МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Кафедра вычислительной математики и программирования

Лабораторная работа №3
по спецкурсу «Нейроинформатика»

Многослойные сети. Алгоритм обратного распространения ошибки

Выполнил: Дюсекеев А.Е.
Группа: М8О-101М-21
Преподаватель: Леонов С.С.

Москва, 2022

Цель работы

Исследование свойств многослойной нейронной сети прямого распространения и алгоритмов ее обучения, применение сети в задачах классификации и аппроксимации функции.

Основные этапы работы

1. Использовать многослойную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми.
2. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов первого порядка.
3. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов второго порядка.

Оборудование

Параметры процессора:

Name	i9-12900K
Processor Base Frequency	3.20 GHz
Number of Cores	16

Оперативная память:

Всего	16.0 ГБ
Скорость	2133 МГц
Тип памяти	DDR4

Программное обеспечение

Matlab R2015b, 64-bit.

Сценарий выполнения работы

Этап 1

1. Заданы 3 линейно неразделимых класса. Точки, принадлежащие одному классу, лежат на алгебраической линии. Построить и обучить многослойную сеть прямого распространения, которая будет классифицировать точки заданной области.

   Обучающий набор $\{x_i, y_i\}, i = 1, \dots, N$, число классов $K = 3$. Сеть реализует отображение вида:

$$f(x_i, y_i) = \{(z_k)_{k=1}^{K} = (0, \dots, 1, \dots, 0) \big| z_{k=K^*} = 1 \text{ при } (x_i, y_i) \in K^*\}$$

$$t = 0 : 0.025 : 2\pi$$

$$x = f(t)$$

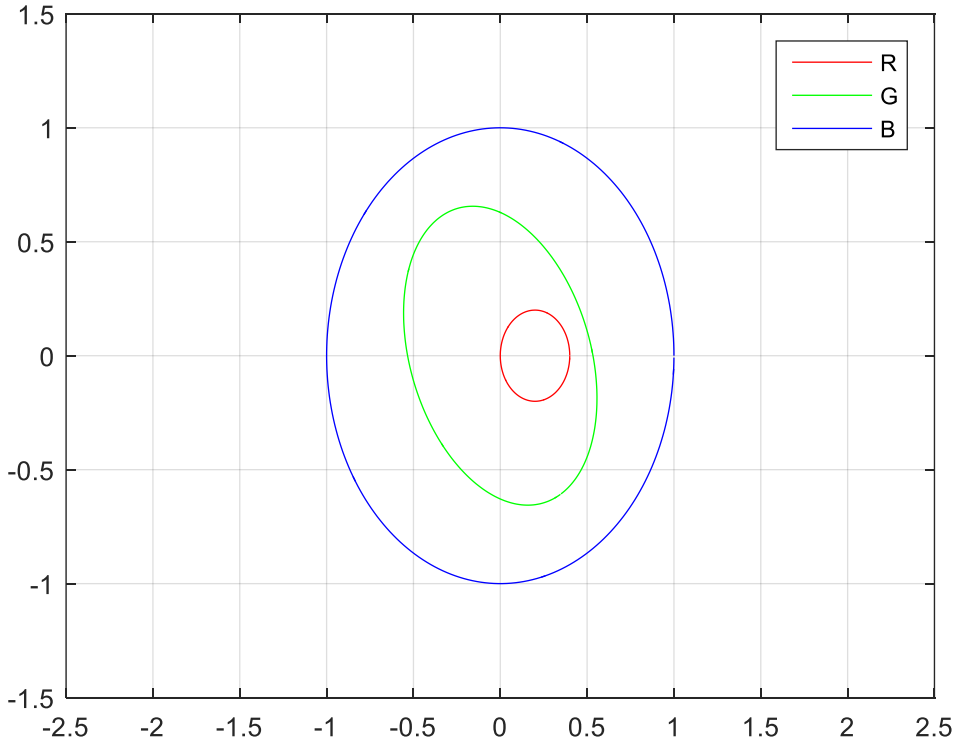
$$y = g(t)$$

   Эллипс: $a = 0.2, b = 0.2, \alpha = 0, x_0 = 0.2, y_0 = 0;$

   Эллипс: $a = 0.7, b = 0.5, \alpha = -\frac{\pi}{3}, x_0 = 0, y_0 = 0;$

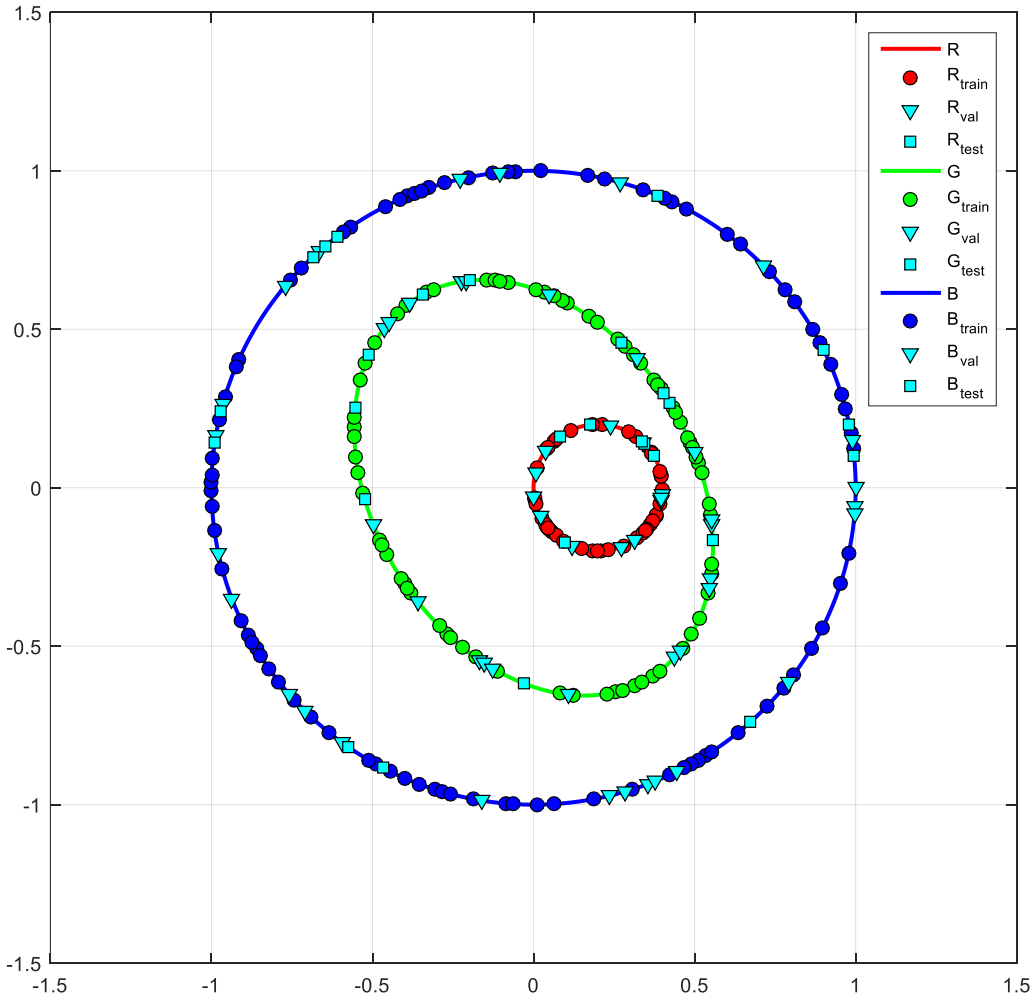
   Эллипс: $a = 1, b = 1, \alpha = 0, x_0 = 0, y_0 = 0.$

   1.1. В соответствии с вариантом задания для каждой линии сгенерировать множество точек. Далее для первого класса выбрать из исходного множества случайным образом 60 точек. Для второго и третьего классов 100 и 120 точек соответственно. Для выбора точек рекомендуется использовать функцию *randperm*, с помощью которой получить псевдослучайную последовательность индексов вектора.



1.2. Множество точек, принадлежащее каждому классу, разделить на обучающее, контрольное, и тестовое подмножества с помощью функции *dividerand* в отношении 70%-20%-10%.

1.3. Отобразить с помощью функции *plot* исходные множества точек для каждого из классов. Задать параметр *LineWidth* равным 2, подписать линии, задать сетку. С помощью *axis* задать границы для входного множества.



1.4. Соответствующие подмножества точек каждого класса объединить в обучающее, контрольное, и тестовое подмножества обучающей выборки. Обучающая выборка состоит из последовательного объединения полученных обучающего, контрольного, и тестового подмножеств.

1.5. Создать сеть с помощью функции *feedforwardnet*. Сконфигурировать сеть (*configure*), указав диапазоны изменения для входного множества и эталонных выходов сети. Точки входного и выходного множеств лежат на отрезках $[-1.5,1.5]$ и $[0,1]$ по каждой из координат соответственно.

   Число нейронов скрытого слоя задать равным 20. Использовать активационные функцию *tansig* для скрытого и выходного слоев. Задать *RProp* в качестве алгоритма обучения.

1.6. Для разделения обучающего множества на подмножества использовать *net.divideFcn ='divideind'*. Также задать параметры:

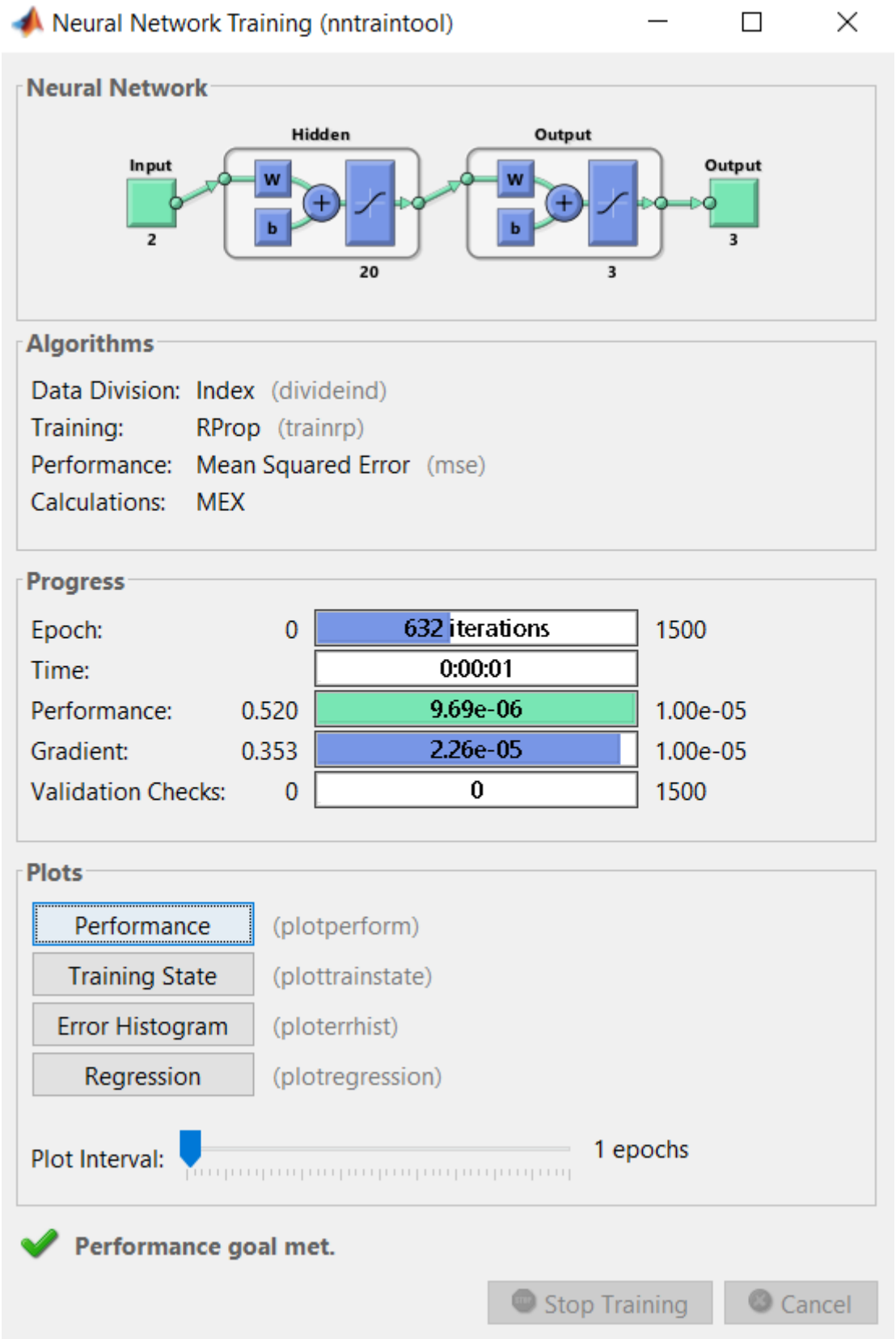
net.divideParam.trainInd = 1 : trnInd;
net.divideParam.valInd = trnInd + 1 : tstInd;
net.divideParam.testInd = tstInd + 1 : proInd;

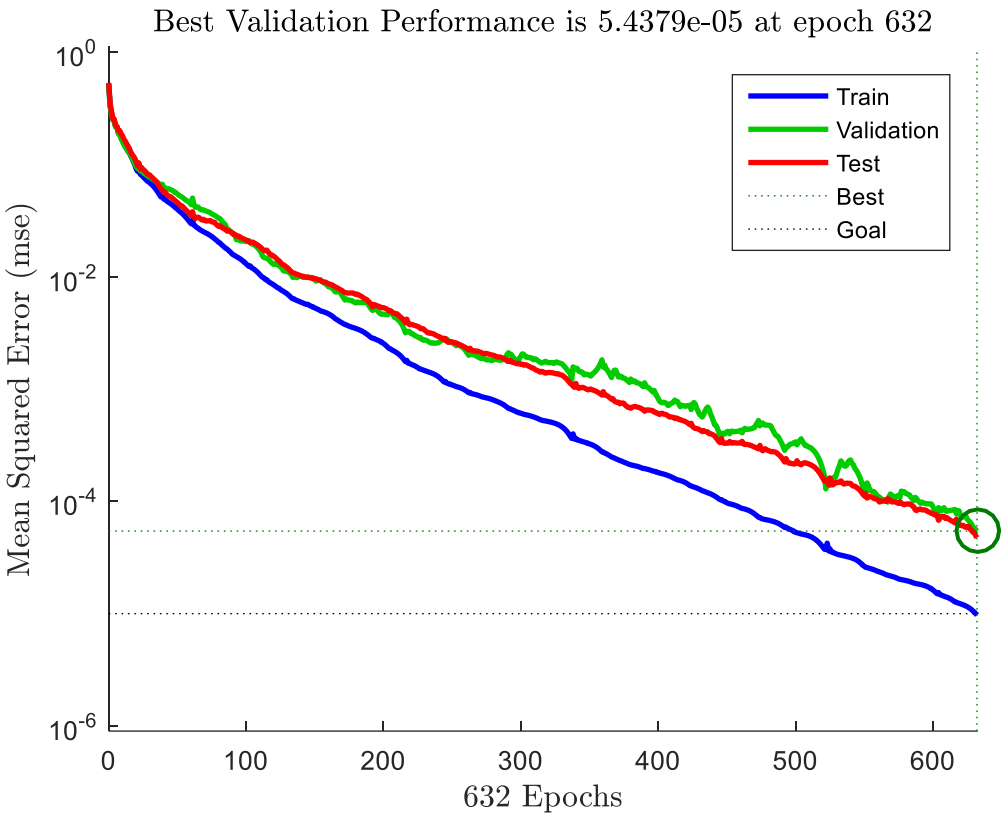
где *trnInd*, *tstInd*, *proInd* задают количество примеров в обучающем, контрольном, и тестовом подмножествах.

1.7. Инициализировать (*init*) весовые коэффициенты и смещения сети с помощью функции, заданной по умолчанию.

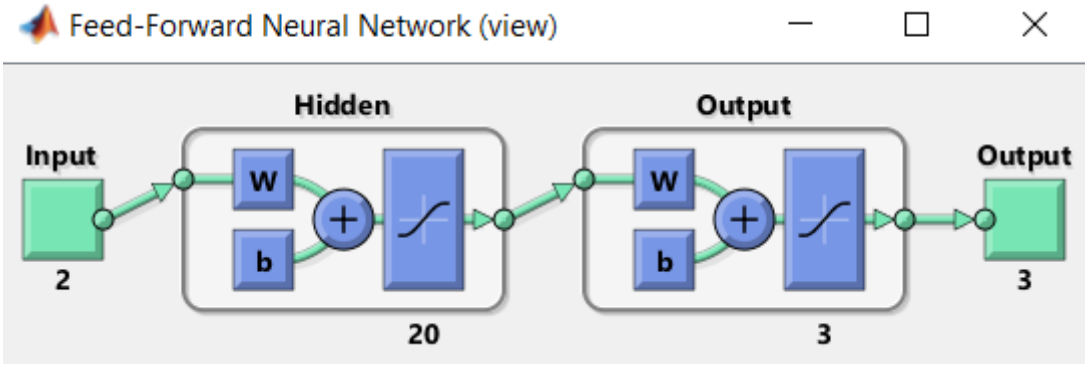
1.8. Задать параметры обучения: число эпох обучения (*net.trainParam.epochs*) и число эпох, в течение которых может расти ошибка на контрольном подмножестве (*net.trainParam.max_fail*), равными 1500, предельное значение критерия обучения (*net.trainParam.goal*) равным 10^{-5}.

1.9. Выполнить обучение сети с помощью функции *train*. Для обучения использовать обучающую выборку. Занести в отчет содержимое *Performance* и *Neural Network Training*.


Neural Network Training (nntraintool)
Neural Network
Input
2
Hidden
W
b
20
Output
W
b
3
Output
3
Algorithms
Data Division: Index (divideind)
Training: RProp (trainrp)
Performance: Mean Squared Error (mse)
Calculations: MEX
Progress
Epoch: 0 632 iterations 1500
Time: 0:00:01
Performance: 0.520 9.69e-06 1.00e-05
Gradient: 0.353 2.26e-05 1.00e-05
Validation Checks: 0 0 1500
Plots
Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Regression (plotregression)
Plot Interval: 1 epochs
Performance goal met.
Stop Training
Cancel



1.10. Отразить структуру сети и проведенное обучение в отчете, заполнив таблицу 1.



Функция создания сети	*feedforwardnet*
Входной слой	[2]
Скрытый слой	[20]
Выходной слой	[3]
Активационные функции	tansig, tansig
Динамика	минимизация *mse*
Функция разделения обучающего множества	*divideind*
Число примеров в подмножествах	*n_train*=196 *n_val*=56 *n_test*=28
Метод обучения	*train/trainrp*
Параметры обучения	*net.trainParam.epochs = 1500* *net.trainParam.max_fail = 1500* *net.trainParam.goal = 1e-5*
Метод инициализации сети	*initlay*
Критерий окончания обучения	goal=1e-5 или *trainParam.epochs = 1500*
Причина окончания обучения	достигнут критерий обучения
Число эпох обучения	632

1.11. Рассчитать выход сети (*sim*) для обучающего подмножества. Преобразовать значения по правилу

$$o_{ij} = \begin{cases} 1, & a_{ij} \geq 0.5; \\ 0, & a_{ij} < 0.5 \end{cases}$$

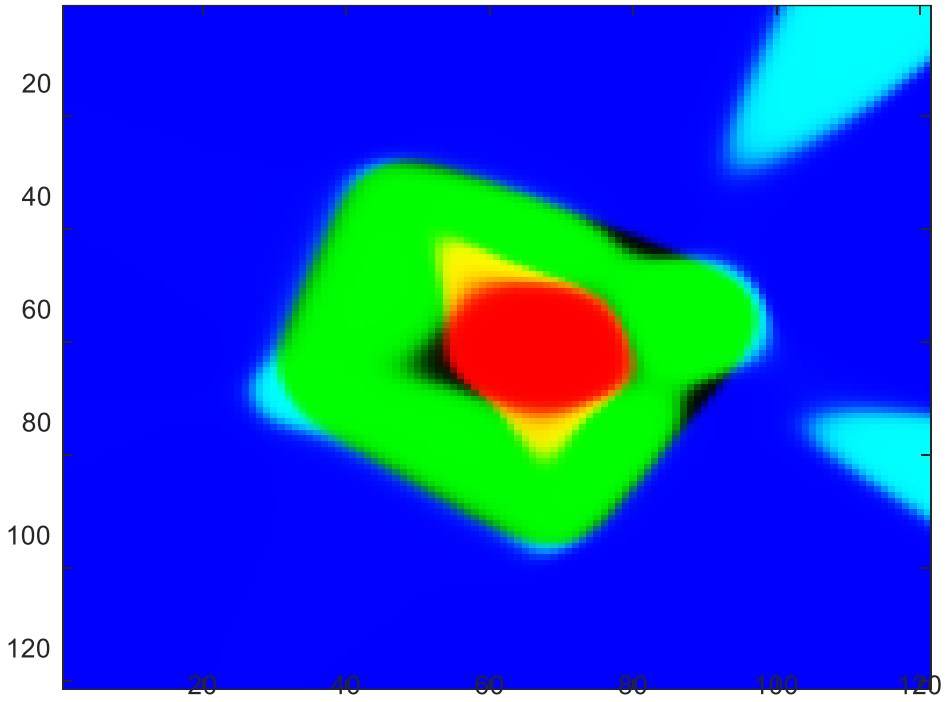
Занести в отчёт количество правильно классифицированных точек.

```
Обучающие: 196/196
Проверочные: 56/56
Тестовые: 28/28
```

1.12. Провести аналогичные расчеты для контрольного и тестового подмножеств.

1.13. Произвести классификацию точек области $[-1.5,1.5] \times [-1.5,1.5]$. Для этого задать сетку для указанной области с шагом $h = 0.025$. Рассчитать выход сети для всех узлов сетки.

1.14. Выход сети для каждой точки задает ее принадлежность к трем классам. Закодировать принадлежности к классам различными цветами и занести полученное изображение в отчет.



Этапы 2 и 3

2. Задан обучающий набор $\{x(i), y(i)\}$. Построить и обучить двухслойную нейронную сеть прямого распространения, которая будет выполнять аппроксимацию функции вида

$$\hat{y}(i) = f\big(x(i)\big)$$

Для обучения использовать алгоритм, реализующий метод поиска экстремума функции многих переменных первого порядка. Функция и метод обучения определяются вариантом задания.

$$x = \sin(t^2 - 7t), \qquad t \in [0{,}5], h = 0.025$$

trainscg, *trainoss*

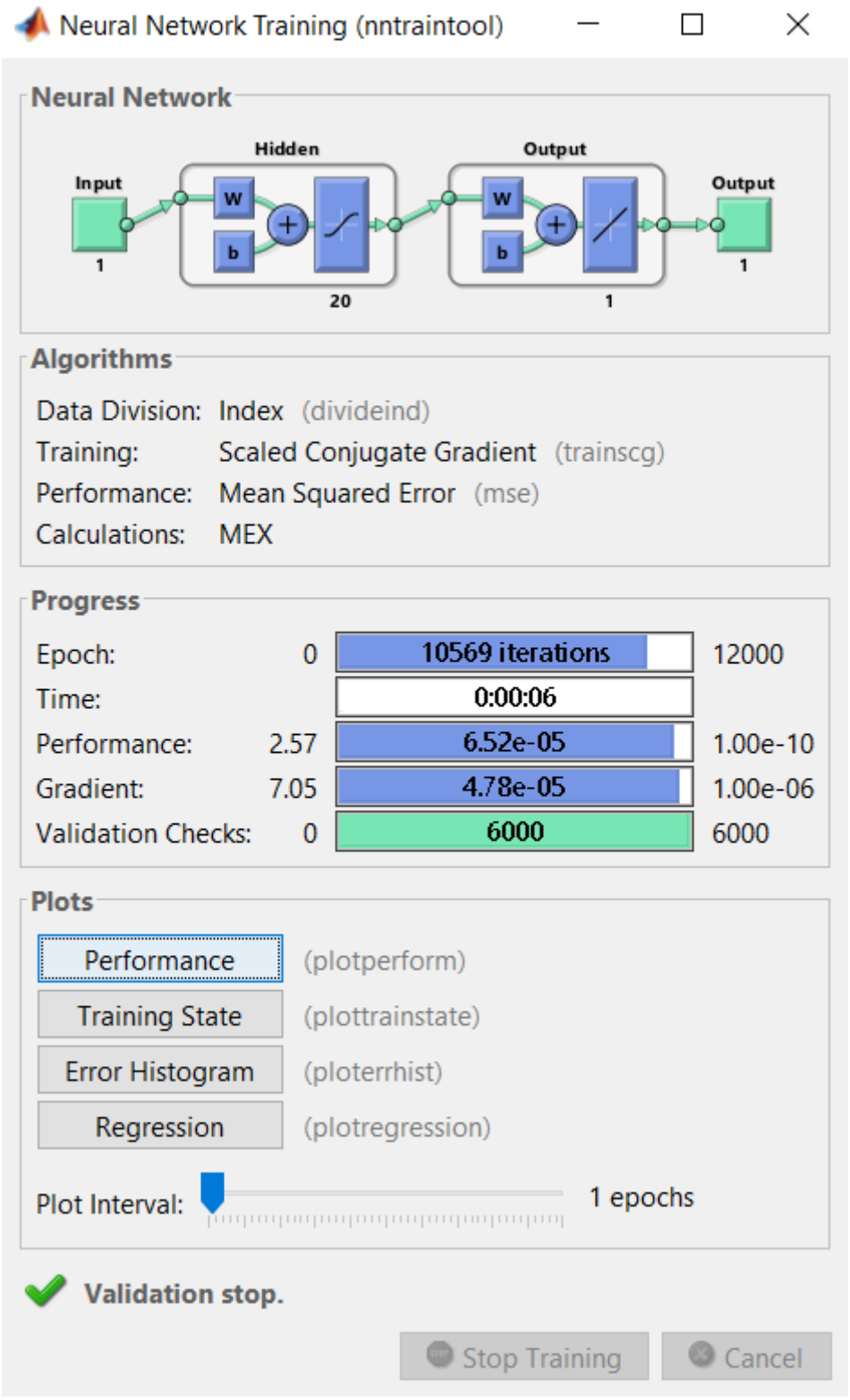
2.1. Создать сеть с помощью функции *feedforwardnet*. Сконфигурировать сеть под обучающее множество с помощью функции configure. Число нейронов скрытого слоя задать равным 10. Использовать активационные функции, заданные по умолчанию (*tansig*, *purelin*). Алгоритм обучения определяется вариантом задания.

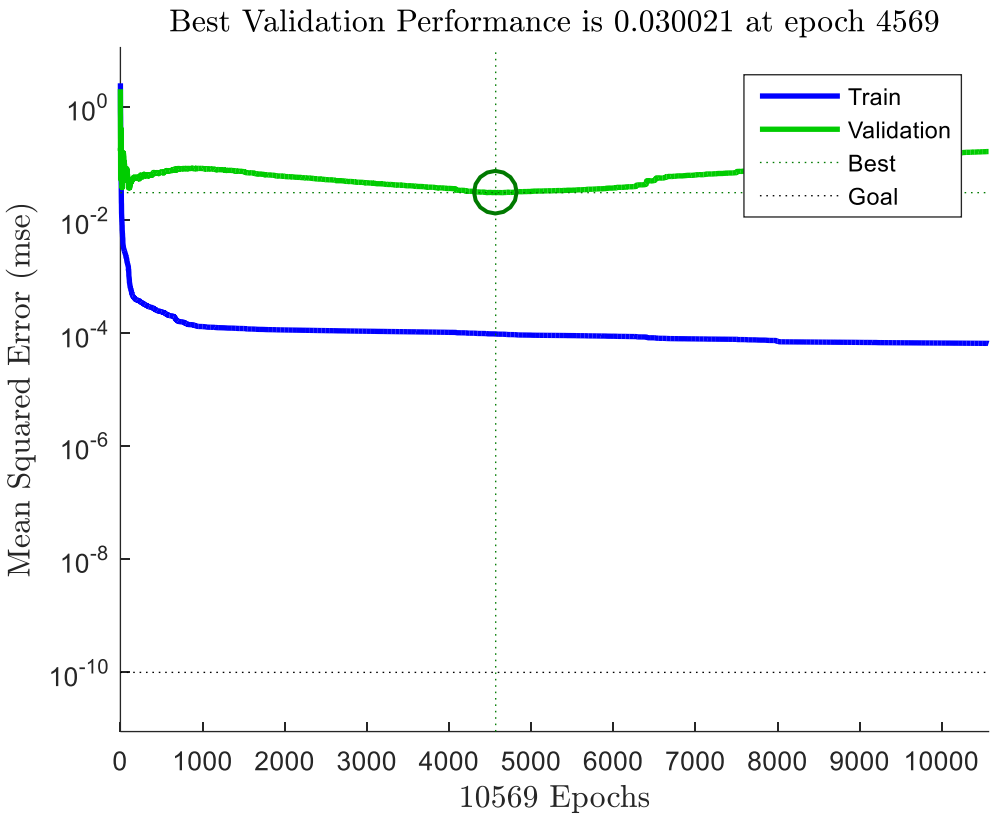
2.2. Для разделения обучающией выборки на обучающее, контрольное, и тестовое подмножества использовать функцию *divideind*. Выделить с конца временной последовательности 10% отсчетов на контрольное подмножество. Тестовое подмножество оставить пустым.
net.divideParam.testInd = []

2.3. Инициализировать сеть (*init*) с помощью функции, заданной по умолчанию

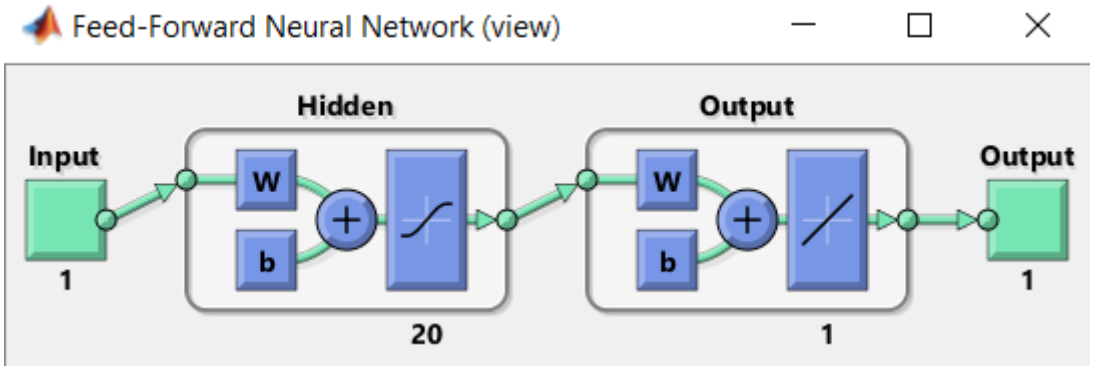
2.4. Задать параметры обучения: значения параметров для некоторых методов обучения описаны выше, число эпох обучения (*net.trainParam.epochs*) и число эпох, в течение которых может расти ошибка на контрольном подмножестве (*net.trainParam.max_fail*), равными 600, предельное значение критерия обучения (*net.trainParam.goal*) равным 10^{-8}

2.5. Выполнить обучение сети с помощью функции *train*. Если необходимо, то произвести обучение несколько раз. Если результаты неудовлетворительные или наблюдается переобучение, то изменить число нейронов в функции *feedforwardnet*, увеличить число эпох обучения или уменьшить предельное значение критерия обучения. Занести в отчет весовые коэффициенты и смещения для двух слоев. Занести в отчет окна *Performance* и *Neural Network Training*, если это возможно для данного метода обучения.


Neural Network Training (nntraintool)
Neural Network
Input
1
Hidden
W
b
20
Output
W
b
1
Output
1
Algorithms
Data Division: Index (divideind)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Mean Squared Error (mse)
Calculations: MEX
Progress
Epoch: 0 10569 iterations 12000
Time: 0:00:06
Performance: 2.57 6.52e-05 1.00e-10
Gradient: 7.05 4.78e-05 1.00e-06
Validation Checks: 0 6000 6000
Plots
Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Regression (plotregression)
Plot Interval: 1 epochs
Validation stop.
Stop Training
Cancel



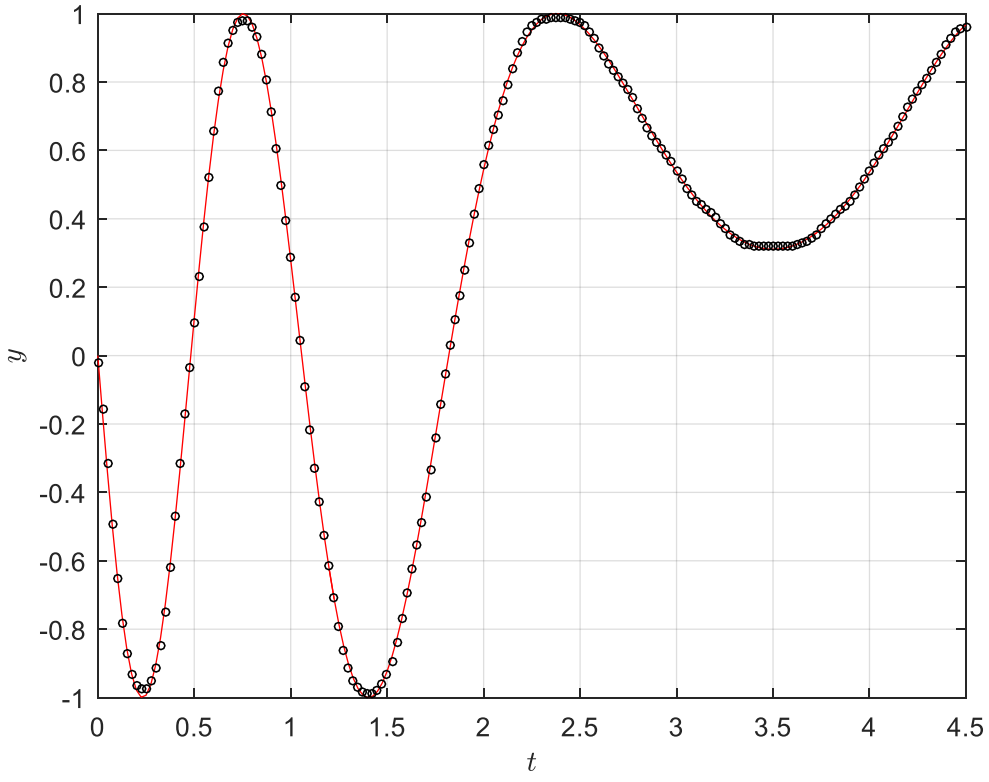
2.6. Отразить структуру сети и проведенное обучение в отчете, заполнив таблицу 1.



Функция создания сети	*feedforwardnet*
Входной слой	[1]
Скрытый слой	[20]
Выходной слой	[1]
Активационные функции	tansig, purelin
Динамика	минимизация *mse*
Функция разделения обучающего множества	*divideind*
Число примеров в	*n_train*=181

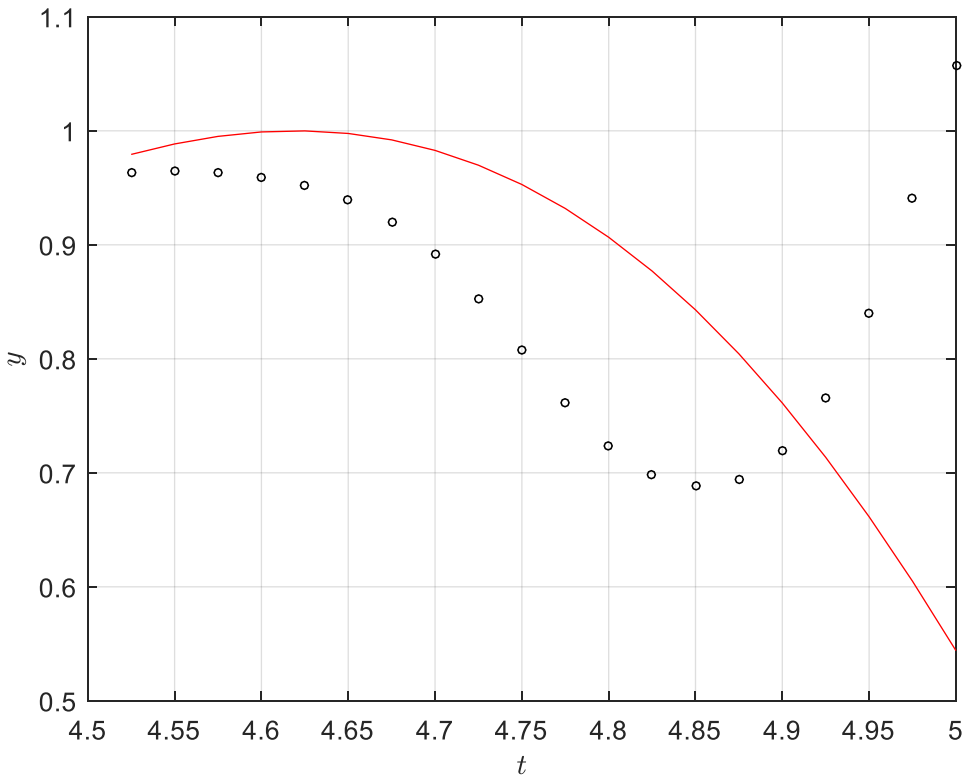
подмножествах	*n_val*=56 *n_test*=0
Метод обучения	*train/trainscg*
Параметры обучения	*net.trainParam.epochs = 12000;* *net.trainParam.max_fail = 6000;* *net.trainParam.goal = 1e-10;*
Метод инициализации сети	*initlay*
Критерий окончания обучения	*trainParam.epochs = 12000* или *net.trainParam.goal = 1e-10*
Причина окончания обучения	достигнуто предельное количество эпох, на которых растёт ошибка на контрольной выборке
Число эпох обучения	10569

2.7. Рассчитать выход сети (*sim*) для обучающего подмножества. Сравнить выход сети с соответствующим эталонным подмножеством: рассчитать показатели качества обучения и заполнить таблицу 2. Отобразить на графике эталонные значения и предсказанные сетью, а также ошибку обучения. Графики занести в отчет.



Обучающее подмножество	
***R* квадрат**	0.999645
MSE	0.000182
RMSE	0.013475
Относительная СКО, %	0.680303
MAE	0.010996
min absolute error	0.000086
max absolute error	0.033018
***MAPE*, %**	3.747539
Доля с ошибкой менее 5%, %	87.610619
Доля с ошибкой от 5% до 10%, %	7.079646
Доля с ошибкой от 10% до 20%, %	2.654867
Доля с ошибкой от 20% до 30%, %	0.884956
Доля с ошибкой более 30%, %	1.769912

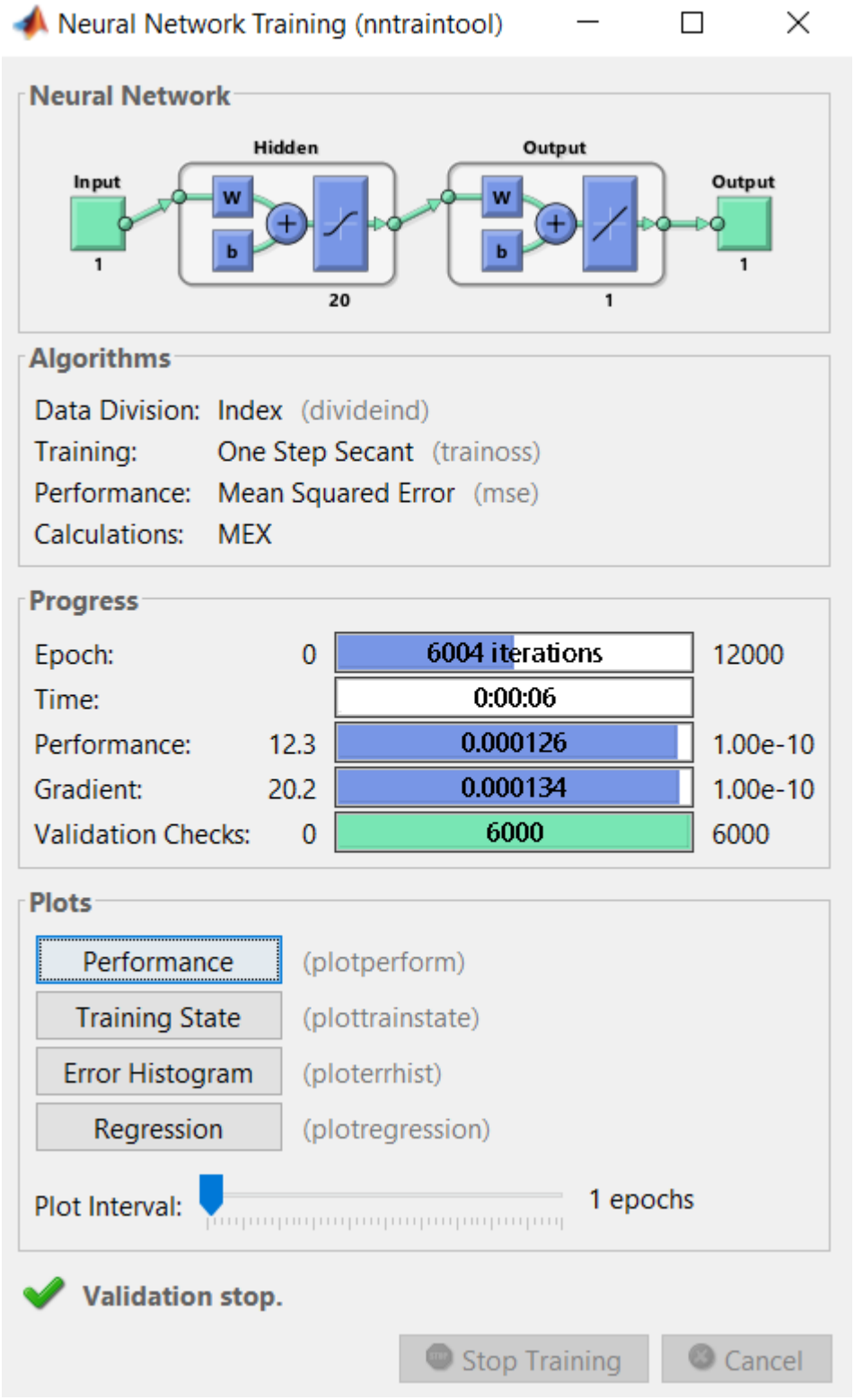
2.8.Проделать тоже самое для контрольного подмножества.

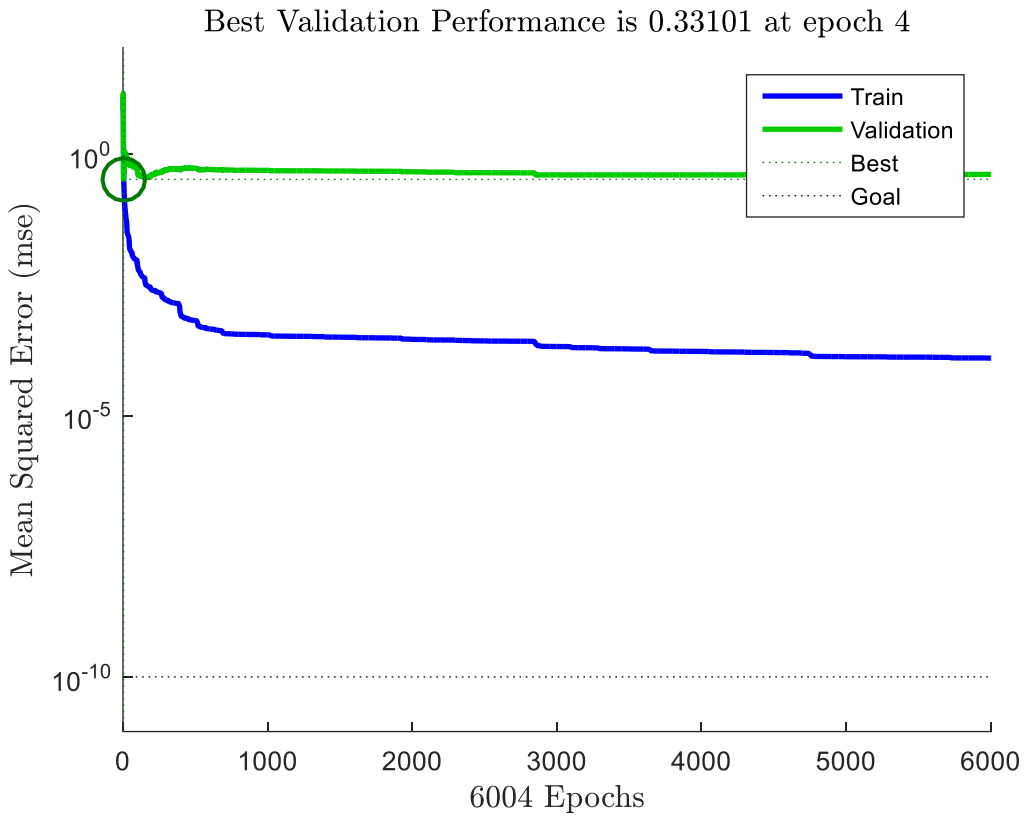


3. Построить и обучить двухслойную нейронную сеть прямого распространения, которая будет выполнять аппроксимацию функции. Для

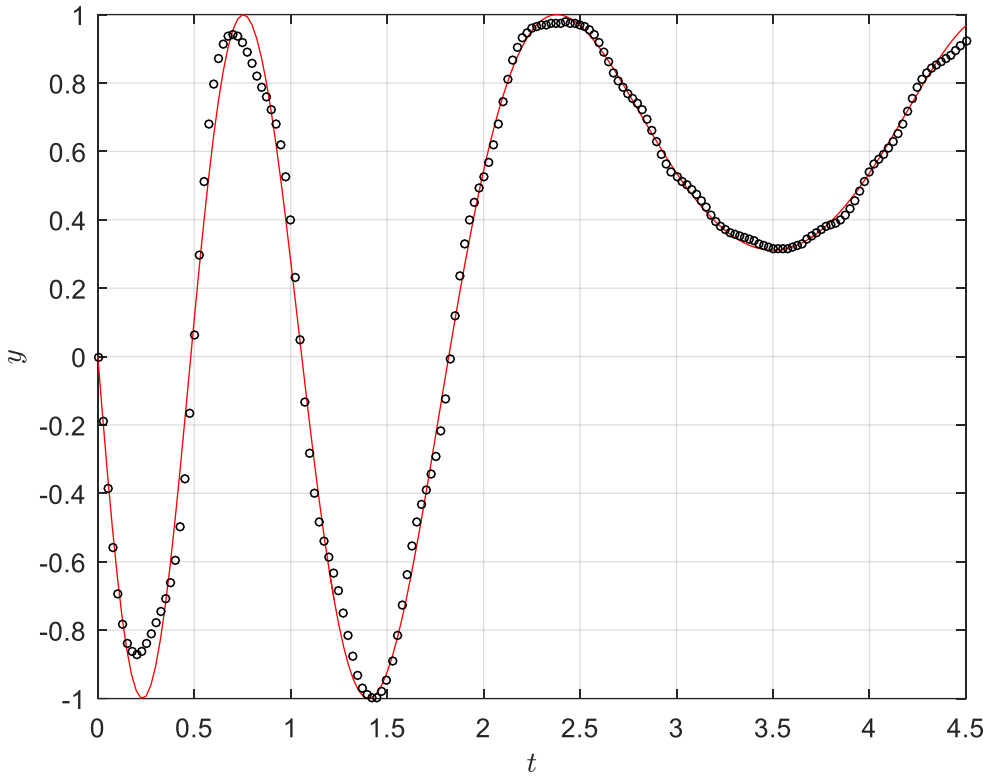
обучения использовать алгоритм, реализующий метод оптимизации функций многих переменных второго порядка. Функция и метод обучения определяются вариантом задания.

Последовательности шагов для выполнения 2 и 3 этапов работы совпадают.

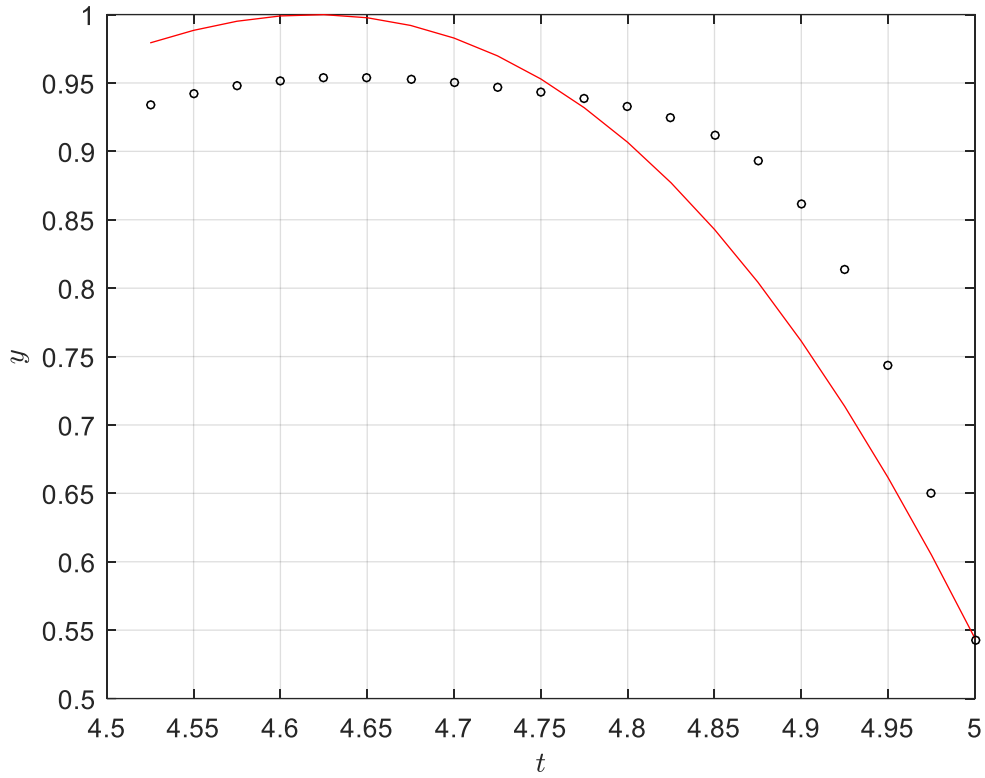




Функция создания сети	*feedforwardnet*
Входной слой	[1]
Скрытый слой	[20]
Выходной слой	[1]
Активационные функции	tansig, purelin
Динамика	минимизация *mse*
Функция разделения обучающего множества	*divideind*
Число примеров в подмножествах	*n_train*=316 *n_val*=35 *n_test*=0
Метод обучения	*train/trainoss*
Параметры обучения	*net.trainParam.epochs = 12000;* *net.trainParam.max_fail = 6000;* *net.trainParam.goal = 1e-10;*
Метод инициализации сети	*initlay*
Критерий окончания обучения	*trainParam.epochs = 12000* или *net.trainParam.goal = 1e-10*
Причина окончания обучения	Превышено число ошибок на контрольном подмножестве
Число эпох обучения	6004



Обучающее подмножество	
***R* квадрат**	0.979528
MSE	0.010563
RMSE	0.102778
Относительная СКО, %	4.874225
MAE	0.085082
min absolute error	0.000950
max absolute error	0.358382
***MAPE*, %**	25.220360
Доля с ошибкой менее 5%, %	21.681416
Доля с ошибкой от 5% до 10%, %	24.778761
Доля с ошибкой от 10% до 20%, %	28.761062
Доля с ошибкой от 20% до 30%, %	12.389381
Доля с ошибкой более 30%, %	12.389381



Контрольное подмножество	
***R* квадрат**	0.783188
MSE	0.022888
RMSE	0.151289
Относительная СКО, %	17.447969
MAE	0.140738
min absolute error	0.010348
max absolute error	0.217037
***MAPE*, %**	53.563128
Доля с ошибкой менее 5%, %	0.000000
Доля с ошибкой от 5% до 10%, %	0.000000
Доля с ошибкой от 10% до 20%, %	8.000000
Доля с ошибкой от 20% до 30%, %	8.000000
Доля с ошибкой более 30%, %	84.000000

Код программы

accuracy.m

```
function res = accuracy(y, yp)
% Высчитывание качественных характеристик обучения
    SSE = sum((y - yp) .^ 2);
    SSyy = sum((y - mean(y)) .^ 2);
    R_square = 1 - SSE/SSyy;

    MSE = mse(y - yp);

    RMSE = sqrt(MSE);

    CKO = RMSE / (max(y) - min(y)) * 100;

    MAE = mae(y - yp);

    MinAE = min(abs(y - yp));
    MaxAE =  max(abs(y - yp));

    MAPE = mean(abs((y - yp) ./ y)) * 100;

    errors = abs((y - yp) ./ y) * 100; % вектор относительных ошибок

    res = sprintf(['R квадрат: %f\n' ...
                   'MSE: %f\n' ...
                   'RMSE: %f\n' ...
                   'Относительная СКО: %f%%\n' ...
                   'MAE: %f\n'...
                   'min abs err: %f\n' ...
                   'max abs err: %f\n' ...
                   'MAPE: %f\n' ...
                   'Доля с ошибкой менее 5%%: %f%%\n' ...
                   'Доля с ошибкой от 5%% до 10%%: %f%%\n'...
                   'Доля с ошибкой от 10%% до 20%%: %f%%\n' ...
                   'Доля с ошибкой от 20%% до 30%%: %f%%\n' ...
                   'Доля с ошибкой более 30%%: %f%%\n'], ...
                  R_square, MSE, RMSE, CKO, MAE, MinAE, MaxAE, MAPE, ...
                  sum(errors < 5) / length(y) * 100, ...
                  sum(5 <= errors & errors < 10) / length(y) * 100, ...
                  sum(10 <= errors & errors < 20) / length(y) * 100, ...
                  sum(20 <= errors & errors < 30) / length(y) * 100, ...
                  sum(errors >= 30) / length(y) * 100);

end
```

main.m

```
% ЛР3
% Вариант 10

set(0, 'DefaultTextInterpreter', 'latex');

%% построение множества точек

t = 0:0.025:2*pi;
```

```
alpha = 0;
x0 = 0.2;
y0 = 0.;
R = [cos(alpha), -sin(alpha); sin(alpha),  cos(alpha)] * [0.2 * cos(t); 0.2 *
sin(t)] + [x0 * ones(1, length(t)); y0 * ones(1, length(t))];

alpha = -pi/3;
x0 = 0.;
y0 = 0.;
G = [cos(alpha), -sin(alpha); sin(alpha),  cos(alpha)] * [0.7 * cos(t); 0.5 *
sin(t)] + [x0 * ones(1, length(t)); y0 * ones(1, length(t))];

alpha = 0.;
x0 = 0.;
y0 = 0.;
B = [cos(alpha), -sin(alpha); sin(alpha),  cos(alpha)] * [1 * cos(t); 1 *
sin(t)] + [x0 * ones(1, length(t)); y0 * ones(1, length(t))];

% нужно оставить только те точки, которые принадлежат области
cond = -1.5 <= B & B <= 1.5;
cond = cond(1, :) & cond(2, :);
B = B(:, cond);

plot(R(1, :), R(2, :), 'r', ...
     G(1, :), G(2, :), 'g', ...
     B(1, :), B(2, :), 'b');
legend('R', 'G', 'B');
axis([-2.5 2.5 -1.5 1.5]);
grid on;

%% формирование обучающего множества и разделение множества на обучающее,
контрольное и тестовое

r = R(:, randperm(end, 60));
g = G(:, randperm(end, 100));
b = B(:, randperm(end, 120));

[r_train, r_val, r_test] = dividerand(r, 0.7, 0.2, 0.1);
[g_train, g_val, g_test] = dividerand(g, 0.7, 0.2, 0.1);
[b_train, b_val, b_test] = dividerand(b, 0.7, 0.2, 0.1);

n_train = length(r_train) + length(g_train) + length(b_train);
n_val = length(r_val) + length(g_val) + length(b_val);
n_test = length(r_test) + length(g_test) + length(b_test);

%% отображение

p = plot(R(1, :),       R(2, :),       '-r', ...
         r_train(1, :), r_train(2, :), 'or',  ...
         r_val(1, :),   r_val(2, :),   'rV', ...
         r_test(1, :),  r_test(2, :),  'rs', ...
         G(1, :),       G(2, :),       '-g', ...
         g_train(1, :), g_train(2, :), 'og',  ...
         g_val(1, :),   g_val(2, :),   'gV', ...
```

```
          g_test(1, :),  g_test(2, :),  'gs', ...
          B(1, :),       B(2, :),       '-b', ...
          b_train(1, :), b_train(2, :), 'ob',  ...
          b_val(1, :),   b_val(2, :),   'bV', ...
          b_test(1, :),  b_test(2, :),  'bs');

p(1).LineWidth = 2;

p(2).MarkerEdgeColor = 'k';
p(2).MarkerFaceColor = 'r';
p(2).MarkerSize = 7;

p(3).MarkerEdgeColor = 'k';
p(3).MarkerFaceColor = 'c';
p(3).MarkerSize = 7;

p(4).MarkerEdgeColor = 'k';
p(4).MarkerFaceColor = 'c';
p(4).MarkerSize = 7;

p(5).LineWidth = 2;

p(6).MarkerEdgeColor = 'k';
p(6).MarkerFaceColor = 'g';
p(6).MarkerSize = 7;

p(7).MarkerEdgeColor = 'k';
p(7).MarkerFaceColor = 'c';
p(7).MarkerSize = 7;

p(8).MarkerEdgeColor = 'k';
p(8).MarkerFaceColor = 'c';
p(8).MarkerSize = 7;

p(9).LineWidth = 2;

p(10).MarkerEdgeColor = 'k';
p(10).MarkerFaceColor = 'b';
p(10).MarkerSize = 7;

p(11).MarkerEdgeColor = 'k';
p(11).MarkerFaceColor = 'c';
p(11).MarkerSize = 7;

p(12).MarkerEdgeColor = 'k';
p(12).MarkerFaceColor = 'c';
p(12).MarkerSize = 7;
axis([-1.5 1.5 -1.5 1.5]);
legend('R', 'R_{train}', 'R_{val}', 'R_{test}', ...
       'G', 'G_{train}', 'G_{val}', 'G_{test}', ...
       'B', 'B_{train}', 'B_{val}', 'B_{test}');
grid on;

%% объединение в выборку с метками
```

```
X = [r_train g_train b_train r_val g_val b_val r_test g_test b_test];
y = [[1; 0; 0] * ones(1, length(r_train)) [0; 1; 0] * ones(1,
length(g_train)) [0; 0; 1] * ones(1, length(b_train)) ...
     [1; 0; 0] * ones(1, length(r_val))   [0; 1; 0] * ones(1, length(g_val))
[0; 0; 1] * ones(1, length(b_val)) ...
     [1; 0; 0] * ones(1, length(r_test))  [0; 1; 0] * ones(1, length(g_test))
[0; 0; 1] * ones(1, length(b_test))];

%% создание сети

net = feedforwardnet(20);

net = configure(net, X, y);
net.inputs{1}.range = [-1.5 1.5; ...
                       -1.5 1.5];
net.outputs{2}.range = [0 1; ...
                        0 1; ...
                        0 1];
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'tansig';

net.trainFcn = 'trainrp';

net.divideFcn = 'divideind';
net.divideParam.trainInd = 1:n_train;
net.divideParam.valInd = n_train+1:n_train+n_val;
net.divideParam.testInd = n_train+n_val+1:n_train+n_val+n_test;

net = init(net); % инициализаций весов

net.trainParam.epochs = 1500;
net.trainParam.max_fail = 1500;
net.trainParam.goal = 1e-5;

%% обучение сети

net = train(net, X, y);

%% расчёт количества правильно классифицированных образцов

n_right_train = sum(sum((sim(net, X(:, 1:n_train)) >= 0.5) == logical(y(:,
1:n_train)), 1) == 3);
n_right_val = sum(sum((sim(net, X(:, n_train+1:n_train+n_val)) >= 0.5) ==
logical(y(:, n_train+1:n_train+n_val)), 1) == 3);
n_right_test = sum(sum((sim(net, X(:, n_train+n_val+1:n_train+n_val+n_test))
>= 0.5) == logical(y(:,n_train+n_val+1:n_train+n_val+n_test)), 1) == 3);

fprintf('Обучающие: %d/%d\nПроверочные: %d/%d\nТестовые: %d/%d\n', ...
        n_right_train, n_train, ...
        n_right_val, n_val, ...
        n_right_test, n_test);

%% пытаемся в картинку

h = 0.025;
```

```
n = int32((1.5 + 1.5) / h) + 1;

x = zeros(2, n * n);

for i = 1:n
    for j = 1:n
        x(:, (i-1)*n + j) = [-1.5 + (double(i)-1)*h; ...
                             1.5 - (double(j)-1)*h];
    end
end

image(permute(reshape(sim(net, x), [3 n n]), [2 3 1]));

%% Создание обучающей выборки и сети

f = @(t) sin(t .^ 2 - 7 * t);
t = 0:0.025:5;

X = t;
y = f(t);

net = feedforwardnet(20);

net = configure(net, X, y);
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'purelin';

net.trainFcn = 'trainoss'; % 'trainscg'/'trainoss'

n_train = ceil(length(X) * 0.9);
net.divideFcn = 'divideind';
net.divideParam.trainInd = 1:n_train;
net.divideParam.valInd = n_train+1:length(X);
net.divideParam.testInd = [];

net = init(net); % инициализаций весов

net.trainParam.epochs = 2 * 6000;
net.trainParam.max_fail = 6000;
net.trainParam.goal = 1e-10;

%% Обучение

net = train(net, X, y);

%% Метрики и графики для обучающего подмножества

disp(accuracy(sim(net, X(1:n_train)), y(1:n_train)));

p = plot(X(1:n_train), y(1:n_train), ...
         X(1:n_train), sim(net, X(1:n_train)), 'o');
p(1).Color = [1 0 0];
p(2).MarkerSize = 3;
p(2).Color = [0 0 0];
```

```
xlabel('$t$');
ylabel('$y$');
grid on;

%% Метрики и графики для контрольного подмножества

disp(accuracy(sim(net, X(n_train+1:length(X))), y(n_train+1:length(X))));

p = plot(X(n_train+1:length(X)), y(n_train+1:length(X)), ...
         X(n_train+1:length(X)), sim(net, X(n_train+1:length(X))), 'o');
p(1).Color = [1 0 0];
p(2).MarkerSize = 3;
p(2).Color = [0 0 0];
xlabel('$t$');
ylabel('$y$');
grid on;
```

Выводы

В лабораторной работе было проведено исследование свойств многослойной нейронной сети прямого распространения и алгоритмов её обучения. Было продемонстрировано применение сети в задачах классификации и аппроксимации функции.