Лабораторная работа 1

Цель работы

Изучение основных свойств и основ работы с GUI – интерфейсом пакета Neural Networks Toolbox в программной среде MatLab.

Содержание

Постановка задачи

Построить и натренировать нейронную сеть для вычисления функции

$$y = x_1 * sin(x_2),\ x_1,\ x_2 \in [-1; 1]$$

Исходные данные

В качестве исходных данных с помощью написанного скрипта были сгенерированы значения для x_1 и x_2, а затем по ним вычислены соответствующие значения y:

x =[-0.971744075369 -0.908085502026 -0.275879929245 -0.34233462519 -0.534516732717 0.166042363216 -0.492364276638 0.557852348271 0.803364729692 0.63603628916 -0.847704356355 0.0515582795791 0.949948818347 0.8234815387 -0.636596416475 -0.233058037265 0.412697908595 -0.259996319331 0.82482797877 -0.026113028086 ; -0.704394141359 0.662720300264 -0.0161233750095 -0.720889931336 0.815524377274 -0.48498938254 -0.685808227007 0.87643916787 -0.081113665884 0.736559535742 0.973447793537 0.223086935397 0.278783480788 -0.0828220159893 -0.0250693227568 0.371137173967 -0.114327560673 0.624237227564 -0.690246562882 0.408250344609]

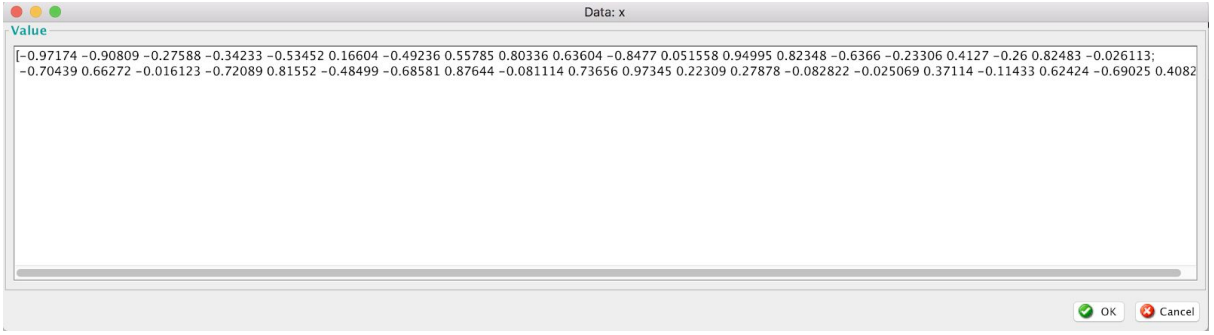
y =[0.629274528104 -0.55871195223 0.00444792283491 0.225959155974 -0.389173696763 -0.0774087781133 0.311813661833 0.428690121478 -0.0650924249649 0.427253084799 -0.700907259645 0.0114068110504

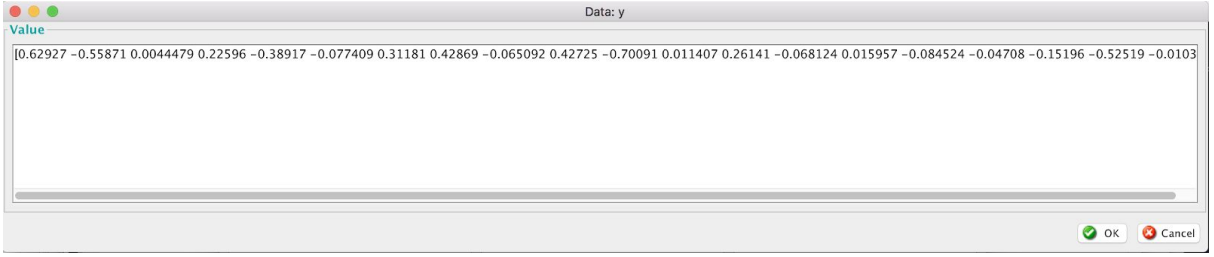
0.261412902368 -0.0681244556635 0.0159573694507 -0.0845244206891 -0.0470800263705 -0.151962264571 -0.52519051128 -0.0103669785159]

Алгоритм решения

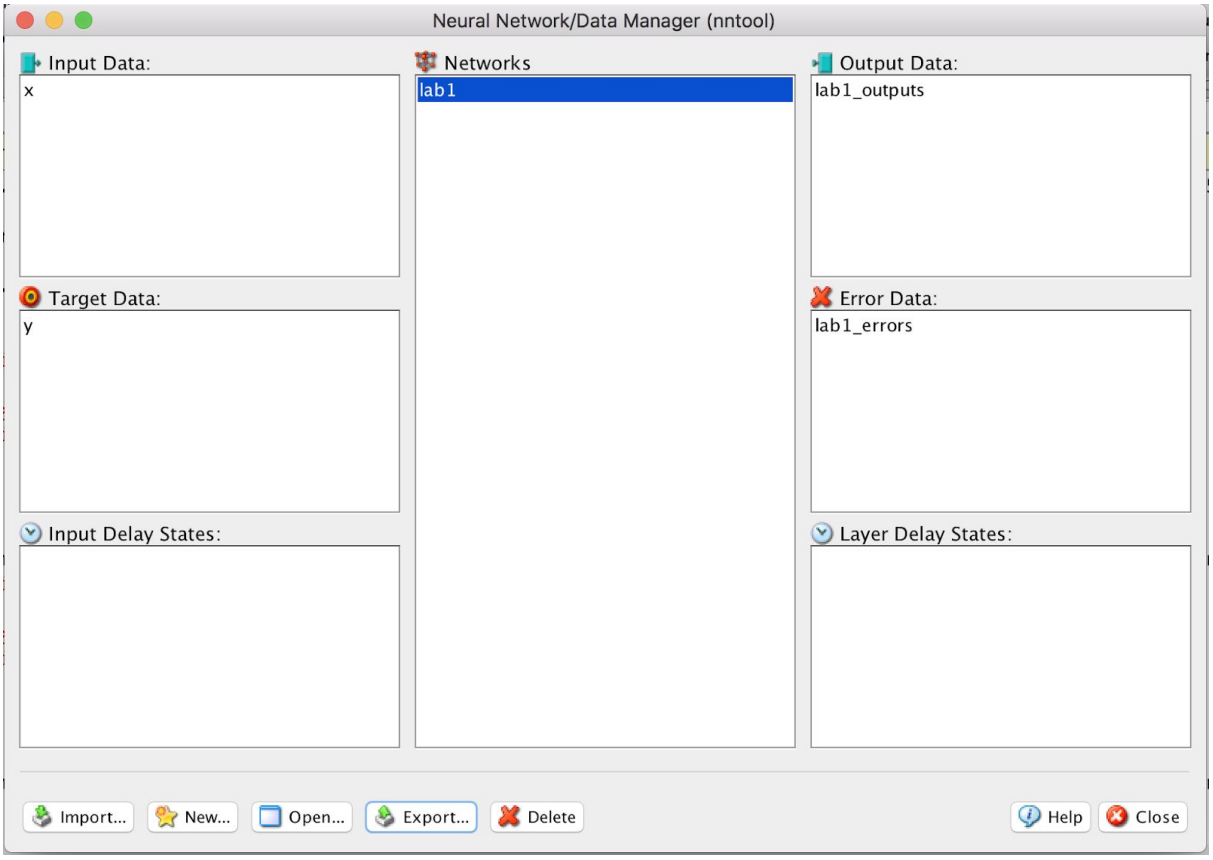
Построение сети

1. С помощью функции **nntool** откроем основное окно интерфейса и сформируем последовательность входов и целей в рабочей области.
2. Внесем исходные данные (*Input Data* и *Target Data)* в окно *Neural Network/Data Manager*.

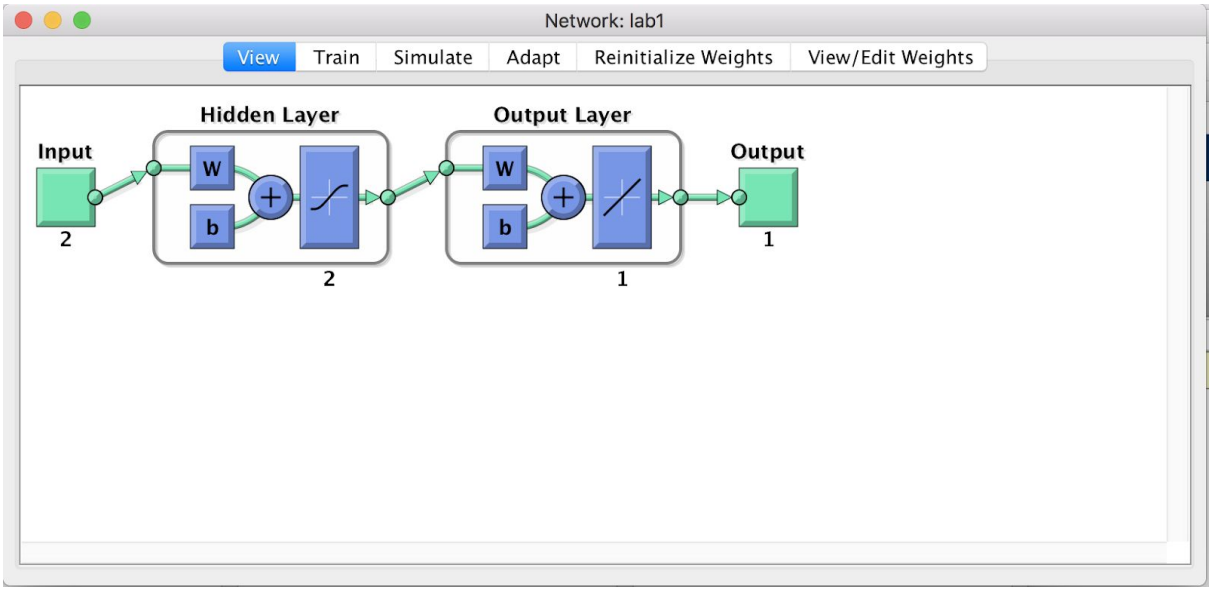




3. Добавим сеть во вкладке *Networks*

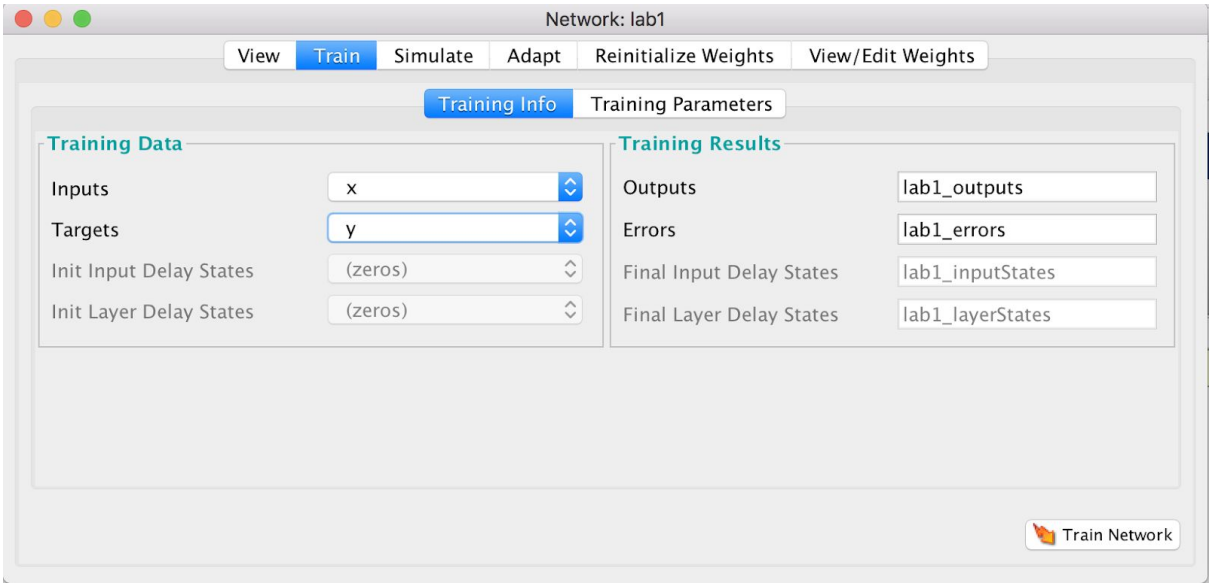


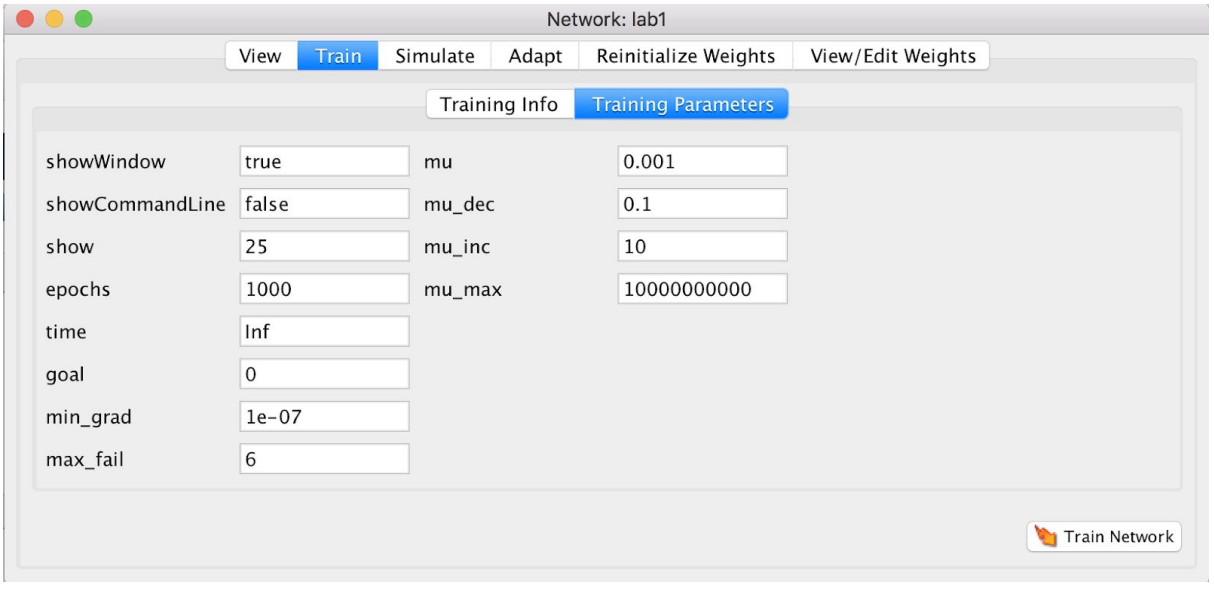
4. Зайдем в настройки сети. Выберем ее тип *Feed-forward backprop* с прямой передачей сигнала и обратным распространением ошибки.
5. Добавим сети *Input Data*, *Target Data*, количество нейронов первого слоя равно 2, функция активации во втором слое - линейная (*PURELIN*).



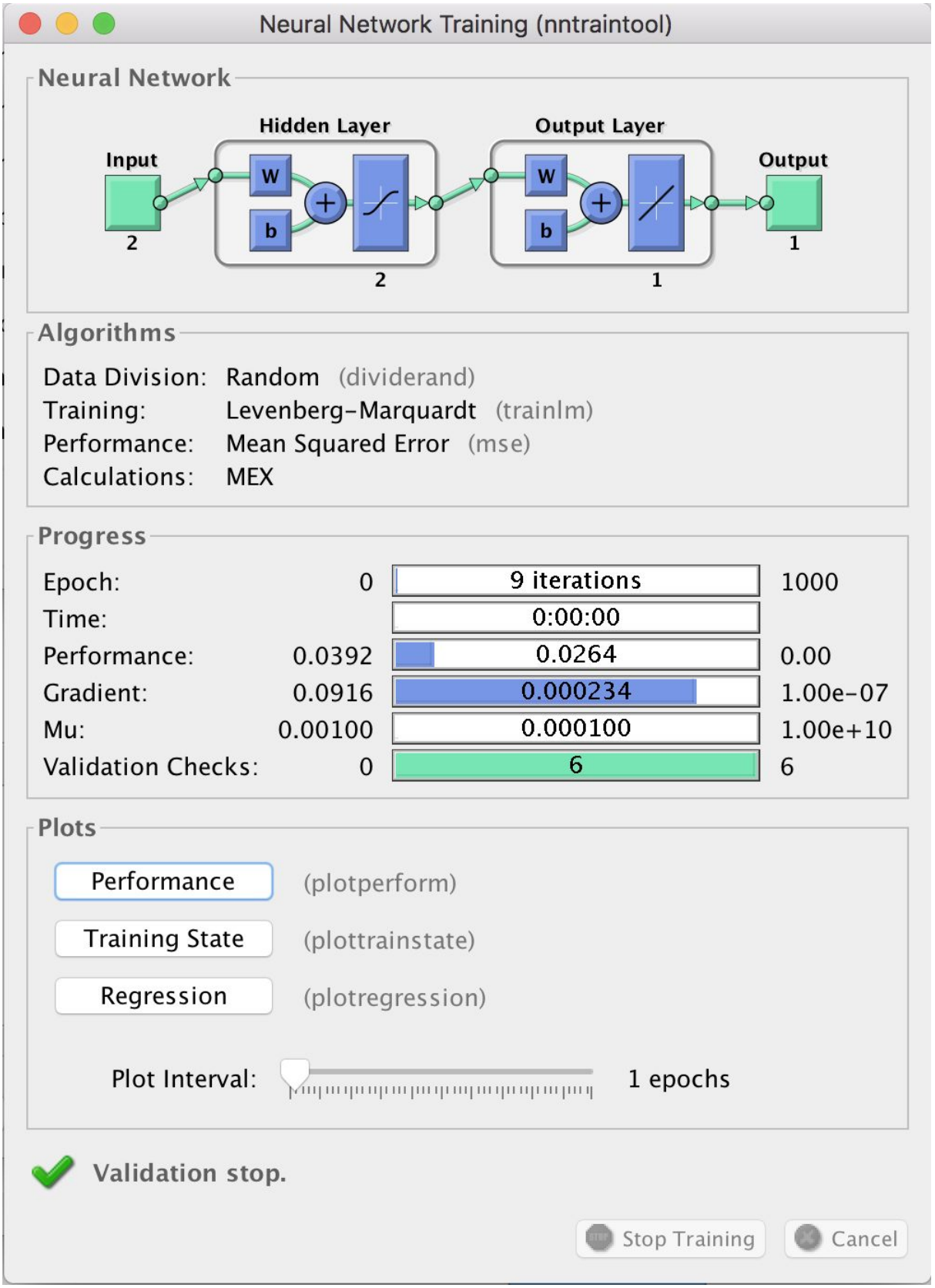
Обучение сети

Установим значения параметров во вкладке Train, как сказано в условии:





И нажмем *Train Network*. Качество обучения сети на выбранной обучающей последовательности показан на следующих графиках:


Neural Network Training (nntraintool)
Neural Network
Hidden Layer
Output Layer
Input
Output
W
b
2
1
Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX
Progress
Epoch: 0 9 iterations 1000
Time: 0:00:00
Performance: 0.0392 0.0264 0.00
Gradient: 0.0916 0.000234 1.00e-07
Mu: 0.00100 0.000100 1.00e+10
Validation Checks: 0 6 6
Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Validation stop.
Stop Training
Cancel

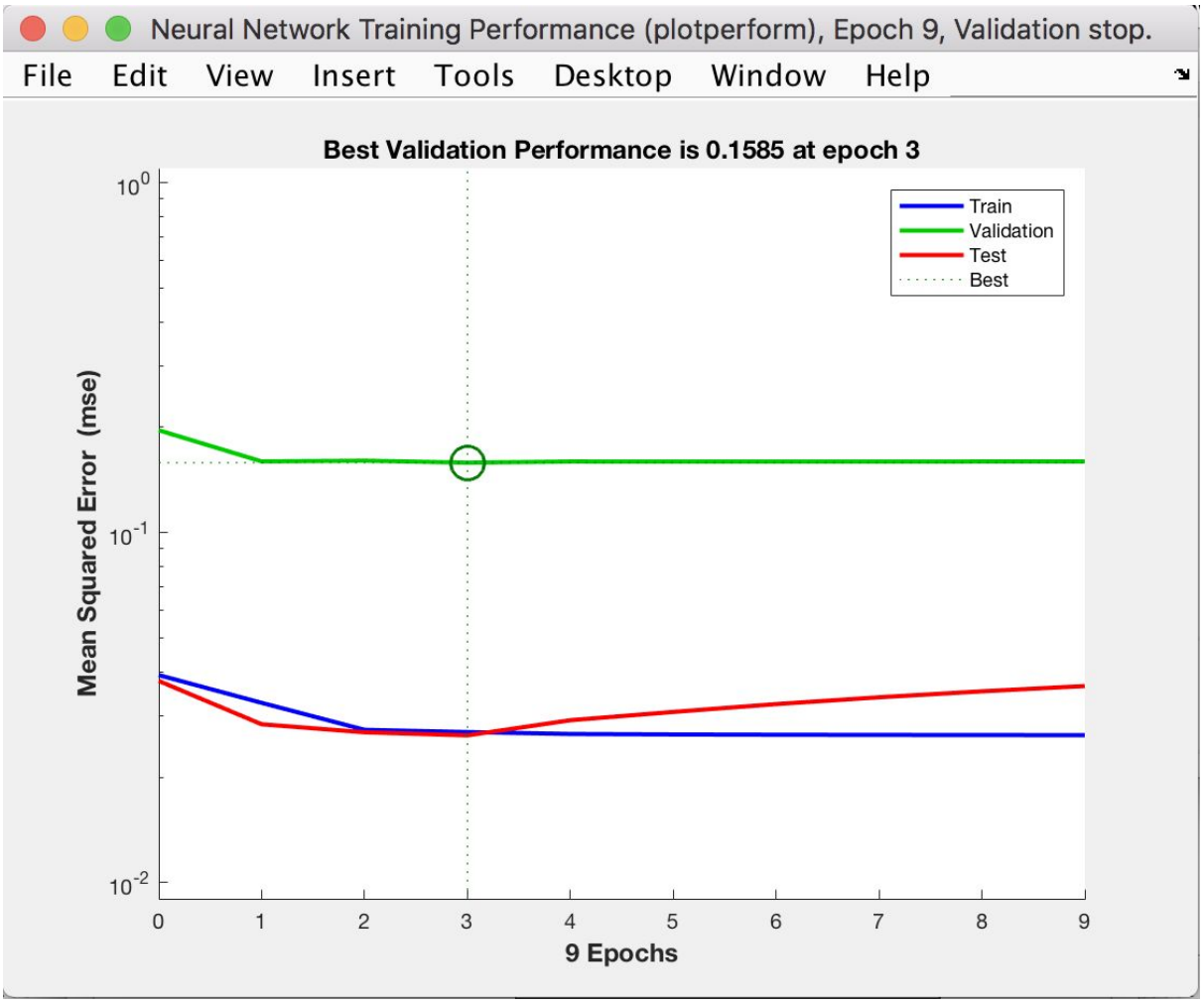


График Performance

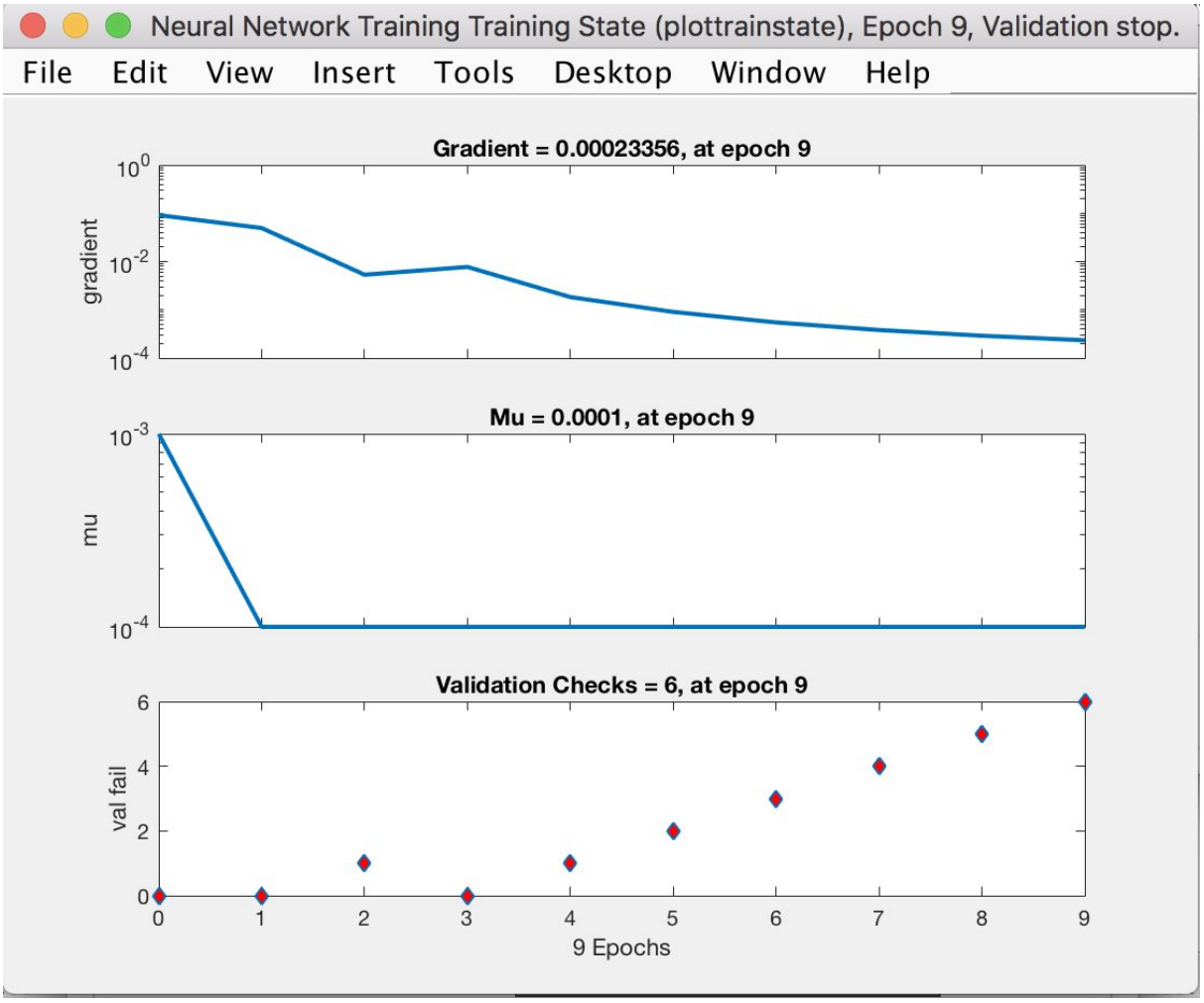


График Training State

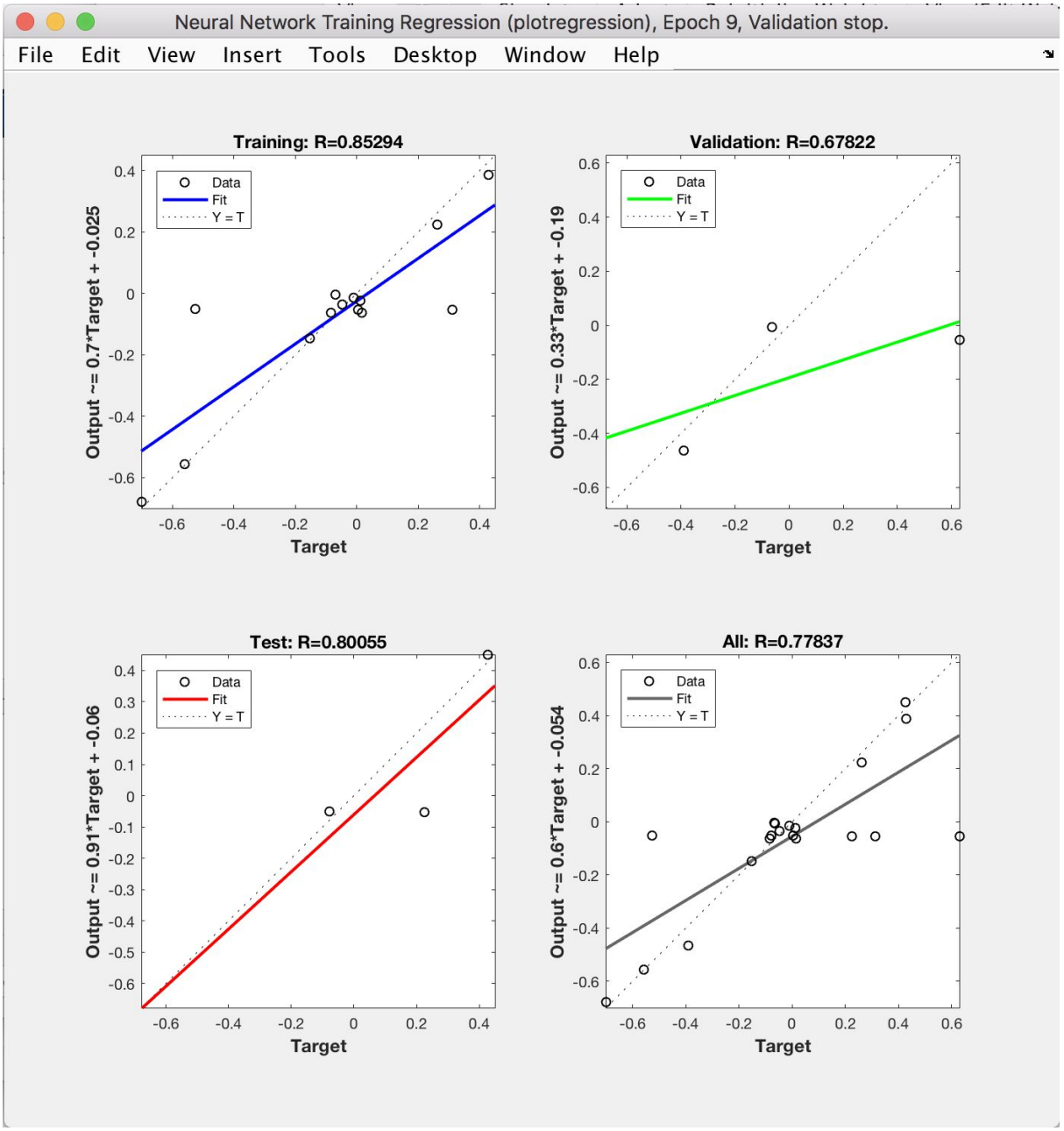


График Regression

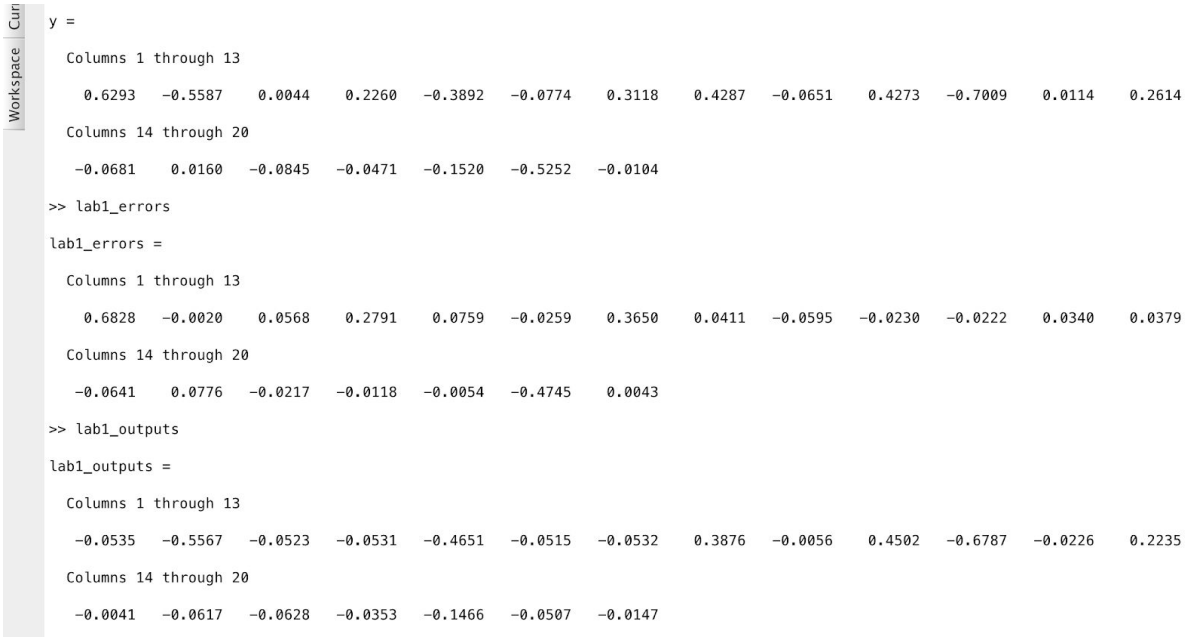
Результаты обучения

Результаты обучения сети можно посмотреть в окне *Network/Data Manager*, а именно выбирая параметры *lab1_outputs* и *lab1_errors*:

$lab_1 outputs$ = [-0.053503 -0.55667 -0.052302 -0.053126 -0.4651 -0.051539 -0.053213 0.3876 -0.005608 0.45023 -0.67868 -0.022636 0.22348 -0.0040626 -0.061654 -0.062779 -0.035276 -0.1466 -0.050654 -0.014699]

$lab_1 errors$ = [0.68278 -0.002041 0.05675 0.27909 0.075925 -0.025869 0.36503 0.041095 -0.059484 -0.022976 -0.022227 0.034042 0.037935 -0.064062 0.077612 -0.021746 -0.011804 -0.0053586 -0.47454 0.0043324]

Кроме того, мы можем выгрузить полученные результаты в рабочую область, что позволяет печатать/обрабатывать значения в дальнейшем:



Результаты работы сети

А также получить сам объект сети:

```
>> lab1

lab1 =

    Neural Network

               name: 'Custom Neural Network'
           userdata: (your custom info)

    dimensions:

          numInputs: 1
          numLayers: 2
         numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 9
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
```

Свойства объекта сети

Выводы

При выполнении первой лабораторной работы были получены необходимые навыки в построении базовой модели нейронной сети, умение устанавливать параметры сети, а также возможности ее обработки.

Лабораторная работа 2

Цель работы

Изучить свойства линейного нейрона

Содержание

Постановка задачи

Построить и натренировать простую линейную нейронную сеть (персептрон) для выполнения логических функций двух переменных OR, AND, NOR, NAND, XOR, NXOR.

Исходные данные

Для задач:

- AND:
  X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
  Y_train = {0, 0, 0, 1};
- OR:
  X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
  Y_train = {0, 1, 1, 1};
- NAND:
  X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
  Y_train = {1, 1, 1, 0};
- NOR:
  X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
  Y_train = {1, 0, 0, 0};
- XOR:
  X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
  Y_train = {0, 1, 1, 0};
- NXOR:

X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
Y_train = {1, 0, 0, 1};

Алгоритм решения

Обучение OR, NOR, AND, NAND

Построим персептрон с помощью функции *newp*:

```
net = newp(inputs, neurons);
```

где `inputs` - [0 1; 0 1], поскольку все логические значения вкладываются в этот интервал, `neurons` - 1, поскольку достаточно одного.

Количество циклов адаптации:

```
net.adaptParam.passes = 20;
```

Тренируем сеть:

```
net = train(net, X_train, Y_train);
```

Симуляция работы нейрона на тестовых входах:

```
Y_test = sim(net, X_test);
```

Все это помещается в один скрипт:

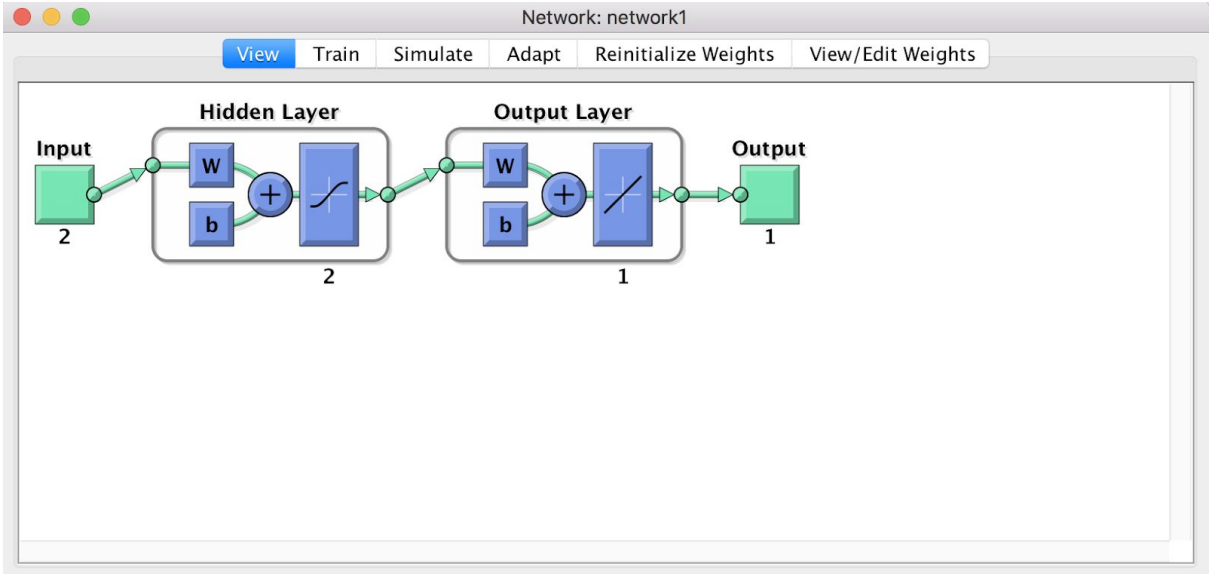
```
function Main()
    disp('Solving for loqical AND');
    X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
    Y_train = {0, 0, 0, 1};
    X_test = X_train;
    Y = build_logiql(X_train, Y_train, X_test, [0 1; 0
1], 1);
    % Output:
    Y

    disp('Solving for loqical OR');
    X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
    Y_train = {0, 1, 1, 1};
```

```
        X_test = X_train;
        Y = build_logiql(X_train, Y_train, X_test, [0 1; 0
1], 1);
        % Output:
        Y

        disp('Solving for loqical NAND');
        X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
        Y_train = {1, 1, 1, 0};
        X_test = X_train;
        Y = build_logiql(X_train, Y_train, X_test, [0 1; 0
1], 1);
        % Output:
        Y

        disp('Solving for loqical NOR');
        X_train = {[0; 0] [0; 1] [1; 0] [1; 1]};
        Y_train = {1, 0, 0, 0};
        X_test = X_train;
        Y = build_logiql(X_train, Y_train, X_test, [0 1; 0
1], 1);
        % Output:
        Y
    end

    function [Y_test] = build_logiql(X_train, Y_train,
X_test, inputs, neurons)
    %     declaring a network with two inputs [0, 1] interval
each
        net = newp(inputs, neurons);
        net.adaptParam.passes = 20;
        net = train(net, X_train, Y_train);
        Y_test = sim(net, X_test);
    end
```

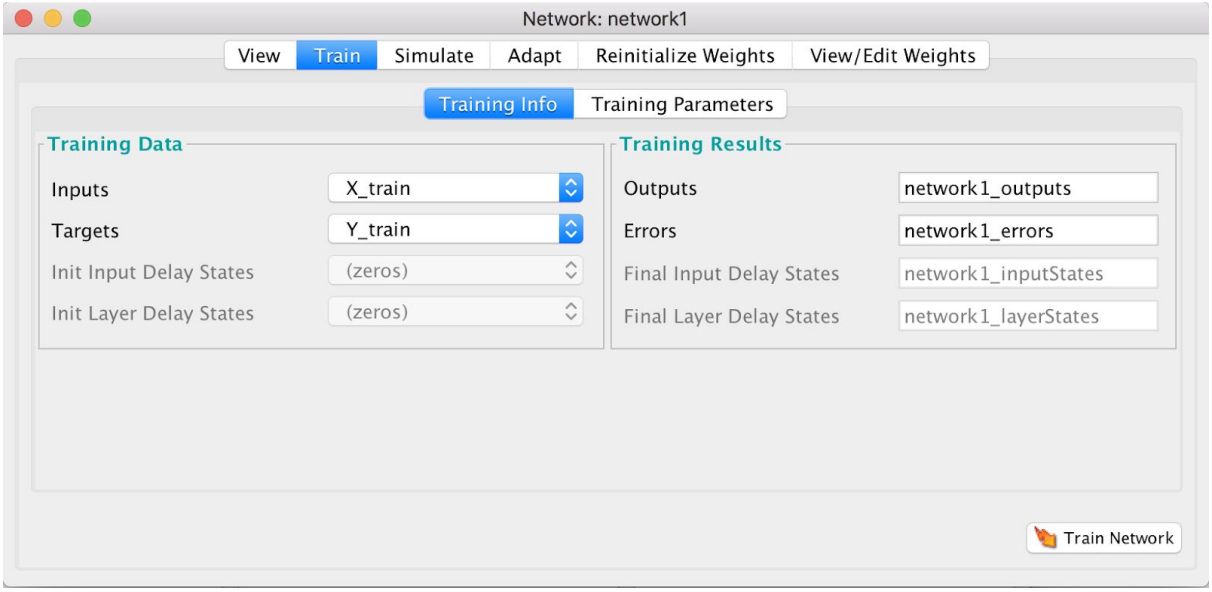
Обучение XOR, NXOR

Как известно, точки выполнения операций XOR, NXOR не могут быть линейно разделяемыми, поэтому использование линейного нейрона (персептрона) не представляется возможным. Для обучения нейросети

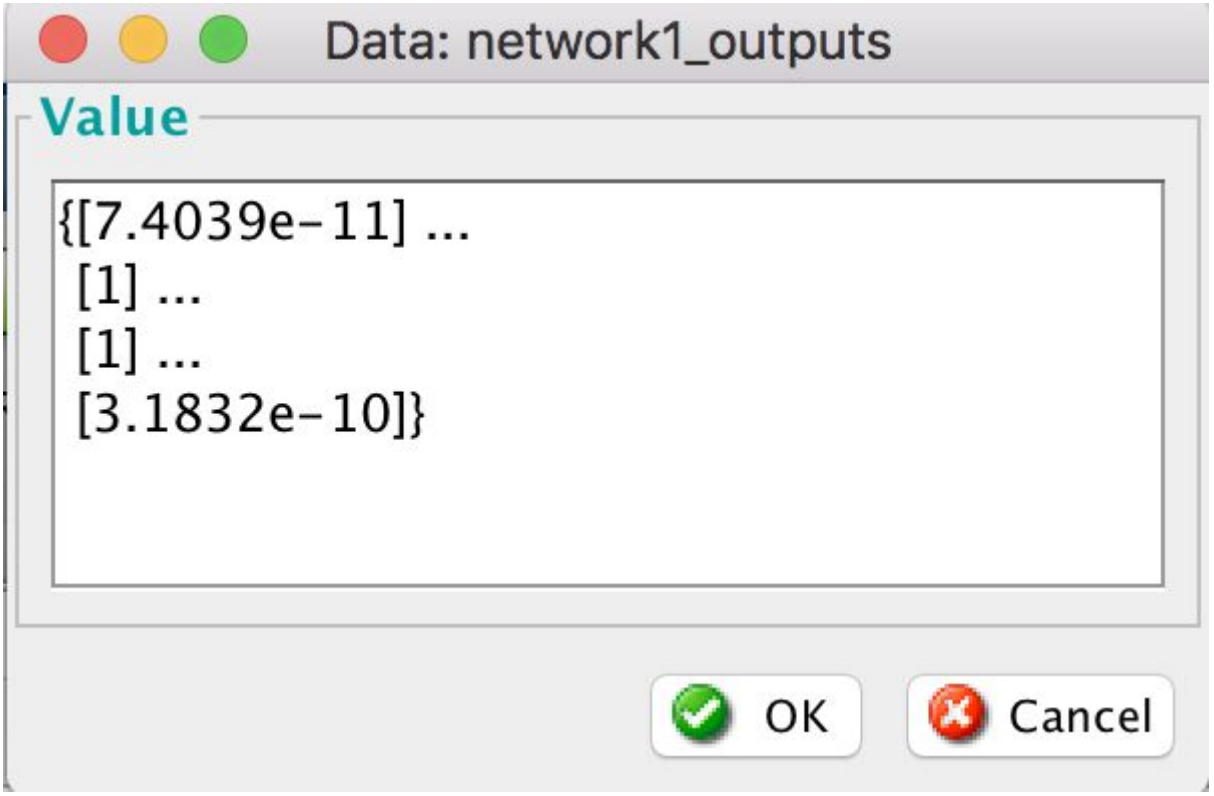
выполнению операций необходимо использовать как минимум 2 слоя. Для этого используем полученные знания и навыки из предыдущей л.р. Шаги будут выполнены похожие, поэтому приведу лишь необходимые скриншоты:



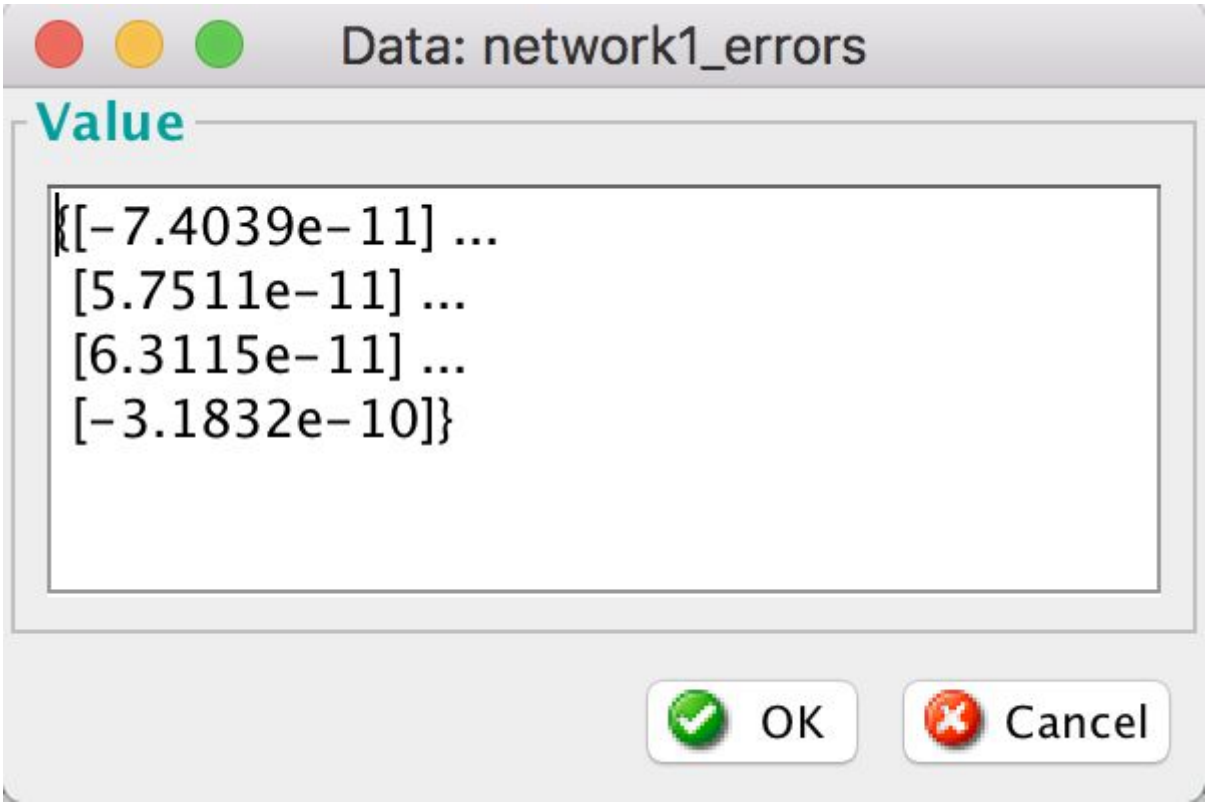
Создали сеть с двумя скрытыми слоями



Указали Inputs/Targets



Как видим, для XOR выход тестовых данных сходится с заложенными тестовыми данными



Ошибки нулевые

Аналогичная процедура проводится для операции NXOR, изменения касаются лишь входных параметров.

Выводы

При выполнении данной лабораторной работы была изучена теория персептрона, были получены навыки в его построении и применении для простых линейно разделяемых функций, а также с использованием графического интерфейса была построена нейронная сеть с двумя скрытыми слоями для обучения с использованием данных, линейно не разделяемых.

Лабораторная работа 3

Цель работы

Изучить возможности многослойного персептрона как универсального аппроксиматора и классификатора.

Содержание

Постановка задачи

Сравнить в производительности и качестве методы обучения многослойного персептрона.

Используемые функции

Функция, которая для переданного исследуемого метода генерирует обучающую выборку и вызывает обработчик метрик:

```
function try_method(method)
    [X,Y] = meshgrid(-2:.2:2, -4:.4:4);
    T = X .* exp(-X.^2 - Y.^2);
    P = [X; Y];

    figure(1);
    surf(X,Y,T, 'FaceColor','g');
    P = num2cell(P, 1);
    T = num2cell(T, 1);

    net = feedforwardnet(2, method);
    net.trainParam.show = 100;
    net.trainParam.lr = 0.001;

    net = train(net, P, T);
    Z_predict = sim(net, P);
    output_res(X, Y, Z_predict, T, 2, 'r');

    [X,Y] = meshgrid(-1:.1:1, -2:.2:2);
    P = [X; Y];
    P = num2cell(P, 1);
    Z_predict = sim(net, P);
    T = X .* exp(-X.^2 - Y.^2);
    T = num2cell(T, 1);
    output_res(X, Y, Z_predict, T, 3, 'b');
end
```

Функция, которая рисует поверхность, а также выводит MSE:

```
function output_res(X, Y, Z, T, fig, color)
    Z = cell2mat(Z);
    e = Z - cell2mat(T);
    disp('MSE');
    disp(mse(e));

    figure(fig);
    surf(X, Y, Z, 'FaceColor', color);
end
```

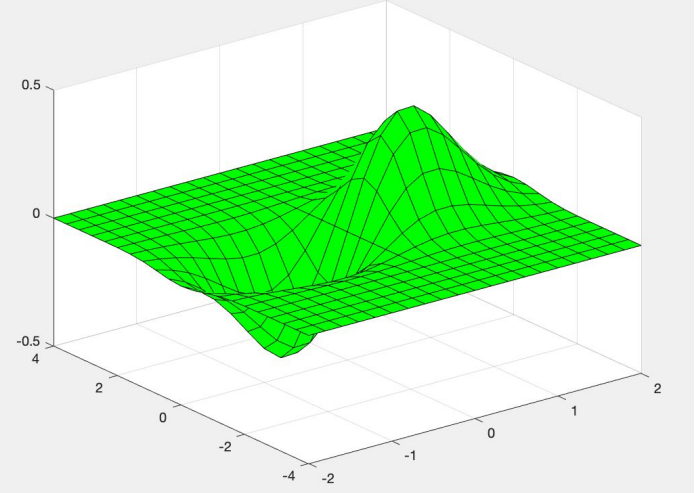
Функция, замеряющая время выполнения:

```
function measure_method(method)
    start = tic;
    try_method(method);
    disp('Time');
    disp(toc(start));
end
```

Главная функция:

```
function Main()
    measure_method('traingd');
    measure_method('trainrp');
    measure_method('trainlm');
end
```

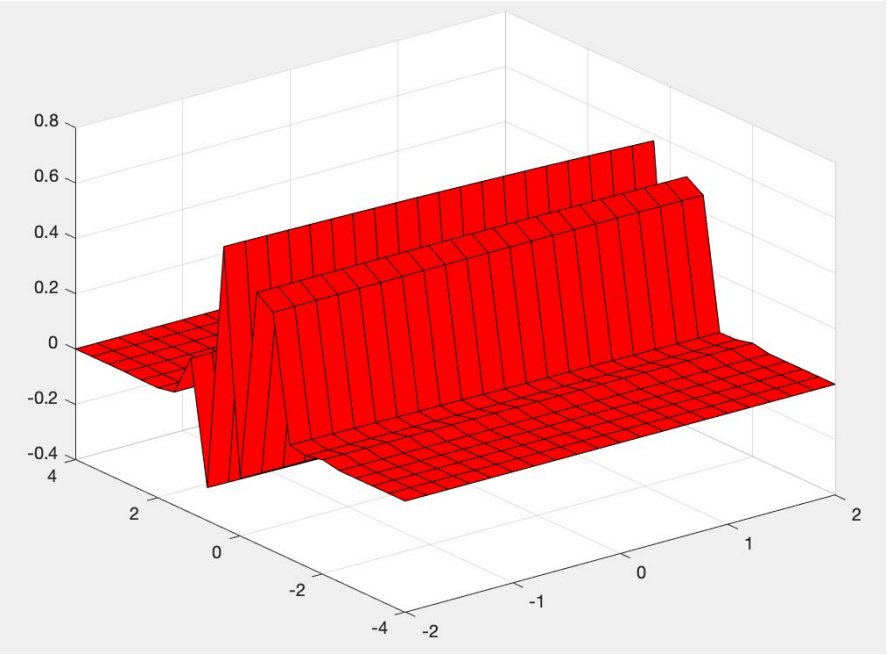
Отрисованная функция для обучения для всех алгоритмов:

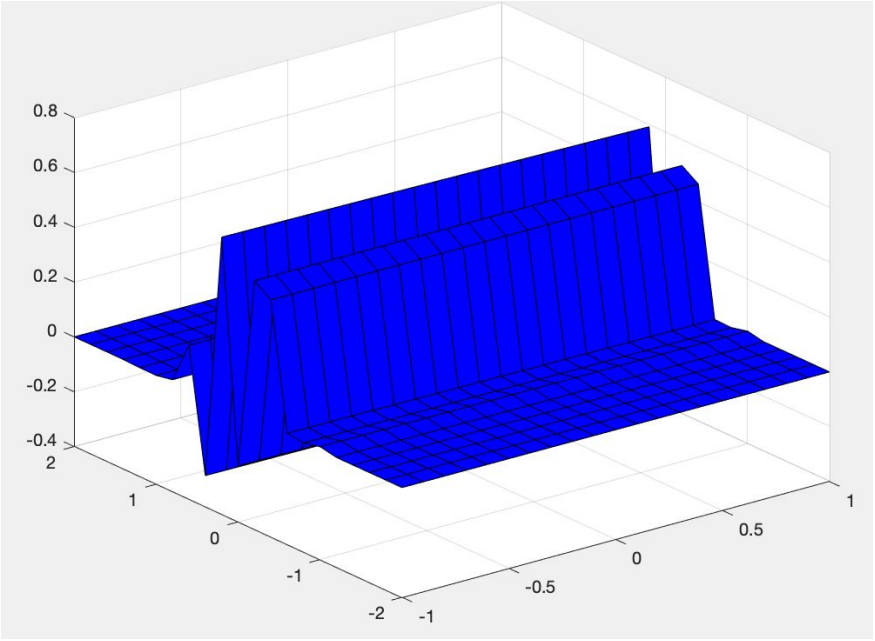


В качестве сравнения выберем алгоритмы `traingd`, `trainrp`, `traingdx`.

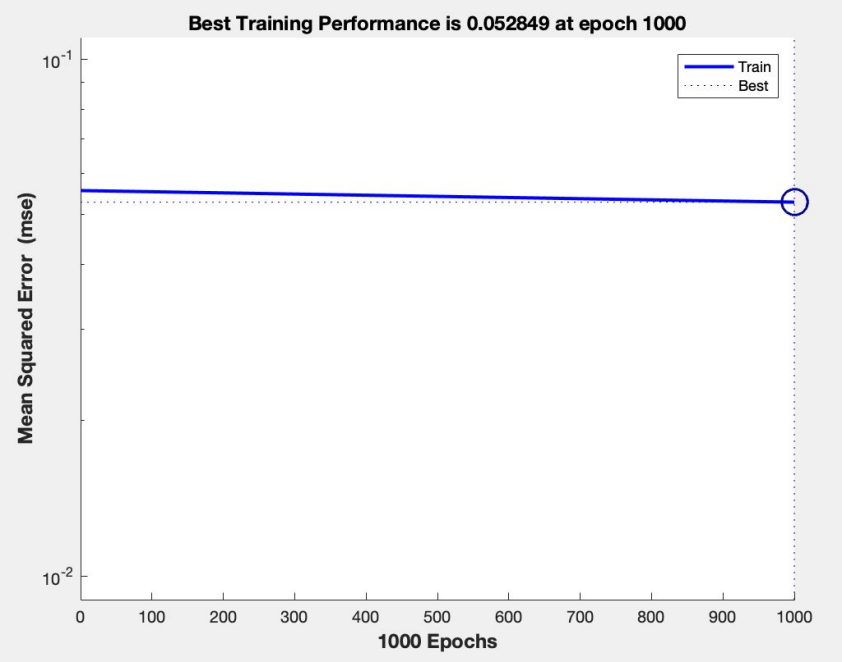
Алгоритм traingd

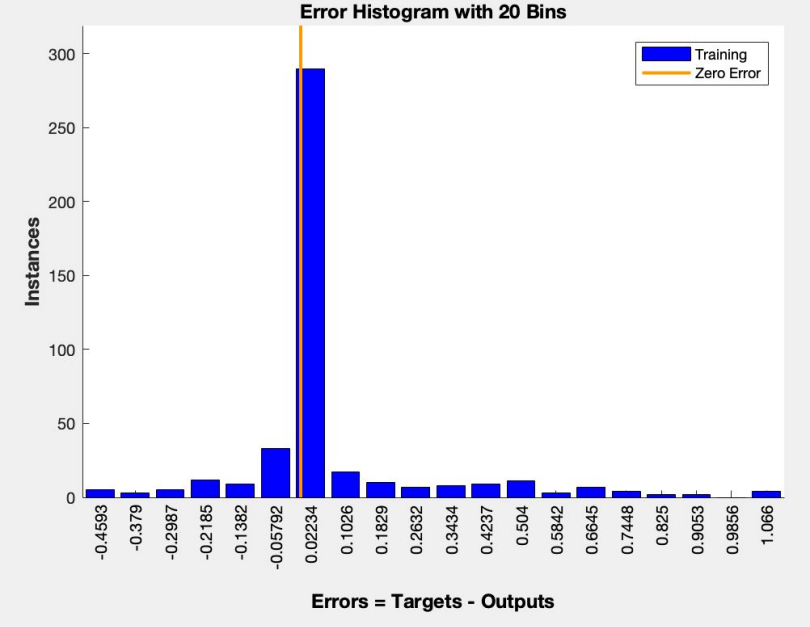
Поверхности при запуске на данных для обучения и тестовых:

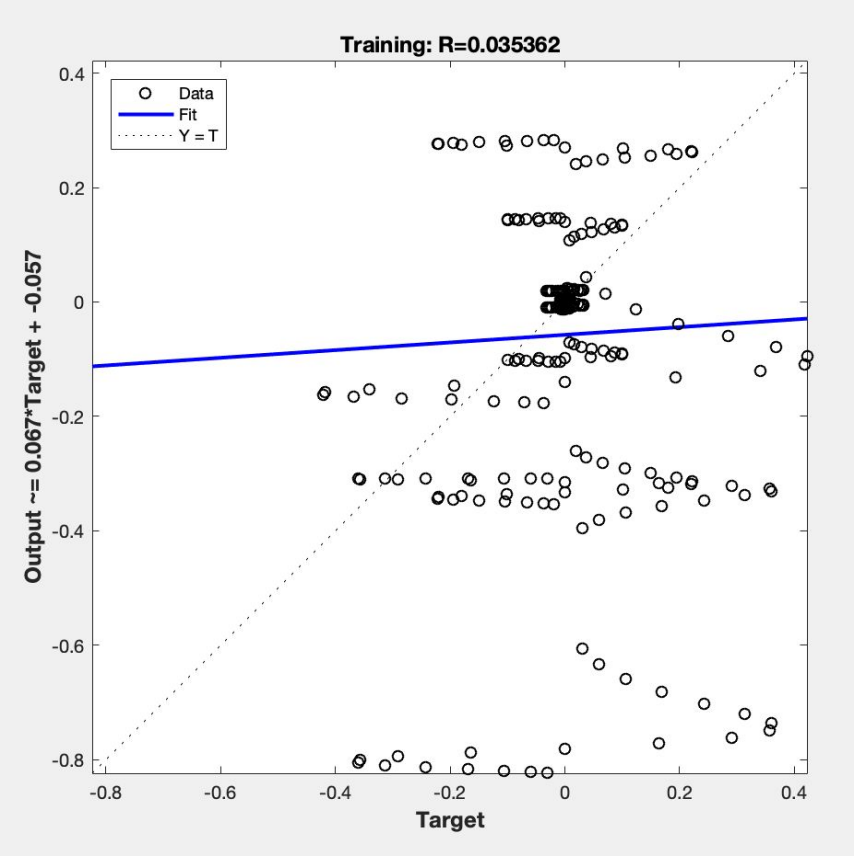




Графики, показывающие качество модели:


Best Training Performance is 0.052849 at epoch 1000
Train
Best
Mean Squared Error (mse)
10^{-1}
10^{-2}
0
100
200
300
400
500
600
700
800
900
1000
1000 Epochs


Error Histogram with 20 Bins
Training
Zero Error
Instances
300
250
200
150
100
50
0
-0.4593
-0.379
-0.2987
-0.2185
-0.1382
-0.05792
0.02234
0.1026
0.1829
0.2632
0.3434
0.4237
0.504
0.5842
0.6645
0.7448
0.825
0.9053
0.9856
1.066
Errors = Targets - Outputs

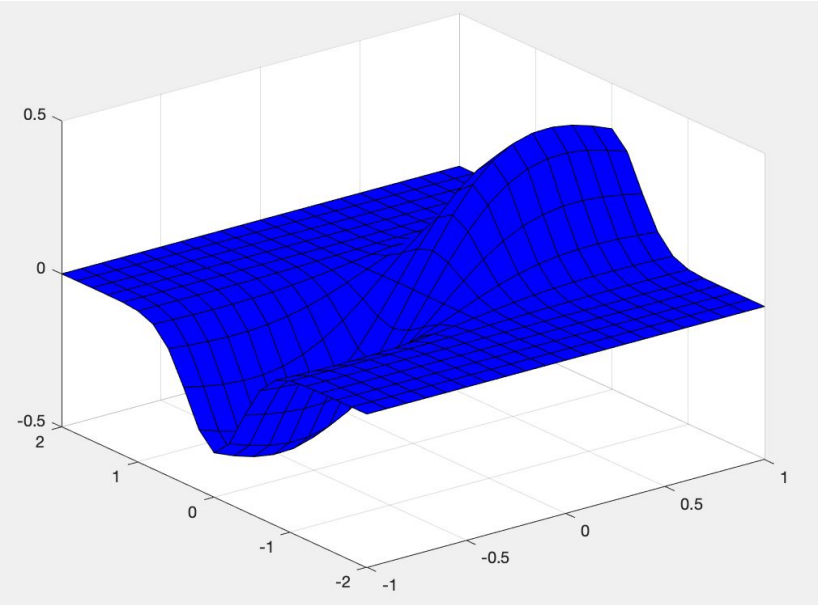


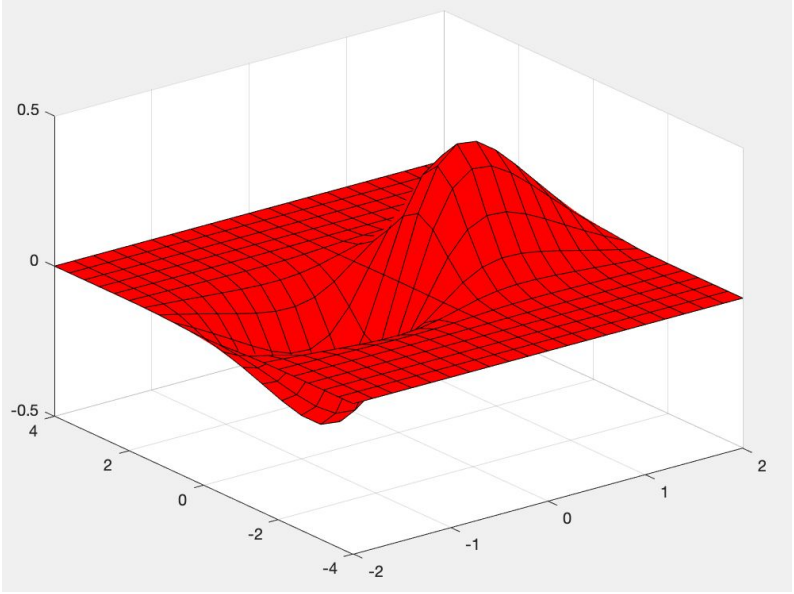
Ошибка и время работы:

```
MSE
    0.0821
Time
    1.9608
```

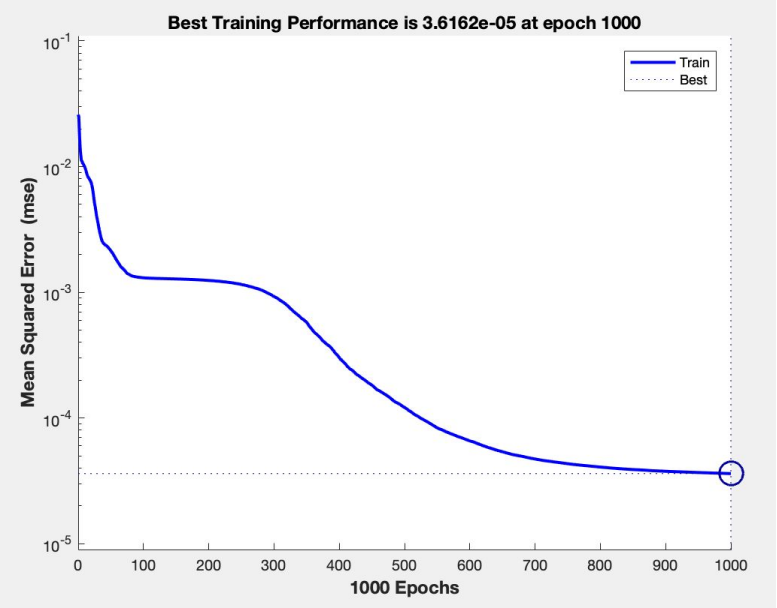
Алгоритм trainrp

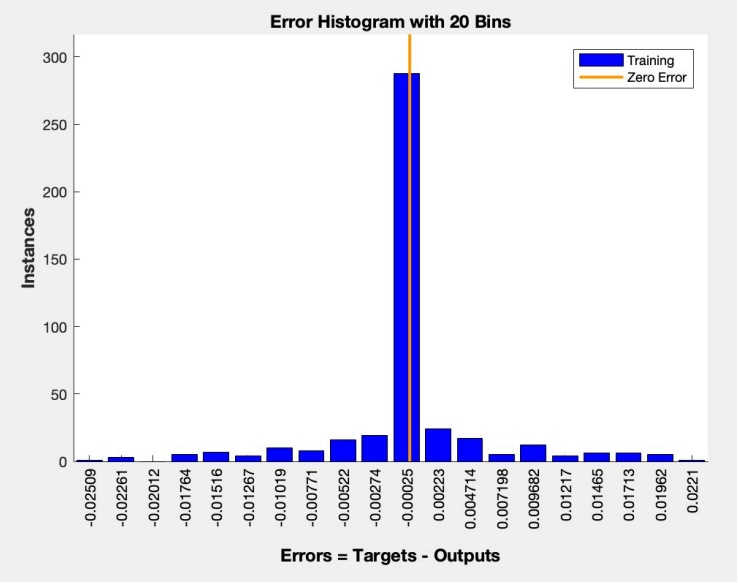
Поверхности при запуске на данных для обучения и тестовых:

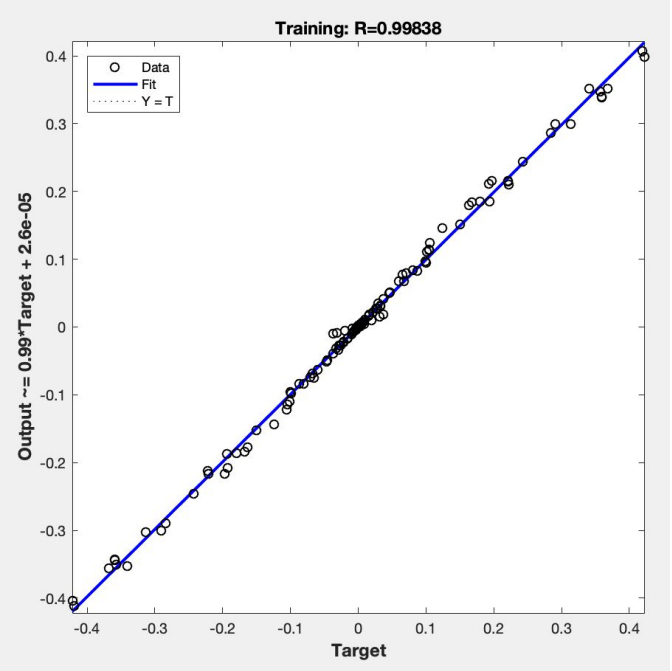




Графики, показывающие качество модели:


Best Training Performance is 3.6162e-05 at epoch 1000
Train
Best
Mean Squared Error (mse)
1000 Epochs


Error Histogram with 20 Bins
Training
Zero Error
Instances
Errors = Targets - Outputs

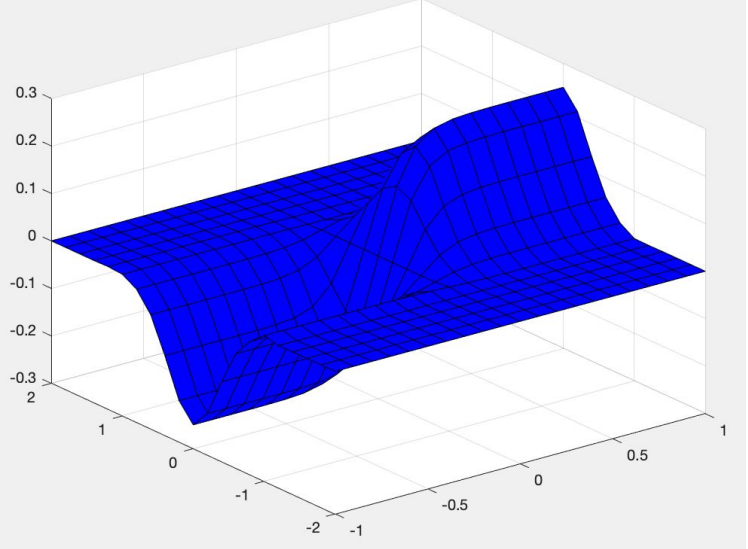


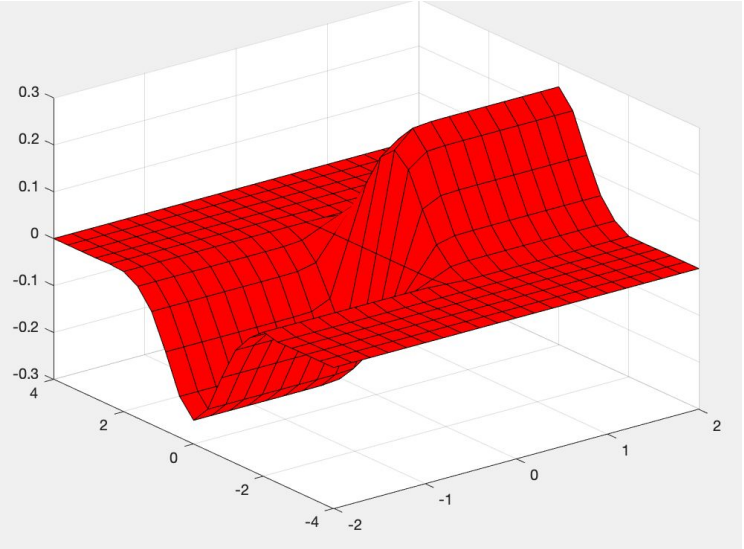
Ошибка и время работы:

```
MSE
    0.0084
Time
    2.5101
```

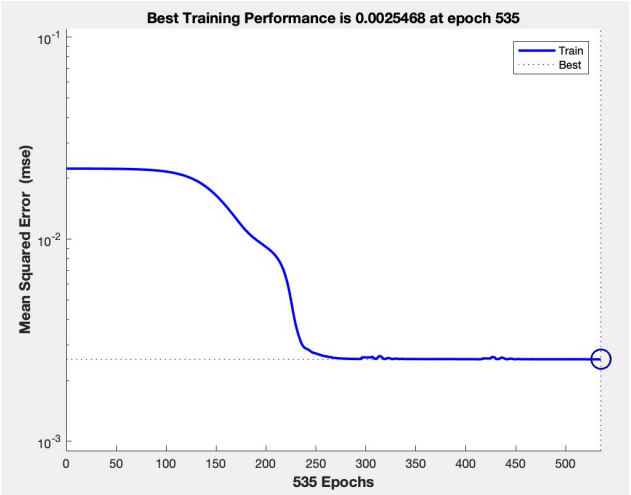
Алгоритм traingdx

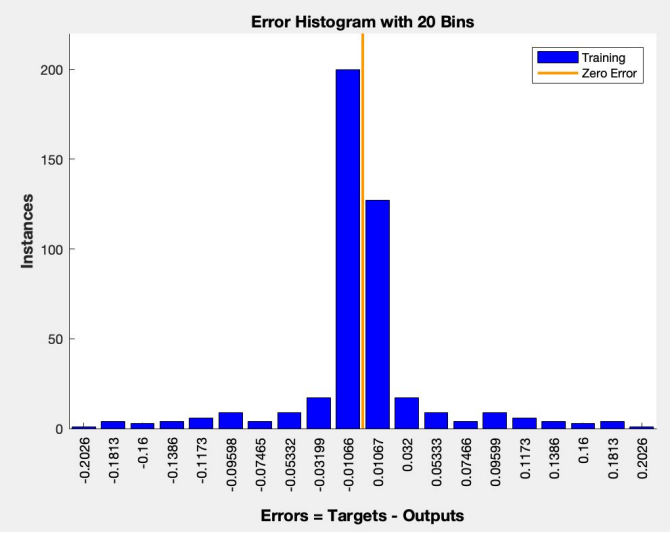
Поверхности при запуске на данных для обучения и тестовых:

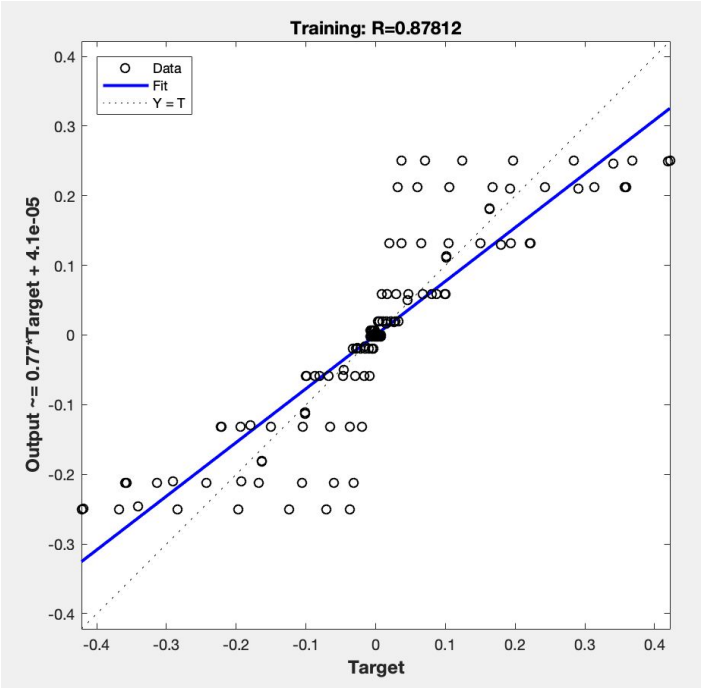




Графики, показывающие качество модели:







Ошибка и время работы:

```
MSE
    0.0137
Time
    3.3786
```

Выводы

Сравним полученные результаты (чем раньше, тем лучше):

1. MSE: `trainrp` - `traingdx` - `traingd`
2. Time: `traingd` - `trainrp` - `traingdx`
3. Сходимость (график MSE): `traingd`, `trainrp` - зависит от коэффициента `lr`, остальные два, `traingdx`, как и видно с его графика, при некотором количестве эпох (~250) сходится и дальше почти не улучшается.

По итогу в этой работе были получены знания в построении моделей нейронных сетей для Backpropagation алгоритмов.

Лабораторная работа 4.1

Цель работы

Выработать практические навыки при исследовании возможностей применения нейронных сетей для решения прикладных задач при помощи пакета прикладных программ MatLab.

Аппроксимировать заданный полином.

Содержание

Постановка задачи

Применение нейронных сетей для аппроксимации полинома

Используемые функции

Главная функция:

```
function Main()
    range = -1:0.1:1;
    N = size(range, 2);

    a = -1; b = -1; c = -1; d = 1; e = 1; f = 1;

    [x1, x2] = meshgrid(range, range);
    x1 = reshape(x1, 1, []);
    x2 = reshape(x2, 1, []);

    Y = a*(x1.^2) + b*(x2.^2) + c*x1.*x2 + d*x1 + e*x2 +
f;
    show_surf(1, x1, x2, Y, N);

%     net = feedforwardnet(2, 'traingda');
%     try_net(net, x1, x2, Y, N);

    net = feedforwardnet(2, 'traingdx');
    try_net(net, x1, x2, Y, N);
end
```

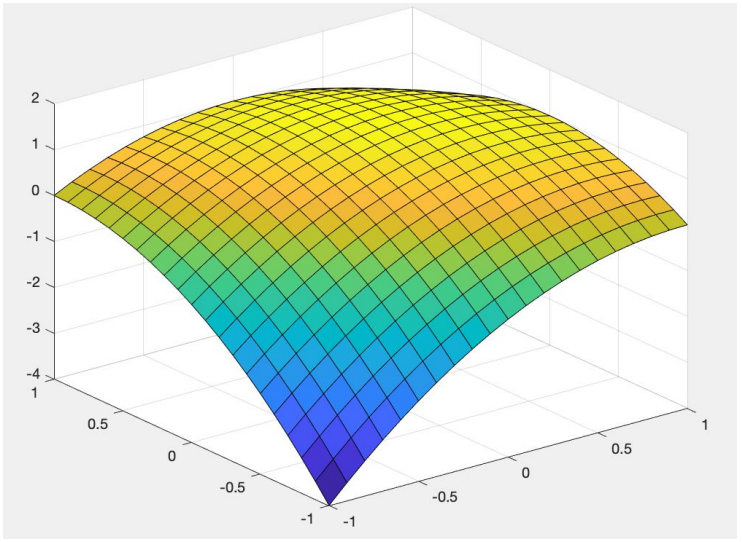
Здесь устанавливаются исходные данные согласно варианту и вызывается метод `try_net` для обучения сети.

Функция обучения и запуска сети:

```
function try_net(net, x1, x2, Y, N)
```

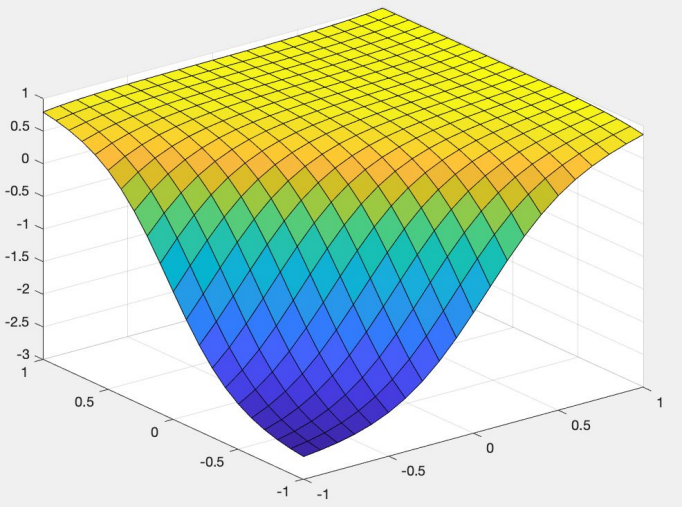
```
    net = configure(net, [x1; x2], Y);

    net.trainParam.epochs = 1000;
    net.trainParam.goal = 0.01;

    net = train(net, [x1; x2], Y);
    outputs = net([x1; x2]);

    show_surf(2, x1, x2, outputs, N);

    errors = outputs - Y;
    figure(3); plot(errors);
end

function show_surf(fignum, x1, x2, y, N)
    figure(fignum);
    surf(reshape(x1, N, N), reshape(x2, N, N), reshape(y,
N, N));
end
```

График исходной функции:



Алгоритм traingdx

График функции на исходных данных, запущенных через обученную сеть:



Сводная информация по обучению:

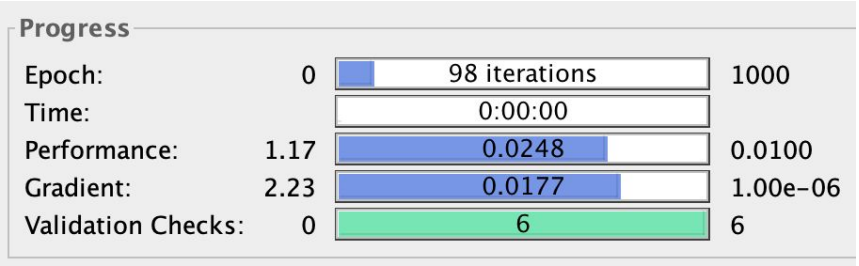


График ошибки:

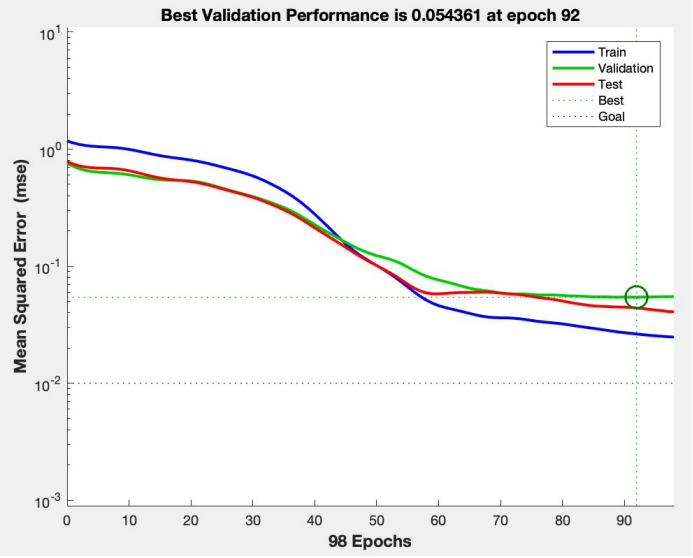


График Training State:

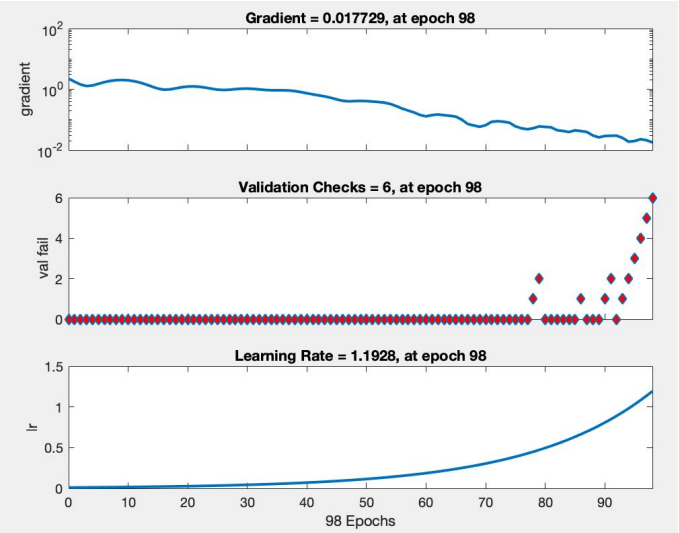
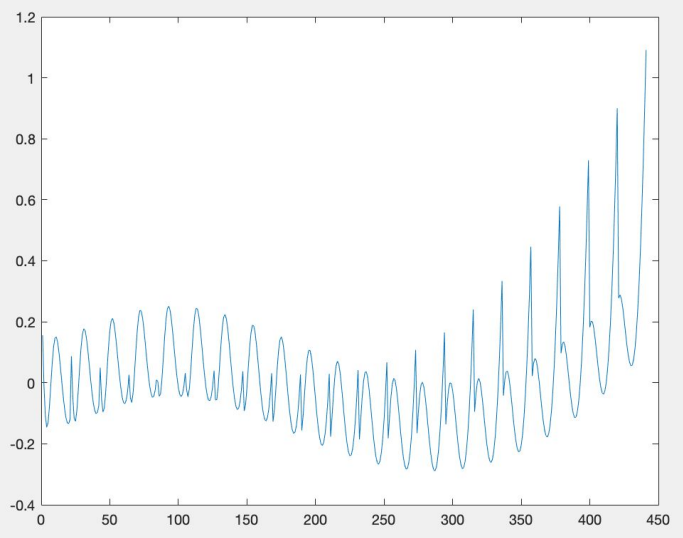
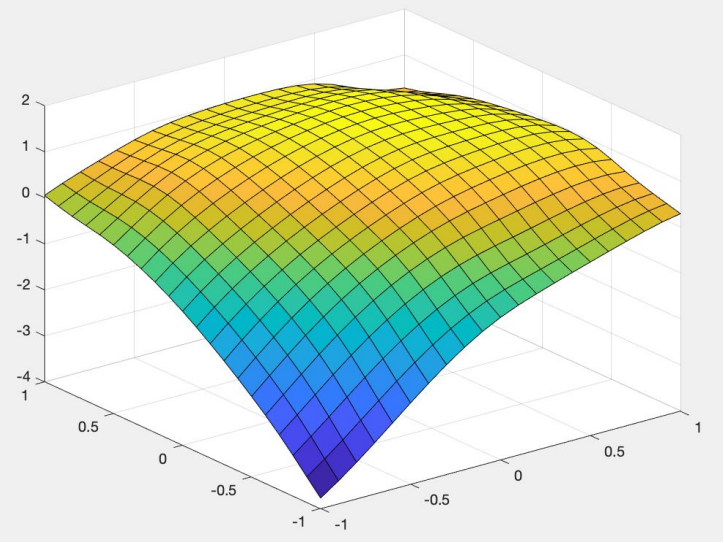


График разности `outputs - Y`:



Алгоритм trainlm (default)

График функции на исходных данных, запущенных через обученную сеть:



Сводная информация по обучению:

Progress

Epoch:	0	3 iterations	1000
Time:		0:00:00	
Performance:	27.2	0.00302	0.0100
Gradient:	40.5	0.125	1.00e-06
Mu:	0.00100	0.000100	1.00e+10
Validation Checks:	0	0	6

График ошибки:

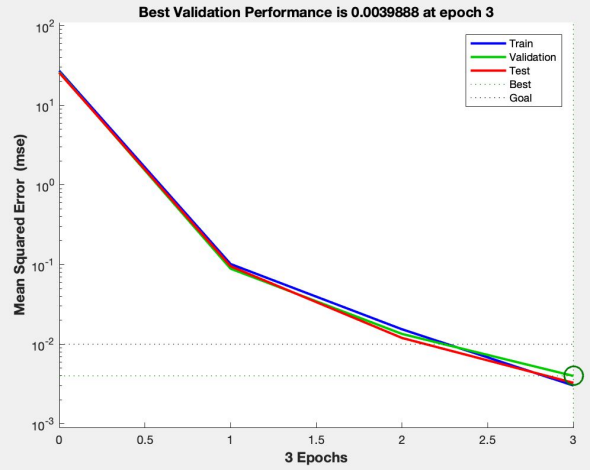


График Training State:

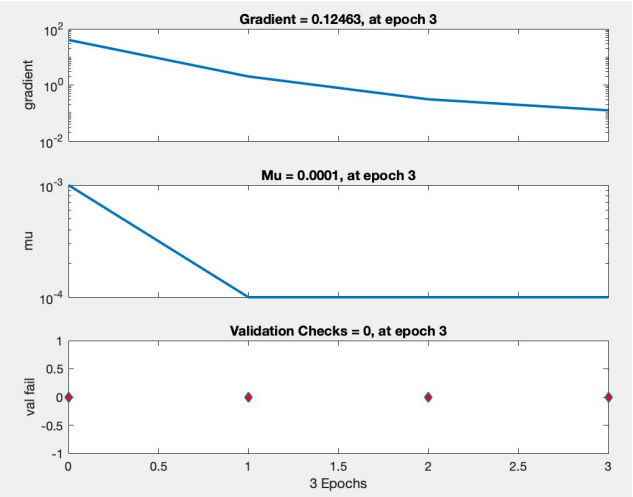
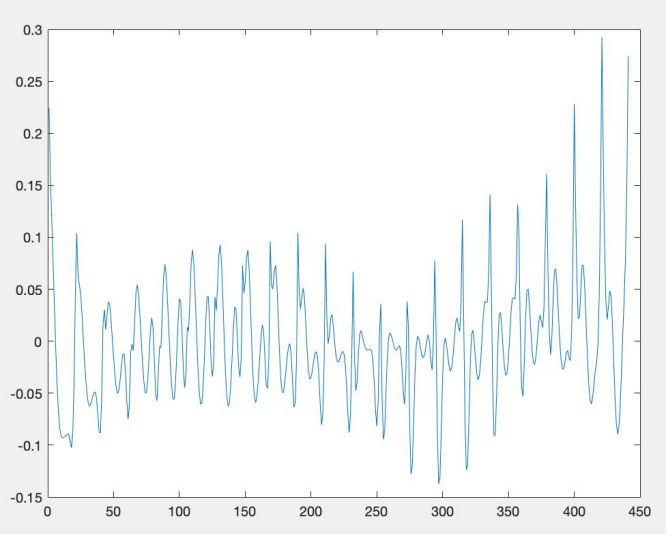


График разности `outputs - Y`:



Выводы

Были опробованы два алгоритма: traingdx и trainlm. По результатам обучения лучше себя показал trainlm, поскольку он быстрее (меньше эпох) и качественнее.

Лабораторная работа 4.2

Цель работы

Выработать практические навыки при исследовании возможностей применения нейронных сетей для решения прикладных задач при помощи пакета прикладных программ MatLab.

Аппроксимировать заданную функцию.

Постановка задачи

Применение нейронных сетей для аппроксимации функции

Используемые функции

Главная функция, где задаются исходные данные:

```
function Main()
    X = 0:0.1:10;
    Y = X .* sin(X.^2 - X .* 5);
%     try_net(X, Y, 20);
    try_net(X, Y, 50);
end
```

Функция запуска сети и вывода результатов:

```
function try_net(X, Y, layers)
    net = feedforwardnet(layers);
    net = configure(net, X, Y);

    net.trainParam.epochs = 1000;
    net.trainParam.max_fail = 100;
    net.trainParam.mu_dec = 0.001;
    net.trainParam.mu_inc = 3.0;

    net = train(net, X, Y);
    outputs = net(X);
    show_performance(X, Y, outputs);
end
```

```
    function show_performance(X, Y, outputs)
        figure(1); plot(X, Y, X, outputs,
'-r','MarkerSize',6,'LineWidth', 2);

        errors = outputs - Y;
        figure(2); plot(errors);
    end
```

Сравнение по количеству внутренних нейронов

1. 20 слоев:

Исходная функция и аппроксимирующая функция:

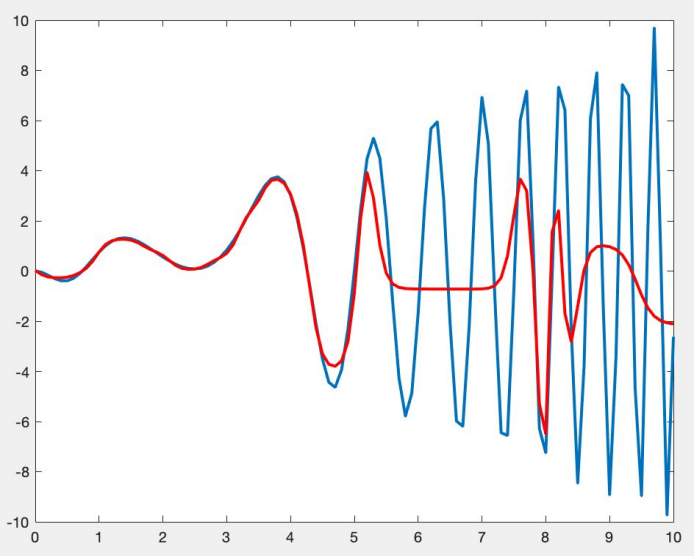
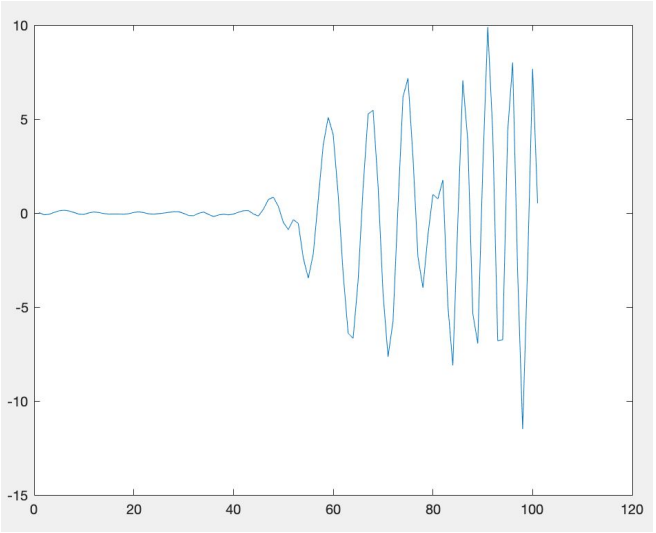


График разности `outputs - Y`:



2. 50 слоев:

Исходная и аппроксимирующая функция:

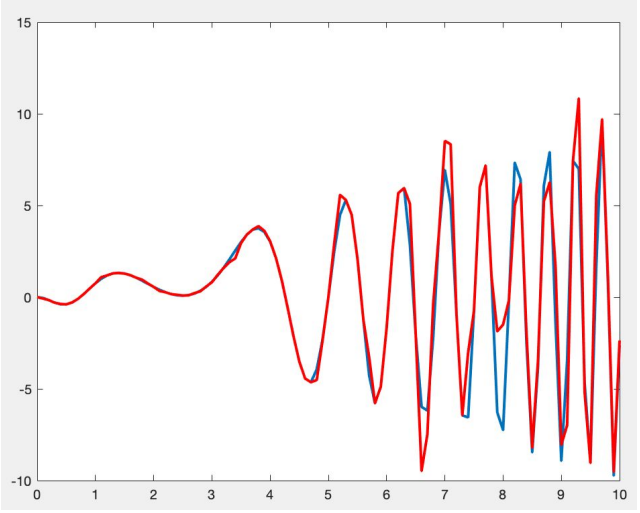
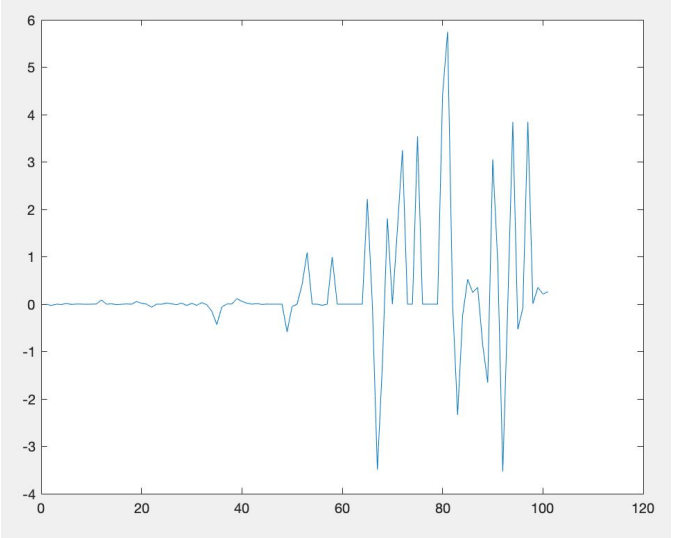


График разности `outputs - Y:`



Выводы

Большинство параметров для сравнения оценены в предыдущей работе, здесь же явно наблюдается улучшение в результате при увеличении количества скрытых нейронов.

Лабораторная работа 5

Цель работы

Анализ работы многослойной нейронной сети по распознава- нию букв методом обратного распространения ошибки.

Содержание

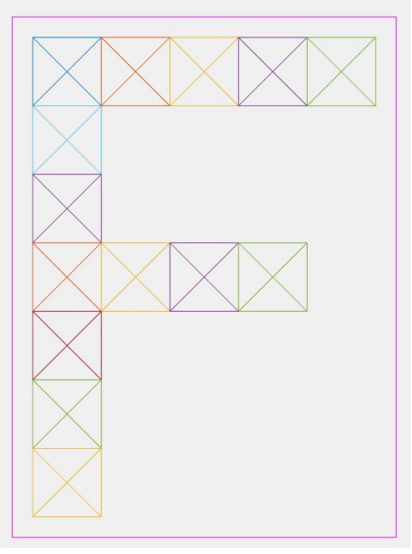
Постановка задачи

Создать нейронную сеть для распознавания 26 символов латинского алфавита.

Алгоритм действий

1. Загрузим исходные данные с помощью М-функции `prprob`:

```
[alphabet, T] = prprob;
plotchar(alphabet(:,6));
```



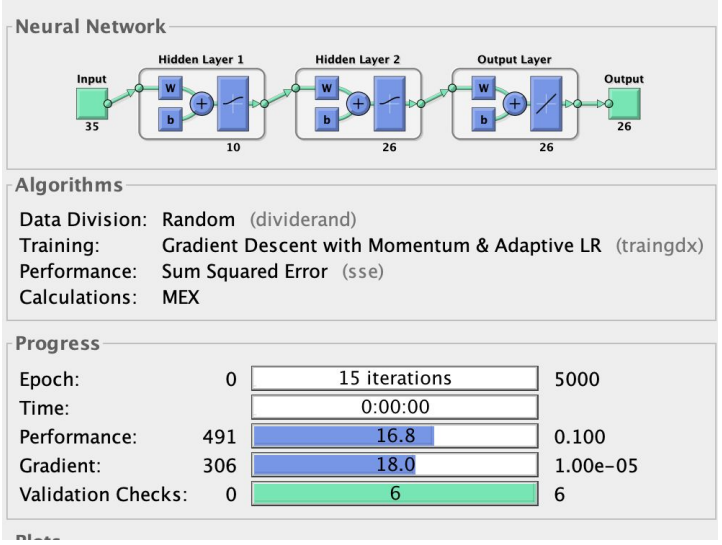
2. Построим необходимую сеть (берем как из примера):

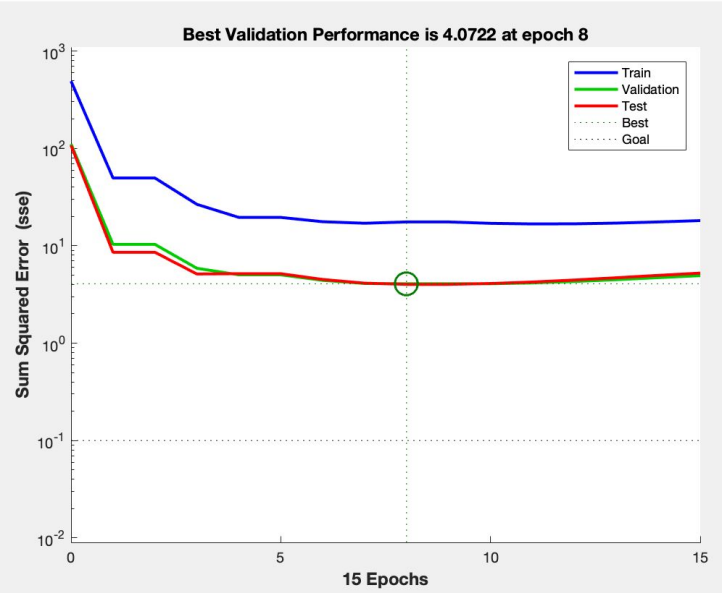
```
[output_layer_size, Q] = size(T);
hidden_layer_size = 10;
net = newff(alphabet, T, [hidden_layer_size out_layer_size],
{'logsig' 'logsig'}, 'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
```

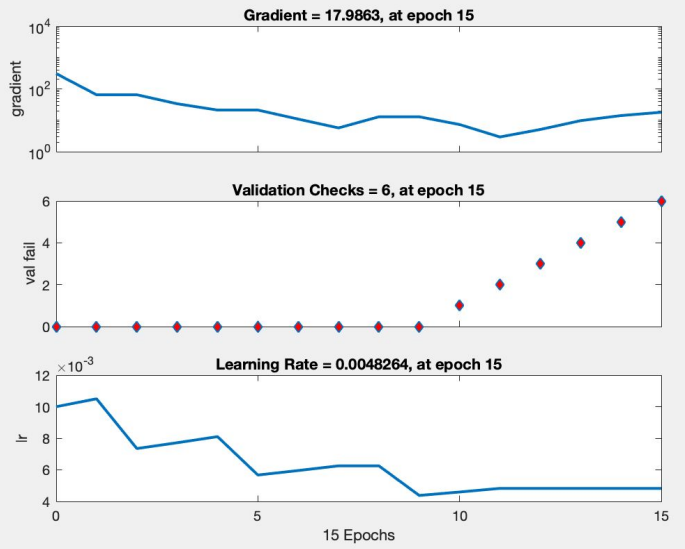
3. Выставляем такие же настройки, как и в разобранном примере:

```
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.95;
[net,tr] = train(net, alphabet, T);
```

И посмотрим на статистику по ее обучению:



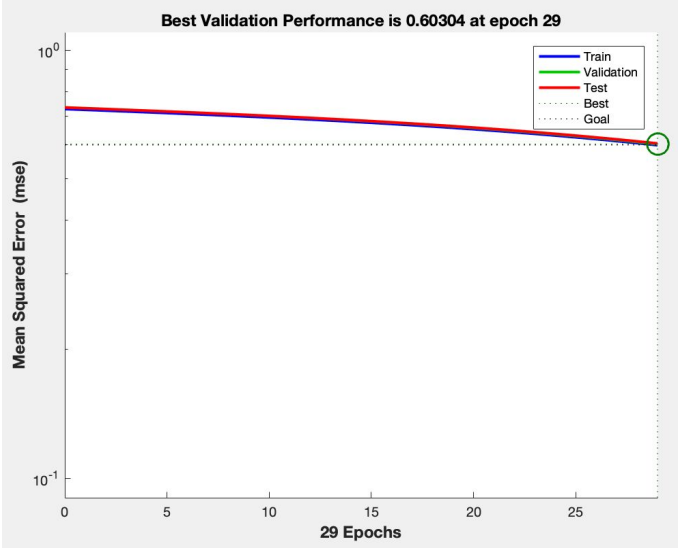


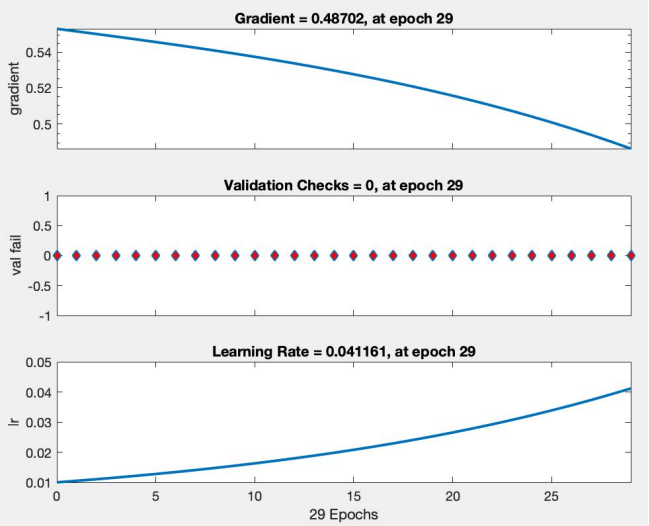


4. Обучим нейронную сеть, не чувствительную к шуму. С этой целью для обучения возьмем пару идеальных и пару зашумленных векторов на вход. Это позволит нейрону хорошо работать на зашумленных символах, но и не потеряет в качестве при чистых данных.

```
noised_n = net;
[R,Q] = size(alphabet);
noised_n.trainParam.goal = 0.6;
noised_n.trainParam.epochs = 300;
T = [targets targets targets targets];
P = [alphabet, alphabet, (alphabet + randn(R,Q)*0.1),
(alphabet + randn(R,Q)*0.2)];
[noised_n,tr] = train(noised_n,P,T);
```

Как видно, ошибка значительно ниже, чем в предыдущем запуске:



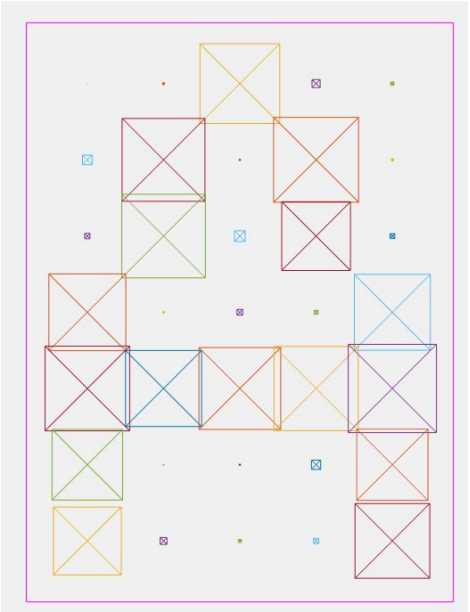


5. Обучим таким образом еще K (=5) раз:

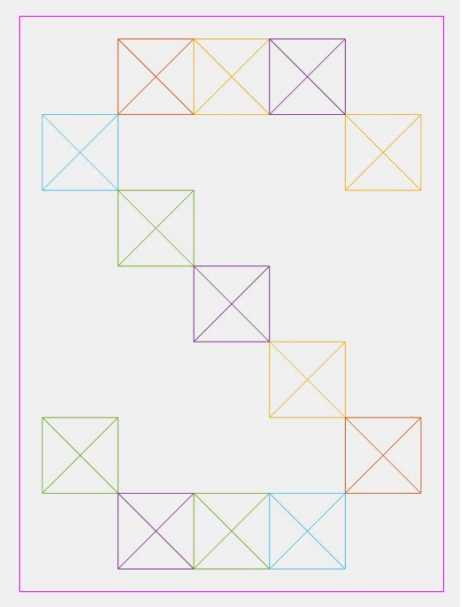
```
for pass = 1:K
    P = [alphabet, alphabet, (alphabet + randn(R,Q)*0.1),
(alphabet + randn(R,Q)*0.2)];
    [noised_n,tr] = train(noised_n,P,T);
end
```

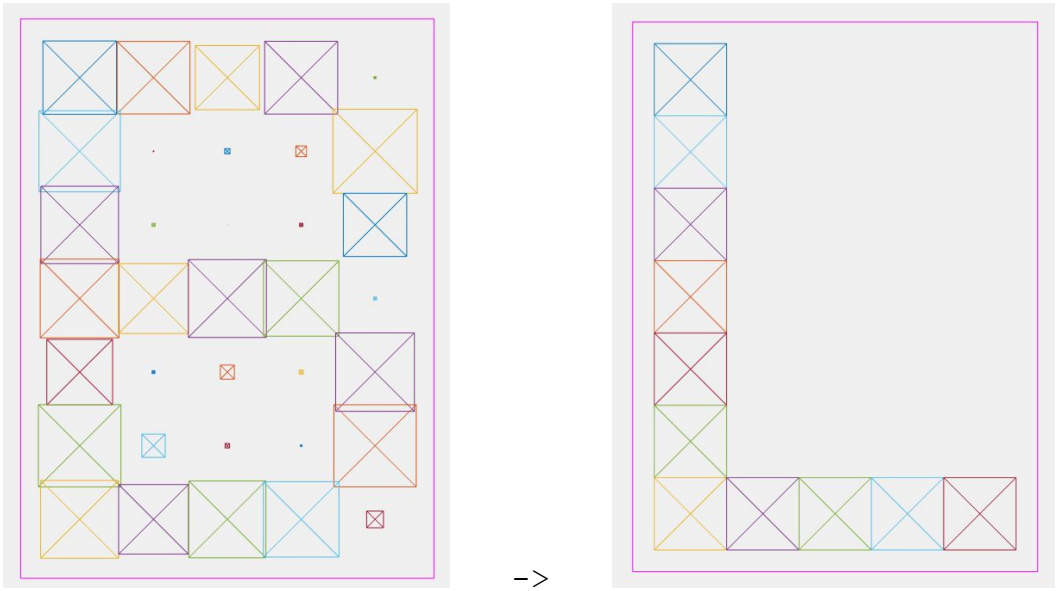
И с помощью скрипта выполним сравнения первых двух символов:

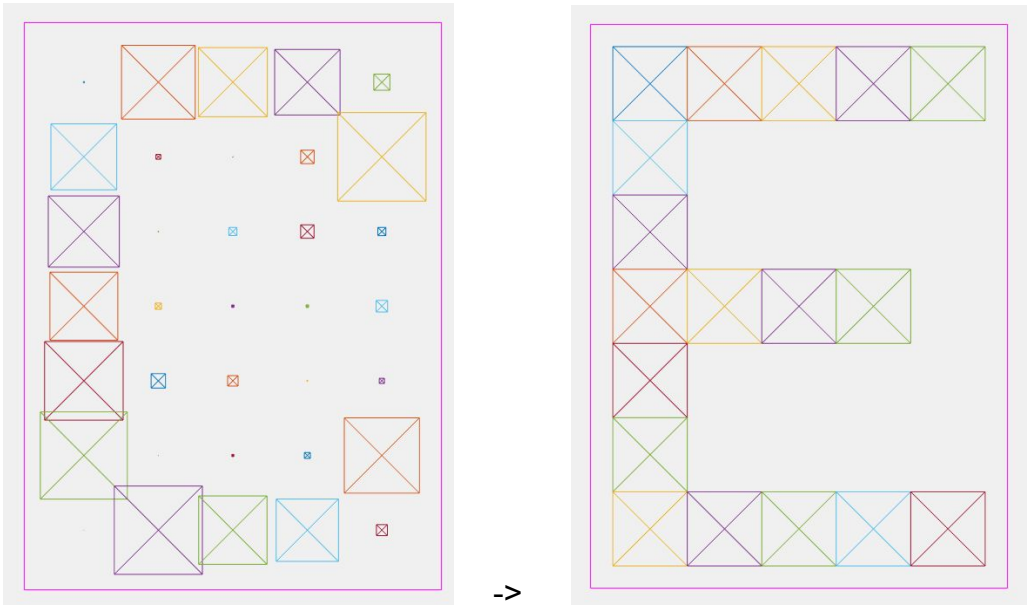
```
function try_letter(net, alphabet, letter)
    noised_letter = alphabet(:,letter) + randn(35,1) *
0.1;
    figure(1); plotchar(noised_letter);
    letter = sim(net, noised_letter);
    answer = find(compet(letter) == 1)
    figure(2); plotchar(alphabet (:,answer));
end
```



->







Опытным путем было установлено, что при K ~= 45 были правильно даны ответы на все три первые буквы. Если изменить шум на:

```
noised_letter = alphabet(:,letter) + randn(35,1) * 0.2;
```

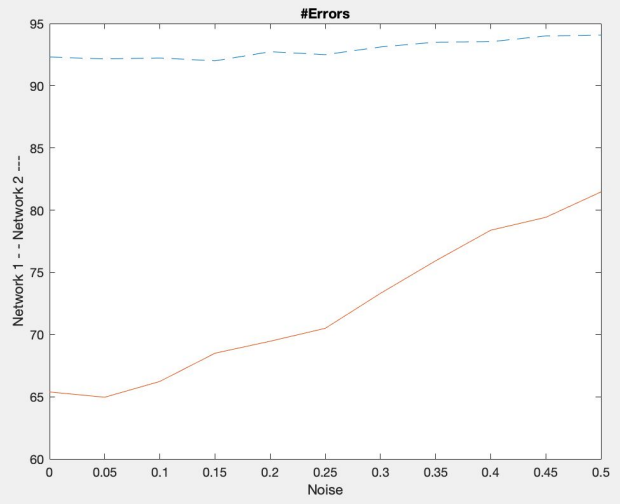
(0.1 -> 0.2), то уже K ~= 60 дает уверенное попадание всех трех букв.

С помощью следующего скрипта:

```
function err_find(net, noised_n, alphabet, T)
    noise_range = 0:.05:.5;
    max_test = 100;
    network1 = [];
    network2 = [];
    for noiselevel = noise_range
     errors1 = 0;
```

```
            errors2 = 0;
            for i = 1:max_test
            P = alphabet + randn(35,26)*noiselevel;
            A = sim(net,P);
            AA = compet(A);
            errors1 = errors1+sum(sum(abs(AA-T)))/2;
            An = sim(noised_n,P);
            AAn = compet(An);
            errors2 = errors2+sum(sum(abs(AAn-T)))/2;
            end
            network1 = [network1 errors1/26/100];
            network2 = [network2 errors2/26/100];
          end
          figure(3);

plot(noise_range,network1*100,'--',noise_range,network2*100);
          title('#Errors');
          xlabel('Noise');
          ylabel('Network 1 - - Network 2 ---');
      end
```

Получим следующий график зависимости количества неотгаданных (в процентах) от уровня шума:



Выводы

В результате выполнения этой работы были разобраны возможности распознавания букв для 35-мерного вектора-входа. Было установлено, насколько влияет обучение с шумом на результат распознавания ответа.

Лабораторная работа 6

Цель работы

Содержание

Выводы

Лабораторная работа 7

Цель работы

Содержание

Выводы