МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт информационных технологий
Кафедра «Информационные системы»

ЛАБОРАТОРНАЯ РАБОТА 6

«Основы нейронных сетей»

Выполнил:
студент гр. ИС/б-21-1-о
Степанишина М.А.

Севастополь
2023

Лабораторная работа №5

Исследование многослойного персептрона: алгоритмы обратного распространения с адаптивной скоростью обучения и моментом

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.

Ход работы:

Задача 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|($ymax - ymin$)|;

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.

Код программы:

```
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_FFBP_gd.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_ffbp_init.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_training_process.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_tansig_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_logsig_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_d_logsig_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_d_tansig_activ.sce');
function [P, T, y, x]=gendata(Q, kd)
    // функция генерирования случайных точек для 2-классов,
    // разделяемых кривой (границей) y = tansig(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 = tansig(x)
    stepx = abs(xmax - xmin) / Q;
    x = xmin:stepx:xmax;
    y = tansig(x);
    ymin = min(y);
    ymax = max(y);

    // диапазон разброса точек относительно значений функции
    range = kd * abs(ymax - ymin);

    // формирование случайных 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 = tansig(datax(1,j));

        if datay(1,j) > border then
            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

    // отображение множества точек 2-х классов и разделяющей границы
    clf(); // очистка графика
    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
    xtitle('Классы и граница (обучающее множество)');
endfunction

function y=tansig(x)
    y = (exp(x) - exp(-x)) ./ (exp(x) + exp(-x)); // Гиперболический тангенс
endfunction

// пример использования функции gendata
Q = 100; // число формируемых точек классов
kd = 0.2; // коэффициент, задающий ширину диапазона разброса точек
[P, T, y, x] = gendata(Q, kd); // генерация данных

//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;
```

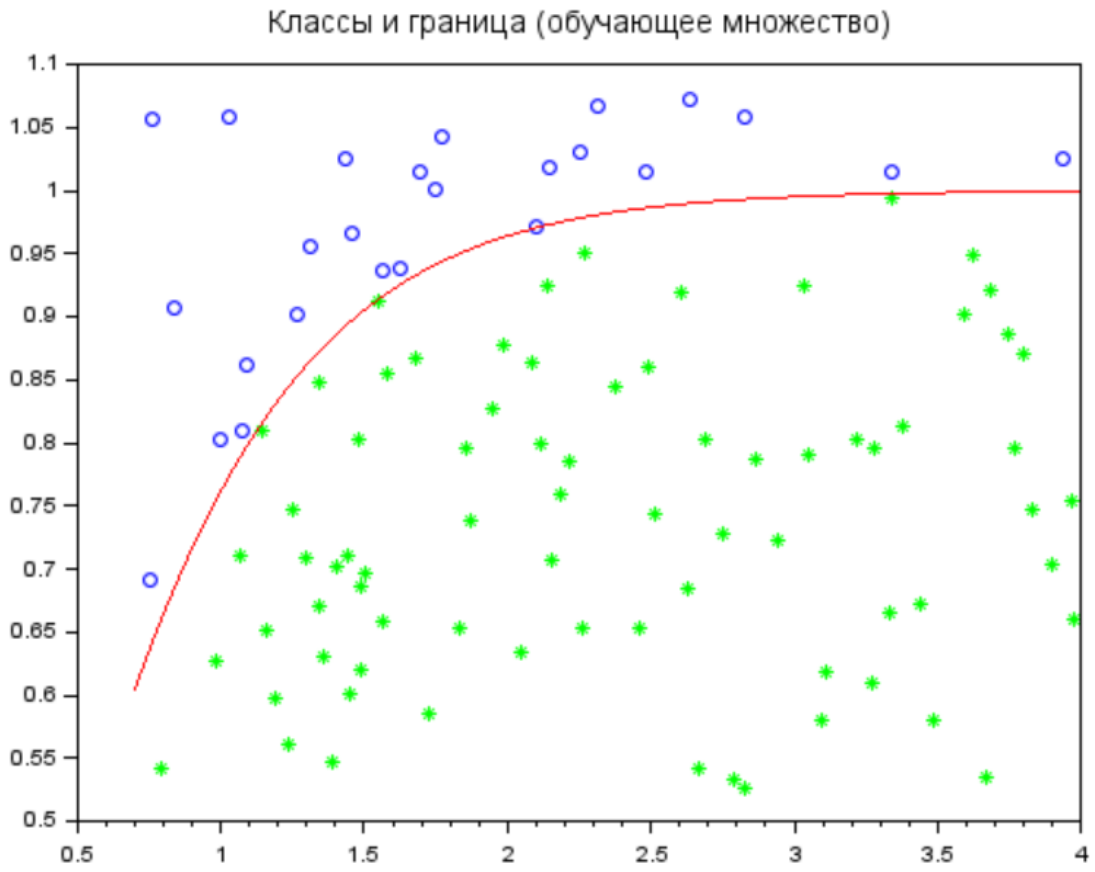

Рисунок 1 - Сгенерированные точки классов и граница между классами

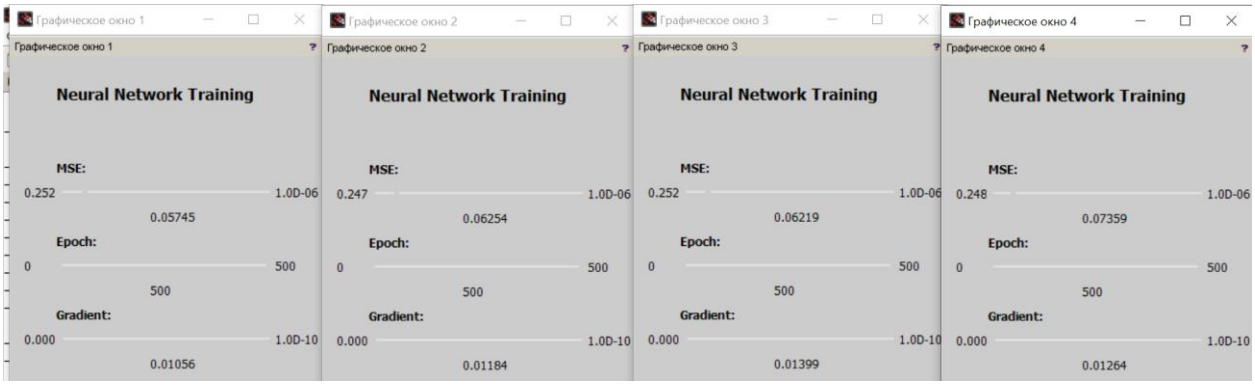

Рисунок 2 - Окно прогресса, отображаемое функциями обучения ann_FFBP_gd

Из расчётов видно, что минимальное СКО при lr=1.5

Код задачи:

```
exec('C:\Users\Мария\Desktop\Универ\3 курс\ДПО\ОСН\Лаба 5\ann_FFBP_gd1.sce');
function [P, T, y, x]=gendata(Q, kd)
    // функция генерирования случайных точек для 2-классов,
```

```
    // разделяемых кривой (границей) y = tansig(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 = tansig(x)
    stepx = abs(xmax - xmin) / Q;
    x = xmin:stepx:xmax;
    y = tansig(x);
    ymin = min(y);
    ymax = max(y);

    // диапазон разброса точек относительно значений функции
    range = kd * abs(ymax - ymin);

    // формирование случайных 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 = tansig(datax(1,j));

        if datay(1,j) > border then
            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

    // отображение множества точек 2-х классов и разделяющей границы
    clf(); // очистка графика
    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
    xtitle('Классы и граница (обучающее множество)');
endfunction

function y=tansig(x)
    y = (exp(x) - exp(-x)) ./ (exp(x) + exp(-x)); // Гиперболический тангенс
endfunction

// пример использования функции gendata
Q = 100; // число формируемых точек классов
kd = 0.2; // коэффициент, задающий ширину диапазона разброса точек
[P, T, y, x] = gendata(Q, kd); // генерация данных

//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;
```

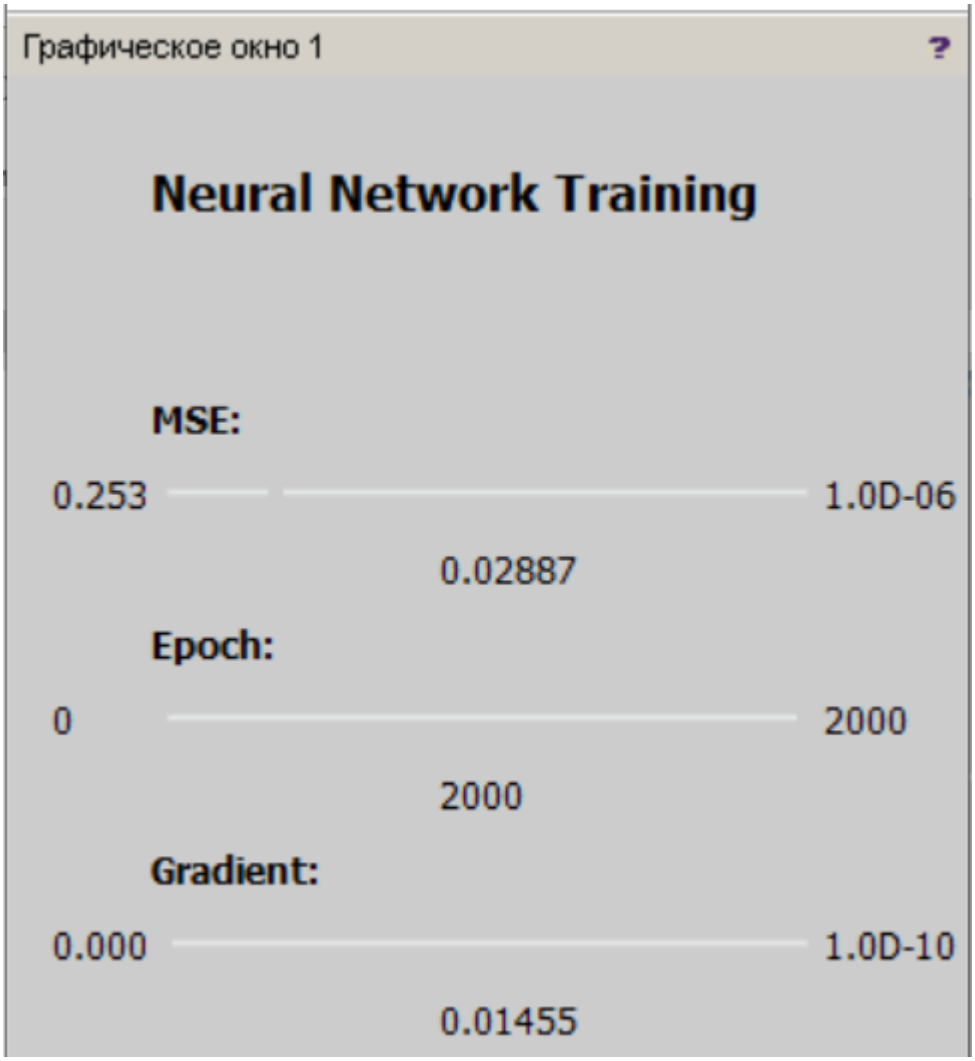

Рисунок 3 - Окно прогресса, отображаемое функциями обучения ann_FFBP_gd1

Код задачи точность классификации:

```
exec('C:\Users\Мария\Desktop\Универ\3 курс\ДПО\ОСН\Лаба 5\ann_FFBP_gd1.sce');
exec('C:\Users\Мария\Documents\ОСН\Лаба 5\ann_FFBP_run.sce');
[P_test,T_test,y_test,x_test]=gendata(Q,kd);

//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); //
оценкаточностиь
 end;
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');
```

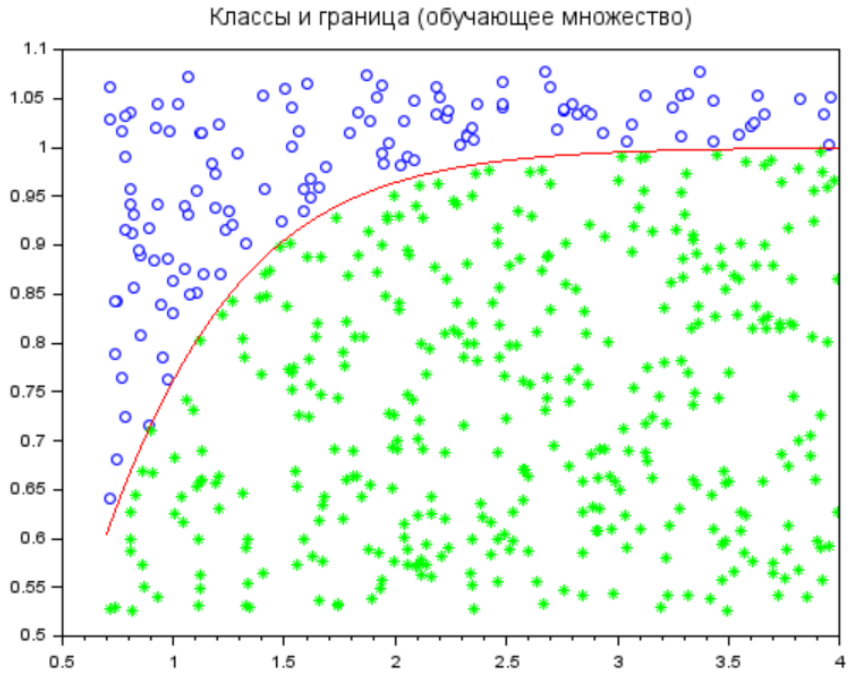

Рисунок 4 – Результаты классификации для тестового множества, точность классификации 0,9675 (S=30)

Задача 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 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.

Код программы:

```
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_ffbp_init.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_FFBP_gd.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_ffbp_init.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_training_process.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_tansig_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_logsig_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_d_logsig_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_d_tansig_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_purelin_activ.sce');
exec('C:\Users\Мария\Documents\OCH\Лаба 5\ann_d_purelin_activ.sce');
```

```
function [W, out_mse]=ann_FFBP_gdx1(P, T, N, af, lr, lr_inc, lr_dec, Mr,
itermax, mse_min, gd_min, mse_diff_max)
    rhs=argn(2);

    // Error Checking
    if rhs < 3; error("Expect at least 3 arguments, P, T and N");end
    if rhs < 4; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if rhs < 5; lr = 0.01; end
    if rhs < 6; lr_inc = 1.05; end
    if rhs < 7; lr_dec = 0.75; end
    if rhs < 8; Mr = 0.9; end
    if rhs < 9; itermax = 1000; end
    if rhs < 10; mse_min = 1e-5; end
    if rhs < 11; gd_min =  1e-5; end
    if rhs < 12; mse_diff_max =  0.01; end

    mse_diff_max

    if af == []; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if lr == []; lr = 0.01; end
    if lr_inc == []; lr_inc = 1.05; end
    if lr_dec == []; lr_dec = 0.75; end
    if Mr == []; Mr = 0.9; end
    if itermax == []; itermax = 1000; end
    if mse_min == []; mse_min = 1e-5; end
    if gd_min == []; gd_min =  1e-5; end
    if mse_diff_max == []; mse_diff_max =  0.01; end

    // Initialization
    format(8);
    W = ann_ffbp_init(N,[-0.1 0.1]);
    itercnt = 0;
    af_d = strsubst(af,'ann_','ann_d_');
    mse = %inf;
    gd = %inf;
    tempW = ann_ffbp_init(N,[0 0]);
    preW = W;

    layers = size(N,2)-1; // layers here counted from 1st hidden layers to
output layer
    n = list(0);
    a = list(0);
    m = list(0);
    s = list(0);

    while mse > mse_min & itercnt < itermax & gd > gd_min
        // Simulate Phase
        n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(T,2)); // This could
be save in temp n to save memory
        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)); // This could be save in temp n to save
memory
            a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');
        end
        e = T - a($);

        // Back Propagate
        m(layers) = evstr(af_d(layers)+'(a('+string(layers)+'))');
        //        s(layers) = (-2*m(layers).*e); // Change on 6/7/2011 to
accommodate purelin function changes
        s(layers) = -e;
```

```
        for cnt = layers-1:-1:1
            //            m(cnt) = evstr(af_d(cnt)+'(a('+string(cnt)+'))');
            m(cnt) = evstr(af_d(cnt)+'(n('+string(cnt)+'))');
            s(cnt) = m(cnt).*(W(cnt+1)(:,1:$-1)'*s(cnt+1));
        end

        // Saving Previous Weight
        pre2W = preW;
        preW = W;

        // Temporary Update

        tempW(1)(:,1:$-1) = (1+Mr)*preW(1)(:,1:$-1) - Mr*pre2W(1)(:,1:$-1) -
((1-Mr)*lr*s(1)*P')./size(P,2);
        tempW(1)(:,$) = (1+Mr)*preW(1)(:,$) - Mr*pre2W(1)(:,$) - (1-
Mr)*lr*mean(s(1),2); //b

        for cnt = 2:layers
            tempW(cnt)(:,1:$-1) = (1+Mr)*preW(cnt)(:,1:$-1) -
Mr*pre2W(cnt)(:,1:$-1) - ((1-Mr)*lr*s(cnt)*a(cnt-1)')./size(P,2);
            tempW(cnt)(:,$) = (1+Mr)*preW(cnt)(:,$) - Mr*pre2W(cnt)(:,$) -
(1-Mr)*lr*mean(s(cnt),2); //b
        end
        // Simulate Phase 2 to check MSE
        n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(T,2)); // This could
be save in temp n to save memory
        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)); // This could be save in temp n to save
memory
            a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');
        end
        e2 = T - a($);

        mse = mean(e.^2);
        mse2 = mean(e2.^2);

        // Update Weights
        mse_diff = (mse2 - mse)/mse;

        if  mse_diff <= 0
            W = tempW;
            lr = lr*lr_inc;
            if lr>1
                lr = 1
            end
        elseif mse_diff > 0 | mse_diff < mse_diff_max
            W = tempW;
        elseif mse_diff >= mse_diff_max then
            lr = lr*lr_dec;
            if lr<0.005
                lr = 0.005
            end

        end

        // Stopping Criteria
        mse = mean(e.^2);
        itercnt = itercnt + 1;
        out_mse(itercnt)=mse; // добавленная строка программы
        gd = mean(s(1).^2);

        if  modulo(itercnt,round(itermax/20)) == 0
```

```
            mprintf('Epoch %3i / %i',itercnt,itermax);
            mprintf(' MSE: %f\n',mse);
        end
        mse_all(itercnt) = mse;

    end
    mprintf('\n');
    mprintf('Epoch %3i / %i',itercnt,itermax);
    mprintf(' MSE: %f\n',mse);
    plot(1:size(mse_all,1),mse_all');
    xlabel('Epoch(n)');
    ylabel('MSE');
endfunction

function [W, out_mse]=ann_FFBP_gdm1(P, T, N, af, lr, Mr, itermax, mse_min,
gd_min)

    // Checking Input Arguement
    rhs=argn(2);

    // Error Checking
    if rhs < 3; error("Expect at least 3 arguments, P, T and N");end
    if rhs < 4; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if rhs < 5; lr = 0.01; end
    if rhs < 6; Mr = 0.9; end
    if rhs < 7; itermax = 1000; end
    if rhs < 8; mse_min = 1e-5; end
    if rhs < 9; gd_min =  1e-5; end

    if af == []; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if lr == []; lr = 0.01; end
    if Mr == []; Mr = 0.9; end
    if itermax == []; itermax = 1000; end
    if mse_min == []; mse_min = 1e-5; end
    if gd_min == []; gd_min =  1e-5; end

    // Initialization
    format(8);
    W = ann_ffbp_init(N,[-0.01 0.01]);
    itercnt = 0;
    af_d = strsubst(af,'ann_','ann_d_');
    mse = %inf;
    gd = %inf;
    preW = W;

    layers = size(N,2)-1; // layers here counted from 1st hidden layers to
output layer
    n = list(0);
    a = list(0);
    m = list(0);
    s = list(0);

    while mse > mse_min & itercnt < itermax & gd > gd_min

//        if  modulo(itercnt,round(itermax/20)) == 0
//            mprintf('Epoch %3i / %i',itercnt,itermax);
//        end

        // Simulate Phase
        n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(T,2)); // This could
be save in temp n to save memory
        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)); // This could be save in temp n to save
memory
            a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');
        end
        e = T - a($);

        // Back Propagate
        m(layers) = evstr(af_d(layers)+'(a('+string(layers)+'))');
        s(layers) = (-2*m(layers).*e); // Change on 6/7/2011 to accommodate
purelin function changes
        for cnt = layers-1:-1:1
            m(cnt) = evstr(af_d(cnt)+'(a('+string(cnt)+'))');
            s(cnt) = m(cnt).*(W(cnt+1)(:,1:$-1)'*s(cnt+1));
        end

        // Etra Lines for Momentum
        pre2W = preW;
        preW = W;

        // Update Weights
        W(1)(:,1:$-1) = (1+Mr)*preW(1)(:,1:$-1) - Mr*pre2W(1)(:,1:$-1) - ((1-
Mr)*lr*s(1)*P')./size(P,2);
        W(1)(:,$) = (1+Mr)*preW(1)(:,$) - Mr*pre2W(1)(:,$) - (1-
Mr)*lr*mean(s(1),2); //b
        for cnt = 2:layers
            W(cnt)(:,1:$-1) = (1+Mr)*preW(cnt)(:,1:$-1) -
Mr*pre2W(cnt)(:,1:$-1) - ((1-Mr)*lr*s(cnt)*a(cnt-1)')./size(P,2);
            W(cnt)(:,$) = (1+Mr)*preW(cnt)(:,$) - Mr*pre2W(cnt)(:,$) - (1-
Mr)*lr*mean(s(cnt),2); //b
        end

        // Stopping Criteria
        mse = mean(e.^2);
        itercnt = itercnt + 1;
        out_mse(itercnt)=mse; // добавленная строка программы
        gd = mean(s(1).^2);

        if  modulo(itercnt,round(itermax/20)) == 0
            mprintf('Epoch %3i / %i',itercnt,itermax);
            mprintf(' MSE: %f\n',mse);
        end
        mse_all(itercnt) = mse;

    end
    mprintf('\n');
    mprintf('Epoch %3i / %i',itercnt,itermax);
    mprintf(' MSE: %f\n',mse);
    plot(1:size(mse_all,1),mse_all');
    xlabel('Epoch(n)');
    ylabel('MSE');
endfunction

function [W, out_mse]=ann_FFBP_gda1(P, T, N, af, lr, lr_inc, lr_dec, itermax,
mse_min, gd_min, mse_diff_max)

    // Checking Input Arguement
    rhs=argn(2);

    // Error Checking
    if rhs < 3; error("Expect at least 3 arguments, P, T and N");end
    if rhs < 4; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if rhs < 5; lr = 0.01; end
    if rhs < 6; lr_inc = 1.05; end
```

```
    if rhs < 7; lr_dec = 0.75; end
    if rhs < 8; itermax = 1000; end
    if rhs < 9; mse_min = 1e-5; end
    if rhs < 10; gd_min =  1e-5; end
    if rhs < 11; mse_diff_max =  0.01; end

    lr_max = 0.2;
    lr_min = 0.0005;

    if af == []; af = ['ann_tansig_activ','ann_purelin_activ']; end
    if lr == []; lr = 0.01; end
    if lr_inc == []; lr_inc = 1.05; end
    if lr_dec == []; lr_dec = 0.75; end
    if itermax == []; itermax = 1000; end
    if mse_min == []; mse_min = 1e-5; end
    if gd_min == []; gd_min =  1e-5; end
    if mse_diff_max == []; mse_diff_max =  0.01; end

    // Initialization
    format(8);
    W = ann_ffbp_init(N,[-0.01 0.01]);
    itercnt = 0;
    af_d = strsubst(af,'ann_','ann_d_');
    mse = %inf;
    gd = %inf;
    tempW = ann_ffbp_init(N,[0 0]);

    layers = size(N,2)-1; // layers here counted from 1st hidden layers to
output layer
    n = list(0);
    a = list(0);
    m = list(0);
    s = list(0);

    while mse > mse_min & itercnt < itermax & gd > gd_min

        // Simulate Phase
        n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(T,2)); // This could
be save in temp n to save memory
        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)); // This could be save in temp n to save
memory
            a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');
        end
        e = T - a($);

        // Back Propagate
        m(layers) = evstr(af_d(layers)+'(a('+string(layers)+'))');
        //        s(layers) = (-2*m(layers).*e); // Change on 6/7/2011 to
accommodate purelin function changes
        s(layers) = -e;

        for cnt = layers-1:-1:1
            //            m(cnt) = evstr(af_d(cnt)+'(a('+string(cnt)+'))');
            m(cnt) = evstr(af_d(cnt)+'(n('+string(cnt)+'))');
            s(cnt) = m(cnt).*(W(cnt+1)(:,1:$-1)'*s(cnt+1));
        end

        // Temporary Update
        tempW(1)(:,1:$-1) = W(1)(:,1:$-1) - (lr*s(1)*P')./size(P,2);
        tempW(1)(:,$) = W(1)(:,$) - lr*mean(s(1),2);
        for cnt = 2:layers
```

```
            tempW(cnt)(:,1:$-1) = W(cnt)(:,1:$-1) - (lr*s(cnt)*a(cnt-
1)')./size(P,2);
            tempW(cnt)(:,$) = W(cnt)(:,$) - lr*mean(s(cnt),2);
        end

        // Simulate Phase 2 to check MSE
        n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(T,2)); // This could
be save in temp n to save memory
        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)); // This could be save in temp n to save
memory
            a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');
        end
        e2 = T - a($);

        mse = mean(e.^2);
        mse2 = mean(e2.^2);

        // Update Weights
        mse_diff = (mse2 - mse)/mse;

        if  mse_diff <= 0
            W = tempW;
            lr = lr*lr_inc;
            if lr>lr_max
                lr = lr_max
            end
        elseif mse_diff > 0 | mse_diff < mse_diff_max
            W = tempW;
        elseif mse_diff >= mse_diff_max then
            lr = lr*lr_dec;
            if lr<lr_min
                lr = lr_min;
            end

        end

        // Stopping Criteria
        mse = mean(e.^2);
        itercnt = itercnt + 1;
        out_mse(itercnt)=mse; // добавленная строка программы
        gd = mean(s(1).^2);
        if  modulo(itercnt,round(itermax/20)) == 0
            mprintf('Epoch %3i / %i',itercnt,itermax);
            mprintf(' MSE: %f\n',mse);
        end
        mse_all(itercnt) = mse;

    end
    mprintf('\n');
    mprintf('Epoch %3i / %i',itercnt,itermax);
    mprintf(' MSE: %f\n',mse);
    plot(1:size(mse_all,1),mse_all);
    xlabel('Epoch(n)');
    ylabel('MSE');
endfunction

function [W, out_mse]=ann_FFBP_gd1(P, T, N, af, lr, itermax, mse_min, gd_min)
// Обучение на основе блочного алгоритма градиентного спуска с обратным
распространением.
//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
слояr
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));
    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
    out_mse(itercnt)=mse; // добавленная строка программы
    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

function z=f(x, y)
    z=4.*x.*y-cos((exp(-y)-exp(x))./(exp(-x)+exp(y)))^2;
endfunction

minx=0;
miny=0;
maxx=1;
maxy=1;
mse_min=1e-6;
gd_min=1e-10;
itermax=300;
S=[5,10,15,20];
lr=0.005;
lrgd=0.05;
lr_inc=1.05;
lr_dec= 0.75;
Mr=0.9;
mse_diff_max=0.01;
af=['ann_tansig_activ', 'ann_purelin_activ'];
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);
mse_gd_history = zeros(size(S,2),itermax);
N=[2 10 1];
[W_gd, out_mse_gd] = ann_FFBP_gd1(P_train,T_train,N,af,lrgd,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,ms
e_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
);


clf(3);
figure(3);
plot(1:length(out_mse_gd), out_mse_gd, 'b', 1:length(out_mse_gda),
out_mse_gda, 'g', 1:length(out_mse_gdm), out_mse_gdm, 'r',
1:length(out_mse_gdx), out_mse_gdx, 'm');
xtitle('Кривые обучения MPL с помощью разных алгоритмов', 'Эпоха', 'СКО')
legend(['gd';'gda';'gdm';'gdx']);
```

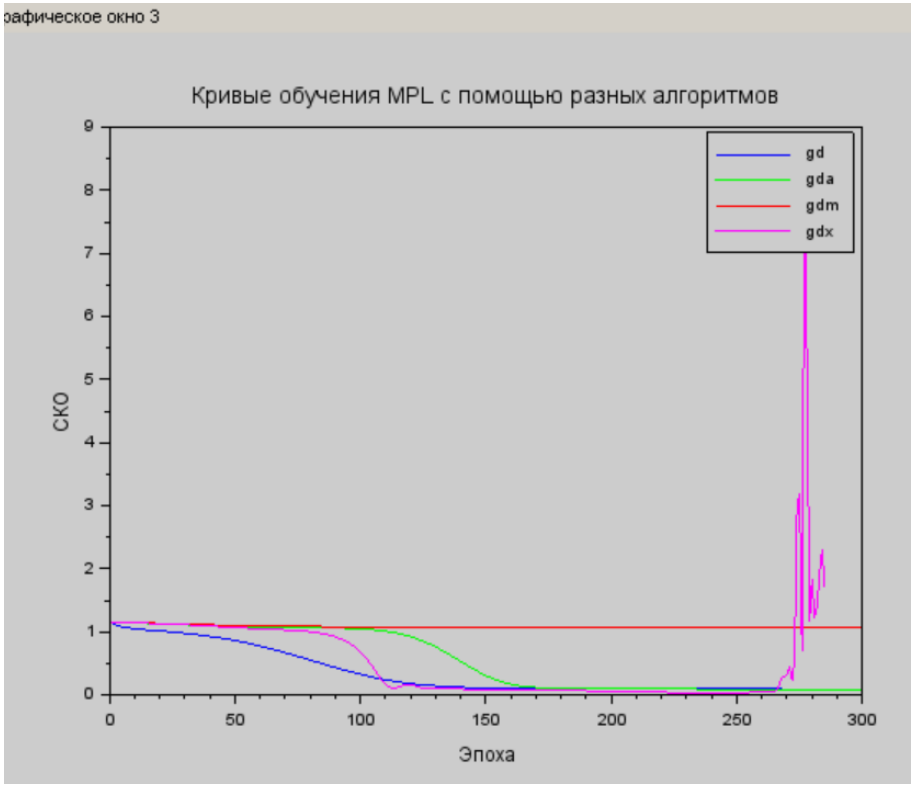

Рисунок 5 - Кривые обучения MLP

Вывод:

В ходе лабораторной работы были углублены теоретические знания в области архитектуры многослойных нейронных сетей прямого распространения.

Были исследованы свойства алгоритмов обучения многослойных нейронных сетей.

Были приобретены практические навыки обучения многослойного персептрона при решении задач классификации и аппроксимации функций.

Были сгенерированы точки классов и граница между классами. Так же был выведен прогресс, отображающий функции обучения ann_FFBP_gd. Был построен график отображающий результаты классификации для тестового множества, точность классификации 0,9675 (S=30). Затем были построены кривые обучения MLP.