3);     // экспоненциальное распределение, лямбда=3
x=grand(1,5000,'chi',10);    // распред. хи квадрат с 10-ю степенями свободы
x=grand(1,5000,'poi',2);     // распределение Пуассона с мат. ожиданием 2
```

3 Описание лабораторной установки

В качестве лабораторной установки используется персональный компьютер с инсталлированной системой Scilab и модулем моделирования нейронных сетей Neuralnetworks 3.0.

Скачайте пакет Scilab с веб-страницы по ссылке: Previous versions | Scilab (https://www.scilab.org/download/previous-versions) и выполните его стандартную установку (для корректного взаимодействия с модулем Neuralnetworks 3.0. рекомендуется установить Scilab 6.0.2)

При запуске Scilab появляется главное окно, показанное на рисунке 3.1.

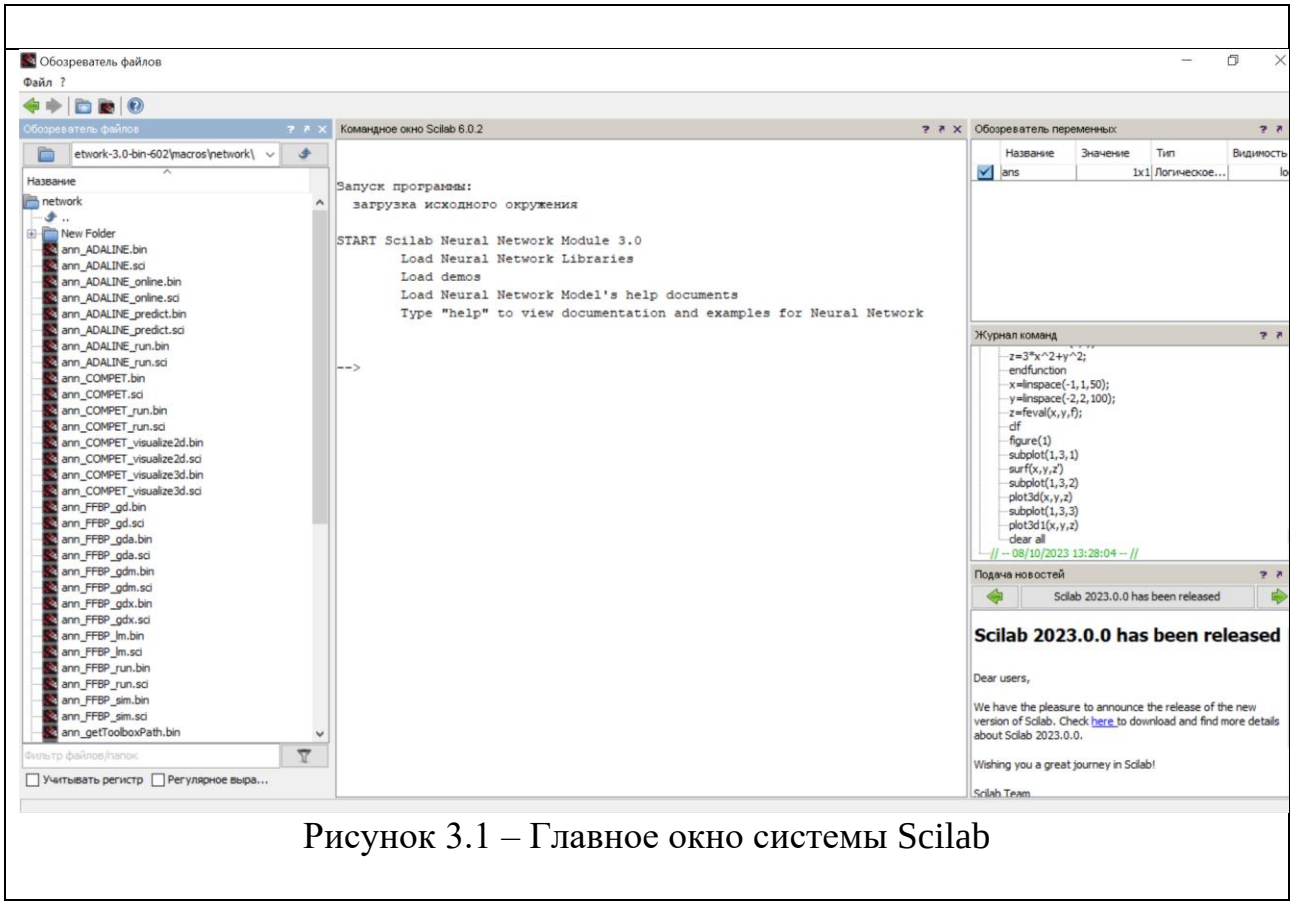


Рисунок 3.1 – Главное окно системы Scilab

Оно содержит окно *обозревателя файлов*, окно *обозревателя переменных*, *журнал команд*, *окно новостей* и *командное окно*. При выборе одного из окон его рамка окрашивается голубым цветом. Управление Scilab осуществляется через систему меню и командное окно. Появление стрелки вида --> (*prompt*) свидетельствует о готовности Scilab воспринимать команду.

Для установки модуля моделирования нейронных сетей необходимо скачать архив **neuralnetwork-3.0-bin-602.zip** с Neural Network Module 3.0., расположенный по адресу: https://atoms.scilab.org/toolboxes/neuralnetwork/3.0

Разархивируйте zip файл в папку **neuralnetwork-3.0-bin-602** на вашем компьютере и запустите пакет Scilab. В его командном окне выполните установку модуля Neuralnetwork-3.0, набрав команду

exec("*путь к папке*\neuralnetwork-3.0-bin-602\loader.sce"),

где в строке "*путь к папке*" необходимо указать конкретный путь к папке **neuralnetwork-3.0-bin-602** на вашем компьютере. Проверьте установку модуля Neuralnetwork-3.0 , набрав в командном окне Scilab команду help. Откроется окно справки, в котором должен появиться раздел NeuralNet**.**

4 Варианты заданий и программа работы

4.1 Ознакомиться с основными командами и функциями Scilab, а также правилами построения сценариев [2].

4.2 Выбрать вариант задания, взяв остаток от деления номера студента в списке группы на 15. Выполнить задания 4.3-4.6.

4.3 Определить функцию *f*(*x*) и вычислить N её значений на заданном отрезке. Вывести значения аргумента и значения функции. Построить график функции *f*(*x*) с необходимыми надписями и координатной сеткой.

1. Функция $f(x)=\frac{\sin x\cos x}{x+\cos x}$, N=15, на отрезке [0, 2π].

2. Функция $f(x)=\ln(2x-1)\sqrt{e^{-x}+4e^{x}}$, N=10, на отрезке [0.7, 4].

3. Функция $f(x)=(\sin x-1)\sqrt{e^{-\cos x}+tg\ x\ e^{x}}$, N=20, на отрезке [0.05, 1].

4. Функция $f(x)=x\sin x+\frac{e^{-x}-e^{x}}{e^{-x}+e^{x}}$, N=30, на отрезке [0, 1].

5. Функция $f(x)=x\ (\sin x+\cos x)\frac{x-ctg\ 2x}{1+sin^{2}\ x}$, N=10, на отрезке [0, π/2].

6. Функция $f(x)=\frac{x}{1+\sqrt[3]{x+1}}$., N=30, на отрезке [10,100].

7. Функция $f(x)=\frac{sh\ x+ch\ x}{th\ x+5}$, N=20, на отрезке [-3,3].

8. Функция $f(x)=x^{3}\,|\sin x+\cos x\,|+x-tg\ 2x$, N=10, на отрезке [0, π/2].

9. Функция $f(x)=\log_{2}(\sin x+1)\sqrt{e^{-\cos x}}$, N=30, на отрезке [0, π/2].

10. Функция $f(x)=\frac{\sqrt[3]{x^{2}+1}}{\sqrt{|x|+0.5}}\frac{x}{1+sin^{2}\ x}$, N=10, на отрезке [-π/2, π/2].

11. Функция $f(x)=\frac{sin^{2}(2x+\frac{\pi}{2})}{\sqrt{\frac{|x|+1}{4}}}+\frac{x\ln x}{1+x^{2}}$, N=20, на отрезке [-π/2, π/2].

12. Функция $f(x)=\frac{\sqrt{(3x+1)^{3}}}{x+2}+\sin^{2}(\frac{2x-1}{x+1})$, N=15, на отрезке [0, 1].

13. Функция $f(x)=\sqrt{\log_{2}(\sin x+1)}+\sqrt{1+x^{3.5}}$, N=30, на отрезке [0, π/2].

14. Функция $f(x)=4cos^{2}\left(\frac{x+\frac{\pi}{3}}{x+2}\right)-\ln(\frac{e^{-x}-e^{x}}{e^{-x}+e^{x}})$, N=20, на отрезке [0, 1].

15. Функция $f(x)=\ln\left(\sqrt{\frac{2x-1}{x+1}}\right)+\sin(\sqrt{2+x})$, N=10, на отрезке [0.7, 4].

4.4 Если возможно, то вычислить матрицу обратную D

1. $D = 2(A^2 + B)(2B - A)$, где

$$A = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 5 & 2 \\ -1 & 0 & 7 \end{pmatrix}, \qquad B = \begin{pmatrix} -1 & 0 & 5 \\ 0 & 1 & 3 \\ 2 & -2 & 4 \end{pmatrix}$$

2. $D = 3A - (A + 2B)B^2$, где

$$A = \begin{pmatrix} 4 & 5 & -2 \\ 3 & -1 & 0 \\ 4 & 2 & 7 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 1 & -1 \\ 0 & 1 & 3 \\ 5 & 7 & 3 \end{pmatrix}$$

3. $D = 3A^2 - (A + 2B)B$, где

$$A = \begin{pmatrix} 4 & 5 & -2 \\ 3 & -1 & 0 \\ 4 & 2 & 7 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 1 & -1 \\ 0 & 1 & 3 \\ 5 & 7 & 3 \end{pmatrix}$$

4. $D = (A - B^2 2)(2A + B^3)$, где

$$A = \begin{pmatrix} 5 & 2 & 0 \\ 10 & 4 & 1 \\ 7 & 3 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 3 & 6 & -1 \\ -1 & -2 & 0 \\ 2 & 1 & 3 \end{pmatrix}$$

5. $D = 2(A - B)(A^2 + B)$, где

$$A = \begin{pmatrix} 5 & 1 & 7 \\ -10 & -2 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 4 & 1 \\ 3 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix}$$

6. $D = (A - B)^2 A + 2B$, где

$$A = \begin{pmatrix} 5 & -1 & 3 \\ 0 & 2 & -1 \\ -2 & -1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 3 & 7 & -2 \\ 1 & 1 & -2 \\ 0 & 1 & 3 \end{pmatrix}$$

7. $D = (A^2 - B^2)(A + B^2)$, где

$$A = \begin{pmatrix} 7 & 2 & 0 \\ -7 & -2 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 0 & 2 & 3 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{pmatrix}$$

8. $D = 2(A - B)(A^2 + B)$, где

$$A = \begin{pmatrix} 5 & 1 & 7 \\ -10 & -2 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 4 & 1 \\ 3 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix}$$

9. $D = 2A - (A^2 + B)B$, где

$$A = \begin{pmatrix} 1 & 4 & 2 \\ 2 & 1 & -2 \\ 0 & 1 & -1 \end{pmatrix}, \qquad B = \begin{pmatrix} 4 & 6 & -2 \\ 4 & 10 & 1 \\ 2 & 4 & -5 \end{pmatrix}$$

10. $D = 2(A - 0,5B) + A^3B$, где

$$A = \begin{pmatrix} 5 & 3 & -1 \\ 2 & 0 & 4 \\ 3 & 5 & -1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 4 & 16 \\ -3 & -2 & 0 \\ 5 & 7 & 2 \end{pmatrix}$$

11. $D = (A - B)A^2 + 3B$, где

$$A = \begin{pmatrix} 3 & 2 & -5 \\ 4 & 2 & 0 \\ 1 & 1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} -1 & 2 & 4 \\ 0 & 3 & 2 \\ -1 & -3 & 4 \end{pmatrix}$$

12. $D = 3(A^2 + B^2) - 2AB$, где

$$A = \begin{pmatrix} 4 & 2 & 1 \\ 3 & -2 & 0 \\ 0 & -1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 0 & 2 \\ 5 & -7 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

13. $D = 2A^3 + 3B(AB - 2A)$, где

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 5 & 3 & 1 \\ -1 & 2 & 0 \\ -3 & 0 & 0 \end{pmatrix}$$

14. $D = A(A^2 - B) - 2(B + A)B$, где

$$A = \begin{pmatrix} 2 & 3 & 1 \\ -1 & 2 & 4 \\ 5 & 3 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 7 & 13 \\ -1 & 0 & 5 \\ 5 & 13 & 21 \end{pmatrix}$$

15. $D = (2A - B)(3A + B) - 2A^2B$, где

$$A = \begin{pmatrix} 1 & 0 & 3 \\ -2 & 0 & 1 \\ -1 & 3 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 7 & 5 & 2 \\ 0 & 1 & 2 \\ -3 & -1 & -1 \end{pmatrix}$$

4.5. Вычислить 3 значения функции на заданном отрезке. Вывести значения аргумента и значения функции на экран. Построить 3-D график функции:

1. Функция $f(x,y)=\frac{\sin x\cos y}{x+\cos x}$, на отрезке x∈ [0, π],y∈[0, π]
2. Функция $f(x,y)=x\sin xy+\frac{e^{-x}-e^{y}}{e^{-y}+e^{x}}$, на отрезке x∈ [0, π],y∈[0, 1]
3. Функция $f(x,y)=\left(\frac{e^{-x}-e^{y}}{2}\right)^2+\left(\frac{e^{-y}+e^{x}}{2}\right)^2$, на отрезке x,y∈[-3, 3]
4. Функция $f(x,y)=4xy-\cos^2(\frac{e^{-y}-e^{x}}{e^{-x}+e^{y}})$, на отрезке x,y∈[0, 1]
5. Функция $f(x,y)=e^{\frac{y+5}{x^2}}-\sqrt{x+y}$, на отрезке x,y∈[0, 3]
6. Функция $f(x,y)=\operatorname{tg}\left(\sqrt{\frac{2x-y}{x+y+1}}\right)+\sqrt{y+x}$, на отрезке x,y∈ [0.7, 4]
7. Функция $f(x,y)=\frac{x+2y}{1+\sqrt[3]{xy+1}}$., на отрезке x,y∈[10,100]
8. Функция $f(x,y)=\frac{sin^2(2xy+\frac{\pi}{2})}{\sqrt{\frac{|xy|+1}{4}}}$, на отрезке x,y∈[-π/2, π/2]
9. Функция $f(x,y)=\log_2(\sin x+\sin y+1)\sqrt{e^{-\cos x}}$, на отрезке x,y∈[0, π/2]
10. Функция $f(x,y)=|\sin x+\cos y|+x+y$, на отрезке x,y∈[0, π/2]
11. Функция $f(x,y)=\frac{sh\,x+ch\,y}{th\,xy+5}$, на отрезке x,y∈[-3,3]
12. Функция $f(x,y)=4cos^2\left(\frac{x+y+\frac{\pi}{3}}{xy+2}\right)$, на отрезке x,y∈[0, 1]
13. Функция $f(x,y)=x\,(\sin y+\cos x)\frac{y}{1+sin^2\,y}$, на отрезке x,y∈[0, π/2]
14. Функция $f(x,y)=5x^2-2y^2+xy$, на отрезке x,y∈[0, 3]
15. Функция $f(x,y)=(\sin x^2+\cos y^2)^y$, на отрезке x,y∈[-1, 1]

4.6 Сгенерировать матрицу из (N+10)x(100*N) одинаково распределенных случайных чисел, где N – номер студента в списке группы. Построить гистограмму распределения этих чисел. Вычислить среднее, дисперсию и стандартное отклонение, медиану. Варианты распределений случайных чисел:

1. Равновероятное распределение на интервале (0,1).
2. Нормальное распределение с мат. ожиданием 1 и дисперсией 2.
3. Равновероятное распределение на интервале (-5,5).
4. Нормальное распределение с мат.ожиданием 5 и дисперсией 0.25.
5. Экспоненциальное распределение с параметром лямбда=5.
6. Экспоненциальное распределение с параметром лямбда=10.

7. Распределение Пирсона хи-квадрат с 5-ю степенями свободы.
8. Распределение Пирсона хи-квадрат с 20-ю степенями свободы.
9. Распределение Пирсона хи-квадрат с 1, 10, 30-ю степенями свободы.
10. Распределение Пирсона хи-квадрат с 1, 5, 20-ю степенями свободы.
11. Распределение Пуассона с мат. ожижанием 2.
12. Распределение Пуассона с мат. ожижанием 10.
13. Нормальное распределение с мат. ожиданием -5 и дисперсией 5.
14. Равновероятное распределение на интервале (-3,10).
15. Экспоненциальное распределение с параметром лямбда=1.

5 Методические указания по выполнению работы

5.1. При выполнении задания 4.3 рекомендуется воспользоваться функцией **plot2d.** Синтаксис функции (параметры в квадратных скобках не обязательны):

```
plot2d([logflag,][x,],y[,style[,strf[,leg[,rect[,nax]]]]])
plot2d([logflag,][x,],y,<opt_args>)
```

Здесь x –действительная матрица или вектор; если x отсутствует, то предполагается, что вектор изменяется от 1:n, где n - число пунктов кривой, определяемых параметром y;

y - действительная матрица или вектор;

<opt_args> - последовательность ключей key1=value1, key2=value2,..., где key1, key2,... могут иметь значения:

logflag - устанавливает масштаб вдоль осей (линейный или логарифмический), возможные значения: "nn", "nl", "ln" и "ll";

style - устанавливает стиль для каждой кривой, целочисленное положительное или отрицательное значение;

strf - управляет отображением заголовка, задается в виде последовательности из трех цифр "xyz" (по умолчанию strf = "081").

Значения этих цифр и остальных ключей можно посмотреть в справке к функции. Ниже приведен пример сценария для построения графиков 3-х синусоидальных функций с использованием plot2d:

```
clf;
x=[0:0.1:2*%pi]';
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
```

5.3 В ходе выполнения задания 4.4. используйте встроенные функции для работы с матрицами.

5.4. При построении 3-D графиков (задание 4.5) рекомендуется изучить возможности пункта меню «Инструменты» графического окна, в частности, вращения изображения и увеличения области, а также возможностей контекстного

меню, которое появляется при нажатии правой кнопки мышки. Используйте эти инструменты для более детального анализа свойств функции и редактирования изображения и надписей.

5.5 При построении гистограмм в соответствии с заданием 4.6 воспользуйтесь расширенными возможностями функции histplot, приведенными в примере:

```
d=rand(1,10000,'normal');
clf(1); figure(1); histplot(20,d)
xgrid
clf(2); figure(2); histplot(20,d,normalization=%f)
clf(3); figure(3); histplot(20,d,leg='rand(1,10000,"normal")',style=5)
clf(4); figure(4); histplot(20,d,leg='rand(1,10000,"normal")',style=16,
rect=[-3,0,3,0.5]);
```

6 Содержание отчета

6.1 Цель работы.
6.2 Вариант задания, описание используемых формул.
6.3 Листинги программ с комментариями.
6.4 Результаты вычислений и графики функций.
6.5 Выводы по результатам исследований.

7 Контрольные вопросы

7.1 Дайте общую характеристику системы моделирования Scilab и приведите основные отличия от системы Matlab.
7.2 Какие типы данных, используются в Scilab и каким образом задаются константы в программах для Scilab?
7.3 Как в Scilab определяются пользовательские функции?
7.4 Приведите пример задания вектора-строки, вектора столбца и матрицы.
7.5 Поясните, как осуществляются операции конкатенации матриц и как они реализуются инструкциями Scilab?
7.6 Как математически записываются комплексные числа и какие операции существуют в Scilab для работы с такими числами?
7.7 Какие основные статистические функции имеются в Scilab для обработки матриц?
7.8 Каким образом можно сохранить данные в файле или ввести их в Scilab из файла?
7.9 Какие основные команды Scilab используются для построения двумерных графиков?
7.10 Как сделать, чтобы вывод нового графика не стирал предыдущий график?

7.11 Как отобразить на графике координатную сетку, подписать рисунок и названия осей координат?
7.12 Как построить в Scilab 3-D поверхность?
7.13 Напишите сценарий генерирования случайной последовательности с нормальным законом распределения и построения гистограммы распределения.
7.14 Как можно разделить графическое окно на отдельные подокна?
7.15 Как можно разделить переменные на равномерные интервалы? Приведите пример.
7.16 Напишите сценарий генерирования смеси гармонического сигнала с шумом и графического отображения этого процесса.
7.17 Напишите сценарий генерации случайных чисел с экспоненциальным распределением и построения гистограммы такого распределения.

ЛАБОРАТОРНАЯ РАБОТА № 2

ИССЛЕДОВАНИЕ АКТИВАЦИОННЫХ ФУНКЦИЙ НЕЙРОННЫХ ЭЛЕМЕНТОВ

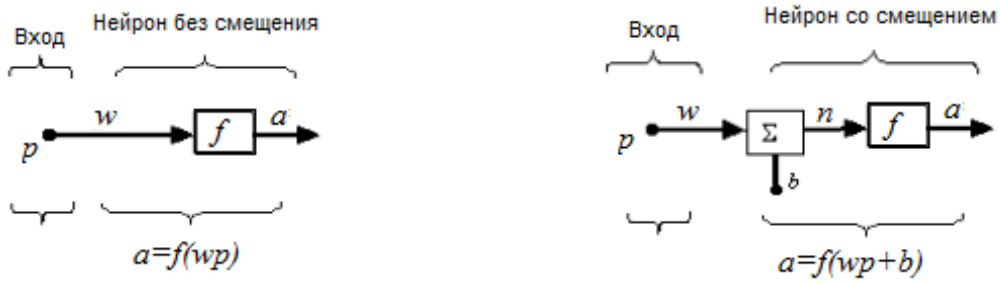
1 Цель работы

Углубление теоретических знаний в области архитектуры нейронных сетей, исследование свойств активационных функций нейронных элементов, приобретение практических навыков моделирования простейшей нейронной сети прямого распространения.

2 Основные теоретические положения

2.1 Структура нейрона с одним входом

Элементарной ячейкой нейронной сети является простейший нейрон. Структурная схема простейшего нейрона с единственным скалярным входом показана на рисунке 2.1*а* [3].



а) *б*)

Рисунок 2.1 – Структура нейрона с единственным входом

Скалярное входное значение p умножается на скалярный весовой коэффициент w и подается на вход функции активации нейрона f, которая порождает скалярный выход a.

Нейрон, изображенный на рисунке 2.1 *б*, дополнен скалярным смещением b. Смещение суммируется со взвешенным входом wp и приводит к сдвигу аргумента n функции f на величину b, т.е. $n = wp + b$. Выходом функции активации по-прежнему является a. Переменные w и b являются скалярными параметрами нейрона. Основной принцип работы нейронной сети состоит в настройке параметров нейрона таким образом, чтобы поведение сети соответствовало некоторому желаемому критерию. Регулируя веса и смещение, можно обучить сеть выполнять определенную функцию.

Уравнение простейшего нейронного элемента с одним входом и смещением можно записывать в виде

$$a = f(w * p + b * 1). \qquad (2.1)$$

Здесь константа 1 рассматривается как константное входное значение. Если ввести в рассмотрение *расширенный вектор весов* **x**=[w b**]** и *расширенный вектор входа* **z=[***p* 1**]**$^{\mathbf{T}}$, то выражение (2.1) можно записать в векторной форме:

$$a = f([w \quad b]\begin{bmatrix} p \\ 1 \end{bmatrix}) = f(\mathbf{x}\,\mathbf{z}). \qquad (2.2)$$

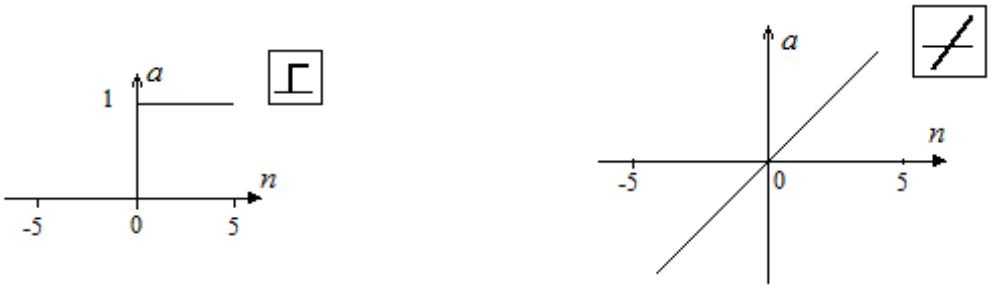
2.2 Активационные функции

Рассмотрим основные виды функций активации *f*, используемых в нейронных сетях и реализованные в модуле NeuralNetworks 3.0 пакета Scilab.

Единичная функция активации с жестким ограничением (hardlim) соответствует функции Хевисайда. Она равна 0, если $n<0$, и 1, если $n \geq 0$. Пример вызова функции в пакете Scilab:

```
n = [-5:0.1:5];
a = ann_hardlim_activ(n);  // вызов функции hardlim
plot(n,a,'.');
```

В результате выполнения этого кода получим график функции hardlim в диапазоне значений входа от –5 до + 5 (рисунок 2.2*а*).



а) *б*)

Рисунок 2.2 – Функции активации нейронных элементов

Линейная функция активации (purelin**)**. Линейная функция описывается выражением $a=n$. График функции изображен на рисунке 2.2*б*. Пример вызова функции в пакете Scilab: a = ann_purelin_activ(n).

Логистическая (сигмовидная) функция активации (logsig). Эта функция описывается соотношением $a = 1/(1 + \exp(-n))$. График функции изображен на рисунке 2.3. Аргумент функции может принимать любое значение в диапазоне от –∞ до +∞, а выход изменяется в диапазоне от 0 до 1. Пример вызова функции в пакете Scilab: a = ann_logsig_activ(n). Благодаря своей дифференцируемости эта функция часто используется в нейросетях.

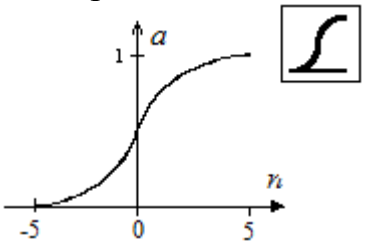


Рисунок 2.3 – График логистической (сигмовидной) функции активации

Тангенциальная (сигмовидная) функция активации (tansig). Эта функция описывается соотношением $a = (exp(n)-(exp(-n))/(exp(n) + exp(–n))$. График функции подобен графику функции logsig. Её аргумент также может принимать любое значение в диапазоне от –∞ до +∞, но значения функции изменяются в диапазоне от -1 до 1. Пример вызова функции в пакете Scilab: a = ann_tansig_activ(n).

В таблице 2.1 представлены дополнительные функции активации, часто используемые в моделях ИНС. При необходимости они легко могут быть запрограммированы самостоятельно.

Таблица 2.1 – Дополнительные функции активации

Симметричная hardlim	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Линейная с насыщением	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Симметричная линейная с насыщением	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Положительная линейная (ReLU)	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Состязательная	$a = 1$ для нейрона с max n $a = 0$ для всех других		compet

Символы, изображенные в квадрате в правом верхнем углу графиков на рисунках 2.2–2.3, а также в 3-ем столбце таблицы 2.1, представляют условное обозначение функций активации. Эти обозначения используются на структурных схемах нейронных сетей.

2.3 Нейрон с векторным входом и сеть прямого распространения

Нейрон с вектором входа **p**, состоящим из R элементов $p_1, p_2, \ldots, p_R$, изображен на рисунке 2.4. Здесь каждый элемент вектора **p** умножается на веса $w_{11}, w_{12}, \ldots, w_{1R}$. Взвешенные значения элементов вектора входа **p** поступают на сумматор. Их сумма равна скалярному произведению вектора-строки **W** на вектор входа **p**.

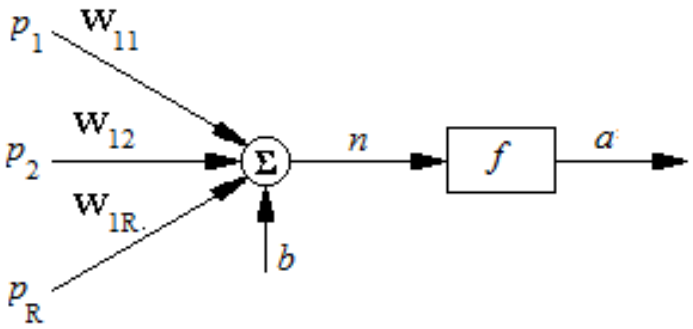


Рисунок 2.4 – Схема простейшего нейрона с векторонным входом

Смещение b суммируется со взвешенной суммой входов. Результирующая сумма n, называемая *сетевым значением*, равна

$$n = w_{11}\, p_1 + w_{12}\, p_2 + \ldots + w_{1R}\, p_R + b\,. \qquad (2.3)$$

Сетевое значение n служит аргументом функции активации f. Выражение (2.3) удобно записывать в векторно-матричной форме:

$$n = \mathbf{W}\mathbf{p} + b, \qquad (2.4)$$

где $\mathbf{W} = [w_{11}, w_{12}, \ldots, w_{1R}]$.

При изображении нейросетей, содержащих нейроны с векторными входами, часто используется обобщенная векторно-матричная схема нейрона (рисунок 2.5), которая структурно соответствует выражению (2.4). Вход нейрона изображается в виде темного прямоугольника, под которым указывается количество элементов R входного вектора **p**. Размер вектора входа **p** указывается ниже символа **p** и равен $R\times1$. Вектор входа умножается на вектор-строку **W** длины R. Как и прежде, константа 1 рассматривается как вход, который умножается на скалярное смещение b.

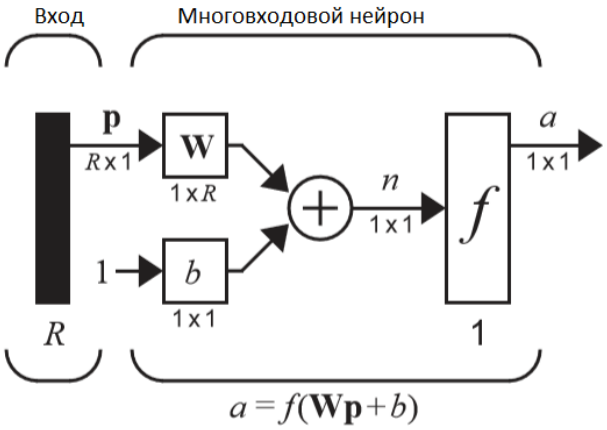


Рисунок 2.5 – Обобщенная векторно-матричная схема нейронного элемента

Входом n функции активации нейрона служит сумма смещения b и произведения **Wp**. Эта сумма преобразуется функцией активации f в выходное значение нейрона a, которое в данном случае является скалярной величиной. Структурная схема, изображенная на рисунке 2.5, соответствует однослойной сети с одним нейроном в слое. Если слой содержит S нейронов, то он характеризуется *матрицей весов* **W** размером SxR, и вектором смещений **b** размером Sx1.

С помощью схемы однослойной сети, изображенной на рисунке 2.5, можно построить многослойную сеть. В качестве примера на рисунке 2.6 изображена трехслойная сеть прямого распространения (FF – feed forward). Для этой сети выход предыдущего слоя является входом следующего слоя. Входом сети является вход первого слоя, т.е. вектор **p**, а выходом – выход последнего слоя, т.е. вектор $\mathbf{a}^3$. Соответственно, прямое распространение входного вектора по сети опишется уравнением:

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3) \quad . \qquad (2.5)$$

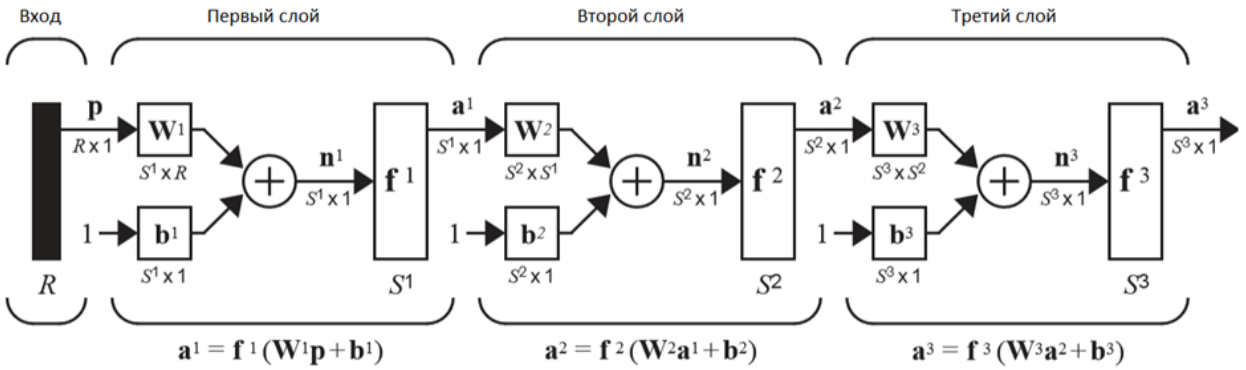


Рисунок 2.6 – Трехслойная сеть прямого распространения

Внутренние слои сети называются *скрытыми слоями* (hidden layers). Вход сети, представленный вектором **p**, называют *входным слоем*. Сокращенно структуру многослойной сети обозначают, указывая последовательно размерность входного слоя и количество нейронов в последующих слоях. Например, *структура сети*, изображенная на рисунке 2.6, обозначается в виде: R-S^1-S^2-S^3.

2.4. Производные активационных функций и инициализация весов нейросети

В ходе обучения нейронной сети выполняют минимизацию некоторой функции потерь (целевой функции) сети, которая определяется целью функционирования сети. Поиск минимума функции потерь сопряжен с её дифференцированием, что приводит к необходимости вычисления производных функции потерь и распространению их значений в обратном направлении для определения чувствительности функции потерь к настраиваемым параметрам в слоях сети. Важно, чтобы при обратном распространении значения производных не становились нулевыми (например, это возможно из-за наличия зон насыщения функций активации), т.к. это приводит к потере эффективности рассматриваемых далее градиентных алгоритмов обучения.

Производные некоторых рассмотренных выше линейных функций активации равны 0 в зоне насыщения и равны 1 на линейном участке (satlin, satlins, poslin, purelin).

Производные сигмовидных функций вычисляются на основе следующих выражений [3]:

- для логистической сигмовидной функции (logsig) :

$$a(n) = \frac{1}{1+e^{-n}}$$

производная равна

$$\frac{da(n)}{dn} = a(n)(1 - a(n)) ; \quad (2.6)$$

- для тангенциальной сигмовидной функции (гиперболический тангенс, tansig):

$$a(n) = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$$

производная равна

$$\frac{da(n)}{dn} = (1 - a^2(n)). \quad (2.7)$$

Под инициализацией нейронной сети подразумевается задание начальных значений матрицы весов **W**. Обычно начальным весам сети присваивают небольшие случайные значения, а смещениям – нули. Для этого могут непосредственно использоваться генераторы случайных чисел (rand, grand), рассмотренные в лабораторной работе 1. В модуле NeuralNetworks 3.0 пакета Scilab имеется встроенная функция ann_ffbp_init для инициализации весов и смещений многослойной сети с прямыми связями. Вызов этой функции осуществляется следующим образом:

```
W = ann_ffbp_init(N,r),
```

где N – вектор, задающий количество нейронов в каждом слое сети, включая входной и выходной слои; r– вектор, задающий желаемый диапазон изменения значений весов; W – список, элементами которого являются *расширенные матрицы весов* W(i) слоя i. Функция инициализирует веса сети путём вызова генератора случайных чисел grand с равномерным законом распределения. При этом векторы смещений b(i) каждого слоя получают нулевые значения и включаются в матрицу W(i) в виде её последнего столбца. Пример вызова функции инициализации для сети со структурой 3-3-2-1:

```
N = [3 3 2 1];
r = [-1 1];
W = ann_ffbp_init(N, r)
W  =
      W(1)
           0.9297771   0.4516779   0.9143339   0.
           0.9353899   0.9411856  -0.7802765   0.
          -0.6847738   0.9622194  -0.0292487   0.
     W(2)
           0.5962117  -0.4059411  -0.990433    0.
           0.6005609  -0.7162273  -0.1564774   0.
      W(3)
          -0.775071   0.831471   0.
```

3 Варианты заданий и программа работы

3.1. Повторить теоретический материал по источникам [1, 2, 5].

3.2. Выбрать вариант задания в соответствии с таблицей 2.2. Правила определения номера варианта изложены в лабораторной работе 1.

Таблица 2.2 – Варианты заданий

Вариант	Структура сети	Активационная функция 1	Активационная функция 2	Входной сигнал для i-го входа сети
1	100-2-1	logsig	purelin	sin(2*%pi*0.01*i*t)
2	100-4-1	logsig	satlin	sin(2*%pi*0.02*i*t)
3	100-6-1	logsig	satlins	sin(2*%pi*0.03*i*t)
4	100-8-1	logsig	poslin	sin(2*%pi*0.04*i*t)
5	100-10-1	tansig	purelin	sin(2*%pi*0.01*i*t)
6	100-12-1	tansig	satlin	sin(2*%pi*0.02*i*t)
7	100-14-1	tansig	satlins	sin(2*%pi*0.03*i*t)
8	100-16-1	tansig	poslin	sin(2*%pi*0.04*i*t)
9	50-4-1	logsig	purelin	sin(2*%pi*0.01*i*t)
10	50-8-1	logsig	satlin	sin(2*%pi*0.02*i*t)
11	50-16-1	logsig	satlins	sin(2*%pi*0.03*i*t)
12	50-32-1	logsig	poslin	sin(2*%pi*0.04*i*t)
13	50-16-1	tansig	purelin	sin(2*%pi*0.01*i*t)
14	50-8-1	tansig	satlin	sin(2*%pi*0.02*i*t)
15	50-4-1	tansig	satlins	sin(2*%pi*0.03*i*t)

3.3. Построить графики активационных функций и их производных в соответствии с вариантом задания, используя соответствующие встроенные функции модуля NeuralNetworks 3.0 пакета Scilab. В случае отсутствия указанных встроенных функции определить их самостоятельно на основе выражений, приведенных в таблице 2.1.

3.4. Реализовать две сети прямого распространения заданной структуры, запрограммировав вычисления в соответствии с выражением (2.5) для каждой из двух заданных видов активационных функций.

3.5. Выполнить моделирование двух нейросетей, сгенерировав входные сигналы в соответствии с вариантом задания на интервале времени от 0 до 2/*F*, где *F* – минимальная частота входного гармонического сигнала (например, для варианта 1 F=0.01Гц). При этом инициализацию сетей выполнить двумя способами:

1) случайными значениями из диапазона от -10 до +10;

2) случайными значениями из диапазона от -1/sqrt(R) до +1/sqrt(R), где R число входов нейрона.

3.6. Вычислить выходные значения всех нейронов сети. Построить графики активностей одного из нейронов скрытого слоя и нейрона выходного слоя, а также гистограмму общей активности всех нейронов скрытого слоя и отдельно гистограмму активности выходного нейрона для двух способов инициализации для каждой из двух реализованных сетей. Вычислить значение средней активности и стандартного отклонения на выходах нейронов скрытого и выходного слоёв.

3.7. Вычислить значения локальных производных нелинейностей для всех нейронов в точках, соответствующих значениям входных сигналов этих нейронов. Построить графики вычисленных значений локальных производных для одного из нейронов скрытого слоя и нейрона выходного слоя, а также гистограмму значений локальных производных функции активации скрытых нейронов и выходного нейрона (для двух способов инициализации и для каждой из двух реализованных сетей). Вычислить среднее значение и стандартное отклонение локальных производных для нейронов скрытого и выходного слоёв.

3.8. Выполнить анализ полученных результатов, обратив внимание на характер гистограмм. Сделать выводы о характере распределения значений активностей и значений локальных производных для заданных активационных функций и каждого из способов инициализации. Сравнить результаты для каждой из двух реализованных сетей.

3.9. Подготовить и защитить отчет по работе.

4 Методические рекомендации по выполнению работы

4.1. Модуль NeuralNetwork 3.0 пакета Scilab имеет ограниченный набор встроенных функций активации и их производных:

a = ann_hardlim_activ(n) – единичная с жестким ограничением;
a = ann_logsig_activ(n) – логистическая, однополярная сигмоидальная;
a = ann_purelin_activ(n) – линейная;
a = ann_tansig_activ(n) – биполярная тангенциальная сигмоидальная;
d_a= ann_d_hardlim_activ(y) – производная единичной функции;
d_a = ann_d_logsig_activ(y) – производная логистической функции;
d_a = ann_d_purelin_activ(y) – производная линейной функции;
d_a = ann_d_tansig_activ(y) – производная биполярной сигмоидальной функции.

Остальные функции активации и их производные следует определять самостоятельно на основе выражений, указанных в таблице 2.1.

Обратите внимание на то, что функции ann_d_logsig_activ(y) и ann_d_tansig_activ(y) вычисляют производные на основе выражений (2.6) и (2.7). При этом входные аргументы этих функций должны представлять собой предварительно вычисленные значения однополярной сигмовидной функции ann_logsig_activ(n) или тангенциальной биполярной сигмовидной функции ann_tansig_activ(n).

Указанные выше встроенные функции допускают обработку входных аргументов, заданных в виде векторов или матриц. В этом случае они применяются к векторам и матрицам поэлементно.

4.2. При выполнении задания 3.4 для реализации выражения (2.5) удобно входные векторы представлять матрицей **P**, в которой каждый столбец соответствует очередному входному вектору. В этом случае произведение **WP** будет матрицей, что потребует преобразования вектора смещений **b** в матрицу для обеспечения выполнения операции сложения при вычислении **n**. Преобразование **b** матрицу можно выполнить с помощью встроенной функции repmat пакета Scilab. Пример вызова этой функции см. ниже в п.4.3.

4.3. Для генерации матрицы **P** рекомендуется воспользоваться нижеследующим примером, в котором также показано как выполнить моделирование сети (задание 3.5) на примере одного слоя, построить графики активностей и производных, вычислить необходимые статистики (задание 3.6 и 3.7):

```
//задание исходных данных
F=0.01 //минимальная частота входного сигнала
R=5; //число входов слоя
S=3; // число нейронов слоя

//формирование матрицы p входных данных, размером R x length(t)
t=0:0.1:2/F;          // формируем массив отсчетов времени от 0 до 2/F с шагом 0.1
fi=(2*%pi*F).*t;
p=[sin(fi)];          // формируем массив отсчетов функции sin для 1-го входа
for i=2:R             // формируем массивы отсчетов функции sin для i-го входа, где i=1..R
   p=[p; sin(fi*i)];  // и добавляем эти массисы  в виде очередной строки в матрицу p
end

//выполняем инициализацию весов и смещений
scale=1/sqrt(R);
w=grand(S,R,'unf',-scale,scale)
b=zeros(S,1);

//выполняем моделирование слоя с tansig нелинейностью
n=w*p+repmat(b,1,size(p,2))  // repmat копирует вектор b в виде столбцов
                             // и строит из него матрицу размера size(w*p)= (S x length(t))
a=ann_tansig_activ(n);

//вычисление производных tansig нелинейности
d_a=ann_d_tansig_activ(a);

//вычисление статистик
mean_a=mean(a)
stdev_a=stdev(a)
mean_d_a=mean(d_a)
stdev_d_a=stdev(d_a)

//построение графиков активностей, производных и гистограмм
clf(1);
figure(1);
subplot(2,2,1)
plot(t,a(1,:),t,a(2,:), t,a(3,:));
title('Активность нейронов слоя')
subplot(2,2,2)
```

```
plot(t,d_a(1,:),t,d_a(2,:), t,d_a(3,:));
title('Производные функции активации  слоя')
subplot(2,2,3)
histplot(20,a);
title('Гистограмма активности нейронов слоя')
subplot(2,2,4)
histplot(20,d_a);
title('Гистограмма производных функции активации')
```

Результаты вычислений представлены на рисунке 2.7.

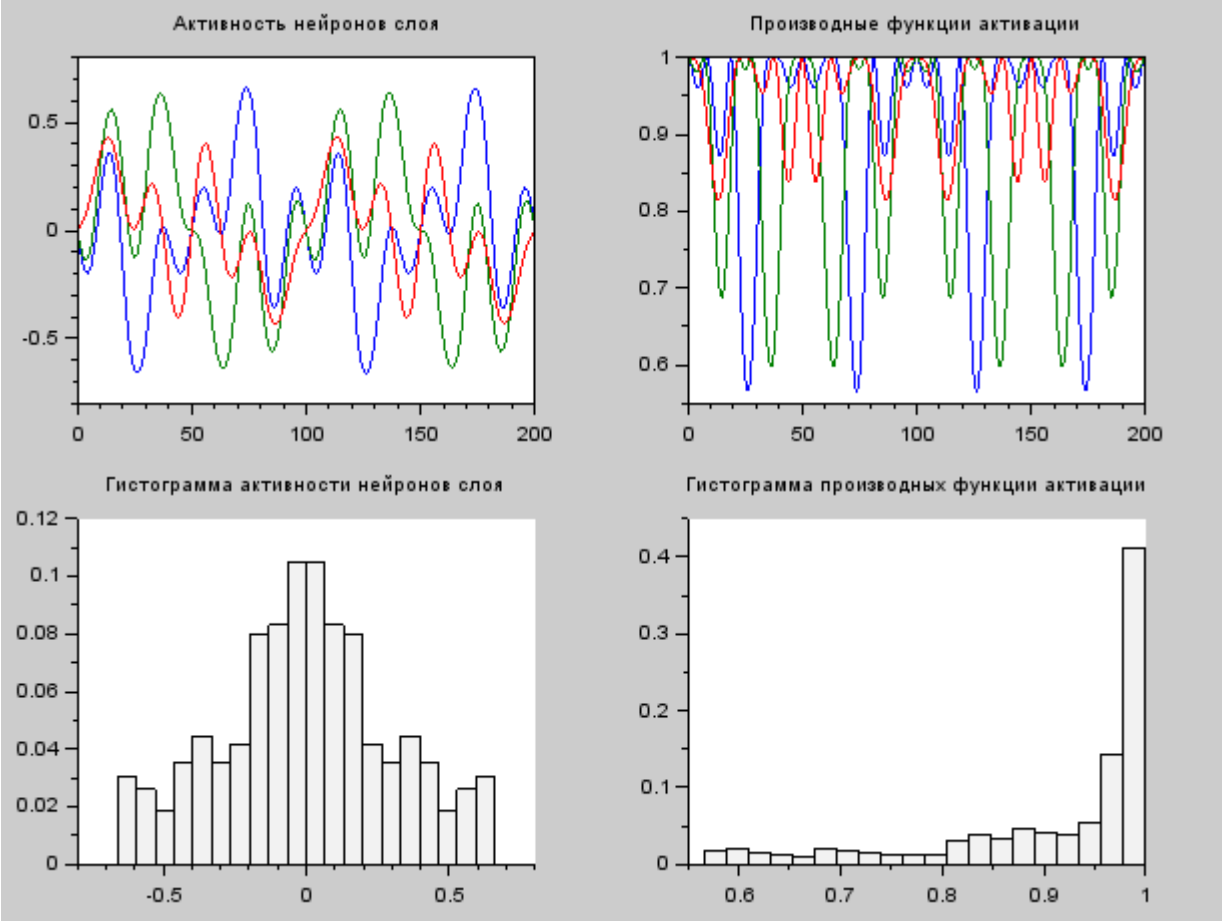


Рисунок 2.7 – Графики активности нейронов и производных функции активации, гистограммы

5 Содержание отчета

5.1. Цель работы.
5.2. Вариант задания.
5.3. Схема нейронной сети, формулы нелинейностей и их производных в соответствии с вариантом задания.
5.4. Листинги программ с комментариями.
5.5. Графики активности на выходе нейрона скрытого и выходного слоёв. Гистограммы, значения основных статистик активности.

5.6. Графики значений производных нелинейности на выходе нейронов скрытого и выходного слоёв. Гистограммы, значения основных статистик производных
5.7. Выводы по результатам исследований.

6 **Контрольные вопросы**

6.1. Нарисуйте схему нейронного элемента с векторным входом и запишите выражение для вычисления выходного значения.
6.2. Приведите выражение функции активации с жестким ограничением (функция Хевисайда) и ее графическое представление.
6.3. Приведите выражения для линейных функций активации (satlin, satlins) с насыщением и их графическое представление.
6.4. Приведите выражение для знаковой функции активации и ее графическое представление.
6.5. Приведите выражения логистической и тангециальной (сигмовидных) функций активации и их графическое представление.
6.6. Продемонстрируйте на компьютере, каким образом проводились исследования функций активации?
6.7. Нарисуйте структурные схемы однослойной и многослойной искусственных нейронных сетей с прямыми связями.
6.8. Поясните понятия входного, скрытого и выходного слоев. Как сокращенно обозначают структуру сетей прямого распространения?
6.9. Приведите выражения для вычисления производных активационных функций, указанных в таблице 2.1.
6.10. Используя правила дифференцирования, выведите выражения для вычисления производных логистической и тагенциальной сигмовидных активационных функций.
6.11. Что такое инициализация нейронной сети и как её выполняют? Приведите примеры кода на языке Scilab.
6.12. Нарисуйте структурную схему 3-х слойной сети прямого распространения и запишите на языке Scilab фрагмент кода для вычисления её выходного значения, если последовательность входных векторов p представляется в виде матрицы размером R x Q.
6.13. Какой желательный характер распределений должны иметь выходные значения нейронов нейросети и почему? Как этого добиваются?
6.14. Почему нежелательно, чтобы производные активационных функций принимали нулевые значения?

ЛАБОРАТОРНАЯ РАБОТА № 3

ИССЛЕДОВАНИЕ ОДНОСЛОЙНОГО ПЕРСЕПТРОНА

1 Цель работы

Углубление теоретических знаний в области архитектуры нейронных сетей с пороговыми активационными функциями, исследование свойств однослойного персептрона и правила его обучения, приобретение практических навыков обучения и моделирования однослойной сети при решении простых задач классификации.

2 Основные теоретические положения

2.1 Структура однослойного персептрона

Общая структурная схема персептрона изображена на рисунке 3.1. Персептрон представляет собой однослойную сеть, которая состоит из S нейронов с R входами, характеризуется матрицей весов $\mathbf{W}_{SxR}$, и вектором смещений $\mathbf{b}_{Sx1}$, использует пороговую активационную функцию с жестким ограничением hardlim.

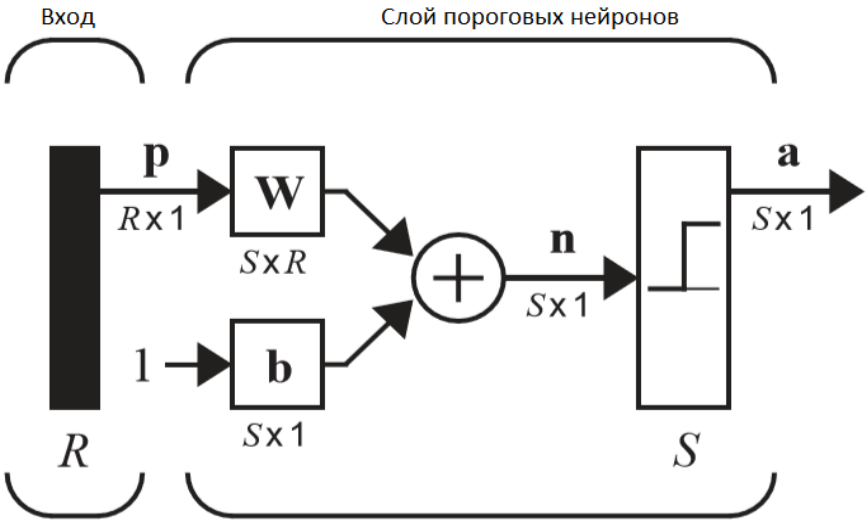


Рисунок 3.1 ─ Структурная схема однослойного персептрона

Каждая срока матрицы W соответствует набору весов одного из нейронов персептрона. Если обозначить i-ую строку матрицы как $_i\mathbf{w}^T$, то выход отдельного нейрона сети запишется в виде:

$$a_i = hardlim(n_i) = hardlim(_i\mathbf{w}^T\mathbf{p} + b_i). \quad (3.1)$$

Если скалярное произведение i-ой строки матрицы весов $_i\mathbf{w}^T$ на входной вектор $\mathbf{p}$, будет больше или равно $-b_i$, то на выходе нейрона будет 1, иначе ─ 0. Таким образом, каждый нейрон персептрона делит входное пространство на

две области. Персептрон, состоящий из S нейронов, способен распознать 2^S классов.

2.2 Граница решения

Рассмотрим персептрон из одного нейрона (**простой персептрон**) с 2-мя входами (рисунок 3.2.)

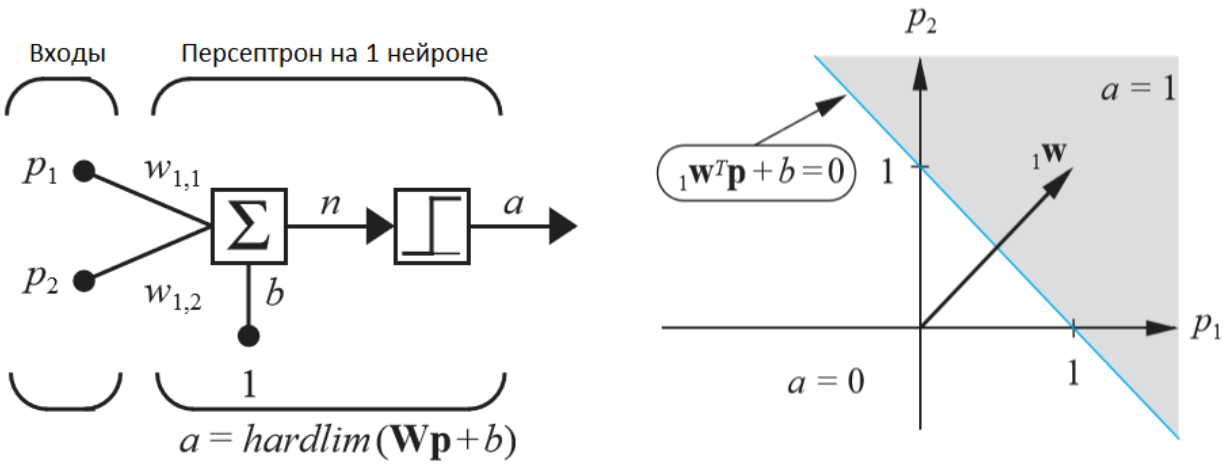


Рисунок 3.2 — Простой персептрон и его граница решения

С учетом только 2-х входов выходной сигнал нейрона запишется в виде:

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p} + b) = hardlim(w_{1,1}p_1 + w_{1,2}p_2 + b) \quad (3.2)$$

Так как функция hardlim(n) при n=0 меняет значение с 0 на 1, то ***граница решения*** будет определяться условием:

$$n = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = 0. \quad (3.3)$$

Пусть

$$w_{1,1} = 1,\ w_{1,2} = 1,\ b = -1.$$

Для этого случая граница решения запишется в виде уравнения прямой линии (рисунок 3.2.):

$$n = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = p_1 + p_2 - 1 = 0.$$

Чтобы изобразить её, найдеч точки пересечения линии с осями координат (p_1, p_2):

$$p_1 = -\frac{b}{w_{1,1}} = -\frac{-1}{1} = 1 \qquad p_2 = -\frac{b}{w_{1,2}} = -\frac{-1}{1} = 1$$

С одной стороны границы выход персептрона равен a=1, а с другой — a=0. Чтобы определить область, где выход персептрона будет равен 1, рассмотрим точку

$$\mathbf{p} = \begin{bmatrix} 2 & 0 \end{bmatrix}^T,$$

для которой a=1, действительно

$$a = hardlim\left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} - 1 \right) = 1.$$

Таким образом, часть входного пространства, где a=1, будет находиться выше границы решения (на рисунке 3.2. ─ затененная область).

2.3 Графическое построение границы решения

Границу решения можно также построить **графически**. Отметим, что ***граница всегда ортогональна вектору весов*** **w,** как изображено на рисунке 3.2. Действительно, граница решения определяется условием:

$$_1\mathbf{w}^T\mathbf{p} + b = 0. \tag{3.4}$$

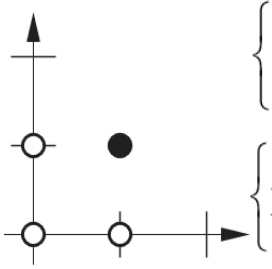
Для всех точек, принадлежащих границе, скалярное произведение входного вектора **p** на вектор весов **w** имеет одно и то же значение, равное -b. Это означает, что соответствующие входные векторы **p** имеют одно и то же значение своей проекции на вектор весов **w**, т.е. *концы таких векторов* **p** *должны лежать на линии ортогональной вектору весов*. Отметим, что *вектор весов всегда направлен в сторону области, где выход нейрона равен 1.*

После того как определено направление вектора весов **w**, значение **смещения** определяется путем выбора точки на границе и решения уравнения (3.4).

Пример.

Построим границу для случая реализации логической функции AND с помощью простого персептрона.

1. Зададим пары входных и выходных значений логического элемента AND в соответствии с рисунком 3.3.



$$\left\{\mathbf{p}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, t_1 = 0\right\} \left\{\mathbf{p}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, t_2 = 0\right\}$$

$$\left\{\mathbf{p}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, t_3 = 0\right\} \left\{\mathbf{p}_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_4 = 1\right\}.$$

Рисунок 3.3 ─ Входные и выходные значения логического элемента AND (зачерненная точка соответствует 1)

2. Выберем возможную границу решения (рисунок 3.4).

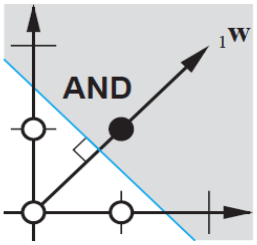


Рисунок 3.4 ─ Граница решения и вектор весов элемента AND

3. Построим вектор весов **w**, ортогональный границе. Вектор может иметь любую длину, например

$$_1\mathbf{w} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}.$$

4. Найдем смещение *b*. Для этого выберем произвольную точку на границе и решим уравнение:

$$\mathbf{p} = \begin{bmatrix} 1.5 & 0 \end{bmatrix}^T \qquad {}_1\mathbf{w}^T\mathbf{p} + b = \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} + b = 3 + b = 0 \quad \Rightarrow \quad b = -3.$$

5. Проверим решение. Например, для $\mathbf{p}_2$ на выходе персептрона должен быть 0. Действительно

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} - 3 \right) = hardlim(-1) = 0$$

2.4. Правило обучения персептрона

Правило обучения персептрона представляет собой процедуру **обучения с учителем** и заключается в поиске параметров (весов и смещений), которые обеспечивают совпадения желаемой **t** и действительной реакции персептрона **a** на заданный входной вектор **p**. Обучение осуществляется на основе обучающего **множества примеров** «вход-выход»:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}.$$

В ходе обучения персептрона на его вход подаются входные векторы **p** из множества примеров, вычисляется реакция **a**, которая сравнивается с требуемым выходом **t** и вычисляется ошибка **e.** Например, для простого персептрона с одним выходом ошибка равна

$$e = t - a.$$

В зависимости от значения ошибки корректируется вектор весов персептрона в соответствии с правилом:

$$_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p} = {}_1\mathbf{w}^{old} + e\mathbf{p} \qquad (3.5)$$

Для коррекции (обновления смещений) (смещение эквивалентно некоторому дополнительному входу с весом b, для которого p=1) используется упрощенное правило (3.5):

$$b^{new} = b^{old} + e\,. \tag{3.6}$$

В том случае, когда персептрон содержит несколько нейронов приведенные правила обучения можно записать в матричной форме:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^{T},$$
$$\mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e}\,. \tag{3.7}$$

В соответствии с правилами (3.7) для каждого очередного вектора **p** из множества примеров новые значения матрицы весов $\mathbf{W}^{new}$ и вектора смещений $\mathbf{b}^{new}$ вычисляются на основе их предыдущих значений $\mathbf{W}^{old}$ и $\mathbf{b}^{old}$. Обычно **начальные значения весов и смещений** инициализируют небольшими случайными числами. Как следует из (3.5), вектор весов отдельного нейрона персептрона в ходе обучения изменяется на величину очередного входного вектора **p** с учетом ошибки, которая равна -1, 0 или 1.

Можно доказать, что ***правило обучения персептрона обеспечивает схождение весов к требуемым значениям, обеспечивающим правильную классификацию, за конечное число шагов при условии существования решения*** (теорема сходимости) [1, 5].

Сетевая функция **n** персептрона играет роль дискриминантной функции. В общем случае эта функция описывает уравнение гиперплоскости. Поэтому простой персептрон может классифицировать только такие образы входного пространства, которые разделимы с помощью гиперплоскости. Иными словами, задачи классификации, решаемые простым персептроном, – это **линейные сепарабельные задачи**.

3. Варианты заданий и программа работы

3.1. Повторить теоретический материал, относящийся к архитектуре персептрона и его обучению [1, 4, 5].

3.2. Для заданной матрицы входных данных **P** и заданного вектора выходных значений **T** (таблица 3.1) разработать простой персептрон, решающий задачу классификации 2-х классов. Решение найти с помощью графического построения границы решения и вычисления весов и смещения вручную. Протестировать с помощью компьютера полученное решение для всех входных векторов (столбцов матрицы **P**).

Таблица 3.1 – Варианты задания п. 3.2

Вариант	Матрица **P**	Вектор **T**
1	[0 1 1 0; 0 0 1 1]	[0 0 1 1]
2	[0 1 1 0; 0 0 1 1]	[0 0 1 1]
3	[0 1 1 0;	[1 1 0 0]

	0 0 1 1]	
4	[0 1 1 0; 0 0 1 1]	[1 0 0 1]
5	[0 1 1 0; 0 0 1 1]	[0 0 1 0]
6	[0 1 1 0; 0 0 1 1]	[0 0 0 1]
7	[0 1 1 0; 0 0 1 1]	[1 0 0 0]
8	[0 1 1 0; 0 0 1 1]	[0 1 0 0]
9	[0 1 1 0; 0 0 1 1]	[1 1 0 1]
10	[0 1 1 0; 0 0 1 1]	[1 1 1 0]
11	[0 1 1 0; 0 0 1 1]	[1 0 1 1]
12	[0 1 1 0; 0 0 1 1]	[0 1 1 1]
13	[0 1 1 0 2; 0 0 1 1 0]	[0 0 1 0 1]
14	[0 1 1 0 2; 0 0 1 1 0]	[0 1 0 0 1]
15	[0 1 1 0 2; 0 0 1 1 0]	[1 1 0 0 1]

3.3. Даны четыре класса, каждый из которых представлен 2-мя точками (столбцами матрицы **P**), указанными в таблице 3.3. Необходимо:

- разработать структурную схему персептрона, распознающего эти 4 класса;
- выполнить предварительный анализ задачи, изобразив все точки четырех классов и построив графически возможные границы решений персептрона;
- анализируя графическую границу решений, задать допустимые целевые значения выходов персептрона (определить матрицу **T**) для всех входных точек, представленных столбцами матрицы данных **P**;
- изучить функцию ann_PERCEPTRON и, используя её, обучить персептрон правильному распознаванию входных классов;
- написать программу, которая:
  - отображает диаграмму размещения входных точек из **P** на плоскости с координатами (*p1, p2*);
  - обучает персептрон;
  - накладывает на диаграмму входных точек границы решения после обучения персептрона;
  - выполняет тестирование полученного решения для всех заданных входных данных, а также для дополнительно выбранных точек из областей принадлежности входных классов.

Таблица 3.3 – Варианты задания п.3.3

Вари-ант	Класс 1 **P(:,1), P(:,2),**	Класс 2 **P(:,3), P(:,4),**	Класс 3 **P(:,5), P(:,6),**	Класс 4 **P(:,7), P(:,8),**	
1	[1; 1], [2; 2]	[-1; 1], [-2; 2]	[-1; -1], [-2; -2]	[1; -1], [2; -2]	
2	[2; 1], [2; 2]	[-2; 1], [-2; 2]	[-2; -1], [-2; -2]	[2; -1], [2; -2]	
3	[1; 2], [2; 2]	[-1; 2], [-2; 2]	[-1; -2], [-2; -2]	[1; -2], [2; -2]	
4	[1; 1], [2; 2]	[-1; 1], [-2; 2]	[-1; 0], [-2; -1]	[1; 0], [2; -1]	
5	[2; 1], [2; 2]	[-2; 1], [-2; 2]	[-2; 0], [-2; -1]	[2; 0], [2; -1]	
6	[1; 2], [2; 2]	[-1; 2], [-2; 2]	[-1; -1], [-2; -1]	[1; -1], [2; -1]	
7	[1; 0], [2; 1]	[-1; 0], [-2; 1]	[-1; -1], [-2; -2]	[1; -1], [2; -2]	
8	[2; 0], [2; 1]	[-2; 0], [-2; 1]	[-2; -1], [-2; -2]	[2; -1], [2; -2]	
9	[1; 1], [2; 1]	[-1; 1], [-2; 1]	[-1; -2], [-2; -2]	[1; -2], [2; -2]	
10	[2; 1], [3; 2]	[-1; 1], [-2; 2]	[-1; -1], [-2; -2]	[2; -1], [3; -2]	
11	[3; 1], [3; 2]	[-2; 1], [-2; 2]	[-2; -1], [-2; -2]	[3; -1], [3; -2]	
12	[2; 2], [3; 2]	[-1; 2], [-2; 2]	[-1; -2], [-2; -2]	[3; -2], [3; -2]	
13	[1; 1], [2; 2]	[-2; 1], [-3; 2]	[-2; -1], [-3; -2]	[1; -1], [2; -2]	
14	[2; 1], [2; 2]	[-3; 1], [-3; 2]	[-3; -1], [-3; -2]	[2; -1], [2; -2]	
15	[1; 2], [2; 2]	[-2; 2], [-3; 2]	[-2; -2], [-3; -2]	[1; -2], [2; -2]	

3.4. Выполнить анализ полученных результатов, обратив внимание на то, что при каждом новом запуске функции обучения границы решения меняются. Сравнить результаты нескольких запусков программы. Сохранит ли программа работоспособность, если матрицу целевых значений выходов **T** формировать произвольным образом (например, используя в качестве целевого выходного вектора номер его класса в двузначном двоичном коде)?

3.5. Подготовить и защитить отчет по работе.

4. Методические рекомендации по выполнению работы

4.1. Модуль NeuralNetwork 3.0 пакета Scilab содержит встроенные функции для обучения и моделирования персептрона:

[w,b]= ann_PERCEPTRON(P,T) – функция обучения персептрона, код которой с комментариями приведен в приложении А;
a= ann_PERCEPTRON_run(P,w,b) – функция моделирования персептрона.

Ниже приведен пример использования этих функций для обучения и моделирования простого персептрона, реализующего логическую операцию AND:

```
//обучающие данные для элемента AND
P = [0 0 1 1 ;
       0 1 0 1];
 T = [0 0 0 1];
 // вызов функции обучения
[w,b] = ann_PERCEPTRON(P,T);
 Epoch: 1
 Epoch: 2
 Epoch: 3
 Epoch: 4
 Epoch: 5
 Epoch: 6
 // отображение рез-тов
 --> w
            w = 2.2113249 1.7560439
 --> b
           b = -2.9997789
 // проверка результатов обучения
 // вызов функции моделирования персептрона:
 // a = ann_hardlim_activ(w*P+repmat(b,1,size(P,2));
 a = ann_PERCEPTRON_run(P,w,b)
            a = 0. 0. 0. 1.
```

4.2. При выполнении задания 3.2 для графического построения границы решения простого персептрона и вычисления его весов и смещения следует руководствоваться примером, рассмотренным в п. 2.3 настоящей лабораторной работы. Для тестирования полученного решения необходимо использовать функцию моделирования персептрона:

```
a = ann_PERCEPTRON_run(P,w,b).
```

4.3 При выполнении задания 3.3 необходимо провести предварительный анализ расположения точек, представляющих классы, на плоскости ($p1$, $p2$) и нарисовать вручную возможные границы решения персептрона. Поскольку задано 4 класса, то персептрон должен состоять из 2-х нейронов, каждый из которых создаёт границу, которая делит входное пространство на 2 области. При этом первая граница разделяет множество входных примеров на 2 подмножества, а вторая граница разделяет эти подмножества еще раз на две части. Это позволяет персептрону успешно распознавать 4 заданных класса, которые должны быть линейно-разделимыми. Чтобы обучить этому персептрон, необходимо корректно сформировать матрицу целевых выходов персептрона **T**.

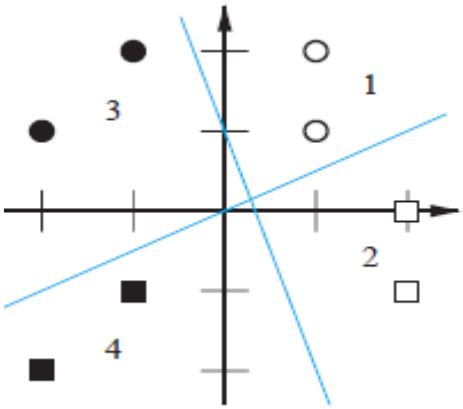


Рисунок 3.5

Рассмотрим пример. На рисунке 3.5 изображены «точки», представляющие входные классы (1, 2, 3, 4) и возможные границы решений персептрона. Необходимо сформировать целевые значения выходов двух нейронов персептрона. Для этого примем решение относительно того, с какой стороны границы на выходе нейрона будет 1, а с какой – 0. Пусть для границы, отделяющей зачерненные «точки» от не зачернённых, 1 будет со стороны зачерненных точек. Пусть для границы, отделяющей кружочки от квадратов, на выходе нейрона будет 1 со стороны, где изображены квадраты. Тогда векторы целевых значений выходов будут следующими:

класс 1: t = [0;0];
класс 2: t = [0;1];
класс 3: t = [1;0];
класс 4: t = [1;1].

Отсюда для заданной матрицы входных данных **P** (каждый столбец соответствует «точке» на рисунке)

P= [1 1 2 2 -1 -2 -1 -2;
1 2 -1 0 2 1 -1 -2]

матрица **T** целевых значений выходов будет равна:

T= [0 0 0 0 1 1 1 1;
0 0 1 1 0 0 1 1].

Теперь, используя значения матриц **P** и **T**, можно обучить персептрон, вызвав функцию обучения:

```
[w,b]=ann_PERCEPTRON(P,T)
```

Ниже приведен пример кода программы, обеспечивающей обучение персептрона для рассматриваемого примера, отображение диаграммы расположения точек и границ решений (рисунок 3.6):

```
clear //очистка рабочего пространства
K=4; //количество классов
N=2; //количество точек в классе
S=2; //число нейронов в слое

P=[ 1 1 2 2 -1 -2 -1 -2;
    1 2 -1 0  2  1 -1 -2];

T=[0 0 0 0 1 1 1 1;
   0 0 1 1 0 0 1 1];

// вызов функции обучения
[w,b] = ann_PERCEPTRON(P,T);
```

```
//построение диаграммы расположения входных точек
clf;
j=-3;
for i=1:K*N //цикл для всех точек данных из P
    // 3-й  аргумент plot2d ( , ,  j )  определяет отображаемый символ
   //  его допустимые значения -1,-2,…,-15
    plot2d(P(1,i),P(2,i),j);
    if modulo(i,2)==0 // если i четное
        j=j-1; // меняем j для отображения класса из P другим символом
    end
end
xgrid;
xtitle("Границы решений", "p1", "p2");
mtlb_hold('on')  // запрет очистки граф. окна

//отображение границ решений
p1 = min(P)-0.1:0.1:max(P)+0.1;
len=length(p1);
p2=zeros(S, len);
for i=1:S
    p2(i,:) = -w(i,1)/w(i,2).*p1 - b(i)./w(i,2); //уравнение прямой
    plot(p1,p2(i,:));
end

// тестирование персептрона
// вызов функции моделирования персептрона:
a = ann_PERCEPTRON_run(P,w,b)
a  =
  0.   0.   0.   0.   1.   1.   1.   1.
  0.   0.   1.   1.   0.   0.   1.   1.
```

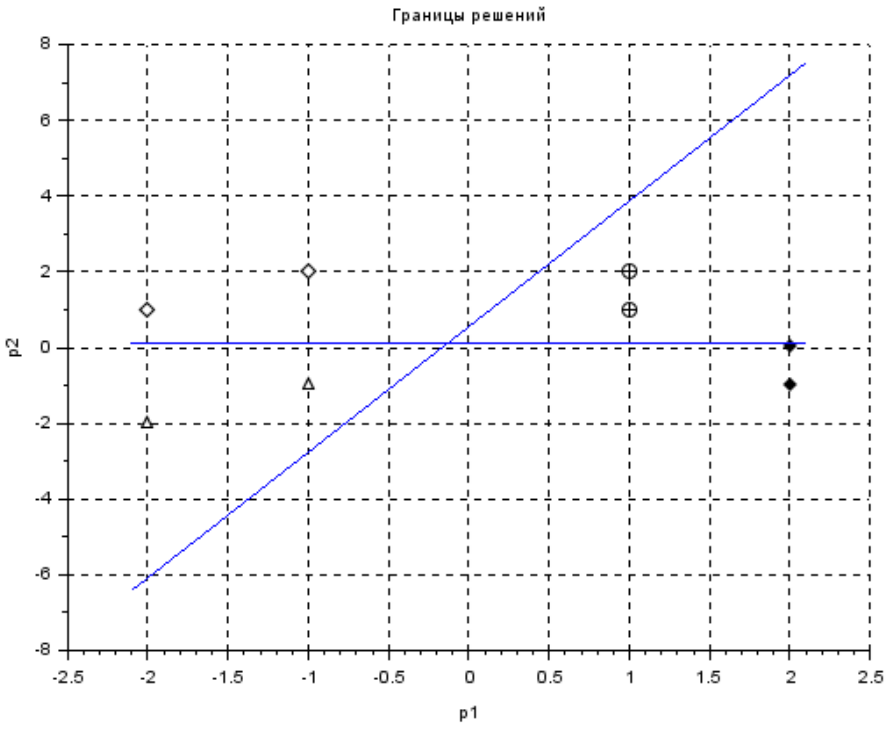


Рисунок 3.6 – Диаграмма расположения входных точек и границы решений

5. Содержание отчета

5.1. Цель работы.
5.2. Вариант задания.
5.3. Схема простого персептрона, решающего задачу классификации 2-х классов; графические построения границы решения и вычисления его весов и смещения, выполненные вручную; листинг программы с результатами тестирования простого персептрона.
5.4. Схема однослойного персептрона, решающего задачу классификации 4-х классов; пояснение выбора значений матрицы T; результаты обучения персептрона, правила обучения; диаграмма расположения классов с границами решений, соответствующими обученному персептрону; листинг программы с комментариями.
5.5. Выводы по результатам исследований.

6. Контрольные вопросы

6.1 Нарисуйте схему однослойного персептрона, объясните все обозначения, запишите формулу вычисления вектора выходных значений и объясните её.
6.2 Сколько линейно-разделимых классов способен распознать персептрон, содержащий S нейронов? Объясните почему?
6.3. Изобразите простой персептрон. Запишите уравнение границы решения.
6.4. Как построить границу решения простого персептрона, если известны его веса и смещения? Как определить область входного пространства, где выход персептрона будет равен 1?
6.5. Как можно графически построить границу решения персептрона. Приведите пример построения.
6.6. Как связана ориентация вектора весов с границей решения нейрона персептрона?
6.7. Запишите правило обучения персептрона в векторной и матричной форме?
6.8. Чему равны значения ошибки персептрона?
6.9. В каком направлении корректируются значения векторов весов нейронов персептрона?
6.10. Сформулируйте теорему сходимости персептрона.
6.11. Какой класс задач распознавания способен решать персептрон?
6.12. Как при известном расположении распознаваемых классов, корректно сформировать матрицу целевых значений выходов персептрона? Приведите пример.
6.13. Какие встроенные функции модуля Neuralnetwork 3.0 используются для обучения и моделирования персептрона? Приведите примеры их вызова.
6.14. Объясните алгоритм, реализуемый функцией ann_PERCEPTRON.

Приложение А. Код функции ann_PERCEPTRON(P, T)

```
function [w,b] = ann_PERCEPTRON(P, T)
//инициализация матрицы весов и вектора смещений
w = rand(size(T,1),size(P,1));
b = rand(size(T,1),1);
iter = %t; // присвоение флагу iter=True, контроль эпох
itercnt = 0; // счетчик итераций
while iter == %t  // цикл по эпохам, пока iter=True
no_err = 0          // счетчик безошибочных классификаций
    for cnt = 1:size(P,2) // цикл по всем входным примерам (одна эпоха)
        e = T(:,cnt) - ann_hardlim_activ(w*P(:,cnt)+b);  // вектор ошибки текущего примера
        w = (w + e*P(:,cnt)'); // обучение матрицы весов
        b = b + e; // обучение вектора смещений
        if sum(e) == 0 then  // сумма текущих ошибок равна нулю
            no_err = no_err + 1;  // число безошибочных классификаций
        end
        if  no_err == size(P,2) then //число безошиб. классификаций = числу примеров
            iter = %f; // сбрасываем флаг цикла эпох
        end
    end
    itercnt = itercnt + 1; // счетчик эпох
    disp('Epoch: ' + string(itercnt));
end
endfunction
```

ЛАБОРАТОРНАЯ РАБОТА № 4
ИССЛЕДОВАНИЕ АДАПТИВНОГО ЛИНЕЙНОГО ЭЛЕМЕНТА

1 Цель работы

Углубление теоретических знаний в области разработки нейронных сетей с линейной активационной функцией, исследование свойств квадратичной целевой функции и LMS-алгоритма обучения, приобретение практических навыков обучения однослойной сети линейных адаптивных элементов при решении задачи классификации и адаптивной фильтрации.

2 Основные теоретические положения

2.1 Структура однослойной сети из адаптивных линейных элементов

Адаптивный линейный элемент (АЛЭ) (ADALINE - Adaptive Linear Element) представляет собой нейрон с линейной функцией преобразования purelin. Общая структурная схема однослойной сети из АЛЭ изображена на рисунке 4.1.

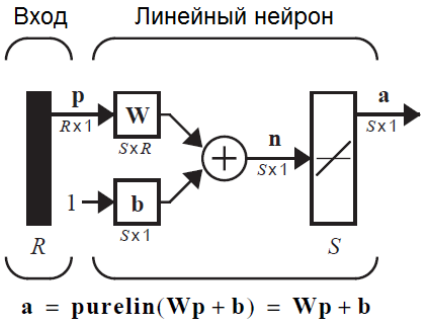


Рисунок 4.1 — Схема однослойной сети из адаптивных линейных элементов

Выходное значение отдельного нейрона сети вычисляется в соответствии с выражением

$$a_i = purelin(n_i) = purelin({}_i\mathbf{w}^T\mathbf{p} + b_i) = {}_i\mathbf{w}^T\mathbf{p} + b_i \qquad (4.1)$$

2.2 Классификация с использованием АЛЭ

Рассмотрим АЛЭ с 2-мя входами и 1 выходом (рисунок 4.2.).

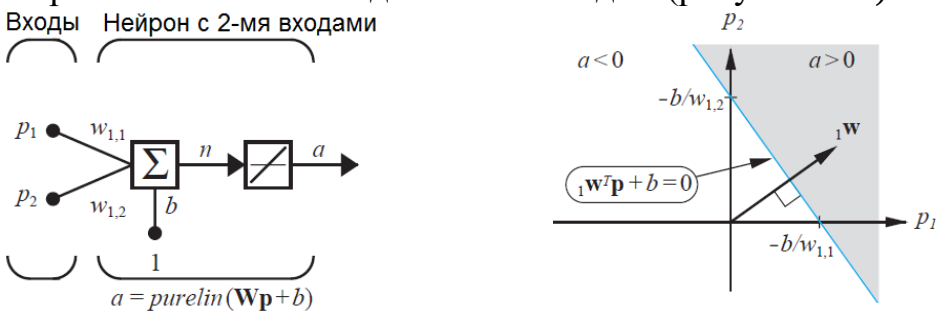


Рисунок 4.2 — АЛЭ и его граница решения

С учетом только 2-х входов выходной сигнал нейрона будет равен

$$a = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b\,. \tag{4.2}$$

Граница решения определится из условия

$$n = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = 0\,. \tag{4.3}$$

АЛЭ разделяет входное пространство на две области, где $a>0$ и $a<0$ (см. рисунок 4.2). Таким образом, АЛЭ (как и персептрон) может использоваться для классификации **линейно сепарабельных** объектов**.**

2.3 Целевая функция нейронной сети

Задача обучения нейронных сетей сводится к поиску весов и смещений, которые обеспечивают необходимую эффективность сети. Количественная мера эффективности сети определяется **целевой функцией** или **функцией потерь**. Функция потерь зависит от параметров сети и имеет малые значения, когда сеть функционирует хорошо, и большие значения, когда сеть функционирует плохо. Поэтому целью обучения является поиск параметров сети, которые обеспечивают минимум функции потерь.

Пусть $F(\mathbf{x})$ непрерывная функция потерь, где $\mathbf{x} = [x_1,x_2,\ldots, x_n]^T$ — вектор столбец параметров сети в n-мерном евклидовом пространстве. Необходимо найти минимум $F(\mathbf{x})$ по **x.** В точке $\mathbf{x}=\mathbf{x}^*$ будет минимум целевой функции, если градиент функции (вектор первых частных производных $F(\mathbf{x})$ по **x**) будет равен нулю в этой точке (условие *первого порядка*)

$$\nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} = \mathbf{0}\,. \tag{4.4}$$

Точки **x**, удовлетворяющие этому условию, называются *стационарными*. В точке x* будет существовать строгий минимум, если (*условие 2-го порядка)*

$$\Delta\mathbf{x}^T\nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\Delta\mathbf{x} > 0\,, \tag{4.4}$$

где $\nabla^2 F(\mathbf{x})$– квадратная *матрица Гессе* функции потерь (матрица вторых частных производных F(**x**) по **x,** называемая гессианом**),** $\mathbf{\Delta x=x - x^*}$ – окрестность точки минимума. Чтобы условие (4.4) выполнялось для любых $\|\mathbf{\Delta x}\|>\mathbf{0}$ достаточно, чтобы матрица Гессе была *положительно определена.* Для того чтобы в точке $\mathbf{x^*}$ находился минимум (строгий или слабый) достаточно, чтобы матрица Гессе была *положительно полуопределена*. Напомним, что матрица положительно определена, если все собственные числа λ матрицы положительны. Если все собственные числа матрицы не отрицательны, то матрица положительно полуопределена.

При обучении нейронных сетей часто используют *квадратичную целевую функцию*. Общая форма квадратичной целевой функции

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c\,, \tag{4.5}$$

где A – симметричная матрица. Градиент и гессиан квадратичной целевой функции $F(\mathbf{x})$ соответственно равны:

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d}\,, \quad \nabla^2 F(\mathbf{x}) = \mathbf{A}\,. \tag{4.6}$$

2.4 Алгоритм наискорейшего спуска

Цель оптимизации заключается в поиске вектора параметров сети **x***, который минимизирует функцию $F(\mathbf{x})$. Для поиска **x*** применяют алгоритмы последовательного приближения – *итеративные алгоритмы*. В соответствии с такими алгоритмами поиск начинается с начального значения $\mathbf{x}_0$ и на каждом шаге k вектор параметров корректируют в соответствии с формулой

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{4.7}$$

или

$$\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k\,, \tag{4.8}$$

где $\mathbf{p}_k$ – вектор, определяющий направление поиска; α_k – скорость обучения, определяющая длину шага $\Delta\mathbf{x}_k$.

Итеративные алгоритмы оптимизации отличаются выбором вектора направления поиска $\mathbf{p}_k$ и способами вычисления значений скорости обучения. Так, в *алгоритме наискорейшего спуска* (**SDA**– **steepest descent algorithm**) направление поиска обратно направлению вектора градиента, т.е. $\mathbf{p}_k = -\mathbf{g}_k$, где $\mathbf{g}_k = \nabla F(\mathbf{x})$ – значение вектора градиента целевой функции на k–ой итерации алгоритма. Соответственно процедура SDA определяется выражением:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k\,. \tag{4.9}$$

Если целевая функция $F(\mathbf{x})$ является увадратичной и имеет сильный минимум, то все собственные значения матрицы Гессе **A** будут положительными и условие схождения алгоритма SDA запишется в виде неравенства:

$$\alpha < \frac{2}{\lambda_{max}}\,, \tag{4.10}$$

где λ_{max} — максимальное собственное число матрицы Гессе. Так как собственные числа матрицы Гессе квадратичной целевой функции определяют кривизну поверхности целевой функции, то в соответствии с (4.10) *скорость обучения SDA связана обратно пропорциональной зависимостью с максимальной кривизной целевой функции.*

2.3 Решение минимума среднего квадрата ошибки

Рассмотрим АЛЭ с одним выходом. Для упрощения анализа введем следующие обозначения: $\mathbf{x}^T = [_1\mathbf{w}\ \mathbf{b}]$ и $\mathbf{z}^T = [\mathbf{p}\ \mathbf{1}]$. Тогда выходной сигнал АЛЭ можно представить в виде скалярного произведения $a = \mathbf{x}^T\mathbf{z}$.

В ходе обучения с учителем параметры (веса и смещения) АЛЭ корректируются таким образом, чтобы выходной сигнал *a* имел наилучшее приближение к желаемой реакции *t*. Для этого вычисляют ошибку приближения *e*=(*t*-*a*) и минимизирует **средний квадрат ошибки** (СКО, MSE – mean square error). С учетом введенных обозначений *целевая функция F*(**x**), заданная в виде СКО, будет равна

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2], \quad (4.11)$$

где *E* – символ математического ожидания. Раскроем (4.11):

$$F(\mathbf{x}) = E[t^2 - 2t\mathbf{x}^T\mathbf{z} + \mathbf{x}^T\mathbf{z}\mathbf{z}^T\mathbf{x}] = E[t^2] - 2\mathbf{x}^T E[t\mathbf{z}] + \mathbf{x}^T E[\mathbf{z}\mathbf{z}^T]\mathbf{x}.$$

СКО целевая функция может быть переписана в квадратичной форме

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{C}\,\mathbf{x}, \text{ где } c = E[t^2], \mathbf{h} = E[t\mathbf{z}], \mathbf{C} = E[\mathbf{z}\mathbf{z}^T]. \quad (4.12)$$

Здесь **h** – вектор кросс-корреляции, **C** – корреляционная матрица.

Сравним целевую функцию (4.12) с квадратичной целевой функцией общего вида (4.5). Получим, что

$$\mathbf{d}=-2\mathbf{h},\ \mathbf{A}=2\mathbf{C}, \quad (4.13)$$

т.е. матрица Гессе целевой функции (4.12) равна **2C**. Точка минимума целевой функции *F*(**x**) определится из условия 1-го порядка (4.4):

$$\nabla F(\mathbf{x}) = \nabla\left(c + \mathbf{d}^T\mathbf{x} + \tfrac{1}{2}\mathbf{x}^T\mathbf{A}\,\mathbf{x}\right) = \mathbf{d} + \mathbf{A}\,\mathbf{x} = -2\mathbf{h} + 2\mathbf{C}\mathbf{x} = \mathbf{0}. \quad (4.14)$$

Если корреляционная матрица положительно определена, то будет существовать единственная точка минимума целевой функции (4.12)

$$\mathbf{x}^*=\mathbf{C}^{-1}\,\mathbf{h}. \quad (4.15)$$

Решение (4.15), позволяющее вычислять параметры АЛЭ через матрицу **C** и вектор **h**, называют **решением минимума СКО** (**оптимальным решением Винера**). На практике точные значения **h** и **C** обычно неизвестны. Однако, если входные данные, представляемые вектором **p**, соответствуют стационарному эргодическому процессу, то, используя усреднение по времени, можно оценить **h** и **C**. Если на *k*-м шаге имеются оценки матрицы **C** и вектора **h**, обозначаемые как $\mathbf{C}_k$ и $\mathbf{h}_k$, то для поиска винеровского решения можно использовать алгоритм наискорейшего спуска. Подставив значение градиента из (4.14) в выражение (4.9), можно получить алгоритм наискорейшего спуска, сходящийся к решению (4.15):

$$\mathbf{g}_k = -2\mathbf{h}_k + 2\mathbf{C}_k\mathbf{x}_k, \quad (4.16)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \boldsymbol{\alpha}_k\,\mathbf{g}_k, \quad (4.17)$$

Отметим, что существование решения (4.15) целиком зависит от **C** . Следовательно, характеристики входного вектора **p** определяют возможность существования единственного решения.

2.4. LMS алгоритм

На практике вычислять оценки матрицы **C** и вектора **h** затратно. Поэтому выполняют аппроксимацию алгоритма наискорейшего спуска, в ходе которой используют приближенные оценки градиента. Идея заключается в аппроксимации среднего квадрата ошибки (4.11) более простым выражением

$$\hat{F}(\mathbf{x}) = (t(k) - a(k))^2 = e^2(k), \tag{4.18}$$

в котором вместо математического ожидания *E* квадрата ошибки используется просто квадрат мгновенной ошибки. Тогда оценка градиента, называемая в этом случае *стохастическим градиентом*, будет равна

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k). \tag{4.19}$$

Так как $a = \mathbf{x}^T \mathbf{z}$ и $e=(t-a)$, то стохастический градиент будет равен

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k) = -2e(k)\mathbf{z}(k). \tag{4.20}$$

Подставив стохастический градиент (4.19) в алгоритм SDA, получим получим **LMS-алгоритм** (LMS- least mean square, алгоритм наименьшего среднего квадрата, также называемый **правилом обучения Уидроу-Хоффа)**

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k), \tag{4.21}$$

или

$${}_1\mathbf{w}(k+1) = {}_1\mathbf{w}(k) + 2\alpha e(k)\mathbf{p}(k), \quad b(k+1) = b(k) + 2\alpha e(k). \tag{4.22}$$

Если используется слой из АЛЭ, то LMS-алгоритм записывается в матричной форме:

$$\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha \mathbf{e}(k)\mathbf{p}^T(k), \quad \mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha \mathbf{e}(k). \tag{4.21}$$

Поскольку в выражении (4.21) используются неточные оценки градиента, то в процессе обучения возникает шум, в результате чего траектория движения вектора параметров $\mathbf{x}_k$ к вектору **x*** в LMS-алгоритме не совпадает с траекторией движения в алгоритме наискорейшего спуска. LMS-алгоритм привлекает своей простотой.

Условие сходимости SDA (4.9) определялось значением собственных чисел матрицы Гессе **A** квадратичной целевой функции. Для LMS справедливо, что **A=2C** . Поэтому по аналогии можно записать условие сходимости LMS-алгоритма

$$\alpha < 1/\lambda_i \quad \text{для всех} \quad i \tag{4.22}$$

или

$$0 < \alpha < 1/\lambda_{max} \tag{4.23}$$

где λ_i - собственные числа корреляционной матрицы **C**. Если это условие выполняется, то можно показать, что LMS алгоритм, обрабатывающий на каждой итерации один входной вектор, сходится к решению, *совпадающему с решением минимума СКО* (4.15) .

2.5 Адаптивная фильтрация

На основе АЛЭ реализуют адаптивные фильтры (рисунок 4.3). Для этого на входе АЛЭ размещают линию задержки (tapped delay line – TDL). Один элемент линии задержки, обозначенный на рисунке 4.3 буквой D, представляет собой регистр памяти, который запоминает входное значение на время, равное шагу дискретизации. В этом случае вектор **p**, поступающий на вход АЛЭ, соответствует совокупности значений на выходе линии задержки – $y(k)$,$y(k\text{-}1)$,…,$y(k\text{-}R\text{+}1)$.

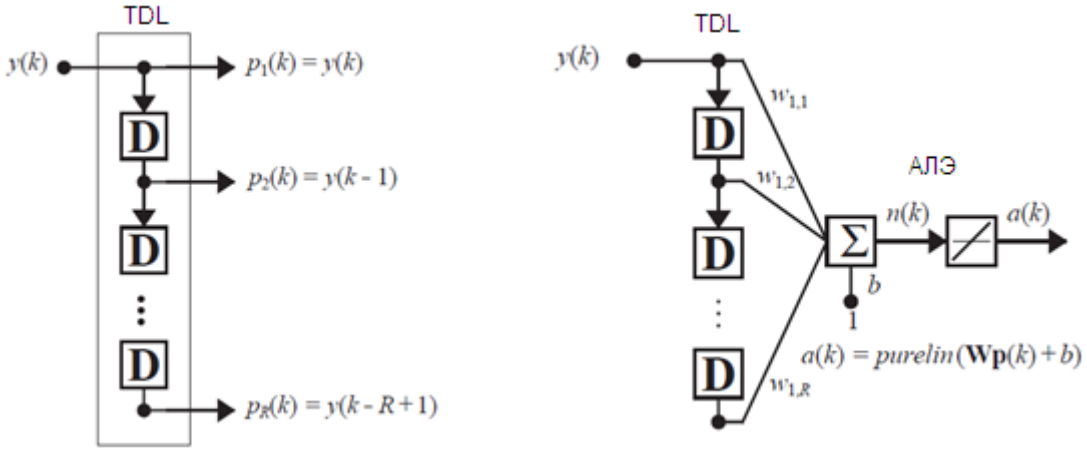


Рисунок 4.3 – Схема адаптивного фильтра на основе АЛЭ

С учетом принятых обозначений выход адаптивного фильтра запишется в виде:

$$a(k) = purelin(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1,i} y(k - i + 1) + b . \qquad \ldots\ldots(4.24)$$

Уравнение (4.24) соответствует фильтру с конечной импульсной характеристикой (КИХ), который также называют *трансверсальным фильтром.*

Адаптивные фильтры используются для решения задач моделирования, идентификации, подавления шумов, компенсации эхо-сигналов, построения эквалайзеров каналов связи, кодирования речи, управления адаптивными антенными решетками и др. Рассмотрим применение адаптивного фильтра для предсказания сигналов. На рисунке 4.4 изображена схема адаптивного предсказателя, построенного на основе АЛЭ, обеспечивающего предсказание текущего отсчета входного сигнала $y(k)$ по 2-м предыдущим отсчетам: $y(k\text{-}1)$ и $y(k\text{-}2)$.

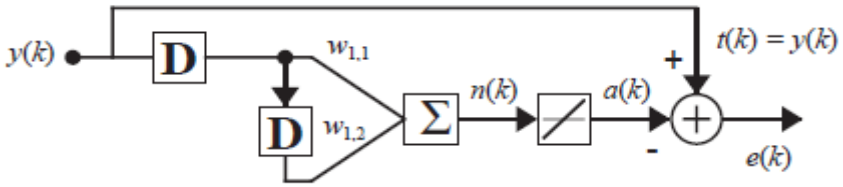


Рисунок 4.4 – Адаптивный предсказатель

Отметим, что адаптивный линейный предсказатель в ходе обучения обеспечивает поиск параметров, которые минимизируют СКО в виде целевой функции (4.12). Так как желаемое значение на выходе предсказателя должно совпадать с текущим входным значением, т.е. t(k)=y(k) и входной вектор предсказателя равен

$$\mathbf{z}(k)=\mathbf{p}(\mathrm{k})=[y(k\text{-}1)\ y(k\text{-}2)]^{\mathrm{T}}\ , \qquad (4.25)$$

то составляющие целевой функции (4.12) запишутся в виде:

$$\mathbf{C} = E[\ \mathbf{z}\mathbf{z}^{\mathbf{T}}\ ] = E\begin{bmatrix} y^2(k-1) & y(k-1)y(k-2) \\ y(k-1)y(k-2) & y^2(k-2) \end{bmatrix}, \qquad (4.26)$$

$$\mathbf{h} = E[\ t\mathbf{z}\ ] = E\begin{bmatrix} y(k)y(k-1) \\ y(k)y(k-2) \end{bmatrix},\ c = E[t^2(k)] = E[y^2(k)]. \qquad (4.27)$$

При известных значениях матрицы **C** и вектора **h** оптимальные веса предсказателя определятся в виде решения минимума СКО (4.15). Например, пусть

$$\mathbf{C} = \begin{bmatrix} 0.54 & 0.45 \\ 0.45 & 0.54 \end{bmatrix}\ , \qquad \mathbf{h} = \begin{bmatrix} 0.54 \\ 0.45 \end{bmatrix}.$$

Тогда в соответствии с (4.15) оптимальный вектор параметров предсказателя будет равен $\mathbf{x}^*=[w_{1,1}\ w_{1,2}]=[1\ 0]$.

При исследовании свойств алгоритмов, базирующихся на СКО целевой функции полезно использовать графики *линий контуров постоянного уровня e=const*. Эти графики для поверхности квадратичной целевой функции, являющейся параболоидом, представляют собой гиперэллипсы (рисунок 4.5). Главные оси этих гиперэллипсов ориентированы в направлении собственных векторов матрицы **C**. Чем больше соответствующее собственное число матрицы **C**, тем больше градиент $F(\mathbf{x})$ вдоль соответствующей оси эллипса и, соответственно, линии контуров равных уровней располагаются ближе друг к другу. Для рассматриваемого примера **C** собственные числа равны λ_1=0,17 и λ_2=1,97. Соответственно, эллипс вытянут вдоль собственного вектора с меньшим собственным значением, т.е λ_1=0,17.

Так как вычисление оптимальных параметров предсказателя в соответствии с (4.15) требует вычисления обратной матрицы, то на практике поиск оптимальных параметров выполняют на основе LMS-алгоритма. При этом скорость схождения, точность и устойчивость LMS-алгоритма сильно зависит от скорости обучения α. Максимальное устойчивое значение α определяется на основе соотношения (4.23). Для рисунка 4.5 α_{max}=1/ λ_2=0,507. На практике при использовании адаптивных фильтров определять собственные числа **C** не представляется

возможным. Однако, учитывая свойство корреляционной матрицы, согласно которому след корреляционной матрицы равен сумме её собственных чисел, т.е. $tr(\mathbf{C}) = \sum_{i=1}^{N} c_{ii} = \sum_{i=1}^{N} \lambda_i$, можно заключить, что для СКО целевой функции $\lambda_{max} \leq tr(\mathbf{C})$. С учетом этого соотношения условие (4.23) можно представить в виде более жесткого условия, *гарантирующего* устойчивую работу LMS-алгоритма для трансверсального фильтра:

$$\alpha \leq \frac{1}{3\sum_{i=1}^{N} c_{ii}} = \frac{1}{3N\sigma_y^2} \qquad (4.28)$$

где σ_y^2 — дисперсия входного сигнала адаптивного фильтра. В соответствии с (4.28) для рассматриваемого примера должно быть $\alpha \leq 1/(3 * 2 * 0{,}54) = 0{,}31$.

Так как LMS-алгоритм аппроксимирует алгоритм наискорейшего спуска, то при малых скоростях обучения α траектория движения вектора параметров LMS-алгоритма будет менее хаотична и в среднем будет соответствовать движению вектора параметров алгоритма наискорейшего спуска, которая направлена перпендикулярно линиям равных контуров, как показано на рисунке 4.5. Однако при значениях α близких к α_{max} продвижение к некоторому малому уровню значений целевой функции происходит быстрее, несмотря на хаотичность. Кружок в центре рисунка 4.5. соответствует решению минимума СКО (4.15). Видно, что при выбранных значениях α решение, получаемое с помощью LMS-алгоритма, приближается к решению минимума СКО.

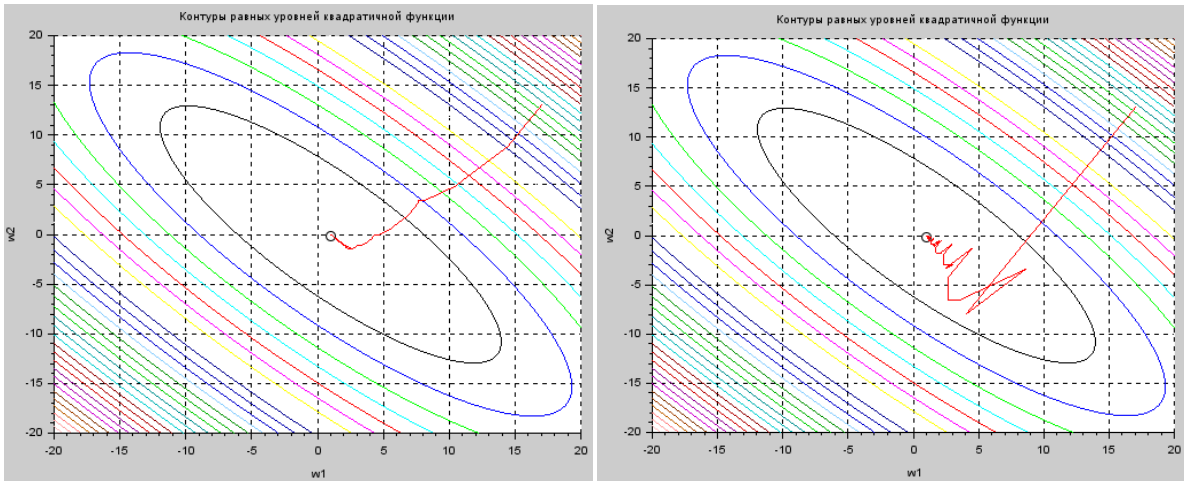


Рисунок 4.5 – Траектория движения вектора параметров предсказателя при использовании LMS-алгоритма: слева $\alpha = 0{,}1$; справа $\alpha = 0{,}507$

3. Варианты заданий и программа работы

3.1. Повторите теоретический материал, относящийся к архитектуре и правилам обучения адаптивного линейного элемента [1, 4, 5].

3.2. Даны четыре класса, каждый из которых представлен 2-мя примерами (столбцами матрицы **P**), указанными в таблице 3.2 к лабораторной работе №3 (*используйте свой вариант*). Необходимо:

- разработать структурную схему классификатора на основе АЛЭ, распознающего эти 4 класса;

- выполнить предварительный анализ задачи, изобразив точки четырех классов и построив графически возможные границы решений;

- задать ту же целевую матрицу **T,** которая использовалась при выполнении задания 3.3 в лабораторной работе №3**,** заменив все нули на -1;

- полагая, что все входные векторы **p** равновероятны, написать программу, вычисляющую корреляционную матрицу **C**, собственные числа гессиана целевой функции **A=2C** и максимальное устойчивое значение параметра α_{max} LMS-алгоритма;

- изучить встроенные функции ann_ADALINE и ann_ADALINE_online пакета NeuralNetworks 3.0, реализующие блочный и последовательный варианты LMS-алгоритма;

- используя указанные функции, написать программу, которая:

  - отображает диаграмму размещения входных точек (примеров) из **P** на плоскости с координатами (*p1*, *p2*);

  - обучает АЛЭ правильному распознаванию входных классов с использованием 2-х указанных функций обучения при разных значениях параметра α;

  - строит кривые обучения - зависимости СКО от номера эпохи для 2-х указанных функций обучения (для этого необходимо модифицировать встроенные функции ann_ADALINE и ann_ADALINE_online);

  - накладывает на диаграмму входных точек данных границы решения после обучения слоя АЛЭ;

  - выполняет тестирование полученного решения для всех заданных входных данных;

- сравнить получаемые границы решения слоя АЛЭ с границами решения персептрона, полученными в лабораторной работе №3, обратив внимание на равноудаленность границ от точек соседних классов для случая слоя АЛЭ;

- сравнить кривые обучения слоя АЛЭ двумя модифицированными функциями ann_ADALINE и ann_ADALINE_online для случая, когда параметр α LMS-алгоритма значительно меньше α_{max} и когда он близок к α_{max} .

3.3. Исследуйте адаптивный линейный предсказатель (рисунок 4.4):

- сгенерируйте входной $y(k)$ и желаемый $t(k)=y(k)$ сигналы адаптивного предсказателя. При этом параметры генератора выбирайте в соответствии с вариантом из таблицы 4.1. Для генерации используйте следующий программный код:

```
//Значения параметров генератора L, F, Fc выбирайте из таблицы 4.1;
//Параметры:
L=     ;
F=     ;
Fc=    ;
//Генератор полигармонического входного сигнала;
td=1/(20*F);
t=0:td:2/F;
fi=(2*%pi*F).*t;
```

```
X=0;
for i=1:L
   X=X+(Fc)/(Fc+2*%pi*F*i)*sin(fi*i);
end
Y=X(2:$)  // Y задержен на такт - вход предсказателя
T=X(1:$-1); // желаемый выход предсказателя
```

Таблица 4.1 – Параметры генератора входного сигнала

Вариант	Число составляющих L	Основная частота F	Частота среза Fc
1	5	0.01	F*5
2	10	0.01	F*6
3	15	0.01	F*7
4	20	0.01	F*8
5	25	0.01	F*9
6	5	0.02	F*5
7	10	0.02	F*6
8	15	0.02	F*7
9	20	0.02	F*8
10	25	0.02	F*9
11	5	0.04	F*5
12	10	0.04	F*6
13	15	0.04	F*7
14	20	0.04	F*8
15	25	0.04	F*9

- запишите развернутые выражения для всех элементов (**C**,**h**,c) целевой функции предсказателя, заданной в виде СКО (4.12);
- вычислите конкретные значения матрицы **C**, вектора **h** и константы *c* для сгенерированного входного сигнала $y(k)$;
- вычислите собственные значения и собственные векторы матрицы Гессе целевой функции предсказателя, точку минимума целевой функции, постройте линии контуров равных уровней целевой функции;
- вычислите максимальное устойчивое значение скорости обучения α_{max} для LMS-алгоритма;
- напишите программу, обучающую предсказатель с использованием встроенной функции ann_ADALINE_predict пакета NeuralNetworks 3.0;
- модифицируйте функцию ann_ADALINE_predict таким образом, чтобы по результатам её работы можно было построить кривую обучения предсказателя и траекторию движения вектора параметров предсказателя на диаграмме контуров равных уровней;
- постройте в одном графическом окне графики входного процесса и его предсказанных значений, кривую обучения, траекторию движения вектора параметров на диаграмме контуров;

- убедитесь, что алгоритм обучения сходится, если $\alpha<\alpha_{max}$ и нестабилен когда $\alpha>\alpha_{max}$;
- убедитесь, что при малых α траектория движения вектора параметров при использовании LMS алгоритма аппроксимирует в среднем траекторию движения вектора параметров алгоритма наискорейшего спуска.

3.4. Подготовьте и защитите отчет по работе.

4. Методические рекомендации по выполнению работы

4.1. Модуль NeuralNetwork 3.0 пакета Scilab содержит встроенные функции для обучения и моделирования АЛЭ:

[w,b]= ann_ADALINE(P, T, alpha, itermax, initfunc) – функция обучения АЛЭ, реализующая блочный (batch) LMS-алгоритм;
[w,b]= ann_ADALINE_online(P, T, alpha, itermax, initfunc) – функция обучения АЛЭ, реализующая последовательный (адаптивный) LMS-алгоритм;
y= ann_ ADALINE _run(P,w,b) – функция моделирования слоя АЛЭ.

Познакомьтесь с описанием параметров этих функций в приложениях А, Б, Г.

4.2. При выполнении задания 3.2 необходимо сначала сформировать матрицу **T** целевых выходов слоя из АЛЭ. Используйте для этого матрицу целевых выходов, построенную в лабораторной работе №3, заменив в ней все нули на -1. Например, для рисунка 3.5 матрица **T** целевых значений выходов классификатора, реализованного в виде слоя из АЛЭ, будет равна:

T= [-1 -1 -1 -1 1 1 1 1;
-1 -1 1 1 -1 -1 1 1].

Так как входные векторы **p** равновероятны, то для вычисления корреляционной матрицы **C** напишите код, который реализует простую формулу

$$\mathbf{C} = 1/Q \sum_{q=1}^{Q} \mathbf{z}_q \mathbf{z}_q^T ,$$

где $\boldsymbol{z}_q^T = [\mathbf{p}_q \;\; 1]$ – расширенный входной вектор слоя АЛЭ (здесь 1 соответствует входу с весом, равным смещению b), Q – число входных векторов **p.**

Для вычисления собственных значений гессиана **A=2C** целевой функции, заданной в виде СКО, используйте вызов встроенной функции Scilab:

evals=spec(2*C)

Максимальное устойчивое значение параметра α_{max} LMS-алгоритма при известных собственных значениях определится из условия (4.10).

Для написания программы, которая выполняет обучение АЛЭ распознаванию входных классов в двух переименуйте и модифицируйте встроенные функции обучения ann_ADALINE и ann_ADALINE_online. Для переименования просто добавьте к имени указанных функций символ “1”. Переименованные и модифицированные функции должны вызываться следующим образом:

```
[w1,b1,mse1] = ann_ADALINE1(P,T,alpha,maxiter,'rand');
[w2,b2,mse2] = ann_ADALINE1_online(P,T,alpha,maxiter,'rand');
```

Функции ann_ADALINE1 и ann_ADALINE1_online модифицируются таким образом, чтобы можно было после обучения построить кривые обучения – зависимости ошибки СКО от номера эпохи обучения. Для этого указанные функции, кроме параметров обученного слоя АЛЭ, должны возвращать массивы значений средних квадратов ошибок, которые в примерах вызовов функций выше обозначены как mse1 и mse2. Для модификации кода функции ann_ADALINE используйте следующую схему:

```
function [w, b, mse1]=ann_ADALINE1(P, T, alpha, itermax, initfunc)
…
itercnt = 0;
mse1=zeros(1,itermax);
while itercnt < itermax
        …
          itercnt = itercnt + 1;
          mse1(itercnt) = mean(e.^2);
          //disp('Epoch:….
end
endfunction
```

Модификация кода функции ann_ADALINE1_online выполняется аналогично:

```
function [w, b, mse2]=ann_ADALINE1_online(P, T, alpha, itermax, initfunc)
…
itercnt = 0;
mse2=zeros(1,itermax);
while itercnt < itermax
   for cnt = 1:size(P,2)
        ….
       e_all(cnt) = e.^2
   end
   itercnt = itercnt + 1;
   mse2(itercnt)=mean(e_all);
  //disp('Epoch:…
end
endfunction
```

В рассматриваемые функции добавляются операторы, которые выделены красным цветом. Эти операторы обеспечивают вычисление СКО на каждой итерации и запоминание вычисленных значений в переменных mse1 и mse2. Операторы отображения *disp* внутри функций рекомендуется исключить для уменьшения объемов данных, выводимых на экран.

Сохраните переименованные функции в файлах ann_ADALINE1.sci и ann_ADALINE1_online.sci. Для загрузки кода модифицированных функций в Scilab не забудьте в начале своей программы разместить операторы

```
exec('ann_ADALINE1_online.sci') // загрузка функции последовательного обучения
exec('ann_ADALINE1.sci')        // загрузка функции блочного обучения
```

Теперь, используя значения матриц **P**, **T** и вычисленное значение α_{max}, можно обучить слой АЛЭ, вызвав модифицированные функции обучения. По результатам обучения следует построить графики кривых обучения – зависимости значений mse1 и mse2 от номера итерации:

```
x_axis=1:maxiter;
plot(x_axis,mse1(x_axis),'r',x_axis,mse2(x_axis),'g');
xtitle('Средний квадрат ошибки', 'Эпоха','СКО');
```

В качестве примера на рисунке 4.6 изображены кривые обучения при разных значениях скорости обучения α: когда α мало и когда близко к максимально допустимому значению (для рассматриваемого примера α_{max} =0,1916) .

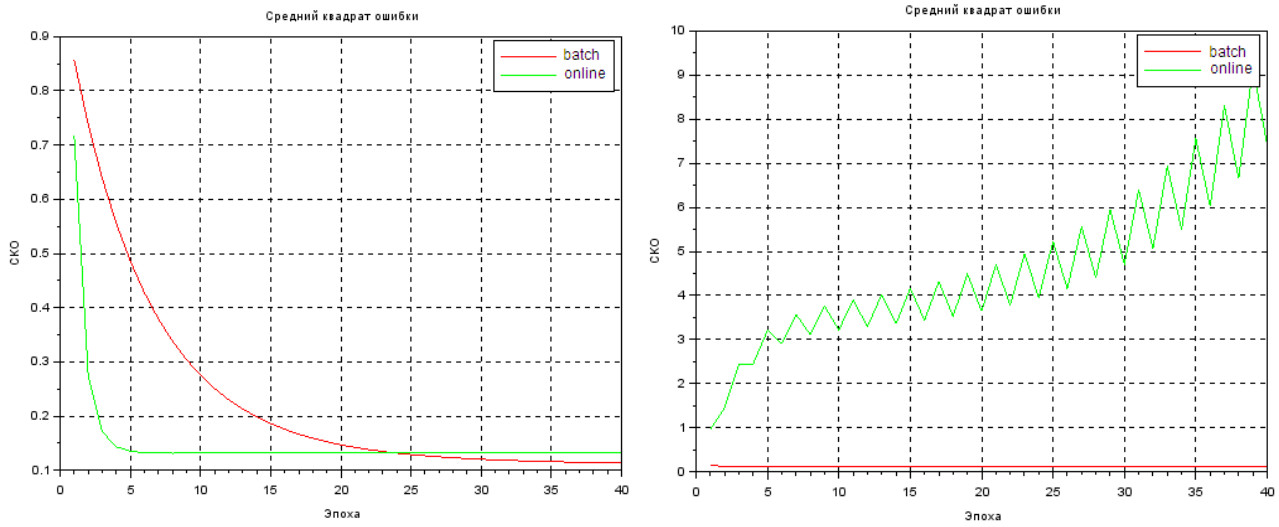


Рисунок 4.6 – Кривые обучения: слева α=0,02; справа α=0,1916

Как следует из рисунка 4.6, блочный LMS-алгоритм при значениях α, обеспечивающих сходимость алгоритма, сходится в начале медленнее, но в итоге дает более точное решение. Последовательный LMS-алгоритм при малых α обучается быстрее, но при значении $\alpha = \alpha_{max}$ теряет устойчивость.

Для отображения диаграммы расположения точек классов и границ полученных решений (рисунок 4.7) используйте код, разработанный при выполнении лабораторной работы №3.

Сравните полученное решение с решением, изображенным на рисунке 3.5. Обратите внимание, что LMS-алгоритм находит границы решения, равноудаленные от центров соседних классов, и получаемое решение не зависит от способа инициализации весов слоя АЛЭ. В то время как однослойный персептрон формирует различные границы решения при разных начальных значениях весов.

4.3. В задании 3.3 требуется обучить линейный предсказатель значений стационарного сигнала $y(k)$ по двум его предыдущим значениям. Соответствующая схема предсказателя с двумя элементами задержки D=2 изображена на ри-

сунке 4.4. В соответствии с заданием вектор отсчетов входного сигнала **Y** и вектор целевых значений **T** предсказателя генерируются с помощью программного кода, приведенного в п.3.3.

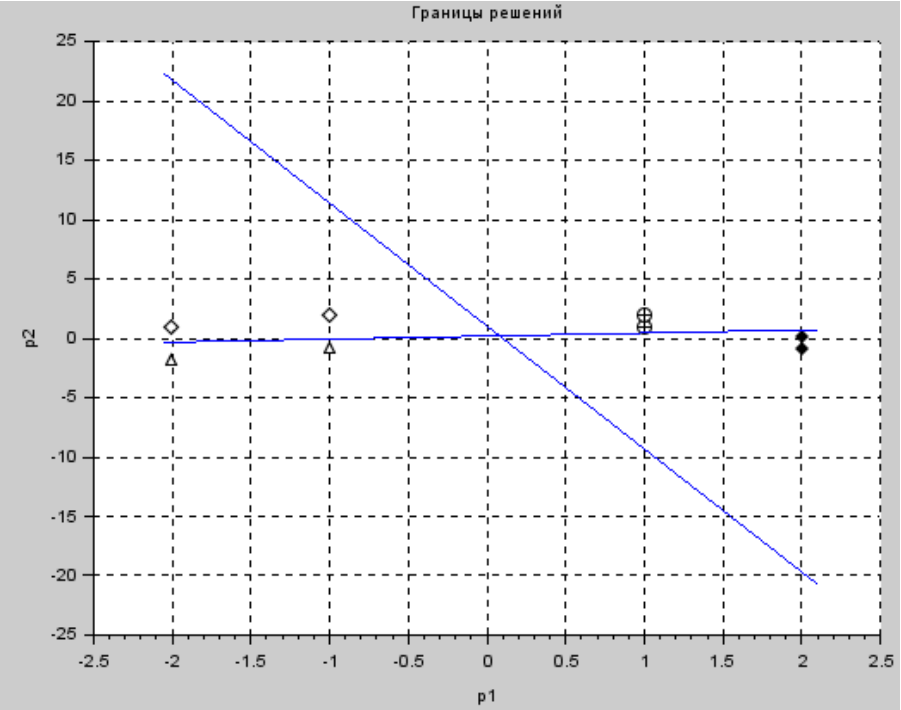


Рисунок 4.7 – Границы решений слоя АЛЭ (блочный LMS-алгоритм)

На первом этапе выполнения задания необходимо вычислить все элементы (**C**,**h**,c) целевой функции предсказателя, заданной в виде СКО (4.12). Для этого следует воспользоваться общими формулами (4.26)-(4.27). Чтобы применить эти формулы, нужно сначала получить значения вектора **p** , элементы которого соответствуют выходам линии задержки (см. рисунок 4.3). Для формирования матрицы **P**, каждый столбец которой равен очередному входному вектору **p** АЛЭ, воспользуйтесь следующим кодом:

```
D=2;            // кол-во элементов задержки предсказателя
P = [];
for cnt = 1:D // для каждого выхода линии задержки
   //формируем строки матрицы P из отсчетов Y
   // очередная строка P – сдвинутая на один отсчет копия предыдущей строки
    P = [P; Y(cnt:$-D+cnt-1)];
end
//формируем вектор целевых значений с длиной, равной длине строки из P
T1 = T(1:$-D+1);
```

Здесь индекс $ означает обращение к последнему элементу массива, а индекс $-D – обращению к элементу с номером D с конца массива.

Теперь, используя формулы (4.26)-(4.27), можно вычислить значения корреляционной матрицы **C**, вектора кросс-корреляции **h** и константы *c***.** Фрагмент кода, обеспечивающий эти вычисления:

```
Q=size(P,2);          // кол-во обучающих примеров
R=size(P,1);          // кол-во входов АЛЭ
Cor=zeros(R,R);    // корреляционная матрица
c=0;
h=0;
for q=1:Q
   p=P(:,q);          // q-ый входной вектор
   Cor=Cor+(p*p').*1/Q;
   c=c+T1(q)^2*1/Q;
   h=h+T1(q)*p*1/Q;
end
```

Для вычисления собственных векторов и собственных значений матрицы Гессе (A=2*Cor) целевой функции предсказателя следует использовать вызов соответствующей встроенной функции Scilab в форме:

```
[evals, diagevals]=spec(2*Cor).
```

Здесь переменная evals – вектор-столбец из собственных значений матрицы 2*Cor, а diagevals – матрица, столбцы которой представляют собственные векторы матрицы 2*Cor**.** При известных значениях матрицы Cor и вектора h *точка минимума целевой функции* (**x***, xstar) находится в виде решения минимума среднего квадрата ошибки (4.15):

```
xstar=inv(Cor)*h; //решение Винера
```

Для построения контуров равных уровней целевой функции следует сначала реализовать (4.12) в виде программной функции от матрицы Cor, вектора h и элементов вектора x=[w(1) w(2)]:

```
function z=f(w1, w2, c, h, Cor)
   x=[w1;w2];
   z=c-2*x'*h+x'*Cor*x;
endfunction
```

После этого используйте функции feval, surf и contour2d для построения 3D поверхности и линий контуров равных уровней:

```
x=linspace(-20,20,100);   // диапазон по оси x
y=linspace(-20,20,100);  // диапазон по оси y
z=feval(x,y,f); //вычисляем значения высот целевой функции f на сетке x,y

clf(1);
figure(1);
subplot(1,2,1);
surf(x,y,z');  //строим 3D поверхность целевой функции
subplot(1,2,2);
contour2d(x,y,z,30); //отображаем линии контуров равных уровней
xset("fpf"," ")          //подавляет отображение чисел над линиями контуров
```

```
xtitle("Контуры равных уровней квадратичной функции","w1","w2");
xgrid;
plot(xstar(1),xstar(2),'*b'); //отображаем точку решения минимума СКО (4.15)
```

В результате работы приведенного фрагмента кода будут построена 3D поверхность и линии контуров равных уровней (рисунок 4.8., см. ниже).

Для обучения адаптивного предсказателя используйте встроенную функцию ann_ADALINE_predict пакета NeuralNetworks 3.0. Чтобы по результатам её работы можно было построить кривую обучения предсказателя и траекторию движения вектора параметров предсказателя измените имя этой функции и модифицируйте её код следующим образом:

```
function [w,b,y,ee,mse,W] = ann_ADALINE1_predict(Y,T,alpha,itermax,D,initfunc)
…
mse=zeros(1,itermax); // создаем вектор СКО
W=[w]; //матрица, каждая строка которой – вектор весов на очередном шаге
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
   for cnt = 1:size(P,2)  // Цикл по всем обучающим примерам из P (1 эпоха)
      …
      w = (w + 2*alpha*e*P(:,cnt)');
      // b = b + 2*alpha*e;  предсказатель не использует смещение
      // Вычисляем и запоминаем квадраты текущих  ошибок
      e_all(cnt) = e.^2;
      W=[W;w];
   end
   itercnt = itercnt + 1;
   mse(itercnt)=mean(e_all); // вычисляем СКО на шаге и запоминаем
   //disp('Epoch: …
    …
end
endfunction
```

Внесенные изменения выделены красным цветом. Сохраните измененное определение функции в файле ann_ADALINE1_predict.sci и добавьте в начало вашей программы орератор загрузки определения новой функции

```
exec('ann_ADALINE1_predict.sci')
```

Теперь функция обучения будет возвращать вектор ошибок **mse** и матрицу **W** из векторов-строк весовых коэффициентов предсказателя на каждой итерации. Вектор **mse** позволяет построить кривые обучения (см. пример на рисунке 4.6), а матрица **W** – траекторию движения вектора параметров предсказателя, наложив её (траекторию) на линии равных контуров (рисунок 4.8):

```
// Создание и обучение предсказателя по значениям  P и T
// Используем модифицированную функцию предсказателя
[w,b,y,ee,mse, W] = ann_ADALINE1_predict(X,T,0.1*alpha_max,40,D,'ones');
plot2d(W(:,1),W(:,2),5); //отображение траектории движения весов
```

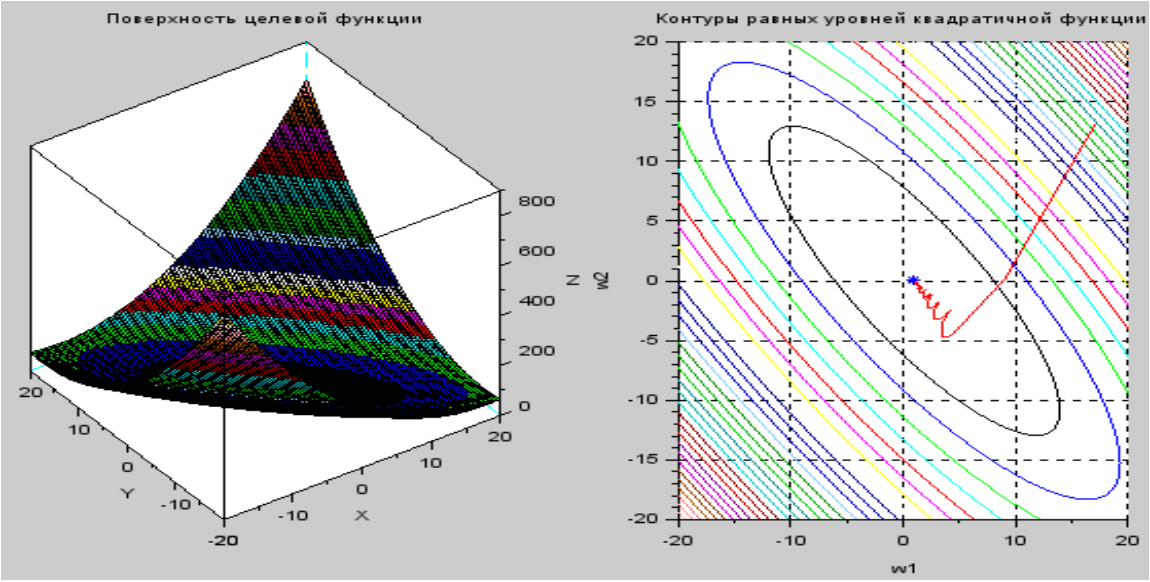


Рисунок 4.8 – Поверхность целевой функции, линии контуров и траектория движения вектора параметров для LMS-алгоритма. (w_0=[17,13]).

5. Содержание отчета

5.1. Цель работы.
5.2. Вариант задания.
5.3. Схема слоя АЛЭ, решающего задачу классификации 4-х классов; вычисленное значение матрицы **C** и её собственных чисел, значения весов и смещений, максимальное значения скорости обучения; диаграмма расположения классов с границами решений, соответствующими обученному слою АЛЭ; графики с кривыми обучения; листинг программы с комментариями.
5.4. Схема адаптивного линейного предсказателя; правила обучения; результаты вычислений: значение корреляционной матрицы **C** и её собственных чисел и векторов, значение максимальной устойчивой скорости обучения; решение минимума СКО; кривые обучения и результирующие значения параметров предсказателя; графики 3D поверхности целевой функции и линий контуров равных уровней, траектории движения вектора параметров; графики входных и выходных сигналов предсказателя, график сигнала ошибки предсказания, листинг программы с комментариями.
5.5. Выводы по результатам исследований.

6. Контрольные вопросы

6.1 Нарисуйте схему однослойной сети из адаптивных линейных элементов, объясните все обозначения, запишите формулу вычисления вектора выходных значений и объясните её.
6.2 Сколько линейно-разделимых классов способна распознать сеть, содержащий S адаптивных линейных нейронов?
6.3. Изобразите АЛЭ с 2-мя входами. Запишите уравнение границы решения.
6.6. Как связана ориентация вектора весов с границей решения АЛЭ?

6.7. Что понимают под целевой функцией сети?
6.8. Сформулируйте условие минимума первого порядка для целевой функции.
6.9. Сформулируйте условие минимума второго порядка для целевой функции.
6.10. Что такое матрица Гессе целевой функции? Каким условиям должна удовлетворять эта матрица, чтобы целевая функция имела строгий минимум?
6.11. Запишите выражение для квадратичной целевой функции в общей форме, найдите градиент и гессиан этой функции.
6.12. Сформулируйте алгоритм наискорейшего спуска. Запишите условие схождения алгоритма.
6.13. Запишите выражение для целевой функции в виде среднего квадрата ошибки. Приведите его к квадратичной форме.
6.14. Запишите условие минимума СКО целевой функции и получите решение минимума СКО.
6.15. Что такое стохастический градиент? Сформулируйте LMS-алгоритм.
6.16. Запишите выражения LMS-алгоритма в матричной форме, сформулируйте условие схождения алгоритма.
6.17 Изобразите схему адаптивного фильтра на основе АЛЭ. Укажите области применения адаптивной фильтрации.
6.18. Изобразите схему адаптивного линейного предсказателя. Запишите выражения для вычисления корреляционной матрицы **C** и вектора кросс-корреляции **h**.
6.19. Сформулируйте гарантирующие условие схождения LMS-алгоритма для трансверсального фильтра.
6.20. Что такое кривая обучения? Как её построить? Какие выводы можно сделать, анализируя кривую обучения?
6.21. Как ориентирована траектория движения вектора параметров на графике линий контуров равных уровней? Чем объясняется хаотичность траектории для LMS-алгоритма?
6.22. Объясните алгоритмы, реализуемые функциями ann_ADALINE, ann_ADALINE_online, ann_ADALINE_predict.

Приложение А. Код функции ann_ADALINE

```
function [w, b]=ann_ADALINE(P, T, alpha, itermax, initfunc)
// Обучение слоя адаптивных линейных нейронов  в блочном режиме
// Длина блока Q  равна числу обучающих примеров
// Примеры вызовов
//    [w,b] = ann_ADALINE(P,T)
//    [w,b] = ann_ADALINE(P,T,alpha,itermax,initfunc)
//Параметры:
//    P : Матрица обучающих примеров (RxQ)
//    T : Матрица целевых выходных значений (SxQ)
//    alpha : Скорость обучения (по умолчанию =0.01)
//    itermaxs : Максимальное число итераций – эпох (по умолчанию – 100)
//    initfunc : Функция инициализации значений w и b: 'rand', 'zeros', или 'ones'
//               По умолчанию используется ‘rand’.
//    w : веса сети (SxR))
//    b : смещения (Sx1)

// 1.===== Обработка списка входных аргументов функции ======
```

```
rhs=argn(2); // argn(2) – возвращает число аргументов  функции
if rhs < 2; error("Должно быть минимум 2 аргумента: P и T"); end
if rhs < 3; alpha = 0.01; end
if rhs < 4; itermax = 100; end
if rhs < 5; w = rand(size(T,1),size(P,1)); b = rand(size(T,1),1); end
//Инициализация весов и смещений
if  rhs == 5 then
   select initfunc
   case 'rand' then
      w = rand(size(T,1),size(P,1));
      b = rand(size(T,1),1);
   case 'zeros' then
      w = zeros(size(T,1),size(P,1));
      b = zeros(size(T,1),1);
   case 'ones' then
      w = ones(size(T,1),size(P,1));
      b = ones(size(T,1),1);
   else
      error("Неверное значение входного аргумента 5");
   end
end

if itermax == []; itermax = 100; end
if alpha == []; alpha = 0.01; end

//2. ====== Реализация правил обучения ==========================
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
   // Вычисляем вектор выхода сети a и вектор ошибки сети e
   // сразу для всего блока данных P (1 эпоха)
   n = w*P + repmat(b,1,size(P,2));
   a = ann_purelin_activ(n);
   e = T - a;
   //Реализуем блочное  правило обучения Уидроу-Хоффа
   // вычисляется среднее обновление для w и b  в пределах блока
   w = w + (2*alpha*e*P')./size(P,2);
   b = b + 2*alpha*mean(e,2);
   // Вычисляем вектор ошибки при обновленных w и b
   n = w*P +  repmat(b,1,size(P,2));
   a = ann_purelin_activ(n);
   e = T - a;
   // Вычисляем значения СКО
   mse = mean(e.^2);
   itercnt = itercnt + 1;
   disp('Эпоха: ' + string(itercnt) + ' СКО: ' + string(mean(mse)));
end
endfunction
```

Приложение Б. Код функции ann_ADALINE_online

```
function [w, b]=ann_ADALINE_online(P, T, alpha, itermax, initfunc)
// Обучение слоя адаптивных линейных нейронов в режиме он-лайн
// Примеры вызовов
//    [w,b] = ann_ADALINE_online(P,T)
//    [w,b] = ann_ADALINE_online (P,T,alpha,itermax,initfunc)
//Параметры:
//    P : Матрица обучающих примеров (RxQ)
//    T : Матрица целевых выходных значений (SxQ)
//    alpha : Скорость обучения (по умолчанию =0.01)
//    itermaxs : Максимальное число итераций – эпох (по умолчанию – 100)
//    initfunc : Функция инициализации значений w и b: 'rand', 'zeros', или 'ones'
//                По умолчанию используется ‘rand’.
```

```
//     w : веса сети (SxR))
//     b : смещения (Sx1)

// 1.===== Обработка списка входных аргументов функции ======
rhs=argn(2); // argn(2) – возвращает число аргументов  функции
if rhs < 2; error("Должно быть минимум 2 аргумента: P и T"); end
if rhs < 3; alpha = 0.01; end
if rhs < 4; itermax = 100; end
if rhs < 5; w = rand(size(T,1),size(P,1)); b = rand(size(T,1),1); end
//Инициализация весов и смещений
if  rhs == 5 then
   select initfunc
   case 'rand' then
      w = rand(size(T,1),size(P,1));
      b = rand(size(T,1),1);
   case 'zeros' then
      w = zeros(size(T,1),size(P,1));
      b = zeros(size(T,1),1);
   case 'ones' then
      w = ones(size(T,1),size(P,1));
      b = ones(size(T,1),1);
   else
      error("Неверное значение входного аргумента 5”);
   end
end

if itermax == []; itermax = 100; end
if alpha == []; alpha = 0.01; end

//2. ====== Реализация правил обучения ==========================
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
   for cnt = 1:size(P,2) // Цикл по всем обучающим примерам  - 1 эпоха
      // Вычисляем вектор текущей ошибки сети e для одного примера P(:,cnt)
      e = T(:,cnt) - ann_purelin_activ(w*P(:,cnt)+b);
     //Реализуем правило обучения Уидроу-Хоффа
      //для каждого примера  P(:,cnt)  вычисляем обновление  w и b
      w = (w + 2*alpha*e*P(:,cnt)');
      b = b + 2*alpha*e;
      // Вычисляем и запоминаем квадраты текущих  ошибок
      e_all(cnt) = e.^2;
   end
   itercnt = itercnt + 1;
   disp('Epoch: ' + string(itercnt) + '  MSE: ' + string(mean(e_all)));
end
endfunction
```

Приложение B. Код функции ann_ADALINE_predict

```
function [w, b, y, ee]=ann_ADALINE_predict(X, T, alpha, itermax, D, initfunc)
// Обучение адаптивного линейного предсказателя с линией задержки
 // в  режиме он-лайн
// Примеры вызовов
//     [w,b] = ann_ADALINE__predict (P,T, alpha)
//     [w,b] = ann_ADALINE__predict (P,T,alpha,itermax, D, initfunc)
//Параметры:
//     X: входные значения предсказателя
//     P : Матрица обучающих примеров (RxQ)
//     T : Матрица целевых выходных значений (SxQ)
//     alpha: Скорость обучения (по умолчанию – 0.01)
//     itermaxs : Максимальное число итераций – эпох (по умолчанию – 100)
//     D – число элементов задержки (по умолчанию – 1)
```

```
//     initfunc : Функция инициализации значений w и b: 'rand', 'zeros', или 'ones'
//                По умолчанию используется ‘rand’.
//     w : веса сети (SxR))
//     b : смещения (Sx1)
//      y : Выход предсказаният
//      ee : Ошибка между  T и  y

// 1.===== Обработка списка входных аргументов функции ======
rhs=argn(2); // argn(2) – возвращает число аргументов  функции
if rhs < 2; error("Должно быть минимум 2 аргумента: P и T"); end
if rhs < 3; alpha = 0.01; end
if rhs < 4; itermax = 100; end
if rhs < 5; D = 1; end
if rhs < 6; w = rand(size(T,1),D); b = rand(size(T,1),1); end

//Инициализация весов и смещений
if  rhs == 6 then
   select initfunc
   case 'rand' then
      w = rand(size(T,1),D);
      b = rand(size(T,1),1);
   case 'zeros' then
      w = zeros(size(T,1),D);
      b = zeros(size(T,1),1);
   case 'ones' then
      w = ones(size(T,1),D);
      b = ones(size(T,1),1);
   else
      error("Неверное значение входного аргумента 5”);
   end
end

if itermax == []; itermax = 100; end
if alpha == []; alpha = 0.01; end
if D == []; D = 1; end

// Формирование выходов линии задержки: реформатирование X в матрицу P
P = [];
for cnt = 1:D // для каждого выхода линии задержки
   //формируем строки  матрицы P из отсчетов X
   // очередная строка P – сдвинутая на один отсчет копия предыдущей строки
   P = [P; X(cnt:$-D+cnt-1)];
end
//формируем вектор целевых значений
T = T(1:$-D+1);

//2. ====== Реализация правил обучения ==========================
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
   for cnt = 1:size(P,2)  // Цикл по всем обучающим примерам из P (1 эпоха)
      n = w*P(:,cnt)+b;  // Сетевая функция АЛЭ
      a = ann_purelin_activ(n);
      e = T(:,cnt) - a;    // Ошибка
      y(cnt) = a;          // Запоминаем выход АЛЭ
      ee(cnt) = e;         // Запоминаем ошибку
      //Реализуем правило обучения Уидроу-Хоффа
      //для каждого примера  P(:,cnt)  вычисляем обновление  w и b
      w = (w + 2*alpha*e*P(:,cnt)');
      b = b + 2*alpha*e;
      // Вычисляем и запоминаем квадраты текущих  ошибок
      e_all(cnt) = e.^2;
   end
   itercnt = itercnt + 1;
```

```
    disp('Epoch: ' + string(itercnt) + '  MSE: ' + string(mean(e_all)));

end

endfunction
```

Приложение Г. Код функции ann_ADALINE_run

```
function y=ann_ADALINE_run(P, w, b)
// Функция моделирования адаптивного линейного элемента.
// Вызов:
//     y = ann_ADALINE_run(P,w,b)
/// Параметры:
//     P : Матрица  входных тестовых примеров (RxQ)
//     w : веса слоя (SxR)
//     b : смещения слоя(Sx1)
//     y : выход слоя (SxQ)
y = ann_purelin_activ(w*P+repmat(b,1,size(P,2)));

endfunction
```

ЛАБОРАТОРНАЯ РАБОТА № 5

ИССЛЕДОВАНИЕ МНОГОСЛОЙНОГО ПЕРСЕПТРОНА: АЛГОРИТМЫ ОБРАТНОГО РАСПРОСТРАНЕНИЯ С АДАПТИВНОЙ СКОРОСТЬЮ ОБУЧЕНИЯ И МОМЕНТОМ

1 Цель работы

Углубление теоретических знаний в области архитектуры многослойных нейронных сетей прямого распространения, исследование свойств алгоритмов обучения многослойных нейронных сетей, приобретение практических навыков обучения многослойного персептрона при решении задач классификации и аппроксимации функций.

2 Основные теоретические положения

2.1 Структура многослойного персептрона и его возможности

Многослойный персептрон (MLP – multi-layer perceptron) состоит из нескольких слоёв. В качестве примера на рисунке 5.1 изображена структурная схема 3-х слойного персептрона. Структуру сети можно описывать сокращенно: R-S^1-S^2-S^3, где R – число входов персептрона, S^1-S^2-S^3 – число нейронов в каждом слое.

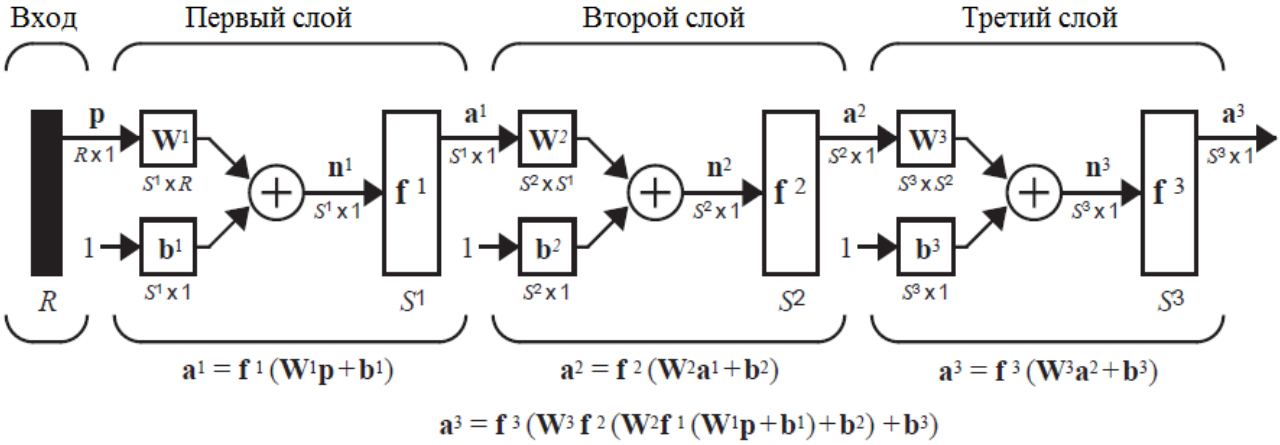


Рисунок 5.1 – Структура 3-х слойного персептрона

В MLP выход одного слоя является входом следующего слоя:

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \quad , \quad m = 0, 1, \ldots, M-1 , \tag{5.1}$$

где m – номер слоя, M – число слоев MLP. На вход нейронов первого слоя подается ввходной вектор **p**:

$$\mathbf{a}^0 = \mathbf{p}.$$

Выход последнего слоя является выходом MLP:

$$\mathbf{a} = \mathbf{a}^M.$$

Для моделирования работы MLP в модуле NeuralNetworks 3.0 пакета Scilab имеется функция ann_FFBP_run, код которой с комментариями приведен в приложении А. Функция ann_FFBP_run(P,W,af) принимает на вход матрицу P, столбцы которой представляют собой входные векторы **p**, расширенную матрицу весов и смещений всех слоев W, список af имен активационных функций каждого слоя MLP. Функция реализует вычисления выходного вектора **a** в полном соответствии с формулой (5.1).

В отличие от однослойного персептрона MLP способен решать задачи классификации, которые не ограничиваются случаем линейной сепарабельности. Например, MLP успешно решает задачу исключающего ИЛИ (рисунок 5.2). Одно из решений – использовать 2 нейрона в первом слое, которые создают две границы. Первая граница отделяет точку p_1=(0,0) от остальных точек, а вторая отделяет точку p_4=(1,1) . Второй слой комбинирует эти две границы, используя AND операцию

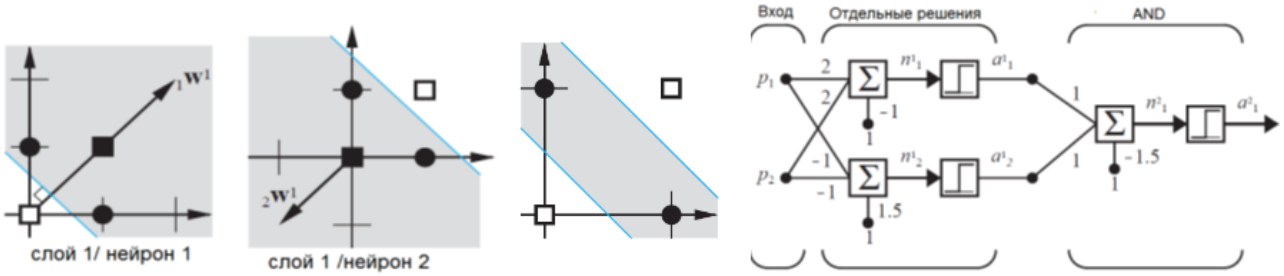


Рисунок 5.2 – Структура MLP для решения задачи исключающего ИЛИ

MLP также широко используют для аппроксимации функций. Можно показать, что *2-х слойная сеть со скрытым сигмоидальным слоем и линейным выходным слоем способна аппроксимировать любую непрерывную функцию с заданной точностью при надлежащем числе нейронов скрытого слоя* (теорема универсальной аппроксимации [3, 4]). Отметим, что теорема универсальной аппроксимации ничего не говорит об оптимальности MLP с одним скрытым слоем для решения задачи аппроксимации в отношении времени обучения, простоты применения, обобщающих способностей обученного MLP.

2.2 Целевая функция и BP-алгоритм обучения MLP

Обучение MLP осуществляется на основе **алгоритма обратного распространения (BP – back propagation)**. Алгоритм BP является развитием алгоритма LMS. Оба алгоритма минимизируют целевую функцию в виде среднего квадрата ошибки (СКО, MSE). Для MLP со многими выходами СКО будет равен:

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]. \tag{5.2}$$

Как и LMS, алгоритм BP аппроксимирует СКО (5.2) квадратами ошибки на каждом шаге k

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k). \tag{5.3}$$

Тогда в соответствии с алгоритмом наискорейшего спуска SDA (4.9) **алгоритм обучения MLP** запишется в виде

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial\hat{F}}{\partial w^m_{i,j}},$$
$$b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial\hat{F}}{\partial b^m_i} \tag{5.4}$$

С учетом того, что целевая функция F в этих формулах является стохастической, то они подобны формулам алгоритма LMS. **Сложность заключается в вычислении частных производных.** Если ввести обозначение

$$s^m_i \equiv \frac{\partial\hat{F}}{\partial n^m_i}, \tag{5.5}$$

где s^m_i **— чувствительность** целевой функции F к изменению n^m_i — i-го сетевозначения в слое m, то можно показать, что выражения для частных производных в формуле (5.4) запишутся в виде

$$\frac{\partial\hat{F}}{\partial w^m_{i,j}} = s^m_i a^{m-1}_j, \quad \frac{\partial\hat{F}}{\partial b^m_i} = s^m_i. \tag{5.6}$$

Соответственно стохастическая аппроксимация SDA (5.4) для MLP запишется в виде

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha s^m_i a^{m-1}_j, \quad b^m_i(k+1) = b^m_i(k) - \alpha s^m_i \tag{5.7}$$

Или векторно-матричной форме

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T,$$
$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m. \tag{5.8}$$

Чувствительности $\mathbf{s}^m$ также вычисляются с помощью рекуррентных соотношений. При этом чувствительность слоя m определяется через чувствительность слоя m+1, т.е. в обратном направлении **(backpropagation).** Чтобы показать это, запишем рекуррентное соотношение для вычисления чувствительностей, используя цепочное правило дифференцирования сложной функции

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \mathbf{s}^{m+1} \quad (5.9).$$

Здесь матрица частных производных от векторной функции $\mathbf{n}^{m+1}$ по $\mathbf{n}^m$, называемая *якобианом* (производная от векторной функции), равна

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m} = \mathbf{W}^{m+1}\dot{\mathbf{F}}^m(\mathbf{n}^m), \quad (5.10)$$

где $\dot{\mathbf{F}}^m(\mathbf{n}^m)$ — диагональная матрица с производными $\dot{f}^m(n_j^m)$ на главной диагонали.

Таким образом, **значения чувствительностей слоев распространяются в обратном направлении – от последнего слоя к первому.** Значение чувствительности выходного слоя

$$s_i^M = -2(t_i - a_i)\dot{f}^M(n_i^M) \quad \text{или} \quad \mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}). \quad (5.11)$$

Объединяя формулы, приведенные выше, получим алгоритм обучения MLP, называемый *алгоритмом обратного распространения* или алгоритмом BP (рисунок 5.3).

Алгоритм BP:

1. Инициализировать веса и смещения небольшими случайными значениями, задать допустимое значение ошибки $E_{\text{доп}}$;
2. Распространить очередной входной вектор по сети в прямом направлении: формула (5.1);
3. Распространить чувствительности в обратном направлении, выполнив вычисления по формулам:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}),$$
$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \qquad m = M-1, \ldots, 2, 1. \quad (5.12)$$

4. Обновить веса и смещения в соответствии с формулой (5.8);
5. Проверить значение ошибки, если ошибка на выходе сети больше $E_{\text{доп}}$, то перейти к п.2., иначе “стоп”.

Рисунок 5.3 – Алгоритм обратного распространения

2.3 Последовательный и блочный алгоритмы BP

Рассмотренный алгоритм BP является последовательным, т.к. предполагает обновление параметров сети после предъявления каждого очередного входного вектора. Возможен альтернативный блочный вариант алгоритма, когда полный градиент вычисляется до обновления весов и смещений (т.е. после подачи

на вход сети всех входных векторов **p**). Например, если все входные векторы равновероятны, то СКО будет равен

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})] = \frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q). \quad (5.13)$$

Полный градиент этой целевой функции будет равен среднему градиенту отдельных квадратов ошибок:

$$\nabla F(\mathbf{x}) = \nabla\left\{\frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\right\} = \frac{1}{Q}\sum_{q=1}^{Q}\nabla\{(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\}. \quad (5.14)$$

Для реализации блочного алгоритма BP выполняются шаги 1-3 алгоритма BP для всех входных векторов. Затем индивидуальные градиенты усредняются, чтобы получить полный градиент, и параметры сети обновляются на основе формул, содержащих средние обновления:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m(\mathbf{a}_q^{m-1})^T, \quad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m. \quad (5.15)$$

Так как рассмотренные алгоритмы BP представляют собой аппроксимацию алгоритма градиентного спуска (gradient descent), то будем обозначать их как BP_GD. В модуле NeuralNetworks 3.0 пакета Scilab имеется функция ann_FFBP_gd, реализующая блочный вариант алгоритма BP_GD. Текст функции с комментариями приведен в приложении Б. В общем случае функция имеет следующий формат вызова:

```
W = ann_FFBP_gd(P,T,N,af,lr,itermax,mse_min,gd_min),
```

где

P- матрица обучающих примеров входных данных;

T – матрица обучающих целевых значений;

N – вектор, задающий число нейронов каждого слоя, включая входной и выходной слои. Например, [2 30 1] – для сети с 2-мя входами, скрытым слоем из 30 нейронов и выходным слоем с 1-м нейроном;

af – список имен активационных функций слоев, например ['ann_tansig_activ','ann_purelin_activ'] для сети с двумя слоями. Число элементов списка функций должно быть равно числу слоёв без учета входного слоя;

lr – скорость обучения (параметр α в формуле 5.8), по умолчанию 0,01;

itermax – максимальное число эпох обучения, по умолчанию 1000;

mse_min – минимальное допустимое значение ошибки $E_{доп}$ (значение СКО-целевой функции), по умолчанию 1e-5;

gd_min – минимальное значение градиента целевой функции, по умолчанию 1e-5.

В отличие от алгоритма, описанного выше, функция ann_FFBP_gd прекращает процесс обучения при выполнении одного из трех условий:

mse < mse_min, itercnt > itermax , gd < gd_min,

где mse, itercnt, gd – соответственно значения СКО, счетчика эпох и значения градиента целевой функции на каждой итерации алгоритма. Обратим внимание на то, что по мере приближения к минимуму целевой функции значения mse должны снижаться, а значения gd должны быть близкими к нулю.

2.4 Эвристические разновидности алгоритмов BP

Основной недостаток алгоритмов BP_GD – медленное схождение. Методы ускорения этих алгоритмов могут быть разделены на 2 группы:

1) эвристические методы:
   - методы, основанные на изменении скорости обучения;
   - методы, основанные на использовании момента (импульса);
2) методы, использующие алгоритмы оптимизации.

Рассмотрим эвристические методы. Алгоритмы, относящиеся к этой группе, используют процедуру BP_GD в качестве базовой. Как отмечалось выше, BP_GD эквивалентен LMS, если сеть однослойная. *Однако BP_GD, применяемый для обучения MLP, обладает совсем другими свойствами.* Это связано с отличием поверхностей целевых функций для однослойной и многослойной сетей. Если поверхность целевой функции однослойной сети имеет одну стационарную точку, то поверхность целевой функции для MLP может иметь **много локальных минимумов** и кривизна её может существенно изменяться в различных областях пространства параметров. Это говорит о том, что необходимо изменять скорость обучения в процессе обучения. Например, на участках, где целевая функция уменьшается можно скорость обучения повышать, на плоских участках – оставлять неизменной, а на участках с растущими значениями целевой функции скорость обучения следует снижать.

Другой путь улучшения сходимости – сглаживание траектории изменения параметров сети. Когда алгоритм расходится, он начинает осциллировать (обычно поперек некоторой длинной узкой ложбины целевой функции). Если выполнить фильтрацию (сглаживание) траектории спуска путем усреднения обновления параметров, то получим более устойчивую траекторию спуска.

Для преодоления проблемы нескольких минимумов, следует запускать алгоритм с разными начальными условиями. При этом необходимо контролировать минимум достигнутой ошибки и выбирать те параметры сети, которые обеспечивают минимальный уровень СКО.

В соответствии с указанными подходами сформулируем две эвристические разновидности алгоритма BP для многослойной сети прямого распространения.

2.4.1 Алгоритм BP с моментом

Как указывалось, схождение алгоритма BP можно улучшить, если сглаживать траекторию спуска. Это можно сделать с помощью фильтра нижних частот. Рассмотрим усредняющий фильтр 1-го порядка:

$$y(k) = \gamma y(k-1) + (1-\gamma)w(k), \tag{5.16}$$

где $w(k)$ – вход фильтра, $y(k)$ – выход фильтра, γ – коэффициент момента ($0 \le \gamma < 1$). Поскольку фильтр, в общем, суммирует очередные входные значения $w(k)$ с предыдущими выходами $y(k\text{-}1)$, то на выходе фильтра будут формироваться значения $y(k)$, соответствующие некоторому среднему значению $w(k)$. Иными словами, фильтр добавляет в динамику обновления $w(k)$ некоторый момент инерции. Чем больше γ, тем более инерционен фильтр. При $\gamma=0$ выход фильтра $y(k)=w(k)$ и фильтр не будет вносить никакой инерции в динамику $w(k)$.

В соответствии с алгоритмом BP обновление параметров определяется выражениями

$$\Delta \mathbf{W}^m(k) = -\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T, \quad \Delta \mathbf{b}^m(k) = -\alpha \mathbf{s}^m.$$

При введении в эти уравнения момента инерции в соответствии с (5.16) получим *алгоритм BP_GD с моментом*, который будем обозначать как BP_GDM

$$\begin{aligned} \Delta \mathbf{W}^m(k) &= \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T, \\ \Delta \mathbf{b}^m(k) &= \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m. \end{aligned} \tag{5.17}$$

BP_GDM делает траекторию спуска более гладкой при большей скорости обучения по сравнению с алгоритмом BP_GD.

В модуле NeuralNetworks 3.0 пакета Scilab имеется функция ann_FFBP_gdm, реализующая этот алгоритм. В общем случае формат вызова функции требует задания следующих параметров:

```
W = ann_FFBP_gdm(P,T,N,af,lr,Mr,itermax,mse_min,gd_min),
```

где Mr – коэффициент момента (γ), по умолчанию 0,9. Остальные параметры функции аналогичны параметрам функции ann_FFBP_gd.

2.4.2 Алгоритм BP с адаптивной скоростью обучения

Как отмечалось, скорость обучения из-за изменения кривизны поверхности целевой функции MLP следует изменять в процессе обучения. Сформулируем правила управления скоростью обучения $\alpha(k)$ для блочного алгоритма BP:

1. Если в ходе обучения квадрат ошибки (по всему обучающему блоку) увеличивается более, чем на ξ процентов (1-5%) после обновления весов, то обновление весов следует отменить, скорость обучения уменьшить, умножив её на значение $0<\rho<1$, а коэффициент момента γ (если используется) обнулить.

2. Если квадрат ошибки уменьшается после обновления весов, то обновление весов принимается, скорость обучения умножается $\eta>1$; если γ было обнулено, то его значение восстанавливается.

3. Если квадрат ошибки увеличивается менее чем на ξ процентов, то обновление весов принимается, скорость обучения не изменяется; если γ было обнулено, то его значение восстанавливается.

В модуле NeuralNetworks 3.0 пакета Scilab имеется функция ann_FFBP_gda, реализующая этот алгоритм. В общем случае формат вызова функции требует задания следующих параметров:

```
W = ann_FFBP_gda(P,T,N,af,lr,lr_inc,lr_dec,itermax,mse_min,gd_min,mse_diff_max),
```

где lr_inc – коэффициент увеличения скорости обучения (η), по умолчанию 1,05; lr_dec – коэффициент уменьшения скорости обучения (ρ), по умолчанию 0,75; mse_diff_max – допустимое относительное увеличение СКО (ξ), по умолчанию 0,01. Остальные параметры функции аналогичны параметрам функции ann_FFBP_gd.

Также в составе модуля NeuralNetworks 3.0 пакета Scilab имеется функция ann_FFBP_gdx, которая комбинирует алгоритм ВР с моментом и алгоритм ВР с адаптивной скоростью обучения. Функция имеет расширенный список параметров:

```
(P,T,N,af,lr,lr_inc,lr_dec,Mr,itermax,mse_min,gd_min,mse_diff_max).
```

Назначение этих параметров было рассмотрена выше.

Эвристические алгоритмы обеспечивают ускорение сходимости для некоторых задач. Однако такие алгоритмы требуют установки значений дополнительных параметров, тогда как BP_GD требует задания только скорости обучения. Выбор значений дополнительных параметров зависит от задачи.

3. Варианты заданий и программа работы

3.1. Повторите теоретический материал, относящийся к архитектуре и алгоритмам обучения многослойного персептрона [1, 4, 5].

3.2. Решите с помощью MLP задачу бинарной классификации, когда граница между двумя классами является нелинейной. Для этого в соотвествии с вариантом из п. 4.3 лабораторной работы №1, где задана одномерная функция $y=f(x)$ на некотором интервале определения, необходимо:

3.2.1. Сформировать два множества случайных точек данных (не менее 400 точек), которые располагаются выше (класс1) и ниже кривой (класс 2) $y=f(x)$ и отстоят от неё на расстояние d= $0{,}3|(y_{max} - y_{min})|$;

3.2.2. Отобразить классы и кривую $y=f(x)$ на двумерной плоскости;

3.2.3. Разработать программу обучения многослойного персептрона с архитектурой *R-S*-1 для классификации этих классов, в качестве активационных функций скрытого слоя использовать функции ann_tansig_active или ann_logsig_active, а в качестве активационной функции выходного слоя ─ ann_logsig_active, обучение персептрона выполнять с использованием функции ann_FFBP_gd;

3.2.4. Выполнить предварительное обучение MLP с архитектурой [*R*-10-1] на небольшом числе эпох itermax =500 при разных значениях параметра скорости обучения lr с целью определения её квазиоптимального значения.

3.2.5. Используя полученное значение скорости обучения lr, выполнить обучение MLP с архитектурой [*R*-*S*-1] при разных *S* (10,20,30,40) на большом числе эпох itermax =2000.

3.2.6. Для различных MLP, обученных в соответствии с п.3.2.5, выполнить моделирование MLP и проверить точность классификации на **тестовом** множестве данных. Для этого сформировать тестовое множество данных аналогично п. 3.2.1, провести классификацию данных с помощью обученного MLP, отобразить точки предсказанных классов и кривую $y=f(x)$ на двумерной плоскости, вычислить точность правильной классификации при разных *S*.

3.3. Решите с помощью MLP задачу аппроксимации нелинейной функции двух переменных. Для этого, используя вариант из п. 4.5 лабораторной работы №1, где задана двумерная функция $z=f(x,y)$, необходимо:

3.3.1. Сформировать подмножества обучающих и тестовых данных. Для этого выбрать на плоскости (x, y) 500 (или более) случайных точек и определить в этих точках значение функции $z=f(x,y)$. В качестве входного вектора использовать вектор $\mathbf{p}=[x;y]$, в качестве значений элементов целевого вектора значения z. Полученные данные разделить на 2 подмножества: обучающее (80% данных) и тестовое (20% данных).

3.3.2. Разработать программу обучения многослойного персептрона с архитектурой *R*-*S*-1для аппроксимации функции $z=f(x,y)$, в качестве активационных функций скрытого слоя использовать функции ann_tansig_active или ann_logsig_active, а в качестве активационной функции выходного слоя – ann_purelin_active, обучение персептрона выполнять с использованием функции ann_FFBP_gd;

3.3.3. Построить кривые обучения MLP при разных значениях S (например, S = [5, 10, 15, 20]) и фиксированной скорости обучения lr (например, lr =0.005); выбрать квазиоптимальное значение S для дальнейшего использования. Построение кривых обучения выполнить при значении параметра itermax=300;

3.3.4. Выполнить обучение MLP (при квазиоптимальном *S*) также с помощью функций ann_FFBP_gdm, ann_FFBP_gda, ann_FFBP_gdx, сравнить получаемые кривые обучения с кривыми, полученными в п. 3.3.3;

3.3.5. Используя тестовое подмножество данных, выполнить моделирование 4-х вариантов MLP, обученных с помощью 4-х разных функций, указанных выше. Построить графики для сравнения значений функции $z=f(x,y)$ и соответствующих значений на выходе MLP, вычислить СКО аппроксимации функции на тестовом подмножестве, сравнить со значениями СКО, полученными при обучении MLP.

3.4. Подготовьте и защитите отчет по работе.

4. Методические рекомендации по выполнению работы

4.1. В задании 3.2. исследуется задача бинарной классификации. Цель задания – убедиться в том, что MLP способен строить нелинейные границы решения между двумя классами.

Для этого в соответствии с п.3.2.1 задания необходимо сначала сформировать два множества случайных точек, разделяемых заданной по варианту нелинейной функцией. Ниже приведен пример кода функции для формирования случайных точек, принадлежащих 2-м классам, разделяемых кривой $y=f(x)$. Точки классов данных отстоят от разделяющей кривой на величину range=kd*abs(ymax-ymin), где ymax, ymin – максимальное и минимальное значения функции y=f(x) на интервале её определения. Фрагменты кода, отмеченные курсивом, определяются вариантом задания и требуют уточнения. Модернизируйте этот код для формирования множеств входных данных и их меток с учетом варианта задания.

```
function [P, T, y, x]=gendata(Q, kd)
//функция генерирования случайных точек для 2-классов,
// разделяемых кривой (границей) y=f(x)
//Q - число формируемых точек классов
//kd - коэффициент, задающий ширину диапазона разброса точек

 //задание области определения функции - координата x
 xmin=0.7;
 xmax=4;
 //определение ymin и ymax для заданной функции y=f(x)
 stepx=abs(xmax-xmin)/Q;
 x=xmin:stepx:xmax;
 y=f(x); // используйте определение функции из ЛР №1
 ymin=min(y);
 ymax=max(y);
 //диапазон разброса точек относительно значений функции
 range=kd*abs(ymax-ymin);
 //формирование случ. координат точек вдоль осей x и y
 datax=grand(1,Q,'unf',xmin,xmax);
 datay=grand(1,Q,'unf',ymin-range,ymax+range);
 // формирование обучающего множества {P,T}
 P=[];
 T=[];
 for j=1:1:Q
    //ордината границы border для случайной точки datax(1,j)
    border=f(datax(1,j));
    if datay(1,j)>border
       t_class=1  //точка выше границы – класс 1
    else
       t_class=0  //точка ниже границы – класс 2
    end
    //формируем массив входных данных P и массив меток классов T
    P=[P [datax(1,j);datay(1,j)]];
```

```
    T=[T t_class];
  end
endfunction
```

Для отображения точек 2-х классов (задание п.3.2.2) и разделяющей границы (рисунок 5.4) можно использовать следующий код:

```
//отображение множества точек 2-х классов и разделяющей границы
drawlater(); // прорисовка  будет позже
plot(P(1,T==1),P(2,T==1),'o'); //отображать точки 1-го класса знаком 'o'
plot(P(1,T==0),P(2,T==0),'g*'); //отображать точки 2-го класса знаком '*'
plot(x,y,'r'); // отображать границу красным цветом
drawnow(); // включаем прорисовку
xtitle('Классы и граница (обучающее множество)');
```

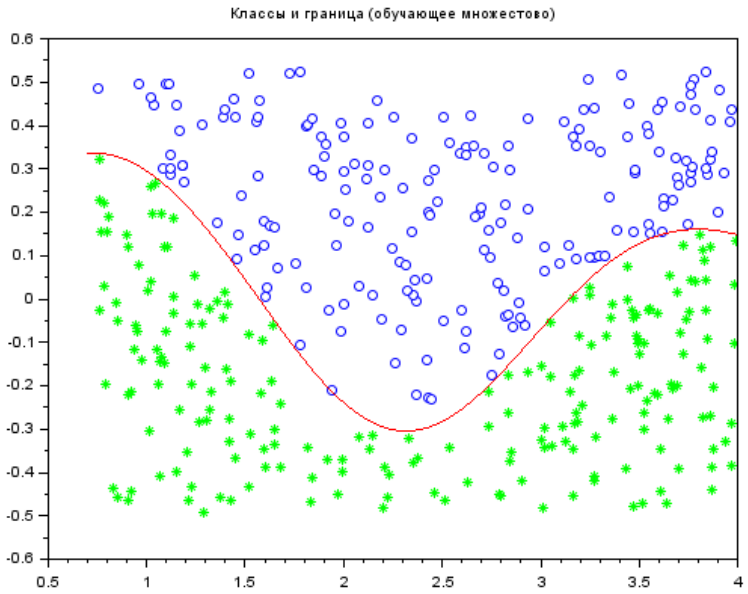


Рисунок 5.4 – Сгенерированные точки классов и граница между классами

В соответствии с заданием обучаемый двухслойный MLP содержит в выходном слое сигмовидную активационную функцию, значения которой лежат в диапазоне от 0 до 1. Эти значения можно интерпретировать как вероятности принадлежности классам. Если значение на выходе MLP будет больше 0,5, то входная точка принадлежит классу 1, иначе классу 2.

Выполнение п. 3.2.4 заключается в предварительном обучении MLP с использованием функции ann_FFBP_gd при разных значениях скорости обучения lr. При каждом запуске функции будет отображаться прогресс обучения в отдельном графическом окне (рисунок 5.5). Для оценки результатов обучения при разных значениях скорости обучения lr контролируйте значение среднего квадрата ошибки MSE в окне прогресса. Рекомендуется значение скорости обучения выбирать из списка lr= [1.5, 1.25, 1.0, 0.75]. В качестве квазиоптимального значения lr следует выбрать то значение, которое обеспечивает минимальное конечное значение MSE.

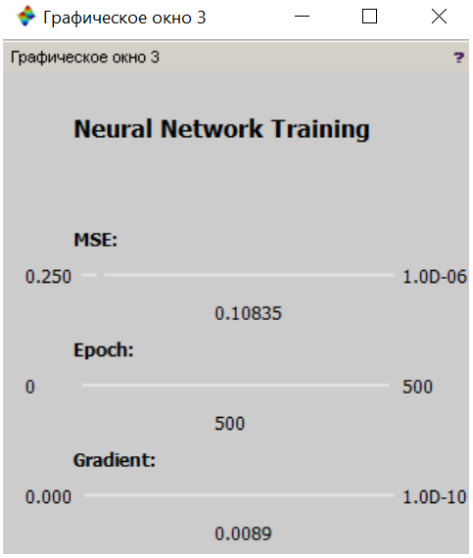


Рисунок 5.5 – Окно прогресса, отображаемое функциями обучения ann_FFBP

Проведите предварительное обучение на 500 итерациях и определите квазиоптимальное значение lr в ходе выполнения следующего кода:

```
//обучение MLP при разных значениях lr
 lr= [1.5, 1.25, 1.0, 0.75];
 N=[2 10 1];  // архитектура сети
 af=['ann_tansig_activ','ann_logsig_activ']; // активационные функции слоев
 itermax=500;
 mse_min=1e-6; // обеспечивают завершение алгоритма по числу итераций
 gd_min=1e-10;
 for k=1:length(lr)
   W = ann_FFBP_gd(P,T,N,af,lr(k),itermax,mse_min,gd_min);
 end;
```

При выполнении п.п. 3.2.5-3.2.6 следует обучать MLP с разным числом нейронов в скрытом слое и тестировать результаты обучения. Обучение необходимо выполнять при itermax =2000 и квазиоптимальной скорости обучения, полученной ранее. Чтобы вычислить точность нейросети во время обучения train_accuracy необходимо вычислять выходные значения обученного MLP с помощью функции моделирования ann_FFBP_run(P,W,af). При этом выходные значения, полученные в ходе моделирования, необходимо привести к бинарным значениям 0 и 1 с учетом интерпретации значений сигмовидной функции в выходном слое MLP. Для этого следует их округлить с помощью функции round, а затем посчитать train_accuracy, как отношение числа случаев правильной классификации sum(T==a_train_pred) к числу точек данных length(T):

```
 //Обучаем MLP при разных значениях S
 S=[10 20 30 40]; // число нейронов скрытого слоя
 len_S=length(S);
 itermax=2000;
 lr=1.0 // квазиоптимальное значение, определенное выше
```

```
W_all=cell(); // создаем клеточный массив для хранения матриц весов W
train_accuracy=zeros(len_S); // создаем массив для хранения точности обучения
for k=1:len_S
  N=[2 S(k) 1];
  W = ann_FFBP_gd(P,T,N,af,lr,itermax, mse_min,gd_min); // обучение
  a_train_pred = round(ann_FFBP_run(P,W,af)); // моделирование
  train_accuracy(k)=sum(T==a_train_pred)/length(T); // вычисление точности
  W_all{k}= W;
end;
```

Для оценки обобщающих свойств персептрона следует использовать множество данных, на котором MLP не обучался. Сформируйте тестовое множество входных данных с помощью функции gendata(Q, kd):

```
[P_test,T_test,y_test,x_test]=gendata(Q,kd);
```

Чтобы вычислить точность правильной классификации на тестовых данных при разных S используйте следующий код:

```
//тестирование MLP
test_accuracy=zeros(len_S); // создаем массив для хранения точностей
for k=1:len_S
    a_test_pred = round(ann_FFBP_run(P_test,W_all{k},af)); // предсказание
    test_accuracy(k)=sum(T_test == a_test_pred)/length(T_test); // оценка точностиь
```

Для отображения результатов классификации, полученных с помощью обученного MLP, используйте код, который применялся для построения рисунка 5.4, заменив целевой вектор T, на вектор предсказания классов тестовых данных a_test_pred:

```
plot(P_test(1,a_test_pred==1),P_test(2,a_test_pred==1),'o'); // точки 1-го класса
plot(P_test(1,a_test_pred==0),P_test(2,a_test_pred==0),'g*'); // точки 2-го класса
plot(x_test,y_test,'m'); // отображать границу малиновым цветом
xtitle('Результаты классификации на тестовом множестве');
```

Пример отображения выходных классов, формируемых обученным MLP на тестовом множестве данных, приведен на рисунке 5.6. Обратите внимание, что ряд точек тестового множества MLP классифицировал неправильно. Точность правильной классификации при S=30 для рассматриваемого примера равна 0,9675 на тестовом множестве (на обучающем множестве 0,9775).

4.2. В задании 3.3. исследуется задача аппроксимации двумерной функции с помощью MLP. Цель задания – убедиться в том, что MLP способен обучаться аппроксимации нелинейных функций на основе множества обучающих данных. Для обучения MLP в соответствии с п.3.3.1 задания требуется сформировать подмножества обучающих и тестовых данных. Тестовые данные используются только в ходе тестирования обученной нейросети с целью оценки качества её функционирования и обобщающих способностей.

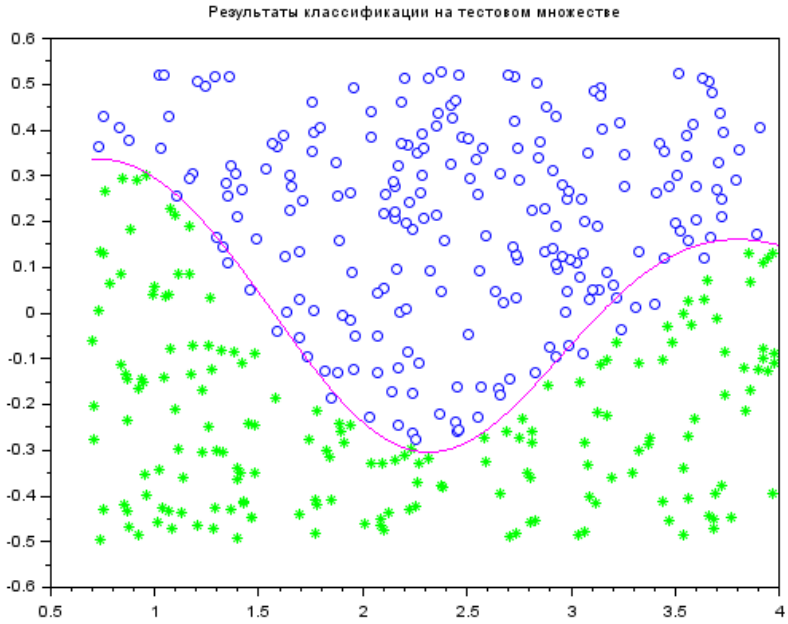


Рисунок 5.6 – Результаты классификации для тестового множества, точность классификации 0,9675 (S=30)

Ниже приведен фрагмент кода для генерации обучающих P_train, T_train и тестовых P_test, T_test подмножеств данных:

```
Q=500; //Общее число элементов множества данных
Q_train=floor(0.8*Q); //число эл-тов обучающего мно-ва = 80% от Q
//формирование случайных координат x,y
x=grand(1,Q,'unf',minx,maxx);
y=grand(1,Q,'unf',miny,maxy);
//формирование обучающего подмно-ва
x_train=x(1:Q_train);
y_train=y(1:Q_train);
P_train=[x_train;y_train]; // мно-во входных векторов
T_train=f(x_train,y_train); //желаемое значение на выходе MLP= f(x,y)
//формирование тестового подмно-ва
x_test=x(Q_train+1:$);
y_test=y(Q_train+1:$);
P_test=[x_test;y_test];
T_test=f(x_test,y_test);
```

В задании п. 3.3.3 требуется обучить MLP на основе алгоритма BP_gd при разных значениях S и построить кривые обучения. Построение кривых обучения потребует модификации кода встроенной функции ann_FFBP_gd. Необходимо, чтобы функция возвращала вектор значений СКО (в программе обозначен out_mse). Для этого выполните модернизацию функции ann_FFBP_gd по следующей схеме:

```
function [W, out_mse]=ann_FFBP_gd1(P, T, N, af, lr, itermax, mse_min, gd_min)
        …
// Stopping Criteria
      mse = mean(e.^2);
      itercnt = itercnt + 1;
      out_mse(itercnt)=mse; // добавленная строка программы
      gd = mean(s(1).^2);
        …
endfunction
```

Изменения, внесенные в функцию, отмечены курсивом. Для исследования MLP и запоминания значений СКО при разных *S* организуйте циклический вызов модифицированной функции ann_FFBP_gd1:

```
af=['ann_tansig_activ','ann_purelin_activ'];
lr=0.005
itermax=300
S=[5,10,15,20]; // выбор значений S зависит от вида функции z=f(x,y)
for i=1:length(S)
    N=[2 S(i) 1];
    [W_gd, out_mse_gd] = ann_FFBP_gd1(P_train,T_train,N,af,lr,itermax);
    mse_gd_history(i,:)=out_mse_gd;
    printf("S=%d CKO=%f\n",S(i),out_mse_gd($)) //конечное значение СКО

end;
```

В этом случае векторы значений СКО будут запоминаться в строках матрицы mse_gd_history. Это позволит после завершения цикла построить кривые обучения для каждого *S* (рисунок 5.7). Из анализа кривых обучения (для примера на рисунке 5.7) следует, что квазиоптимальным будет значение *S*=15, т.к обеспечивает наиболее быстрое снижение СКО и наименьшее конечное значение СКО в ходе обучения (out_mse_gd($)).

Для сравнения эвристических алгоритмов обучения, рассмотренных в п.2.4, с алгоритмом BP_gd (п. 3.3.4 задания) следует обучить MLP с одной и той же структурой, используя также функции обучения, указанные ниже:

```
[W_gd, out_mse_gd] = ann_FFBP_gd1(P_train,T_train,N,af,lr,itermax,mse_min,gd_min);
[W_gda,out_mse_gda] =
   ann_FFBP_gda1(P_train,T_train,N,af,lr,lr_inc,lr_dec,itermax,mse_min,gd_min,mse_diff_max);
[W_gdm, out_mse_gdm]=ann_FFBP_gdm1(P_train,T_train,N,af,lr,Mr,itermax,mse_min,gd_min);
[W_gdx, out_mse_gdx] = ann_FFBP_gdx1(P_train,T_train,N,af,lr,lr_inc,lr_dec,Mr,itermax,mse_min,gd_min);
```

Код всех этих функций модифицируется аналогично функции ann_FFBP_gd1. Результаты обучения MLP разными алгоритмами запоминаются в матрицах весов и векторах ошибок (хранят значения СКО), возвращаемых указанными функциями. Запоминание весов MLP необходимо для тестирования обученных моделей нейросетей, а запоминание векторов значений СКО – для построения кривых обучения (рисунок 5.7).

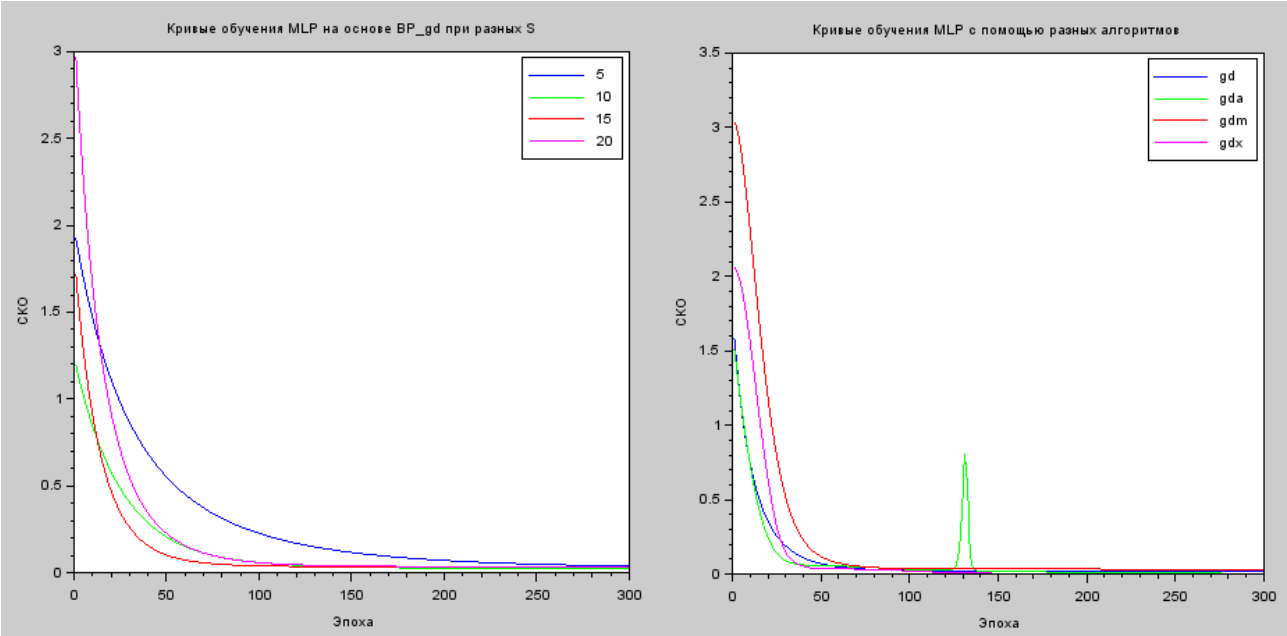


Рисунок 5.7 – Кривые обучения MLP

Тестирование обученных моделей (п. 3.3.5 задания) выполняется исключительно на тестовых данных, которые не использовались для обучения. Это позволяет оценить обобщающие свойства полученных моделей. Для тестирования нейросетей используйте функцию ann_FFBP_run:

```
y_test_gd = ann_FFBP_run(P_test,W_gd,af);       // сеть, обученная алг-м BP_gd
y_test_gda = ann_FFBP_run(P_test,W_gda,af);   // сеть, обученная алг-м BP_gda
y_test_gdm = ann_FFBP_run(P_test,W_gdm,af); // сеть, обученная алг-м BP_gdm
y_test_gdx = ann_FFBP_run(P_test,W_gdx,af);   // сеть, обученная алг-м BP_gdx

// вычисление СКО на тестовых данных
mse_gd=((T_test-y_test_gd)*(T_test-y_test_gd)')/Q_test
mse_gda=((T_test-y_test_gda)*(T_test-y_test_gda)')/Q_test
mse_gdm=((T_test-y_test_gdm)*(T_test-y_test_gdm)')/Q_test
mse_gdx=((T_test-y_test_gdx)*(T_test-y_test_gdx)')/Q_test
```

По результатам тестирования необходимо построить графики функции $z=f(x,y)$, определенной на тестовом множестве входных данных (x_test, y_test), и графики выходных значений MLP при этих же входных данных для разных алгоритмов обучения (рисунок 5.8). Сравнить вычисленные выше СКО аппроксимации функции на тестовом подмножестве со значениями СКО, полученными на этапе обучения соответствующих MLP.

Из рисунка 5.8 следует, что алгоритмы BP_gd и BP_gdm дают неудовлетворительные результаты. Для построения кривых на рисунке 5.8 для этих алгоритмов скорость обучения была увеличена до 0,1 (против 0,005 для алгоритмов BP_GDA и BP_GDX). Эти алгоритмы медленно сходятся и требуют значительно большего числа эпох обучения по сравнению с алгоритмами, основанными на управлении скоростью обучения. Наименьшее значение СКО получается при обучении MLP с помощью алгоритма BP_GDX.

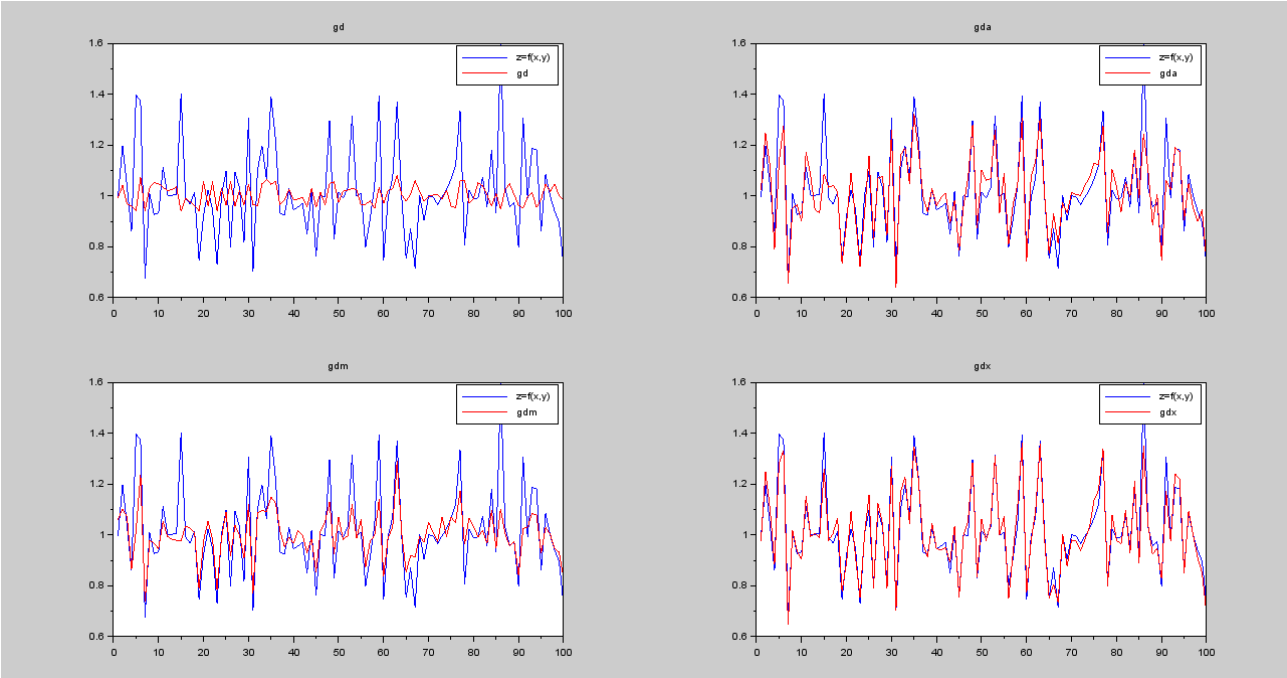


Рисунок 5.8 – Графики тестовых значений функции $z=f(x,y)$ и значений на выходе MLP, обученных разными алгоритмами

На рисунке 5.9 изображены поверхности заданной $z=f(x,y)$ и аппроксимирующей функции, реализуемой с помощью MLP, который был обучен с использованием алгоритма BP_GDX. Для построения таких поверхностей можно воспользоваться кодом, представленным ниже:

```
//построение 3D поверхности функций
x=linspace(-1,1,50);
y=linspace(-1,1,50);
z=feval(x,y,f);
figure(1)
subplot(1,2,1)
surf(x,y,z')
title('Исходная функция')

//визуализация MLP аппроксиматора
Pz=[];
// формируем входные векторы P в каждом узле сетки точек (x,y)
for i=1:50
    for j=1:50
        Pz=[Pz [x(i);y(j)]];
    end
end
// вычисляем выход обученного MLP в узлах (x,y)
z_gdx = ann_FFBP_run(Pz,W_gdx,af);
// реформатируем вектор выхода MLP в матрицу
zz=matrix(z_gdx,[50,50]);
// отображаем поверхность функции аппроксимации
scf(1)
```

```
subplot(1,2,2)
surf(x,y,zz)
title('Аппроксимированная функция')
```

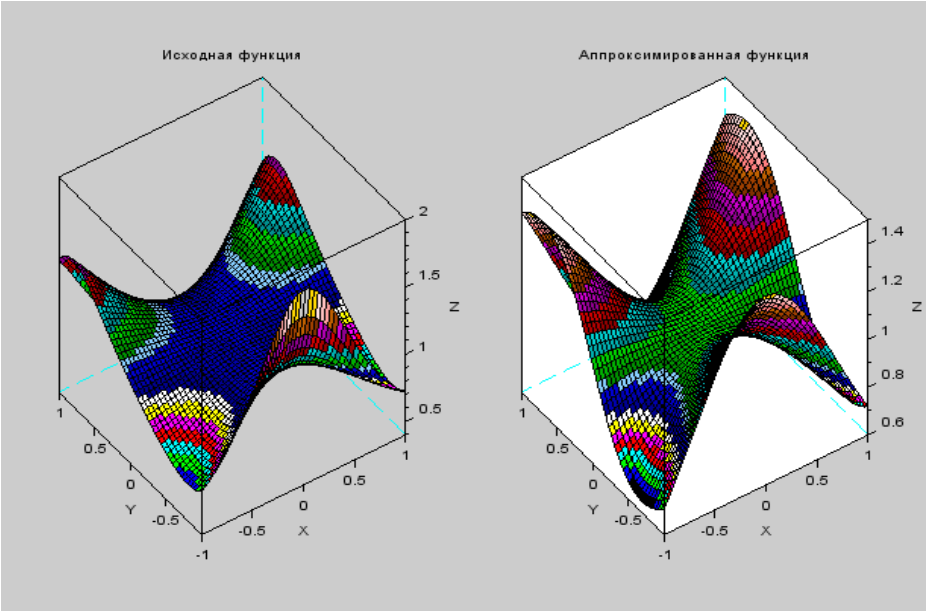


Рисунок 5.8 – Поверхность исходной функции $z=f(x,y)$ и её аппроксимация с помощью MLP

5. Содержание отчета

5.1. Цель работы.
5.2. Вариант задания.
5.3. Схема MLP, решающего задачу классификации 2-х классов; программа обучения MLP бинарной классификации с использованием функции ann_BP_gd; результаты предварительного обучения MLP при разных значениях параметра lr и выбор квазиоптимального lr; результаты обучения MLP при разных *S* и большом числе эпох; результаты тестировании обученного бинарного классификатора на основе MLP и точности правильной классификации при разных *S*.
5.4. Схема MLP для решения задачи аппроксимации функций; программа обучения MLP; кривые обучения MLP при разных значениях S и выбор квазиоптимального *S*; кривые обучения MLP (при квазиоптимальном *S*) с помощью функций ann_FFBP_gd, ann_FFBP_gdm, ann_FFBP_gda, ann_FFBP_gdx; результаты тестирования MLP, обученных указанными функциями (сравнительные графики), значения СКО на этапе обучения и на этапе тестирования.
5.5. Выводы по результатам исследований.

6. Контрольные вопросы

6.1. Нарисуйте схему многослойного персептрона и запишите выражения для вычисления его выходных значений в векторно-матричной форме.

6.2. Объясните алгоритм, положенный в основу функции ann_FFBP_run(P,W,af), и назначение её параметров.
6.3. Нарисуйте структуру MLP для решения задачи исключающего ИЛИ.
6.4. Сформулируйте общий смысл теоремы об универсальной аппроксимации в отношении MLP.
6.5. Запишите выражение целевой функции, используемой в алгоритме BP.
6.6. Запишите общие выражения алгоритма наискорейшего спуска применительно к MLP.
6.7. Что понимают под чувствительностью целевой функции MLP?
6.8. Запишите выражения для производных целевой функции MLP с использованием чувствительности.
6.9. Запишите векторно-матричные выражения стохастической аппроксимации SDA для MLP.
6.10. Запишите рекуррентное соотношение для вычисления чувствительностей, используя цепочное правило дифференцирования.
6.11. Чему равен якобиан применительно к MLP?
6.12. Сформулируйте последовательный алгоритм BP.
6.13 Сформулируйте и запишите выражения для блочного алгоритма BP.
6.14. Объясните параметры функции ann_FFBP_gd, реализующей блочный вариант алгоритма BP_GD.
6.15. При каких условиях функция ann_FFBP_gd прекращает процесс обучения?
6.16. Какой основной недостаток алгоритма BP_GD?
6.17. На какие группы подразделяются эвристические методы улучшения BP_GD? Дайте их краткую характеристику.
6.18. Сформулируйте алгоритм BP с моментом инерции.
6.19. Объясните параметры функции ann_FFBP_gdm.
6.20. Сформулируйте алгоритм BP с адаптивной скоростью обучения.
6.21. Объясните параметры функции ann_FFBP_gda.
6.22. Объясните назначение и параметры функции ann_FFBP_gdx.
6.23. Как формируются обучающее и тестовые подмножества данных? Зачем нужно тестовое подмножество?
6.24. Почему отличаются СКО обучения и тестирования? Какая из ошибок обычно меньше?

Приложение А. Функция ann_FFBP_run

```
function y=ann_FFBP_run(P, W, af)
// Функция моделирования сети прямого распространения
// Пример вызова:
//    [y] = ann_FFBP_run(W,P,af)
// Параметры:
//    P : Тестовые входные значения
//    W : Веса и смещения
//    af : Список активационных функций
//    y : Выходные значения

//1. ================Обработка списка аргументов функции=======================
rhs=argn(2);
```

```
if rhs < 3; af = ['ann_tansig_activ','ann_purelin_activ']; end

if size(W) ~= size(af,2) then
   error('Numbers of activation functions must match numbers of layers (N-1)');
end

//2.================= Фаза моделирования====================================
// Определяем число слоев сети по размерам W
layers = size(W);
//Создаем списки для хранения сетевых значений и активностей слоев
n = list(0);
a = list(0);
//Вычисляем сетевые значения 1-го слоя: n=Wp+b
n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(P,2)); // Можно временно хранить в n для экономии памяти
// Оцениваем значения на выходе 1-го слоя: a=af(n), где af – нелинейность слоя
a(1) = evstr(af(1)+'(n('+string(1)+'))');
// Повторяем вычисления для каждого следующего слоя
for cnt = 2:layers
   n(cnt) = W(cnt)(:,1:$-1)*a(cnt-1) + repmat(W(cnt)(:,$),1,size(P,2)); // Можно временно хранить в n для экономии памяти
   a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');
end
// Результирующие значения соответствуют выходам последнего слоя
y = a($);
endfunction
```

Приложение Б. Функция ann_FFBP_gd

```
function W=ann_FFBP_gd(P, T, N, af, lr, itermax, mse_min, gd_min)
// Обучение на основе блочного  алгоритма градиентного спуска с обратным распространением.
//
// Примеры вызова:
//    W = ann_FFBP_gd(P,T,N)
//    W = ann_FFBP_gd(P,T,N,af,lr,itermax,mse_min,gd_min)
//
// Параметры
//    P : Обучающие входные данные
//    T : Обучающие целевые значения
//    N : Вектор, задающий число нейронов каждого слоя, включая входной и выходной слои
//    af : Список имен активационных функций от 1-го до выходного слоев
//    lr : Скорость обучения
//    itermax : Максимальное число эпох обучения
//    mse_min : Минимальная ошибка  (Значение целевой функции)
//    gd_min : Минимальное значение градиента
//    W : Результирующие веса и смещения

//1. ================Обработка списка аргументов функции=======================
rhs=argn(2);

// Проверка ошибки списка аргументов
if rhs < 3; error("Expect at least 3 arguments, P, T and N");end
// Выбор значений аргументов по умолчанию
if rhs < 4; af = ['ann_tansig_activ','ann_purelin_activ']; end
if rhs < 5; lr = 0.01; end
if rhs < 6; itermax = 1000; end
if rhs < 7; mse_min = 1e-5; end
if rhs < 8; gd_min =  1e-5; end

if af == []; af = ['ann_tansig_activ','ann_purelin_activ']; end
if lr == []; lr = 0.01; end
if itermax == []; itermax = 1000; end
if mse_min == []; mse_min = 1e-5; end
if gd_min == []; gd_min =  1e-5; end

// Проверка ошибки списка активационных функций
if size(N,2)-1~= size(af,2) then
   error('Numbers of activation functions must match numbers of layers (N-1)');
end
```

```
//2.=======================Инициализация====================

// Инициализация сети и формирование списка имен производных активационных функций
format(8);
W = ann_ffbp_init(N);
itercnt = 0;
af_d = strsubst(af,'ann_','ann_d_');
mse = %inf;
gd = %inf;

//Инициализация GUI отображения прогресса обучения
handles = ann_training_process();
handles.itermax.string = string(itermax);
handles.msemin.string = string(mse_min);
handles.gdmax.string = 'inf';
handles.gdmin.string = string(gd_min);

// Задание числа слоев и списков промежуточных переменных
layers = size(N,2)-1; // слои считаются от 1-го скрытого слоя до выходного слоя
n = list(0);
a = list(0);
m = list(0);
s = list(0);

// 3. ===================Реализация цикла эпох обучения==========================
while mse > mse_min & itercnt < itermax & gd > gd_min
      // Прямое распространение – моделирование (аналогично функции ann_FFBP_run (формула 5.1))
      n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(T,2));
      a(1) = evstr(af(1)+'(n('+string(1)+'))');
      for cnt = 2:layers
         n(cnt) = W(cnt)(:,1:$-1)*a(cnt-1) + repmat(W(cnt)(:,$),1,size(T,2));
         a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');

      end
      // Вычисление ошибки
      e = T - a($);
      // Обратное распространение значений чувствительности
      m(layers) = evstr(af_d(layers)+'(a('+string(layers)+'))'); // Вычисление производных актив. функций последнего слоя
      s(layers) = (-2*m(layers).*e); // Вычисление чувствительности выходного слоя  (формула 5.12)
      for cnt = layers-1:-1:1  //Вычисляем производные актив. функций слоев и распространяем обратно чувствительности.
         m(cnt) = evstr(af_d(cnt)+'(a('+string(cnt)+'))');
         s(cnt) = m(cnt).*(W(cnt+1)(:,1:$-1)'*s(cnt+1));   (формула 5.12)
      end

      // Обновление весов сети (формула 5.15)
      W(1)(:,1:$-1) = W(1)(:,1:$-1) - (lr*s(1)*P')./size(P,2); // Обновление весов 1-го скрытого слоя
      W(1)(:,$) = W(1)(:,$) - lr*mean(s(1),2); // Обновление смещений 1-го скрытого слоя
      for cnt = 2:layers // Обновление весов и смещений следующих слоев
         W(cnt)(:,1:$-1) = W(cnt)(:,1:$-1) - (lr*s(cnt)*a(cnt-1)')./size(P,2);
         W(cnt)(:,$) = W(cnt)(:,$) - lr*mean(s(cnt),2);
      end

     // Вычисление критериев остановки алгоритма
      mse = mean(e.^2);  // СКО
      itercnt = itercnt + 1; //Число эпох
      gd = mean(s(1).^2); // Средний квадрат значения градиента целевой функции

     // программирование GUI отображения прогресса обучения
     if  itercnt == 1 then
        mse_max = mse;
        handles.msemax.string = string(mse_max);
        gd_max = gd;
        handles.gdmax.string = string(gd_max);
        mse_span = log(mse) - log(mse_min);
        iter_span = itermax;
        gd_span =  log(gd) - log(gd_min);
     end

      // для версии выше Scilab 5.5
      handles.iter.value = round((itercnt/iter_span)*100);
```

```
        handles.mse.value = -(log(mse)-log(mse_max))/mse_span * 100;// round(((log(mse) - log(mse_min))/mse_span)*100);
        handles.gd.value = -(log(gd)-log(gd_max))/gd_span * 100; //round(((log(gd) - log(gd_min))/gd_span)*100);

        handles.itercurrent.string = string(itercnt);
        handles.msecurrent.string = string(mse);
        handles.gdcurrent.string = string(gd);

end
endfunction
```

ЛАБОРАТОРНАЯ РАБОТА № 6

ИССЛЕДОВАНИЕ МНОГОСЛОЙНОГО ПЕРСЕПТРОНА: АЛГОРИТМ ЛЕВЕНБЕРГА-МАРКВАРДТА

1 Цель работы

Углубление теоретических знаний в области алгоритмов обучения многослойных нейронных сетей прямого распространения, исследование свойств алгоритмов обучения второго порядка, приобретение практических навыков обучения многослойного персептрона при решении задач классификации и аппроксимации функций.

2 Основные теоретические положения

2.1 Метод Ньютона

Метод Ньютона основан на поиске стационарной точки квадратичной аппроксимации целевой функции $F(\mathbf{x})$:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k^T \mathbf{A}_k \Delta\mathbf{x}_k . \quad (6.1)$$

Найдем градиент (6.1) по отношению к $\Delta\mathbf{x}_k$ и приравняем его нулю

$$\mathbf{g}_k + \mathbf{A}_k \Delta\mathbf{x}_k = \mathbf{0} . \quad (6.2)$$

Отсюда оптимальный шаг будет равен

$$\Delta\mathbf{x}_k = -\mathbf{A}_k^{-1}\mathbf{g}_k . \quad (6.3)$$

Тогда правило обучения в соответствии с методом Ньютона запишется в виде

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k , \quad (6.4)$$

где $\mathbf{A}_k$ – матрица Гессе целевой функции (матрица вторых производных), $\mathbf{g}_k = \nabla F(\mathbf{x})$ – значение вектора градиента на k–ой итерации алгоритма .

Достоинством метод Ньютона, является то, что он находит минимум квадратичной целевой функции за один шаг. Однако метод Ньютона при других видах $F(\mathbf{x})$ не сходится за один шаг и не гарантирует схождение.

Метод Ньютона, как и SDA, полагается только на локальные свойства поверхности целевой функции (первая и вторая производные). Он «не знает» глобальных свойств целевой функции. Для произвольной целевой функции метод Ньютона может давать непредсказуемые результаты. В то время как SDA гарантирует схождение при малых значениях скорости обучения. Исключить расхождение алгоритма (6.4) можно путем включения в него SDA шагов, когда происходит расхождение. Иным недостатком алгоритма (6.4) является необходимость вычисления обратной матрицы Гессе.

Метод Ньютона совпадает с SDA когда гессиан – это единичная матрица. Это наблюдение приводит к *квазиньютоновским алгоритмам*. В этих алгоритмах обратную матрицу Гессе заменяют положительной определённой матрицей $\mathbf{H}_k$, которая обновляется на каждом шаге без использования инверсии. Эти алгоритмы разрабатываются так, чтобы $\mathbf{H}_k$ сходилась к $\mathbf{A}^{-1}$.

2.2 Алгоритм Левенберга-Марквардта (LM)

2.2.1 Основной алгоритм

LM алгоритм – разновидность алгоритма Ньютона (6.4), для которого ЦФ $F(\mathbf{x})$ представляется суммой квадратов:

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i^2(\mathbf{x}) = \mathbf{v}^T(\mathbf{x})\mathbf{v}(\mathbf{x}), \tag{6.5}$$

где $\mathbf{v}(\mathbf{x})$ – векторная функция параметров сети, N – число элементов в $\mathbf{v}$.
Для этой целевой функции j- ый элемент градиента будет равен

$$[\nabla F(\mathbf{x})]_j = \frac{\partial F(\mathbf{x})}{\partial x_j} = 2\sum_{i=1}^{N} v_i(\mathbf{x})\frac{\partial v_i(\mathbf{x})}{\partial x_j}. \tag{6.6}$$

Градиент (6.6) можно представить в матричной форме

$$\nabla F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{v}(\mathbf{x}), \tag{6.7}$$

где

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{6.8}$$

якобиан векторной функции $\mathbf{v}(\mathbf{x})$.
Элементы матрицы Гессе и сама матрица будут равны:

$$[\nabla^2 F(\mathbf{x})]_{k,j} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_j} = 2\sum_{i=1}^{N}\left\{\frac{\partial v_i(\mathbf{x})}{\partial x_k}\frac{\partial v_i(\mathbf{x})}{\partial x_j} + v_i(\mathbf{x})\frac{\partial^2 v_i(\mathbf{x})}{\partial x_k \partial x_j}\right\}, \tag{6.9}$$

$$\nabla^2 F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x}), \tag{6.10}$$

где

$$\mathbf{S}(\mathbf{x}) = \sum_{i=1}^{N} v_i(\mathbf{x})\nabla^2 v_i(\mathbf{x}).$$

Если **S(x)** мало, то матрицу Гессе можно аппроксимировать

$$\nabla^2 F(\mathbf{x}) \cong 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}). \tag{6.11}$$

Выполняя подстановки (6.7) и (6.11) в формулу метода Ньютона, получим алгоритм **Гаусса-Ньютона**

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} 2\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k). \tag{6.12}$$

Достоинство алгоритма Гаусса-Ньютона: не требуется вычислять вторые производные. Проблема: матрица

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} \tag{6.13}$$

может быть неинвертируемой. Эту проблему преодолевают введением аппроксимации матрицы Гессе:

$$\mathbf{G} = \mathbf{H} + \mu\mathbf{I}.$$

Отсюда получаем основной **алгоритм Левенберга –Марквардта**

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k). \tag{6.14}$$

LM алгоритм обладает полезным свойством: если μ_k увеличивается, то **LM алгоритм трансформируется в SDA** с малой скоростью обучения:

$$\mathbf{x}_{k+1} \cong \mathbf{x}_k - \frac{1}{\mu_k}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k - \frac{1}{2\mu_k}\nabla F(\mathbf{x}), \text{ для больших } \mu_k$$

Если $\mu_k \to 0$, то **LM алгоритм трансформируется в алгоритм Гаусса-Ньютона.**

Алгоритм начинают выполнять при малом μ_k =0.01. Если шаг не приводит к уменьшению целевой функции, то его повторяют с μ_k увеличенным в $\theta>1$ раз (например, $\theta=10$). Если шаг дает уменьшение целевой функции, то μ_k уменьшают в θ раз. При уменьшении μ_k алгоритм будет приближаться к алгоритму Гаусса-Ньютона, что обеспечит быстрое схождение.

LM алгоритм можно применить для обучения MLP. Если целевые значения обучающей выборки равновероятны, то СКО будет пропорционален сумме квадратов ошибок и

$$F(\mathbf{x}) = \sum_{q=1}^{Q}(\mathbf{t}_q - \mathbf{a}_q)^T(\mathbf{t}_q - \mathbf{a}_q) = \sum_{q=1}^{Q}\mathbf{e}_q^T\mathbf{e}_q = \sum_{q=1}^{Q}\sum_{j=1}^{S^M}(e_{j,q})^2 = \sum_{i=1}^{N}(v_i)^2, \tag{6.15}$$

где $e_{j,q}$ j-ый элемент ошибки для q-ой обучающей пары. Таким образом, целевая функция MLP будет эквивалентна той, которая использовалась при выводе LM алгоритма.

2.2.2. Вычисление Якобиана для LM алгоритма

Алгоритм (6.14) требует предварительного вычисления Якобиана. Для его вычисления необходимо вычислять производные ошибок.
Вектор ошибок MLP применительно к LMA запишется в виде:

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_N \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{2,1} & \dots & e_{S^M,1} & e_{1,2} & \dots & e_{S^M,Q} \end{bmatrix}$$

Вектор параметров:

$$\mathbf{x}^T = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \dots & w^1_{S^1,R} & b^1_1 & \dots & b^1_{S^1} & w^2_{1,1} & \dots & b^M_{S^M} \end{bmatrix}.$$

При этом

$$N = Q \times S^M, n = S^1(R+1) + S^2(S^1+1) + \dots + S^M(S^{M-1}+1).$$

Для вычисления якобиана (6.8) необходимо вычислять элементы матрицы:

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial x_l}. \tag{6.16}$$

Определим чувствительность Марквардта:

$$\tilde{s}^m_{i,h} \equiv \frac{\partial v_h}{\partial n^m_{i,q}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}}, \tag{6.17}$$

где $h = (q-1)S^M + k$.
Теперь мы можем вычислить элементы якобиана. Для весов MLP

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w^m_{i,j}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times a^{m-1}_{j,q}, \tag{6.18}$$

или, если параметр – это смещение, то

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b^m_i} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h}. \tag{6.19}$$

Чувствительности Марквардта могут вычисляться в соответствии с рекуррентными соотношениями для стандартных чувствительностей алгоритма BP, за исключением последнего слоя MLP. Для последнего слоя

$$\tilde{s}^M_{i,h} = \frac{\partial v_h}{\partial n^M_{i,q}} = \frac{\partial e_{k,q}}{\partial n^M_{i,q}} = \frac{\partial (t_{k,q} - a^M_{k,q})}{\partial n^M_{i,q}} = -\frac{\partial a^M_{k,q}}{\partial n^M_{i,q}} = \begin{cases} -\dot{f}^M(n^M_{i,q}), & i = k \\ 0, & i \neq k \end{cases} \tag{6.20}.$$

Следовательно, когда на вход MLP подан вектор $\mathbf{p}_q$, а на выходе сформирован вектор $\mathbf{a}^M_q$ LMA инициализируется значениями

$$\tilde{\mathbf{S}}^M_q = -\dot{\mathbf{F}}^M(\mathbf{n}^M_q), \tag{6.21}$$

где $\dot{\mathbf{F}}^m(\mathbf{n}^m)$ — диагональная матрица с производными на главной диагонали.

Каждый столбец матрицы (6.19) должен быть распространен в обратном направлении, используя (5.12), чтобы сформировать одну строку якобиана. Столбцы также могут обратно распространяться совместно

$$\tilde{\mathbf{S}}_q^m = \dot{\mathbf{F}}^m(\mathbf{n}_q^m)(\mathbf{W}^{m+1})^T \tilde{\mathbf{S}}_q^{m+1}. \quad (6.22)$$

Общая матрица чувствительностей Марквардта для каждого слоя создается путем дополнения матрицами, вычисляемыми для каждого входного вектора **p**:

$$\tilde{\mathbf{S}}^m = \left[\tilde{\mathbf{S}}_1^m \middle| \tilde{\mathbf{S}}_2^m \middle| \cdots \middle| \tilde{\mathbf{S}}_Q^m\right]. \quad (6.23)$$

Обратите внимание, что для каждого входного вектора должно выполняться обратное распространение чувствительностей. Это связано с тем, что вычисляются производные для каждой отдельной ошибки, а не производные от суммы квадратов ошибок. Для каждого входного вектора, поданного на вход сети, вычисляется вектор ошибки и для каждого элемента вектора ошибки вычисляется одна строка матрицы Якоби. После того, как чувствительности будут распространены обратно, якобиан рассчитывается по формулам (6.18-6.19).

2.2.3. LM алгоритм обучения MLP

Объединяя формулы, приведенные выше, получим LM-алгоритм обучения MLP (рисунок 6.1).

LM алгоритм

1. Подать на вход MLP очередной входной вектор и вычислить реакцию MLP (5.1) и ошибку

$$\mathbf{e}_q = \mathbf{t}_q - \mathbf{a}_q^M.$$

Вычислить сумму квадратов ошибок для всех входных векторов, используя (6.15).

2. Вычислить якобиан (6.8). Для этого вычислить чувствительности в соответствии с рекуррентным соотношением (6.22), выполнив инициализацию (6.21). Составить общую матрицу чувствительностей Марквардта (6.23). Вычислить элементы матрицы Якоби (6.18-6.19).
3. Вычислить новые значения параметров с помощью (6.14)
4. Заново вычислить сумму квадратов ошибок при новых параметрах MLP. Если сумма квадратов меньше, чем значение, вычисленное на шаге 1, то поделить μ_k на θ. Запомнить обновленные значения параметров и перейти к шагу 1. Если сумма квадратов не уменьшилась, то умножить μ_k на θ и перейти к шагу 3.

Рисунок 6.1 – LM алгоритм

LM алгоритм сходится быстрее, чем любой ранее рассмотренный алгоритм. Конечно, он требует большего числа вычислений на каждой итерации, т.к.

включает инверсию матрицы. Тем не менее, LMA является быстрым **при среднем числе параметров сети**.

Основной недостаток LM алгоритма – это большой объем требуемой памяти (матрица Гессе имеет размер nxn, где n – число параметров). Когда число параметров очень большое, то использовать LMA непрактично (верхний предел использования LMA – несколько тысяч параметров).

В модуле NeuralNetworks 3.0 пакета Scilab имеется функция ann_FFBP_lm, реализующая LM алгоритм. Текст функции с комментариями приведен в приложении А. Формат вызова функции требует задания следующих параметров:

```
W = ann_FFBP_lm(P, T, N, af, mu, mumax, theta, itermax, mse_min, gd_min)
```

где mu - значение параметра μ_k LM алгоритма, по умолчанию mu=0.001; mumax – максимально возможное значение μ_k, по умолчанию mumax=100000000; theta – значение параметра θ LM алгоритма, по умолчанию theta=10. Остальные параметры функции аналогичны параметрам функции ann_FFBP_gd, которые были рассмотрены в лабораторной работе №5.

3. Варианты заданий и программа работы

3.1 Повторить теоретический материал, относящийся к алгоритмам обучения многослойного персептрона, основанных на методах оптимизации [4, 5].

3.2 Выполнить сравнительное исследование MLP, обученного на основе алгоритмов BP_gd и BP_lm, при решении задачи классификации. Для этого по аналогии с заданием п.3.2 лабораторной работы №5 выполнить следующее:

3.2.1 Сформировать два множества случайных точек данных (не менее 400 точек), которые располагаются выше (класс1) и ниже кривой (класс 2) $y=f(x)$ и отстоят от неё на расстояние d= $0{,}3|(y_{max} - y_{min})|$;

3.2.2 Отобразить классы и кривую $y=f(x)$ на двумерной плоскости;

3.2.3 Разработать программу обучения многослойного персептрона с архитектурой *R*-*S*-1 для классификации этих классов, в качестве активационных функций скрытого слоя использовать функции ann_tansig_active или ann_logsig_active, а в качестве активационной функции выходного слоя — ann_logsig_active, обучение персептрона выполнять с использованием функций ann_FFBP_gd и ann_FFBP_lm;

3.2.4 Используя значение скорости обучения lr=0.75 (для алгоритма BP_gd), выполнить сравнительное обучение MLP с архитектурой [*R*-*S*-1] двумя алгоритмами BP_gd и BP_lm при разных *S* (10,20,30,40) и при itermax =100. Построить и сравнить кривые обучения для 2-х алгоритмов при разных *S*.

3.2.5 Для различных MLP (при разных *S*), обученных в соответствии с **LM алгоритмом**, выполнить моделирование этих MLP и проверить качество классификации на **тестовом** множестве данных. Для этого сформировать тестовое множество данных аналогично п. 3.2.1, провести классификацию данных с помощью обученного MLP, отобразить точки полученных выходных классов и кривую $y=f(x)$ на двумерной диаграмме, вычислить точ-

ность правильной классификации при разных *S*. Сравнить точности правильной классификации для MLP, обученного на основе LM алгоритма с аналогичными значениями, полученными в предыдущей лабораторной работе, обратив внимание на значение itermax.

3.3 Выполнить сравнительное исследование MLP, обученного на основе алгоритмов BP_gdx и BP_lm, при решении задачи аппроксимации функции 2-х переменных z=*f*(*x*,*y*). Для этого по аналогии с заданием п.3.3 лабораторной работы №5 выполнить следующее:

3.3.1 Сформировать подмножества обучающих и тестовых данных. Для этого выбрать на плоскости (*x*,*y*) 500 случайных точек и определить в этих точках значение функции *z*=*f*(*x*,*y*). В качестве входного вектора использовать вектор **p**=[x;y], в качестве целевого значения *z*. Полученные данные разделить на 2 подмножества: обучающее (80% данных) и тестовое (20% данных).

3.3.2 Разработать программу обучения многослойного персептрона с архитектурой *R*-*S*-1 для аппроксимации функции *z*=*f*(*x*,*y*), в качестве активационных функций скрытого слоя использовать функции ann_tansig_active или ann_logsig_active, а в качестве активационной функции выходного слоя -- ann_purelin_active, обучение персептрона выполнять с использованием функции ann_FFBP_gdx и ann_FFBP_lm;

3.3.3 Выполнить обучение MLP (при квазиоптимальном *S*, определенном в предыдущей лабораторной работе, itermax=100) с помощью функций ann_FFBP_gdx и ann_FFBP_lm, построить и сравнить кривые обучения;

3.3.4 Используя тестовое подмножество данных, выполнить моделирование 2-х вариантов MLP, обученных с помощью 2-х разных функций, указанных выше. Построить сопоставительные графики значений функции *z*=*f*(*x*,*y*) и соответствующих значений на выходе MLP, вычислить СКО аппроксимации функции **на тестовом подмножестве**, сравнить со значениями СКО, полученными **на обучающем подмножестве**.

3.4 Подготовьте и защитите отчет по работе.

4 Методические рекомендации по выполнению работы

4.1 В задании п.3.2. исследуется задача бинарной классификации. Цель задания – убедиться в том, что MLP, обучаемый на основе LM алгоритма, обучается быстрее, чем BP_gd и строит более точную нелинейную границу решения между классами.

Это задание практически повторяет задание, сформулированное п. 3.2 лабораторной работы №5. Поэтому все, что Вам необходимо сделать – это адаптировать соответствующую часть программного кода лабораторной работы №5.

Поскольку в п.3.2.4. требуется сравнить кривые обучения для функций ann_BP_gd и ann_BP_lm, то необходимо модифицировать эти функции обучения, таким образом, чтобы они возвращали, кроме весов, значение СКО на каждой эпохе. Схема такой модернизации была указана в предыдущей лабораторной работе. Кроме этого, необходимо запоминать веса обученных MLP при разных

S, так как они понадобятся при выполнении моделирования и тестирования MLP (п.3.2.5). Для этого удобно использовать клеточные массивы. Пример соответствующего кода, выполняющего обучение на основе модифицированной функции ann_FFBP_gd1 и запоминающего ошибки и веса при разных *S*, приведен ниже:

```
…
   //Обучаем при разных S
   S=[10 20 30 40];
   mse_min=1e-11  //обеспечивают завершение алгоритма по значению itermax
   gd_min=1e-11

…
   MSE_gd=[];
   W_all_gd=cell(); // клеточный массив для хранения всех весов
   for k=1:length(S)
           N=[2 S(k) 1];
           [W_gd,out_mse_gd] = ann_FFBP_gd1(P,T,N,af,lr,itermax);
           MSE_gd=[MSE_gd; out_mse_gd'];
           W_all_gd{k}= W_gd;
  end;
```

В качестве примера на рисунке 6.2 изображены кривые обучения MLP для 2-х исследуемых алгоримов обучения BP_gd и BP_lm, полученные с помощью подобного кода при itermax=100.

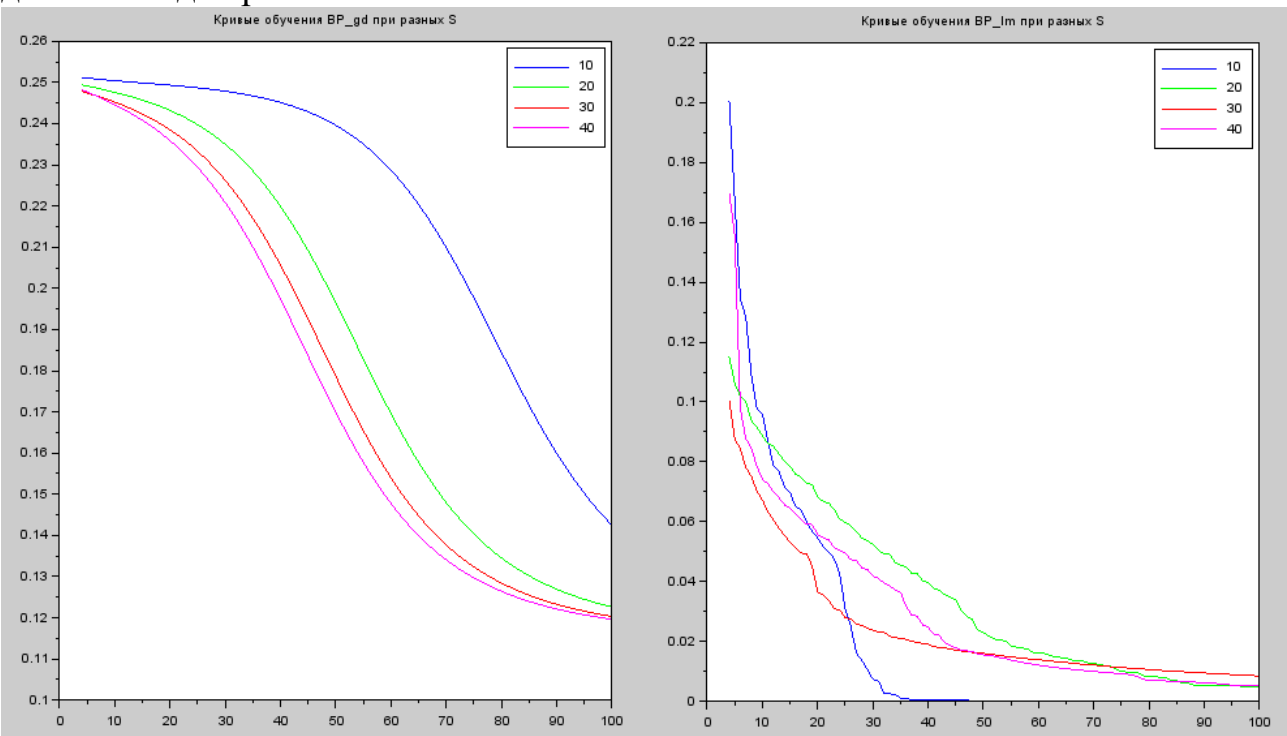


Рисунок 6.2 – Кривые обучения MLP для алгоритмов BP_gd и BP_lm

Из рисунка видно, что LM алгоритм сходится значительно быстрее и обеспечивает в ходе обучения уровень СКО намного ниже, чем алгоритм BP_gd.

Для тестирования MLP (задание п.3.2.5), обученного на основе LM алгоритма, используйте запомненные значения весов и функцию ann_FFBP_run. Тестирование выполняйте на тестовом подмножестве данных, на котором MLP не обучался. Используйте код тестирования, который Вы разработали в предыдущей лабораторной работе.

Пример выходных классов, формируемых обученным MLP на основе LM алгоритма, приведен на рисунке 6.3. Точности правильной классификации при *S*=[10,20,30,40] соответственно равны 0.9925, 0.995, 0.9975, 0.995. Это намного лучше по сравнению с результатами обучения на основе алгоритма BP_gd, полученными в предыдущей лабораторной работе.

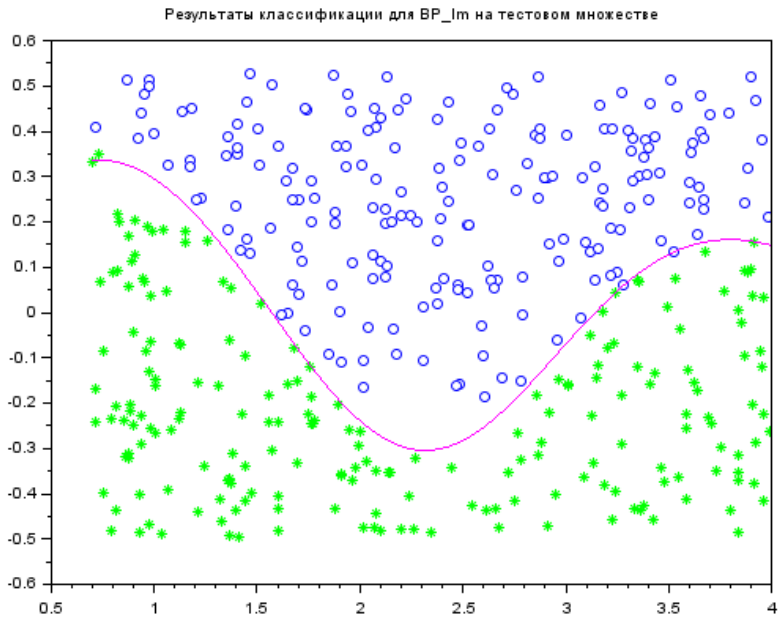


Рисунок 6.3 – Результаты классификации для тестового множества, точность классификации 0,9975 (*S*=30, LM алгоритм)

4.1 В задании 3.3 исследуется задача аппроксимации двумерной функции с помощью MLP. Цель задания – убедиться в том, что MLP, обучаемый на основе LM алгоритма, превосходит по скорости обучения и точности получаемой аппроксимации лучший из ранее рассмотреных алгоритмов обучения BP_gdx.

Это задание практически повторяет задание, сформулированное п. 3.3 лабораторной работы №5. Поэтому все, что Вам необходимо сделать – это адаптировать соответствующую часть программного кода лабораторной работы №5.

При выполнении задания в соответствии с п.3.3.3 отобразите кривые обучения для 2-х исследуемых алгоритмов BP_gdx и BP_lm на одном графике (рисунок 6.4).

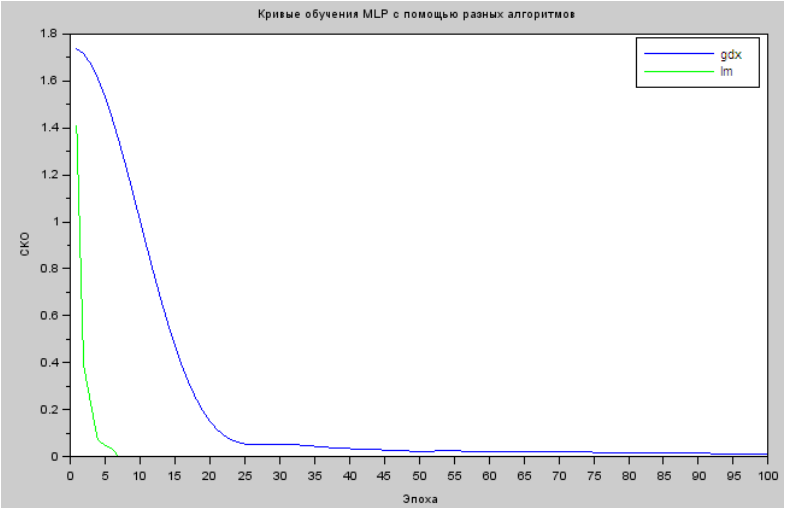


Рисунок 6.4 – Кривые обучения MLP (алгоритмы BP_gdx и BP_lm)

Как следует из рисунка 6.4, алгоритм BP_lm завершил обучение за 7 эпох. При этом обучение завершилось с минимальным значения СКО, равным mse_min=1e-5.

Для тестирования обученных нейросетей используйте функцию ann_FFBP_run (задание п. 3.3.4). По результатам тестирования необходимо построить и сравнить графики функции $z=f(x,y)$, определенной на тестовом множестве входных данных, и графики выходных сигналов MLP, обученных разными алгоритмами BP_gdx и BP_lm . В качестве примера на рисунок 6.5 изображены графики тестовых значений функции $z=f(x,y)$ и аппроксимирующих значений на выходе MLP, обученных функциями ann_BP_gdx и ann_BP_lm.

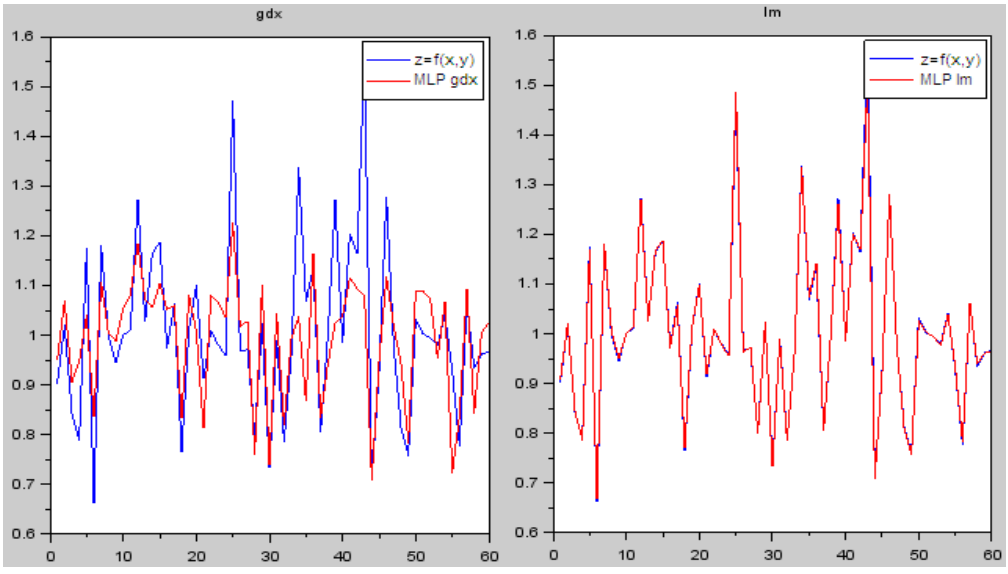


Рисунок 6.5 – Тестовые значения функции $z=f(x,y)$ и значения на выходе MLP, обученного с помощью функций ann_BP_gdx и ann_BP_lm

Из рисунка 6.5 следует, что алгоритм BP_gdx характеризуется бóльшими значениями СКО, чем BP_lm при обучении на 100 эпохах. В частности, для рассматриваемого примера указанные ошибки на тестовом подмножестве данных равны: для BP_gdx – 0.01378, для BP_lm – 0.00008.

5. Содержание отчета

5.1. Цель работы.
5.2. Вариант задания.
5.3. Схема MLP, решающего задачу классификации 2-х классов; программа обучения MLP бинарной классификации с использованием функций ann_BP_gd и ann_BP_lm; результаты обучения MLP при разных *S*; результаты тестировании обученного бинарного классификатора на основе MLP и вероятности правильной классификации при разных *S*.
5.4. Схема MLP для решения задачи аппроксимации функций; программа обучения MLP; кривые обучения MLP для 2-х функций ann_BP_gdx и ann_BP_lm; результаты тестирования MLP, обученных указанными функциями (сравнительные графики), значения СКО на этапе обучения и на этапе тестирования.
5.5. Выводы по результатам исследований.

6. Контрольные вопросы

6.1. Сформулируйте решение задачи оптимизации целевой функции на основе метода Ньютона.
6.2. Сформулируйте достоинства и недостатки метода Ньютона.
6.3. Сформулируйте условия, при которых метод Ньютона совпадает с SDA.
6.4. Запишите выражение для целевой функции алгоритма Левенберга-Марквардта.
6.5. Запишите выражение для градиента целевой функции алгоритма Левенберга-Марквардта.
6.6. Запишите общее выражение для якобиана векторной функции **v**(**x**) в алгоритме Левенберга-Марквардта.
6.7. Запишите выражение, аппроксимирующее матрицу Гессе в алгоритме Левенберга-Марквардта.
6.8. Запишите выражение основного алгоритма Левенберга-Марквардта.
6.9. При каком условии алгоритм Левенберга-Марквардта трансформируется в SDA.
6.10. При каком условии алгоритм Левенберга-Марквардта трансформируется в алгоритм Гаусса-Ньютона.
6.11. Определите чувствительность Марквардта.
6.12. Как вычислить элементы якобиана в алгоритме Левенберга-Марквардта, используя чувствительсность?
6.13. Как вычислить чувствительсность выходного слоя в алгоритме Левенберга-Марквардта?
6.14. Запишите рекуррентное соотношение для вычисления чувствительностей в алгоритме Левенберга-Марквардта.
6.15. Сформулируйте LM алгоритм.
6.16. Объясните назначение параметров функции ann_FFBP_lm.

6.17. При каких условиях функция ann_FFBP_lm прекращает процесс обучения?
6.18. Объясните код функции ann_FFBP_lm.

Приложение А. Функция ann_FFBP_lm

```
function W=ann_FFBP_lm(P, T, N, af, mu, mumax, theta, itermax, mse_min, gd_min)
///Обучение нейросети прямого распространения на основе алгоритма Левенберга-Марквардта.
// Примеры вызова:
//     W = ann_FFBP_lm(P,T,N)
//     W = ann_FFBP_lm(P,T,N,af,mu,mumax,theta,itermax,mse_min,gd_min)
// Параметры
//     P : Обучающий вход
//     T : Обучающие целевые значения
//     N : Вектор, задающий число нейронов каждого слоя, включая входной и выходной слои
//     af : Список имен активационных функций от 1-го до выходного слоев
//    mu : значение "мю" ЛМ алгоритма
//     mumax : максимально возможное "мю"
//     theta : множитель "тета" для "мю"
//     itermax : Максимальное число эпох обучения
//     mse_min : Минимальная ошибка  (Значение целевой функции)
//     gd_min : Минимальное значение градиента
//     W : Выходные веса и смещения
//
//1. ================Обработка списка аргументов функции=========================
    rhs=argn(2);

/ Проверка ошибки списка аргументов
    if rhs < 3; error("Expect at least 3 arguments, P, T and N");end
// Выбор значений аргументов по умолчанию
    if rhs < 4; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if rhs < 5; mu = 0.001; end
    if rhs < 6; mumax = 100000000; end
    if rhs < 7; theta = 10; end
    if rhs < 8; itermax = 1000; end
    if rhs < 9; mse_min = 1e-5; end
    if rhs < 10; gd_min =  1e-5; end

    if af == []; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if mu == []; mu = 0.001; end
    if mumax == []; mumax = 100000000; end
    if theta == []; theta = 10; end
    if itermax == []; itermax = 1000; end
    if mse_min == []; mse_min = 1e-5; end
    if gd_min == []; gd_min =  1e-5; end

//2.=======================Инициализация====================
// Инициализация сети и формирование списка имен производных активационных функций
    format(8);warning('off');
    W = ann_ffbp_init(N,[-1 1]); // Каждый элемент W – матрица весов одного из слоев
    itercnt = 0;
    af_d = strsubst(af,'ann_','ann_d_');
    mse = %inf;
    gd = %inf;
    A = ann_ffbp_init(N,[0 0]);
    tempW = A;
    train_N = size(P,2); // Размер обучающего множества

    // Инициалиазация GUI окна прогресса
    handles = ann_training_process();
    handles.itermax.string = string(itermax);
    handles.msemin.string = string(mse_min);
    handles.gdmax.string = 'inf';
    handles.gdmin.string = string(gd_min);
```

```
// Задание числа слоев и списков промежуточных переменных
layers = size(N,2)-1; // слои считаются от 1-го скрытого слоя до выходного слоя
n = list(0);
a = list(0);
m = list(0);
s = list(0);

// 3. ==================Реализация цикла эпох обучения==========================
   while mse > mse_min & itercnt < itermax & mu <= mumax & gd > gd_min

      mucnt = 0;
      // Прямое распространение – моделирование (аналогично функции ann_FFBP_run (формула 5.1))
      n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(P,2)); // Вычисляем и запоминаем сетевые выходы 1го слоя
      a(1) = evstr(af(1)+'(n('+string(1)+'))'); // Вычисляем и запоминаем активности на выходе 1го слоя
      for cnt = 2:layers
         n(cnt) = W(cnt)(:,1:$-1)*a(cnt-1) + repmat(W(cnt)(:,$),1,size(P,2)); // Вычисляем и запоминаем сетевые выходы
         a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))'); // Вычисляем и запоминаем активности на выходе каждого слоя
      end
      // Вычисление ошибки
      e = T - a($);
      [r,c] = size(a(layers)); //запоминаем размерность выходного слоя

      m(layers) = evstr(af_d(layers)+'(a('+string(layers)+'))'); Вычисление производных актив. функций выходного слоя (6.20)
      // s(layers) = -(m(layers).*.eye(N($),N($)));
      s(layers) = -(m(layers).*.ones(1,r)).*(ones(1,c).*.eye(r,r));  // Чувствительности выходного слоя –формула (6.21)

   for cnt = layers-1:-1:1 //Цикл для вычисления производных слоев и обратного  распространения чувствительностей
      Wpre = W(cnt+1)(:,1:$-1);     //Извлекаем из W и запоминаем веса следующего слоя
      a(cnt) = a(cnt).*.ones(1,N($));//Формируем матрицу активности слоя
      m(cnt) = evstr(af_d(cnt)+'(a('+string(cnt)+'))'); //Матрица производных F'(n) слоя
      s(cnt) = m(cnt).*(Wpre'*s(cnt+1)); // Матрица чувствительности слоя – формула (6.22)

   end

      // Вычисление элементов Якобиана – формула (6.18) и формирование матрицы Якоби
      Jj = [];
      jac = ann_calcjac(kron(P,ones(1,N($))),s(1)); // Для первого слоя
      Jj = [Jj jac s(1)'];
      for cnt = 2:layers
         jac = ann_calcjac(a(cnt-1),s(cnt)); // Для последующих слоев
         Jj=[Jj jac s(cnt)']; // добавление элементов в матрицу Якоби
      end

      mse = (mean(e.^2))
      mse2 = %inf;
      J = Jj;
      J2 = (J' * J);        // Формула (6.13)
      Je = J'*e(:);  // часть формулы (6.14)
      //Обновление параметров
      while  mse2 >= mse & mu <= mumax // пока новый СКО не станет меньше текущего СКО
         dx = -(J2 + (eye(J2)*mu)) \ (Je); // Вычисление обновления (формула 6.14)
         szpre = 0;
         for cnt = 1:layers       //Цикл для послойного обновления параметров
            sz = N(cnt)*N(cnt+1) + N(cnt+1); // Число параметров текущего слоя
            dx_part = dx(szpre+1:szpre+sz)  //Выделение части обновлений для текущего слоя
            A(cnt) = [matrix(dx_part(1:$-N(cnt+1)),N(cnt+1),N(cnt)) dx_part($-N(cnt+1)+1:$)]; // Реформатирование
            tempW(cnt) = W(cnt) + A(cnt); // обновление параметров слоя

            szpre = szpre + sz; // Начало индексов следующего слоя
         end
         //Вычисление выхода при новых значениях параметров
         y = ann_FFBP_run(P,tempW,af);
         e2 = T - y; // Вычисление новой ошибки
         mse2 = (mean(e2.^2)); //Вычисление нового СКО
         if  mse2 >= mse //Если СКО увеличился, то увеличить mu
            mu = mu*theta;
         end

      end
```

```
        mu = mu/theta;  // Т.к. новый СКО меньше текущего, то уменьшить mu
        if (mu < 1e-20)
            mu = 1e-20;
                // break
        end
        W = tempW; //Сохранить обновленные значения параметров

        //Критерий остановки
        mse = mean(e.^2);
        itercnt = itercnt + 1;
        //средний градиент
        gd = 2*sqrt(Je'*Je)/train_N;

       // Отображение  GUI  прогресса обучения
       if  itercnt == 1 then
          mse_max = mse;
          handles.msemax.string = string(mse_max);
          gd_max = gd;
          handles.gdmax.string = string(gd_max);
          mse_span = log(mse) - log(mse_min);
          iter_span = itermax;
          gd_span =  log(gd) - log(gd_min);
       end

        // Scilab 5.5  и выше
        handles.iter.value = round((itercnt/iter_span)*100);
        handles.mse.value = -(log(mse)-log(mse_max))/mse_span * 100;// round(((log(mse) - log(mse_min))/mse_span)*100);
        handles.gd.value = -(log(gd)-log(gd_max))/gd_span * 100; //round(((log(gd) - log(gd_min))/gd_span)*100);

        handles.itercurrent.string = string(itercnt);
        handles.msecurrent.string = string(mse);
        handles.gdcurrent.string = string(gd);

    end
endfunction
```

ЛАБОРАТОРНАЯ РАБОТА № 7

ИССЛЕДОВАНИЕ СОСТЯЗАТЕЛЬНЫХ СЕТЕЙ И СЕТЕЙ ВЕКТОРНОГО КВАНТОВАНИЯ

1 Цель работы

Углубление теоретических знаний в области обучения нейросетей без учителя, исследование свойств алгоритмов обучения состязательных сетей на основе правил Кохонена, приобретение практических навыков обучения самоорганизующихся карт Кохонена и сетей векторного квантования при решении задач классификации.

2 Основные теоретические положения

2.1 Обучение без учителя и самоорганизующиеся ИНС

При обучении без учителя **отсутствует информация о правильности реакции** ИНС на тот или иной входной вектор. Нейронная сеть самостоятельно обнаруживает взаимосвязь входных векторов и преобразует их в ассоциированную с ними выходную реакцию, т.е. в ИНС происходят процессы самоорганизации.

Самоорганизующиеся ИНС способны открывать отношения подобия во входных данных, обеспечивая тем самым **выделение во входном пространстве классов.**

Существует два основных типа самоорганизующихся ИНС:

1) ИНС, основанные на правиле обучения Хебба;

2) состязательные ИНС.

2.2 Правило обучения Хебба

Правило обучения Хебба заимствовано из биологических нейронных сетей. В соответствии с этим правилом **веса синаптических связей изменяются пропорционально активности** нейронов. Правило обучения Хебба можно записать в виде

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q), \tag{7.1}$$

где $a_i(q)$ – выход i- го нейрона; $p_j(q)$ – входное воздействие на j-ом входе нейрона. Изменение веса связи в соответствии с правилом не требует привлечения сведений о желаемой реакции нейрона.

Правило Хебба может быть записано в векторной форме

$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^{\mathrm{T}}(q). \tag{7.2}$$

Как в любом правиле обучения без учителя, обучение выполняется по реакциям на последовательность предъявляемых входных образов **p**(1**),** **p**(2),…,**p**(Q).

Приведенная форма правила Хебба **не ограничивает рост весов** связей по мере обучения. Поэтому в правило Хебба вводят **затухание**

$$\mathbf{W}(q)=\mathbf{W}(q-1)+\alpha\mathbf{a}(q)\mathbf{p}^{\mathrm{T}}(q)-\gamma\mathbf{W}(q-1)=(1-\gamma)\mathbf{W}(q-1)+\alpha\mathbf{a}(q)\mathbf{p}^{\mathrm{T}}(q), \quad (7.3)$$

где γ – скорость затухания. В этом случае максимальный вес связи ограничивается значением w_{ij} =α/ γ (получается, если все **a** и **p** равны 1).

2.3 Простая распознающая сеть Instar

Рассмотрим простую сеть из одного нейрона (рисунок 7.1).

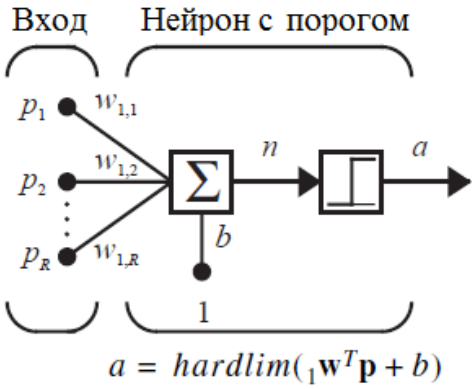


Рисунок 7.1 – Сеть Instar

Нейрон будет активен, когда $_1\mathbf{w}^T\mathbf{p} \geq -b$. Скалярное произведение будет максимальным при $\mathbf{p} = {}_1\mathbf{w}$. Т.е. сеть активна, когда **p** близко к $_1$**w.** Подбирая *b*, мы можем регулировать степень схожести **p** и $_1$**w.**

Если установить $b = -\|_1\mathbf{w}\|\|\mathbf{p}\|$, то нейрон будет активен, когда **p** точно соответствует направлению $_1$**w.** В этом случае нейрон будет распознавать только образ, соответсвующий $_1$**w.**

Если мы хотим, чтобы сеть распознавала образы, близкие к $_1$**w,** надо уменьшить *b*. Чем меньше *b*, тем большое число входных образов будут активировать сеть.

2.4 Состязательные сети

Рассмотрим слой нейронов, в котором на выходе нейрона с наибольшим значением сетевой функции устанавливается единичная активность. Такая сеть называется *состязательной* или *соревновательной*. Нейрон, имеющий наибольшее значение сетевой функции, называют нейроном-победителем, выигравшим “состязание”.

Состязательную функцию преобразования обозначают как **a**= **compet(n)** (рисунок 7.2). Она обеспечивает нахождение индекса *i** нейрона-победителя и устанавливает на его выходе 1:

$$a_i = \begin{cases} 1, i = i^* \\ 0, i \neq i^* \end{cases}, \text{ где } n_{i^*} \geq n_i, \forall i, \quad \text{и} \quad i^* \leq i, \forall n_i = n_{i^*}$$

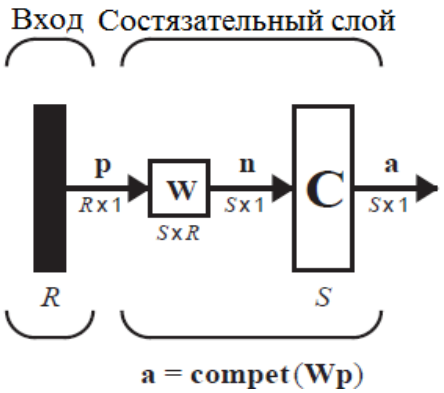


Рисунок 7.2 – Состязательная сеть

В состязательной сети устанавливается 1 на выходе того, нейрона, чей вектор весов **w** *по направлению "ближе" всего к входному вектору* **p.**

2.5 Правило обучения Кохонена

Отдельные нейроны состязательной сети на уровне сетевой функции соответствуют Instar-нейронам. Для решения задачи классификации с помощью состязательной сети можно строкам матрицы весов связей **W** присвоить значения желаемых векторов-прототипов классов. Т.к. векторы-прототипы нам заранее неизвестны, то для обучения сети можно использовать ассоциирующее Instar правило (7.3).

Поскольку для состязательной сети $a_i(q) \neq 0$ только для нейрона-победителя i^*, то

$$_i\mathbf{w}(q) = _i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - _i\mathbf{w}(q-1)) = (1-\alpha)_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q), \quad i = i^*. \tag{7.4}$$

$$_i\mathbf{w}(q) = _i\mathbf{w}(q-1), \quad i \neq i^*. \tag{7.5}$$

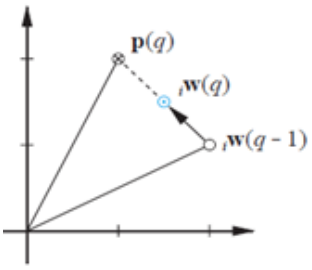


Рисунок 7.3

Правило (7.4-7.5) называют **правилом Кохонена**. В соответствии с этим правилом на каждой итерации вектор-строка $_i\mathbf{w}(q\text{-}1)$ матрицы весов, ближайшая к входному вектору $\mathbf{p}(q)$ (или имеющая наибольшее значение скалярного произведения с $\mathbf{p}(q)$), смещается по направлению к входному вектору (рисунок 7.2), формируя новое значение вектора весов $_i\mathbf{w}(q)$.

2.6 Самоорганизующиеся карты признаков (SOM)

SOM (self-organizing feature maps), предложенные Кохоненым, сначала определяют нейрон-победитель i^* (как в состязательной сети), а затем веса всех

нейронов **в окрестности нейрона-победителя** N_{i*} обновляются на основе правила Кохонена

$$_i\mathbf{w}(q) =_i \mathbf{w}(q-1)+\alpha(\mathbf{p}(q)-_i \mathbf{w}(q-1)) = (1-\alpha)_i\mathbf{w}(q-1)+\alpha\mathbf{p}(q), \quad i \in N_{i*}. \tag{7.6}$$

Окрестность N_{i*} содержит индексы всех нейронов, которые находятся на расстоянии радиуса d от нейрона–победителя i^*: $N_i(d)=\{j, d_{ij} \leq d\}$. Когда предъявляется входной вектор $\mathbf{p}(q)$ **веса** нейрона-победителя и нейронов в его окрестности смещаются в сторону вектора $\mathbf{p}(q)$. После многократных предъявлений соседние нейроны будут обучены распознаванию похожих входных векторов.

Карта признаков отображает образы из входного n-мерного пространства в пространство меньшей размерности. Кроме сокращения размерности пространства, карта признаков обеспечивает сохранение отношений топологического соседства входных образов. При этом геометрическая близость выходных классов на карте признаков означает близость соответствующих образов во входном пространстве признаков.

Примеры круговых окрестностей с радиусами d=1 и d=2 для двумерной карты изображены на рисунке 7.4.

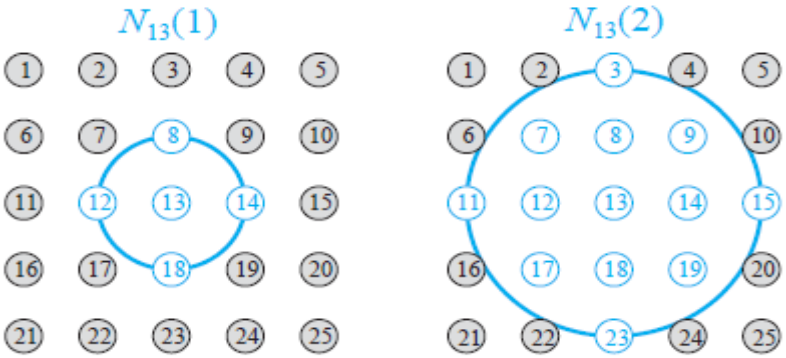


Рисунок 7.4 – Примеры круговых окрестностей

Нейроны могут быть также организованы в 1-мерные и 3-х мерные карты. Кохоненым были предложены **прямоугольные и гексагональные окрестности** с целью повышения эффективности реализации.

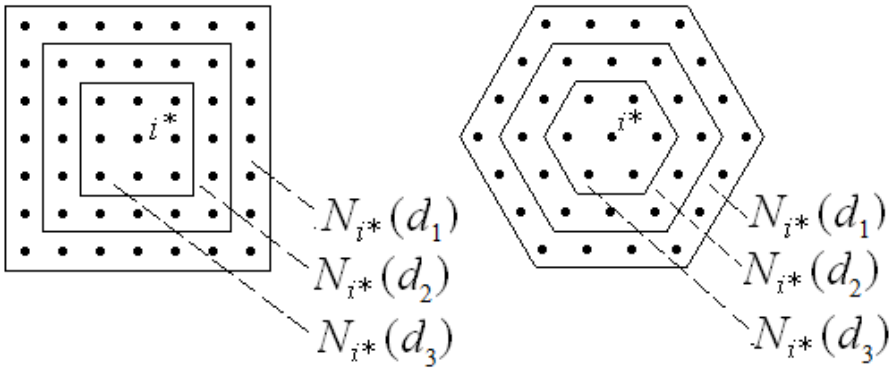


Рисунок 7.5 – Прямоугольная и гексогональные окрестности

На практике множество Ni*(*d*) и коэффициент обучения меняют свои значения динамически в процессе обучения. Процесс обучения начинается при расширенном множестве Ni*(*d*) и больших значениях α. По мере обучения количество НЭ, включаемых в $N_{i*}(d)$ уменьшается, одновременно уменьшается и значение α. При обучении учёт нейронов, образующих окрестность $N_{i*}(d)$, осуществляется с помощью функции соседства Λ(*i, i**), которая равна 1 при *i*=*i** и уменьшается по мере увеличения расстояния от нейрона *i** до нейрона *i* в выходном топологическом пространстве. В этом случае правило обучения записывается в виде:

$$\Delta_i \mathbf{w}(q) = \alpha(q) \cdot \Lambda(i, i^*)(\mathbf{p}(q) -_i \mathbf{w}(q-1)) . \quad \textbf{(7.7)}$$

Например, функция Λ(*i, i**) может быть задана выражением

$$\Lambda(i, i^*) = \exp(-|r_{ii^*}|^2 / 2d^2), \quad (7.8)$$

где r_{ii^*} – расстояние между нейронами *i* и *i** в выходном двумерном пространстве; *d* – параметр, задающий “диаметр” функции соседства. Диаметр *d* и скорость обучения α уменьшаются в зависимости от *q*, например, по экспоненте.

2.7 Обучение на основе векторного квантования (LVQ)

LVQ (learning vector quantization) сети являются гибридными (рисунок 7.6).

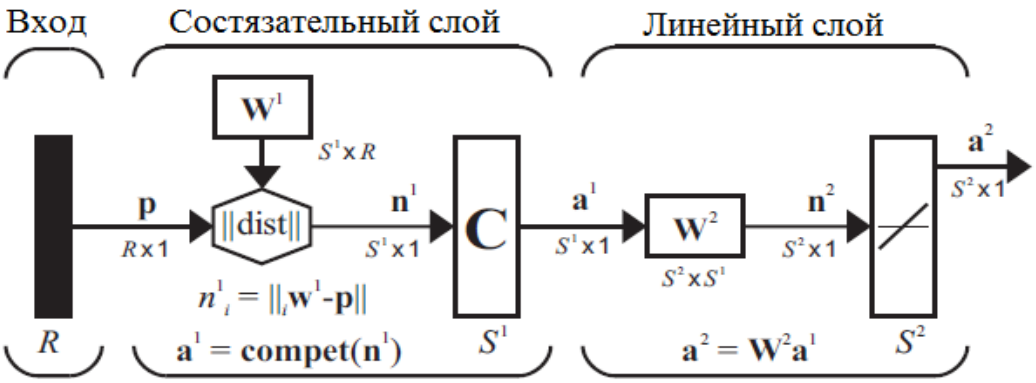


Рисунок 7.6 – Сеть векторного квантования

Как и в состязательной сети, 1-ый слой определяет подклассы входного пространства. При этом вместо скалярного произведения вычисляется прямое расстояние между векторами. Это позволяет не выполнять нормализацию входных векторов.

Второй слой комбинирует подклассы в один класс. Для второго слоя столбцы матрицы $\mathbf{W}^2$ представляют подклассы, а строки – классы. $\mathbf{W}^2$ содержит в каждом столбце по одной 1. Единицы в строках отображают класс подкласса:

$$(w^2_{ki} = 1) \Rightarrow \text{подкласс } i \text{ принадлежит классу } k . \quad (7.9)$$

Процесс комбинации подклассов в классы позволяет сетям LVQ создавать сложные границы между классами.

2.8 Правило обучения сетей векторного квантования

Правило комбинирует обучение без учителя и с учителем. Оно использует множество обучающих примеров $\{\mathbf{p}_1,\mathbf{t}_1\}$, $\{\mathbf{p}_2,\mathbf{t}_2\},\ldots,\{\mathbf{p}_Q,\mathbf{t}_Q\}$. Каждый целевой вектор содержит только одну единицу, которая обозначает класс принадлежности входного вектора. Перед началом обучения формируется матрица $\mathbf{W}^2$ и нейронам 1-го слоя назначается класс принадлежности (выходной нейрон). Обычно равное количество нейронов 1-го слоя связывается с каждым выходным нейроном.

Затем веса 1-го слоя обучаются **на основе модифицированного правила Кохонена**. Если **p** классифицируется корректно ($i^* \in$ классу k), то

$$ {}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)), \text{ если } a^2_{k^*} = t_{k^*} = 1 \quad . (7.10) $$

Если **p** классифицируется не корректно ($i^* \notin$ классу k), то

$$ {}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)), \text{ если } a^2_{k^*} = 1 \neq t_{k^*} = 0 \quad . (7.11) $$

Таким образом, нейроны скрытого слоя смещаются в сторону векторов, которые попадают в класс, для которого они формируют подкласс, и удаляются от векторов, которые относятся к другому классу.

Правило LVQ чувствительно к инициализации весов 1-го первого слоя. Из-за этого некоторые нейроны не могут «попасть» в свои подклассы. Имеется **усовершенствованное правило LVQ2.** Если возникает ситуация неверной классификации, то дополнительно подстраиваются со знаком «+» веса ближайшего нейрона к входному вектору, который даёт верную классификацию.

2.7 Функции Scilab Neural Network 3.0 для обучения состязательных сетей и сетей векторного квантования

В модуле Scilab Neural Network 3.0 реализованы следующие функции для организации обучения без учителя:

- ann_COMPET — функция обучения состязательной сети;
- ann_COMPET_visualize2d — обучение состязательной сети с 2D визуализацией;
- ann_COMPET_visualize3d — обучение состязательной сети с 3D визуализацией;;
- ann_COMPET_run — моделирование состязательной сети;
- ann_SOM — блочное обучение SOM;
- ann_SOM_online — последовательное обучение SOM;
- ann_SOM_visualize2d — обучение SOM с 2D визуализацией;
- ann_SOM_visualize3d — обучение SOM с 3D визуализацией.
- ann_SOM_run — функция моделирования SOM;
- ann_LVQ1 — обучение ИНС векторного квантования ;
- ann_LVQ_run — моделирования ИНС векторного квантования.

Рассмотрим функцию для обучения состязательной сети ann_COMPET (код функции с комментариями приведен в приложении А). Синтаксис вызова функции:

```
[W, b] = ann_COMPET(P,N)
[W, b] = ann_COMPET(P,N,lr,lr_c,itermax),
```

где P – матрица обучающих примеров; N – число нейронов в состязательном слое (число классов); lr – скорость обучения (по умолчанию 0.1); lr_c – скорость обучения для смещений (по умолчанию 0.1); itermax – максимальное число эпох обучения (по умолчанию 100); W – выходная матрица весов связей; b – вектор смещений.

Ниже приведен пример применения функции:

```
x = rand(2,10);
x(:,1:5) = x(:,1:5) + 1;
plot(x(1,:),x(2,:),'.');
[W,b] =  ann_COMPET(x,4);close;
plot(W(:,1), W(:,2), 'or');
```

В примере формируется двумерный случайный массив чисел x в диапазоне от 0 до 1 с равномерным распределением. Затем первая часть массива со столбцами с 1 по 5 увеличивается на 1. Массив x представляет собой матрицу обучающих примеров, которые на рисунке 7.7 обозначены голубыми точками. Вызов ann_COMPET(x,4) обеспечивает обучение состязательной сети для разделения обучающих примеров на 4 класса. Результаты обучения (значения весов нейронов) представлены красными кружочками на рисунке 7.7.

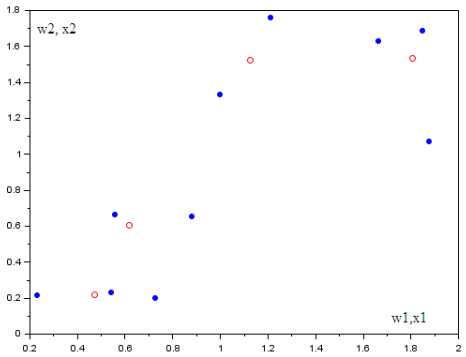


Рисунок 7.7 – Пример обучения состязательной сети

Функции ann_COMPET_visualize2d и ann_COMPET_visualize3d вызываются аналогично. Эти функции позволяют визуализировать несложные примеры обучения либо на плоскости, либо в 3-мерном пространстве.

Функция ann_SOM предназначена для блочного обучения состязательной сети, реализуемой в виде самоорганизующейся карты. Синтаксис вызова функции:

```
W = ann_SOM(P)
W = ann_SOM(P,N,itermax,steps,NS,topfcn,distfcn),
```

где P – матрица обучающих примеров; N – массив, представляющий структуру карты признаков (по умолчанию – [8 8]); itermax – максимальное число эпох обучения (по умолчанию - 200); steps – число шагов сокращения NS (по умолчанию - 100); NS – начальный размер окрестности (по умолчанию - 3); topfcn – функция, определяющая топологию сети (по умолчанию ann_som_gridtop – прямоугольная сетка); distfcn – функция, определяющая способ вычисления расстояния соседства (по умолчанию ann_som_linkdist).

Ниже приведен пример применения функции ann_SOM:

```
x = rand(2,10);
x(:,1:5) = x(:,1:5) + 1;
W =  ann_SOM(x, [2 2]);
[y, classes] = ann_SOM_run(W, x)
```

В примере формируется массив обучающих данных и создается сеть в виде самоорганизующейся карты признаков Кохонена размером 2x2 с прямоугольной топологией. В результате обучения функция ann_SOM вернет матрицу весов W. Вызов функции моделирования ann_SOM_run обеспечивает проверку функционирования обученной сети. Так как сеть содержит 4 нейрона, то обучающие примеры будут отнесены к одному из 4-х классов, распознаваемых сетью. При этом значением переменной y для каждого примера из x являются бинарные векторы с 1 в позиции нейрона-победителя, а значением переменной classes – номер класса каждого входного примера:

```
classes =  4.  4.  4.  3.  3.  1.  1.  2.  2.  1.
 y =       0.  0.  0.  0.  0.  1.  1.  0.  0.  1.
           0.  0.  0.  0.  0.  0.  0.  1.  1.  0.
           0.  0.  0.  1.  1.  0.  0.  0.  0.  0.
           1.  1.  1.  0.  0.  0.  0.  0.  0.  0.
```

Функция ann_SOM_online имеет сходный набор параметров и вызывается аналогично ann_SOM. Отличие заключается в том, что функция осуществляет обучение в последовательном режиме, т.е. матрица весов корректируется после предъявления каждого обучающего примера. Функции ann_SOM_visualize2d и ann_SOM_visualize3d осуществляют визуализацию процесса обучения в блочном режиме (вызываются с тем же набором параметров, что и ann_SOM).

В состав модуля Scilab Neural Network 3.0 для работы с самоорганизующимися картами входят следующие функции, определяющие топологию карт и способ вычисления расстояний соседства:

neuron_pos = ann_som_gridtop(N) — прямоугольная топология;
neuron_pos = ann_som_hextop(N) — гексагональная топология;
neuron_pos = ann_som_randtop(N) — случайная топология;

neuron_dist=ann_som_mandist(neuron_pos) — манхэттенское расстояние;

neuron_dist= ann_som_eudist(neuron_pos) — евклидово расстояние;
neuron_dist= ann_som_linkdist(neuron_pos) — расстояние связи;
neuron_dist= ann_som_boxdist(neuron_pos) — блочное расстояние;

Для визуализации топологии самоорганизующей сети можно использовать функции ann_som_plot2d и ann_som_plot3d. Примеры использования этих функций для визулизации топологии сетей в 2- и 3-х мерном пространствах:

```
N = [5, 5];
neuron_pos = ann_som_hextop(N);
ann_som_plot2d(neuron_pos);

N = [4, 4, 4];
neuron_pos = ann_som_gridtop(N);
ann_som_plot3d(neuron_pos);
```

Результаты визуализации изображены на рисунке 7.8.

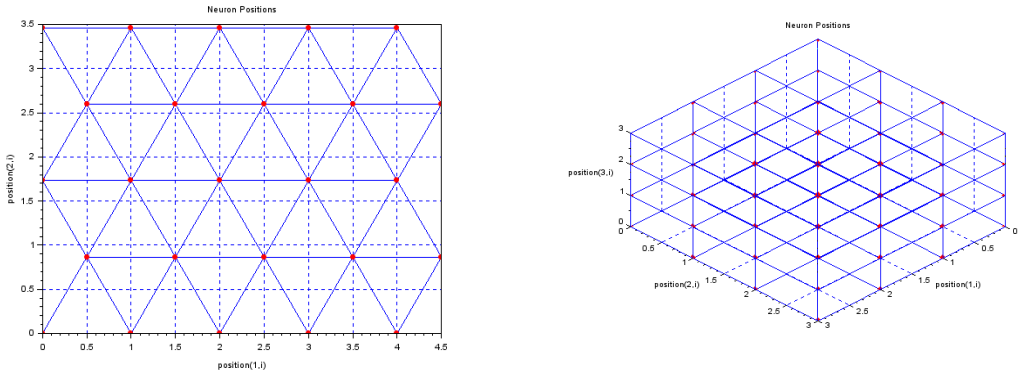


Рисунок 7.8 – 2D и 3D гексагональные топологии

В качестве примера вызова функции, обеспечивающей вычисление расстояний соседства, рассмотрим функцию ann_som_mandist(neuron_pos), вычисляющую манхеттенское расстояние. Пусть заданы позиции 3-х нейронов в 3D в виде столбцов матрицы neuron_pos:

```
neuron_pos = [1 2 3;
              4 5 6;
              7 8 9].
```

Тогда, например, манхеттенское расстояние между нейронами 1 и 2 , 1 и 3 будет равно:

```
neuron_dist(1,2)=sum(|1-2)|+|4-5|+|7-8|)=3,
neuron_dist(1,3)=sum(|1-3)|+|4-6|+|7-9|)=6.
```

Поэтому вызов ann_som_mandist(neuron_pos) вернет матрицу всех расстояний между указанными нейронами:

```
neuron_dist=ann_som_mandist(neuron_pos)
neuron_dist  =
```

```
    0.   3.   6.
    3.   0.   3.
    6.   3.   0.
```

Вызов функций ann_som_eudist(neuron_pos), ann_som_linkdist(neuron_pos), ann_som_boxdist(neuron_pos) выполняется аналогично.

Для обучения и моделирования сетей векторного квантования применяются соответственно функции ann_LVQ1 и ann_LVQ_run. Форматы вызова функций:

```
[W, b] = ann_LVQ1(P, T, N2);
[W, b] = ann_LVQ1(P, T, N2, lr, itermax);
[y, classes] = ann_LVQ_run(W, P),
```

где T – целевые классы, задаваемые в виде бинарных векторов-состояний выходов сети; N2 – количество нейронов состязательного слоя (количество подклассов). Ниже приведен пример применения этих функций для разделения множества входных данных на 4 подкласса с объединением в 2 класса:

```
x = rand(2, 10);
x(:,1:5) = x(:,1:5) + 1;
T = [1 1 1 1 1 0 0 0 0 0;
       0 0 0 0 0 1 1 1 1 1]
[W,b] =  ann_LVQ1(x, T, 4); close;
plot(x(1,:),x(2,:),'.');
plot(W(:,1), W(:,2), 'or');
[y, classes] = ann_LVQ_run(W, x)

classes  =
  1.   1.   1.   1.   1.   2.   2.   2.   2.   2.
 y  =
  1.   1.   1.   1.   1.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   1.   1.   1.   1.   1.
```

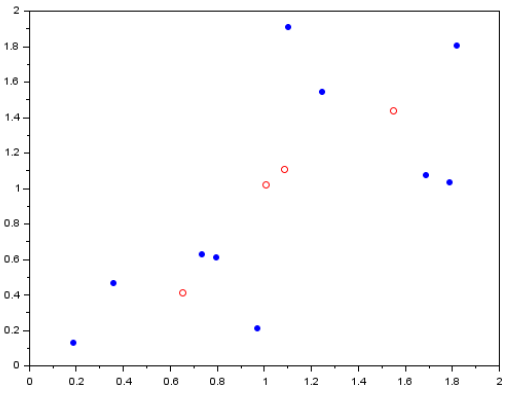


Рисунок 7.9 – Пример обучения LVQ сети

3. Варианты заданий и программа работы

3.1 Повторить теоретический материал, относящийся к алгоритмам обучения состязательных нейронных сетей и сетей векторного квантования [1, 5].

3.2 Сформировать в соответствии с таблицей 7.1 множество входных данных P для обучения SOM. Для генерации данных использовать генератор случайных чисел с равномерным распределением в диапазонах, указанных в таблице 7.1.

Таблица 7.1 – Варианты заданий

Вариант	Пространство	Функции топологии и расстояния	Кол-во классов	Диапазоны значений вх. данных по каждому измерению
1	2D	gridtop, eudist	16	0.0-4.0
2	2D	hextop, linkdist	16	0.0-4.0
3	2D	randtop, mandist	16	0.0-4.0
4	2D	gridtop, linkdist	36	0.0-6.0
5	2D	hextop, boxdist	8	0.0-8.0
6	2D	randtop, eudist	8	0.0-8.0
7	2D	gridtop, mandist	25	0.0-5.0
8	3D	hextop, linkdist	27	0.0-3.0
9	3D	randtop, linkdist	27	0.0-3.0
10	3D	gridtop, eudist	27	0.0-3.0
11	3D	hextop, eudist	8	0.0-2.0
12	3D	randtop, linkdist	8	0.0-2.0
13	3D	gridtop, mandist	8	0.0-2.0
14	3D	hextop, linkdist	64	0.0-4.0
15	3D	randtop, linkdist	64	0.0-4.0

3.3 Написать программу, создающую и обучающую SOM, заданной топологии (в соответствии с таблицей 7.1), на сгенерированных данных.

3.4 Визуализировать топологию SOM в начале и конце обучения.

3.5 Сгенерировать множество входных векторов P в виде кластеров с помощью функции ann_gencluster(M, D, C, Std), где M – число векторов в кластере, D – размерность пространства, C – число кластеров, Std – стандартное отклонение относительно центра кластера. При этом распределить координаты векторов P в пределах заданных диапазонов входного пространства. Обучить SOM на этих данных, определить нейроны-победители, распознающие кластеры, построить диаграмму распределения кластеров и узлов обученной SOM.

3.6 Используя данные, сгенерированные в соответствии с п.3.5, создать и обучить LVQ сеть, которая:

- содержит в скрытом слое такое же число нейронов как SOM, заданная по варианту в соответствии с таблицей 7.1;

- группирует каждые 2 (при четном числе подклассов) или 3 (при нечетном числе подклассов) подкласса, формируемые скрытым слоем, в один класс.

3.7 Оценить точность обучения LVQ классификатора.

3.8. Подготовить и защитить отчет.

4. Методические рекомендации по выполнению работы

4.1 В задании 3.2 небходимо сформировать множество входных данных P для обучения SOM. Используйте для этого генератор равномерно-распределенных случайных чисел grand в соответствии с кодом, приведенным ниже. Задайте число классов C, структуру SOM N и диапазон rng генерируемых значений с учетом варианта задания:

```
C = 16; // число классов (по варианту)
N=[4 4]; // структура сети SOM (исходя из варианта 2D или 3D)
D=max(size(N)); // число измерений

M = 20; // выбираем число точек каждого класса
rng = [0 4] // диапазон значений данных (по варианту)

// генерируем равномерно распределенные данные (матрица размером Dx (CxM))
P=grand(D,C*M,'unf', rng(1),rng(2));

figure(0); clf
title('Исходные данные')
//диаграмма распределения данных
scatter(P(1,:), P(2,:))  //это для 2D; для 3D добавьте в scatter параметр  P(3,:))
```

На рисунке 7.10 изображены примеры диаграмм распределения сгенированных точек данных.

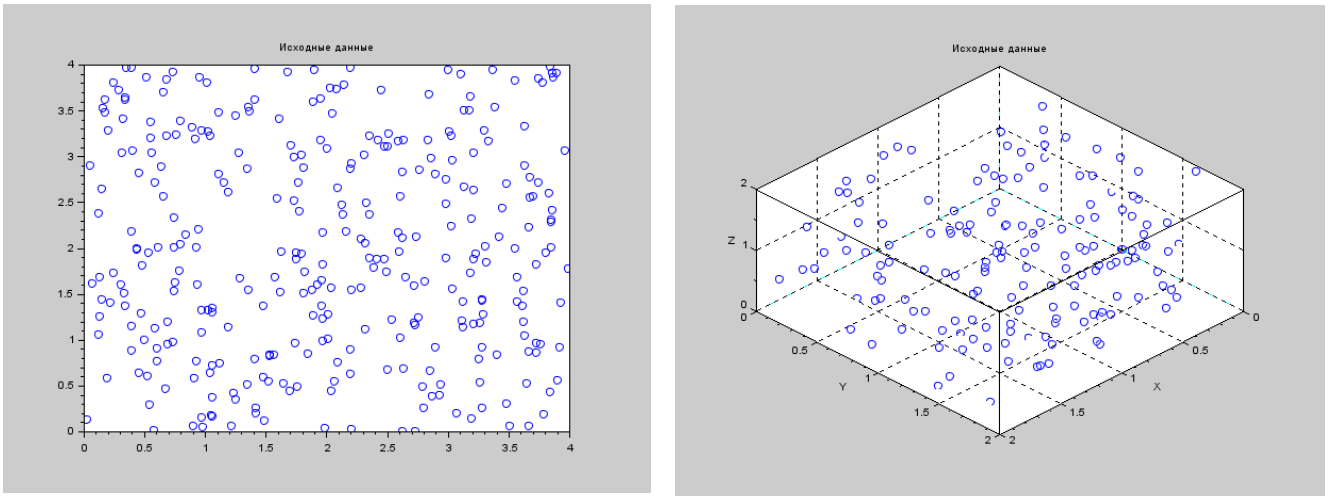


Рисунок 7.10 – Диаграмма распределения данных (слева - 2D, справа - 3D)

4.2. Для создания и обучения SOM заданной топологии в соответствии с заданием 3.2. адаптируйте код, приведенный ниже. На рисунках 7.11 и 7.12 представлены топологии нейросети до обучения и после обучения. Так как входные данные распределены равномерно, то нейроны в ходе обучения должны стремиться к расположению в узлах сетки, заданной топологии (на рисунках 7.11 и 7.12 прямоугольной).

```
//задание функции топологии
topfcn = 'ann_som_gridtop'  // или 'ann_som_hextop" или "ann_som_randtop'
// задание функции расстояния
distfcn = 'ann_som_mandist' // или 'ann_som_eudist' или 'ann_som_linkdist' или
                            // или 'ann_som_boxdist'

//Обучение SOM
W = ann_SOM_visualize2d(P, N, 200, 100, 3, topfcn, distfcn);

//вызов функции топологии для вычисления позиций нейронов
neuron_pos=evstr(topfcn +'(N)');
//вызов функции вычисления расстояний
neuron_dist=evstr(distfcn +'(neuron_pos)');

figure(2);clf
drawlater
subplot(1,2,1)
// визуализация начальной топологии 2D SOM
// для визулизации 3D SOM замените вызов ann_som_plot2d на
// ann_som_plot3d(neuron_pos',neuron_dist)
ann_som_plot2d(neuron_pos',neuron_dist)
title('Начальная топология SOM')
drawnow

drawlater
subplot(1,2,2)
// визуализация топологии SOM после обучения
// для визулизации 3D SOM замените вызов ann_som_plot2d на
// ann_som_plot3d(neuron_pos',neuron_dist)
ann_som_plot2d(W, neuron_dist)
title('Топология SOM после обучения')
drawnow

// классификация входных данных обученной SOM
// y - выход состязательного слоя
// classes - индексы нейрона победителя
[y, classes] = ann_SOM_run(W,P)
```

4.3. Для проверки возможности обнаружения кластеров во входных данных с помощью SOM сгенерируйте множество входных векторов с помощью вызова функции ann_gencluster(M, D, C, Std):

```
// формирование кластеров данных
diap=(rng(2)-rng(1));      // диапазон данных
std=diap/(10*C);           // стандартное отклонение данных ль центра кластера
P = diap*ann_gencluster(M,D,C,std); // генерация кластеров данных

// т.к. функция ann_genclaster формирует кластеры данных по очереди группами
// из M элементов, то выполняем случайную выборку значений из P
maxindP=max(size(P))
randind=int(maxindP*rand(1,maxindP))+1 // генерируем случайные индексы
P1=P(:,randind)  // делаем выборку из P  по случайным индексам
```

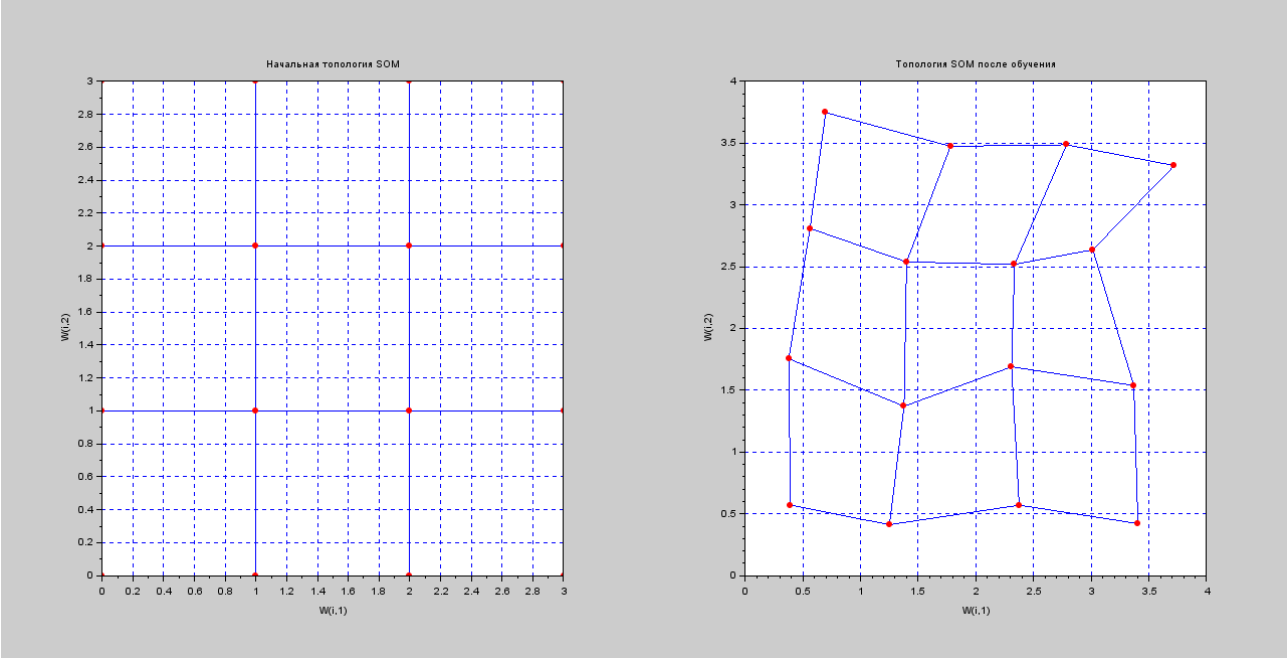


Рисунок 7.11 – Топология 2D SOM в начале (слева) и конце обучения (справа)

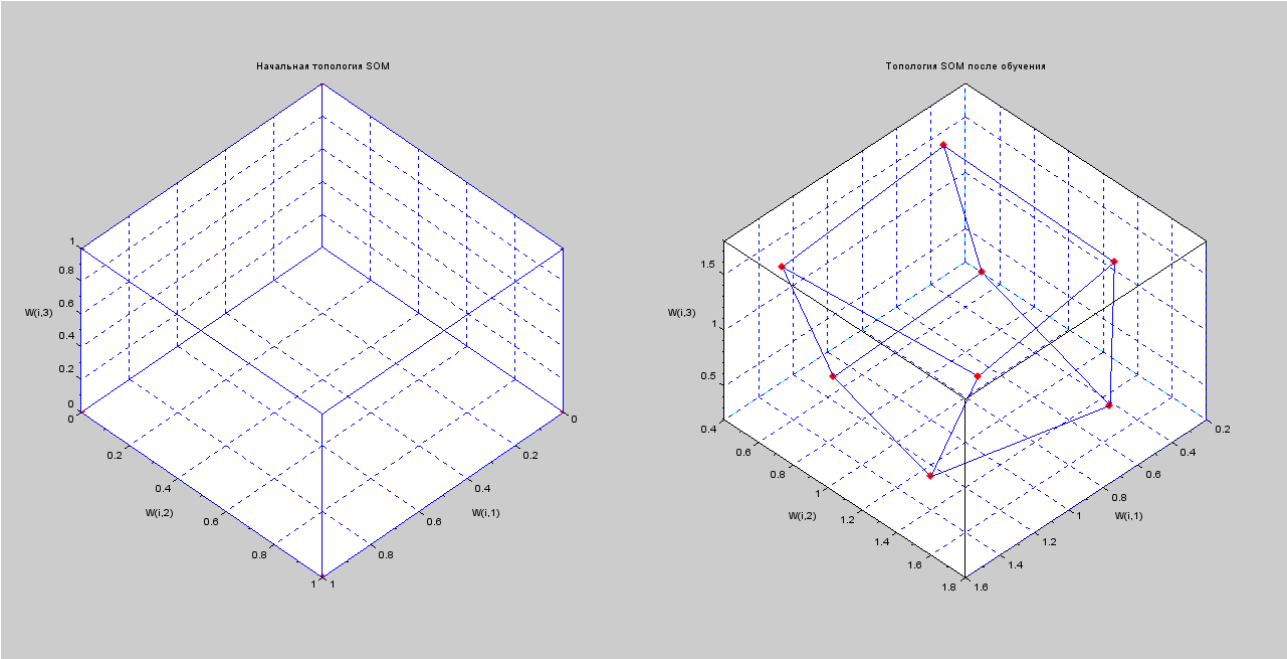


Рисунок 7.12 – Топология 3D SOM в начале (слева) и конце обучения (справа)

После формирования кластеров данных заново обучаем SOM на данных из множества P1. При этом следует воспользоваться кодом из п.4.2, всюду заменив P на P1. В ходе обучения функция ann_SOM_visualize2d (или ann_SOM_visualize3d) автоматически построит диаграмму распределения кластеров и узлов SOM (рисунок 7.13)

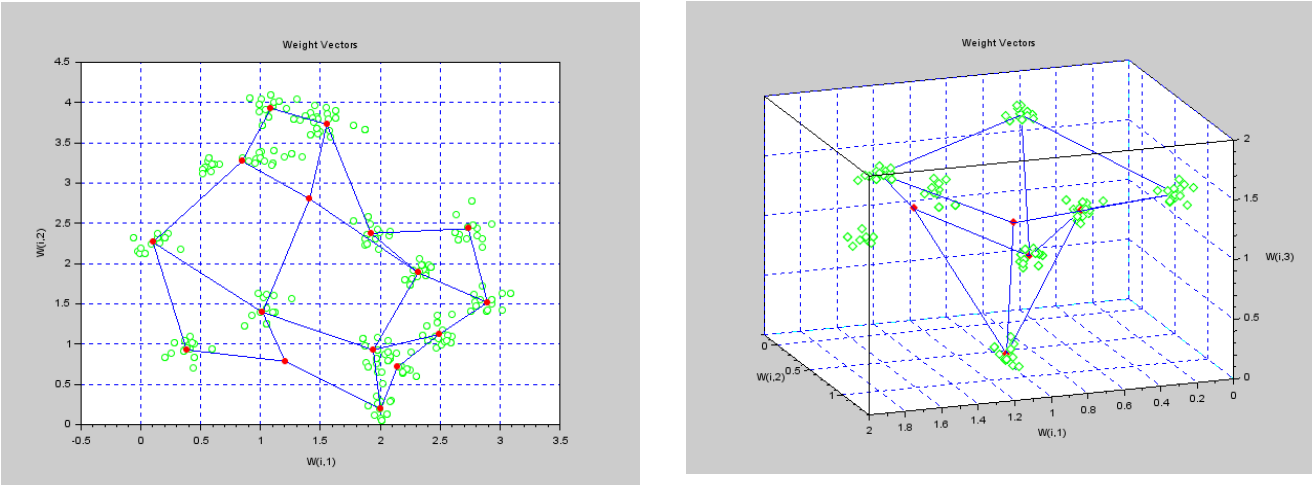


Рисунок 7.13 – Диаграмма распределения кластеров данных и узлов обученной SOM (слева - 2D, справа - 3D)

4.4. Для выполнения задания из п.3.3, в котором требуется обучить LVQ сеть на данных из P1, полученных в предыдущем пункте, адаптируйте код ниже под своё задание:

```
// Формируем обучающще мн-во для LVQ сети на основе P1
Plvq=P1
lenP=max(size(Plvq))
numC=int(C/2)
Tlvq=zeros(numC,lenP) // инициализируем T нулями
//формируем правильное мн-во T, объединив каждый 2-ой подкласс
for t=1:lenP
    Tlvq(ceil(classes(t)/2),t)=1
End

//Обучение LVQ
[Wlvq,blvq] =  ann_LVQ1(Plvq,Tlvq,C,0.01,400);
// проверка на обучающем множестве
[y_lvq, classes_lvq] = ann_LVQ_run(Wlvq, Plvq)
// точность обучения
acc=sum(Tlvq==y_lvq)/(numC*lenP)
```

В результате запуска функции [Wlvq,blvq] = ann_LVQ1(Plvq,Tlvq,C,0.01,400) в отдельном окне будет отображаться прогресс обучения. После обучения в командном окне Scilab можно отобразить точность обучения acc сети LVQ. Для примера с числом классов C=16 она составила 0,967, а для C=8 — 0,996.

5. Содержание отчета

5.1 Цель работы.
5.2 Вариант задания.
5.3 Схемы SOM и LVQ сетей заданной архитектуры, топология SOM до обучения и после обучения, листинги программ с комментариями, таблица с векторами-прототипами и номерами нейронов-победителей для SOM, таблица с векторами-прототипами и номерами классов для LVQ сети.
5.4 Выводы

6. Контрольные вопросы

6.1 Что понимается под термином «обучение без учителя»?
6.2 Назовите основные типы самоорганизующихся ИНС.
6.3 Сформулируйте и запишите правило обучения Хебба в его исходном виде.
6.4 Запишите правило обучения Хебба с затуханием.
6.5 Нарисуйте схему сети Instar и запишите правило её обучения.
6.6 Нарисуте схему состязательной сети и определите функцию её преобразования.
6.7 Запишите правило обучения Кохонена для состязательной сети. Как графически интепретируется правило?
6.8 Запишите правило обучения Кохонена для самоорганизующейся карты.
6.9 Объясните понятие окрестности для нейрона-победителя. Приведите примеры разных видов окрестностей.
6.10 Сформулируйте правило обучения SOM с использованием функции соседства. Определите функцию соседства.
6.11 Нарисуйте схему сети векторного квантования. Объясните процесс вычислений в этой схеме.
6.12 Как назначаются веса второго слоя сети векторного квантования?
6.13 Сформулируйте модифицированное правило Кохонена для сети векторного квантования.
6.14 Какие проблемы возникают при обучении состязательных сетей и как их решают на практике?
6.15 В чем особенность усовершенстовованного правила обучения LVQ2?
6.16 Объясните и приведите пример вызова функции ann_COMPET.
6.17 Объясните и приведите пример вызова функции ann_SOM.
6.18 Объясните и приведите пример вызова функции ann_LVQ1.

Приложение А. Функция обучения состязательной сети

```
function [W, b]=ann_COMPET(P, N, lr, lr_c, itermax)
// Функция обучения состязательной сети.
//
// Обработка входных аргументов функции
rhs=argn(2);

if rhs < 2; error("Expect at least 2 arguments, P and N");end
if rhs < 3; lr = 0.1; end  // Скорость обучения по умолчанию 0.1
if rhs < 4; lr_c = 0.1; end // Скорость обучения смещения по умолчанию-  0.1
if rhs < 5; itermax = 100; end // Число эпох обучения по умолчанию - 100

if lr == []; lr = 0.1; end
if lr_c == []; lr_c = 0.1; end
if itermax == []; itermax = 100; end

[W, b] = ann_compet_init(P, N); // инициализация сети
iter_span = itermax;

// Инициализация GUI прогресса обучения
handles = ann_training_process();
handles.itermax.string = string(itermax);
handles.msecurrent.visible = 'off';
handles.msemin.visible = 'off';
handles.msemax.visible = 'off';
handles.mse.visible = 'off';
handles.msetitle.visible = 'off';
handles.gdcurrent.visible = 'off';
handles.gdmin.visible = 'off';
handles.gdmax.visible = 'off';
handles.gd.visible = 'off';
handles.gdtitle.visible = 'off';
 Q = size(P,2); // Определение размера обучающего множества примеров

for itercnt = 1:itermax // цикл эпох обучения
   for Pcnt = 1:size(P,2) // Цикл по всем  обучающим примерам из P
      //n = W*P(:,Pcnt);
      q = fix(rand(1,'uniform')*Q)+1; // q – формируем случайный номер примера
      p = P(:,q);       // Выбираем случайный пример из P
      n = ann_negdist(W,p);  // вычисление отрицательного расстояния между примером и весами нейронов сети
      a = ann_compet_activ(n + b); // вычисление состязательной функции
      W(find(a),:) = W(find(a),:) +lr*(p'-W(find(a),:)); // Правило обучения Кохонена (формула 7.4)
      b(find(a)) = b(find(a)) - 0.2; // Обновление смещений активного нейрона
      b(find(~a)) = lr_c.*b(find(~a));  // Обновление смещений проигравших нейронов
   end

   handles.iter.value = round((itercnt/iter_span)*100);
   handles.itercurrent.string = string(itercnt);

end

endfunction
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бондарев В. Н. Искусственный интеллект : учеб. пособие для студентов вузов / В. Н. Бондарев, Ф. Г. Аде. — Севастополь : СевНТУ, 2002. — 613 с. — Текст : непосредственный.
2. Ерин С.В. Scilab – примеры и задачи: практическое пособие / С. В. Ерин – М. : Лаборатория «Знания будущего», 2017. – 154 с. — Текст : непосредственный.
3. Медведев, В. С. Нейронные сети. MATLAB 6 / В. С. Медведев, В. Г. Потемкин ; под общ. ред. В. Г. Потемкина. — М. : ДИАЛОГ-МИФИ, 2002. — 496 с. — Текст : непосредственный.
4. Хайкин С. Нейронные сети. Полный курс / С. Хайкин ; пер. с англ. — М. : Вильямс, 2006. — 1104 с. — Текст : непосредственный.
5. Hagan M.T. Neural Network Design. The 2nd edition / M. T. Hagan, H. B. Demuth, M. H. Beale, O. D. Jesus. . — Frisco, Texas, 2014. — 1012 p. — URL: https://hagan.okstate.edu/NNDesign.pdf (дата обращения: 22.04.2024). — Режим доступа: открытый. — Текст : электронный.

НЕЙРОКОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

Методические указания

Составитель : **Бондарев** Владимир Николаевич

В авторской редакции

Изд. № 77/2024. Объем 7,5 п.л. Усл. печ. л. 6,97. Уч.-изд. л. 7,35.
Издательский центр ФГАОУ ВО «Севастопольский государственный университет»