

<b>Лабораторная работа 1</b>	<b>3</b>
Цель работы	3
Содержание	3
Постановка задачи	3
Исходные данные	3
Алгоритм решения	4
Построение сети	4
Обучение сети	6
Результаты обучения	10
Выводы	13
<b>Лабораторная работа 2</b>	<b>14</b>
Цель работы	14
Содержание	14
Постановка задачи	14
Исходные данные	14
Алгоритм решения	15
Обучение OR, NOR, AND, NAND	15
Обучение XOR, NXOR	16
Выводы	20
<b>Лабораторная работа 3</b>	<b>21</b>
Цель работы	21
Содержание	21
Постановка задачи	21
Используемые функции	21
Алгоритм traingd	22
Алгоритм trainrp	25
Алгоритм traingdx	28
Выводы	30
<b>Лабораторная работа 4.1</b>	<b>30</b>
Цель работы	30
Содержание	31
Постановка задачи	31
Используемые функции	31
Алгоритм traingdx	32
Алгоритм trainlm (default)	34
Выводы	36
<b>Лабораторная работа 4.2</b>	<b>37</b>

Цель работы	37
Постановка задачи	37
Используемые функции	37
Сравнение по количеству внутренних слоев	38
Выводы	40
<b>Лабораторная работа 5</b>	<b>40</b>
Цель работы	40
Содержание	40
Выводы	40
<b>Лабораторная работа 6</b>	<b>41</b>
Цель работы	41
Содержание	41
Выводы	41
<b>Лабораторная работа 7</b>	<b>42</b>
Цель работы	42
Содержание	42
Выводы	42

# 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]
```

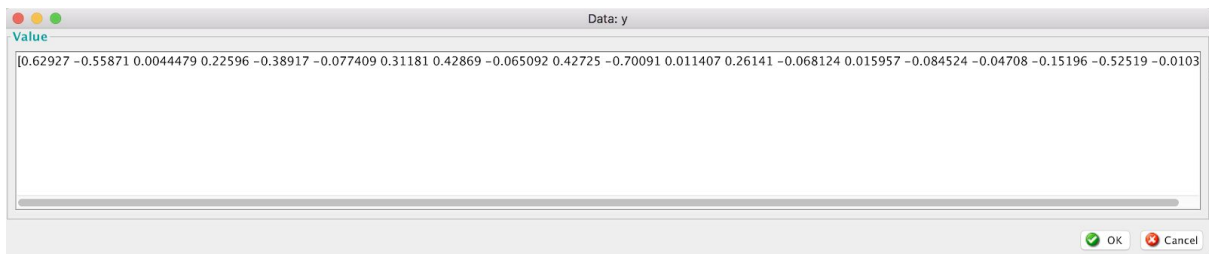
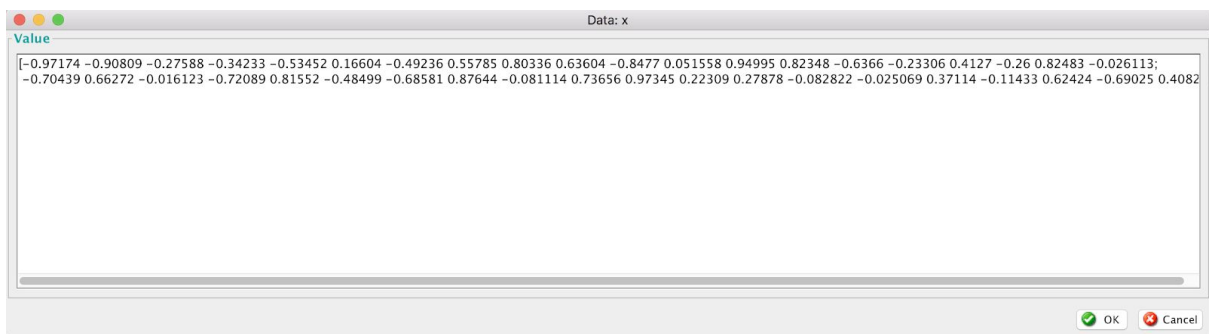
```
y = [0.629274528104 -0.55871195223 0.00444792283491 0.225959155974
-0.389173696763    -0.0774087781133    0.311813661833    0.428690121478
-0.0650924249649    0.427253084799    -0.700907259645    0.0114068110504]
```

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

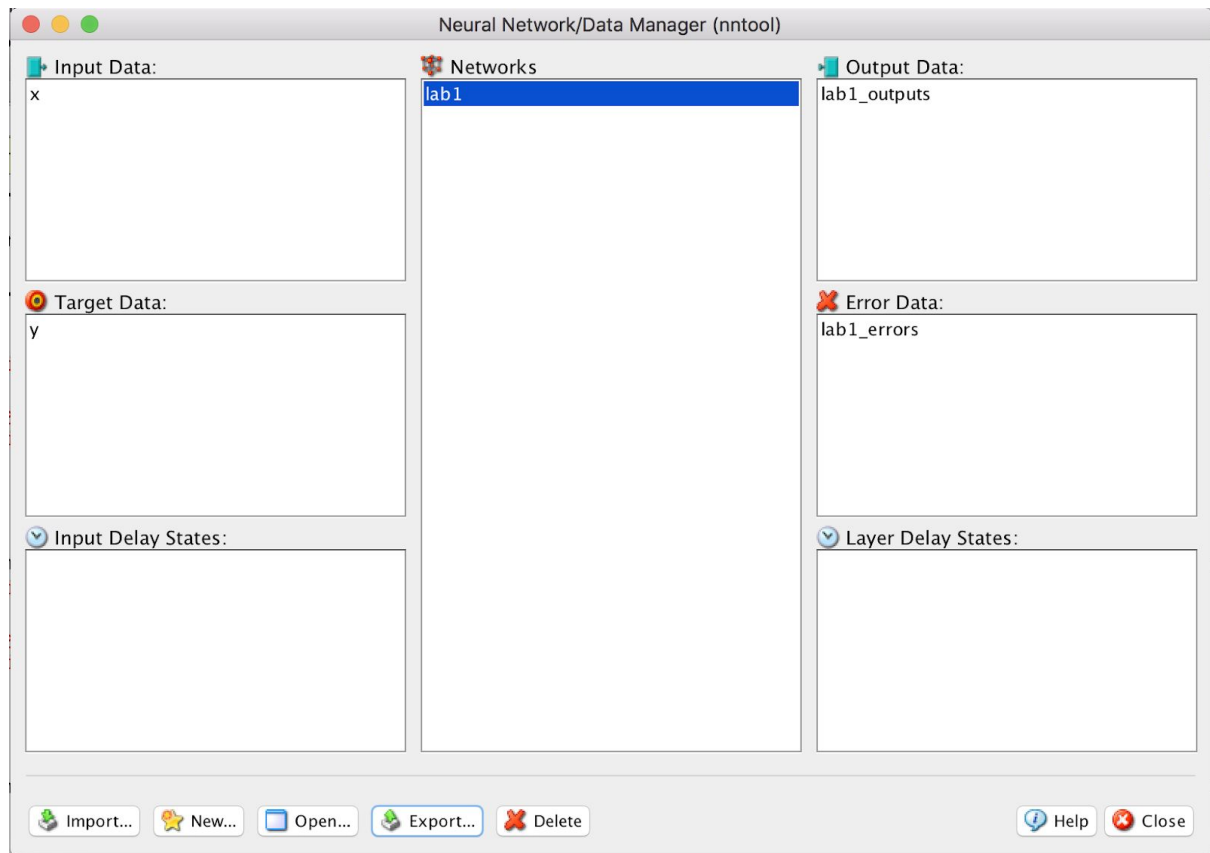
## Алгоритм решения

### Построение сети

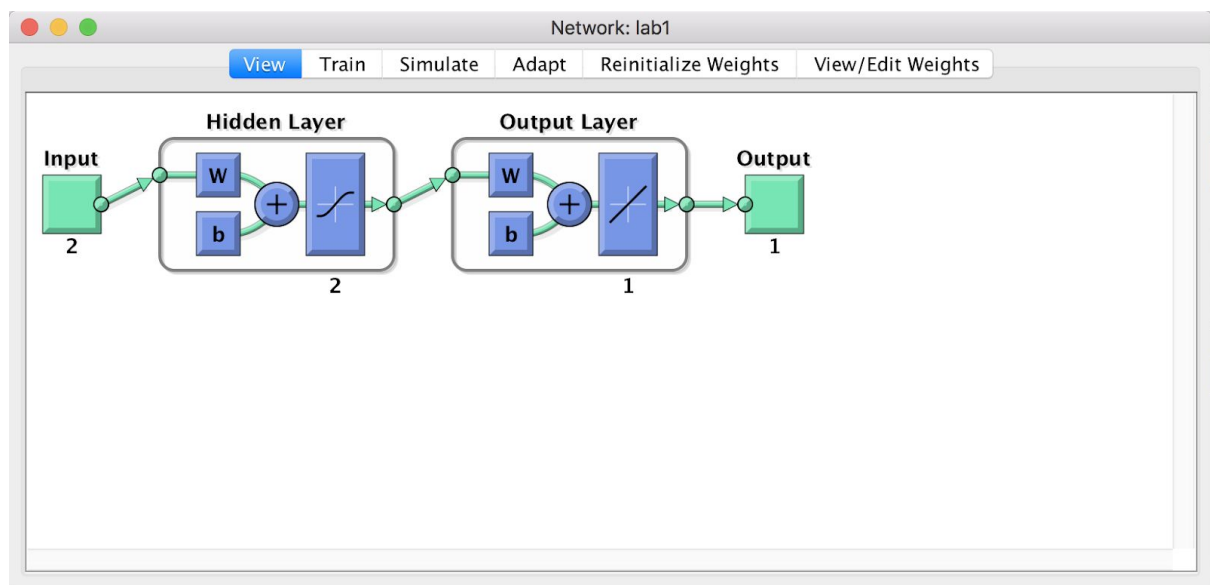
1. С помощью функции **nntool** откроем основное окно интерфейса и сформируем последовательность входов и целей в рабочей области.
2. Внесем исходные данные (*Input Data* и *Target Data*) в окно *Neural Network/Data Manager*.



3. Добавим сеть во вкладке *Networks*

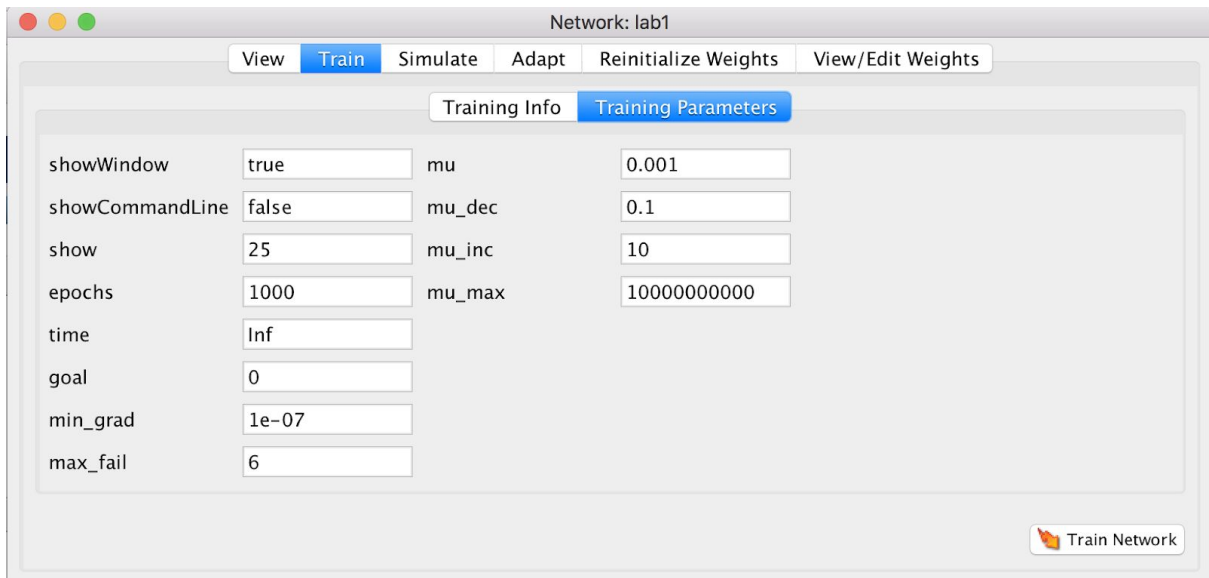
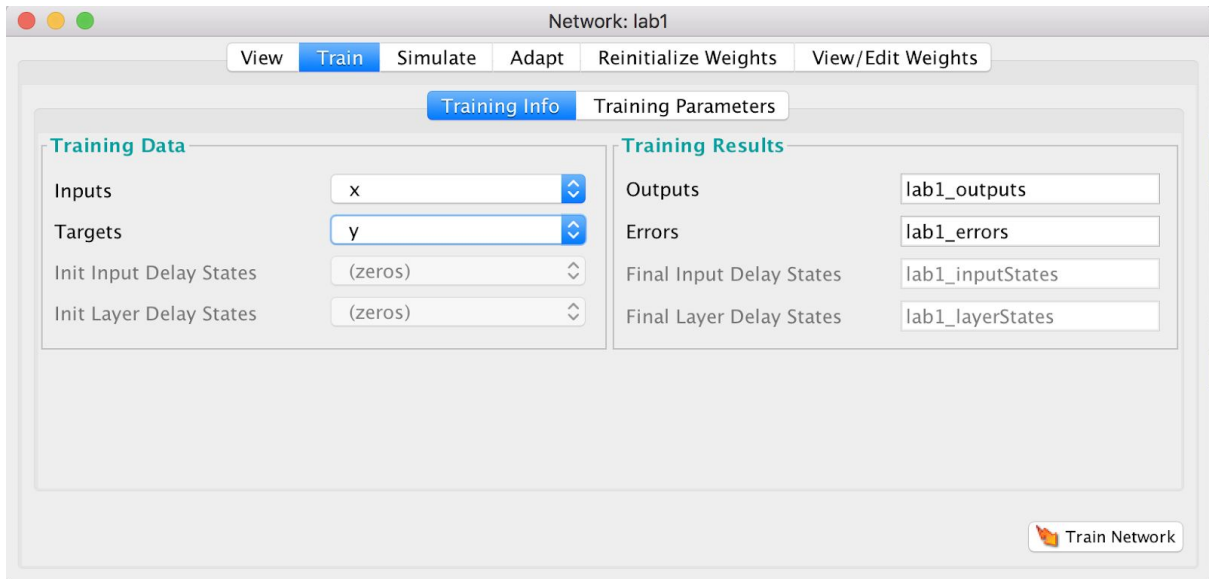


4. Зайдем в настройки сети. Выберем ее тип *Feed-forward backprop* с прямой передачей сигнала и обратным распространением ошибки.
5. Добавим сети *Input Data*, *Target Data*, количество нейронов первого слоя равно 2, функция активации во втором слое - линейная (*PURELIN*).



# Обучение сети

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



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

Neural Network Training (nntraintool)

### Neural Network

The diagram shows a neural network architecture with the following components:

- Input Layer:** 2 nodes.
- Hidden Layer:** 2 nodes, containing weight matrix  $W$  and bias vector  $b$ , with an addition node (+) and an activation function (sigmoid curve).
- Output Layer:** 1 node, containing weight matrix  $W$  and bias vector  $b$ , with an addition node (+) and an activation function (linear slope).

### 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

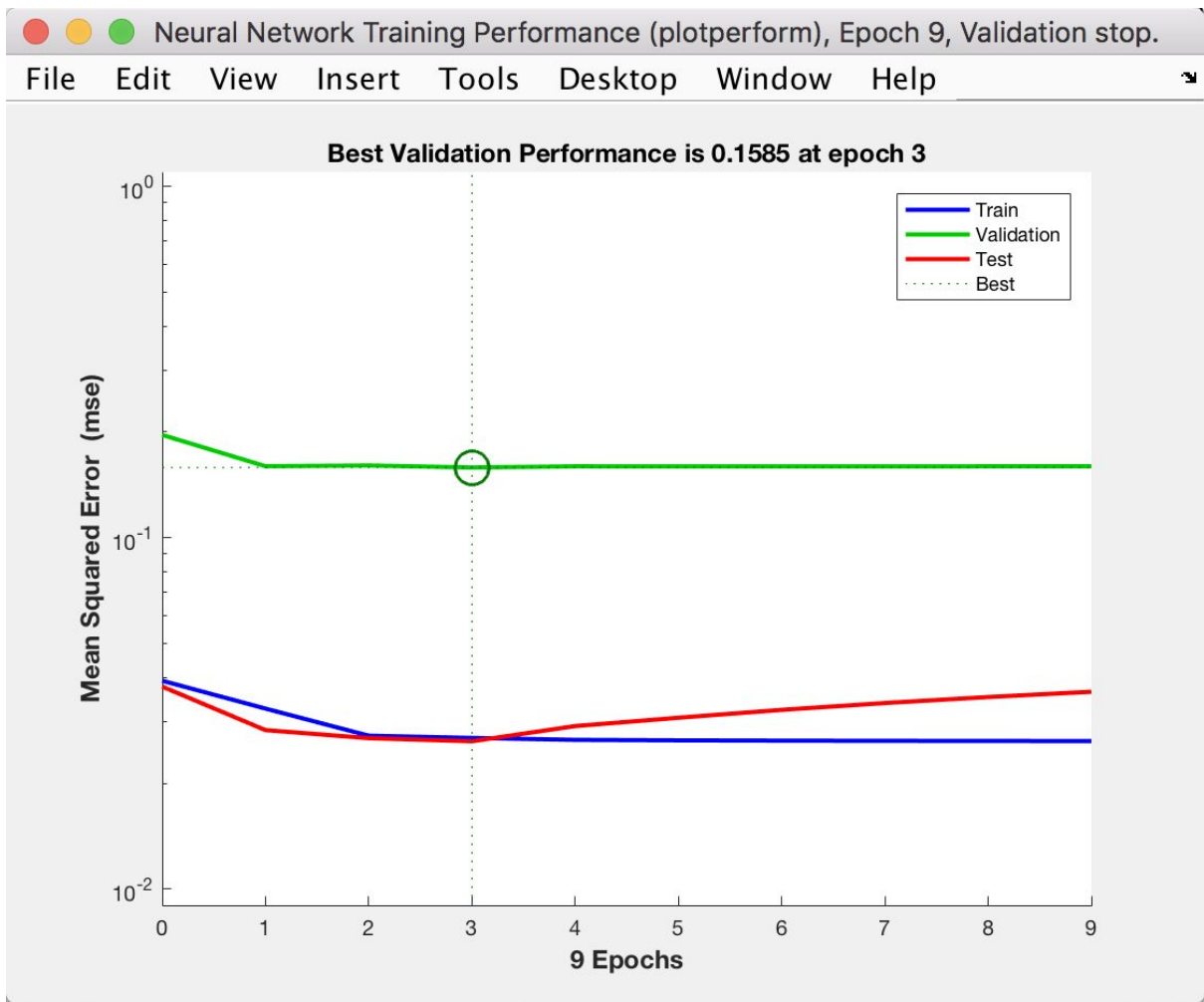
(plotregression)

Plot Interval:  1 epochs

Validation stop.

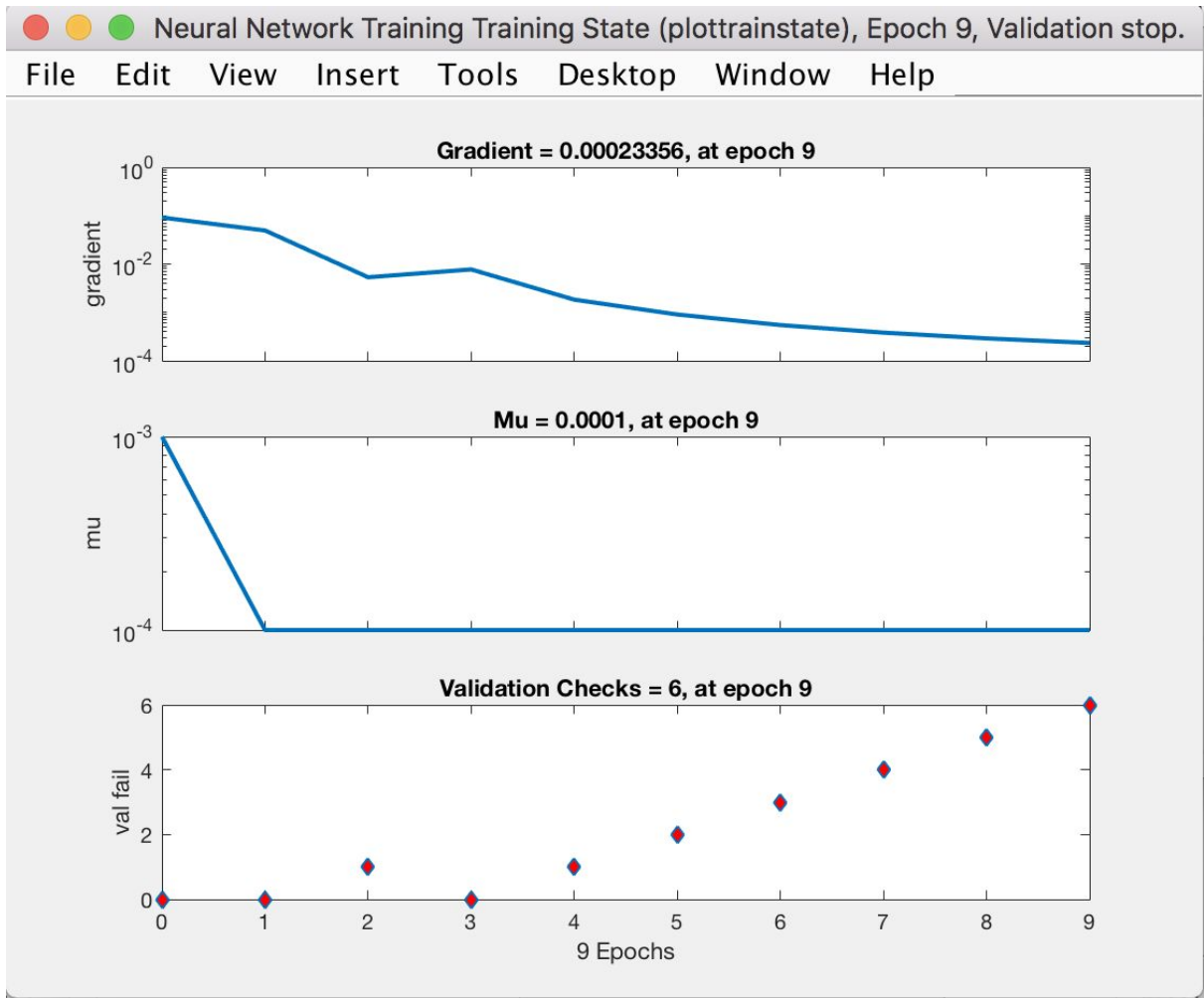
Stop Training

Cancel



*График Performance*





Γραφικ Training State

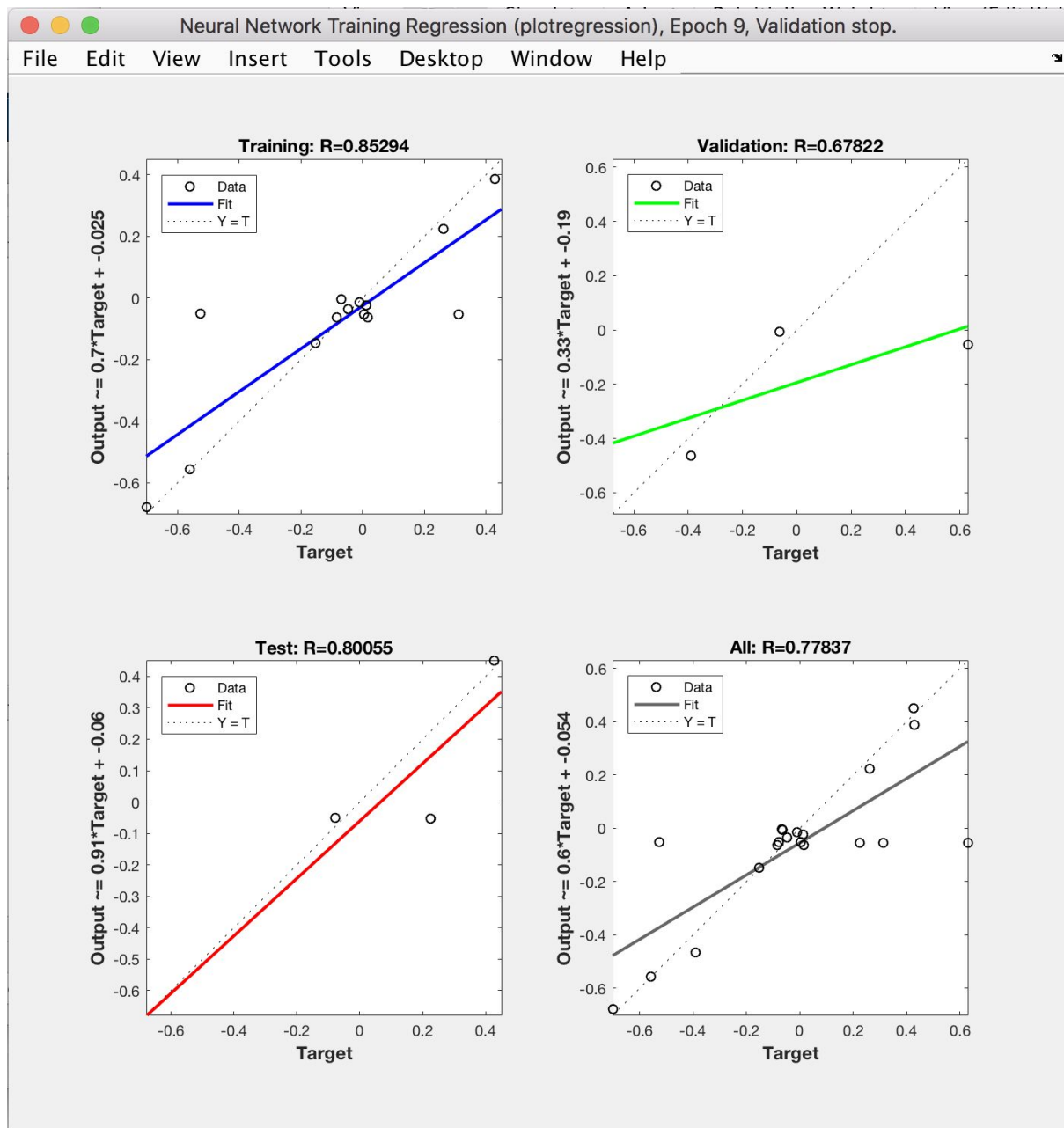


График Regression

## Результаты обучения

Результаты обучения сети можно посмотреть в окне *Network/Data Manager*, а именно выбирая параметры *lab1\_outputs* и *lab1\_errors*:

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

```
lab1_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:

```
Workspace | Command Window
y =
Columns 1 through 13
    0.6293    -0.5587    0.0044    0.2260    -0.3892    -0.0774    0.3118    0.4287    -0.0651    0.4273    -0.7009    0.0114    0.2614
Columns 14 through 20
    -0.0681    0.0160    -0.0845    -0.0471    -0.1520    -0.5252    -0.0104

>> lab1_errors
lab1_errors =
Columns 1 through 13
    0.6828    -0.0020    0.0568    0.2791    0.0759    -0.0259    0.3650    0.0411    -0.0595    -0.0230    -0.0222    0.0340    0.0379
Columns 14 through 20
    -0.0641    0.0776    -0.0217    -0.0118    -0.0054    -0.4745    0.0043

>> lab1_outputs
lab1_outputs =
Columns 1 through 13
    -0.0535    -0.5567    -0.0523    -0.0531    -0.4651    -0.0515    -0.0532    0.3876    -0.0056    0.4502    -0.6787    -0.0226    0.2235
Columns 14 through 20
    -0.0041    -0.0617    -0.0628    -0.0353    -0.1466    -0.0507    -0.0147
```

### *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:  
 $X_{\text{train}} = \{[0; 0] [0; 1] [1; 0] [1; 1]\};$   
 $Y_{\text{train}} = \{0, 0, 0, 1\};$
- OR:  
 $X_{\text{train}} = \{[0; 0] [0; 1] [1; 0] [1; 1]\};$   
 $Y_{\text{train}} = \{0, 1, 1, 1\};$
- NAND:  
 $X_{\text{train}} = \{[0; 0] [0; 1] [1; 0] [1; 1]\};$   
 $Y_{\text{train}} = \{1, 1, 1, 0\};$
- NOR:  
 $X_{\text{train}} = \{[0; 0] [0; 1] [1; 0] [1; 1]\};$   
 $Y_{\text{train}} = \{1, 0, 0, 0\};$
- XOR:  
 $X_{\text{train}} = \{[0; 0] [0; 1] [1; 0] [1; 1]\};$   
 $Y_{\text{train}} = \{0, 1, 1, 0\};$
- 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 logical 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 logical 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 logical 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 logical 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

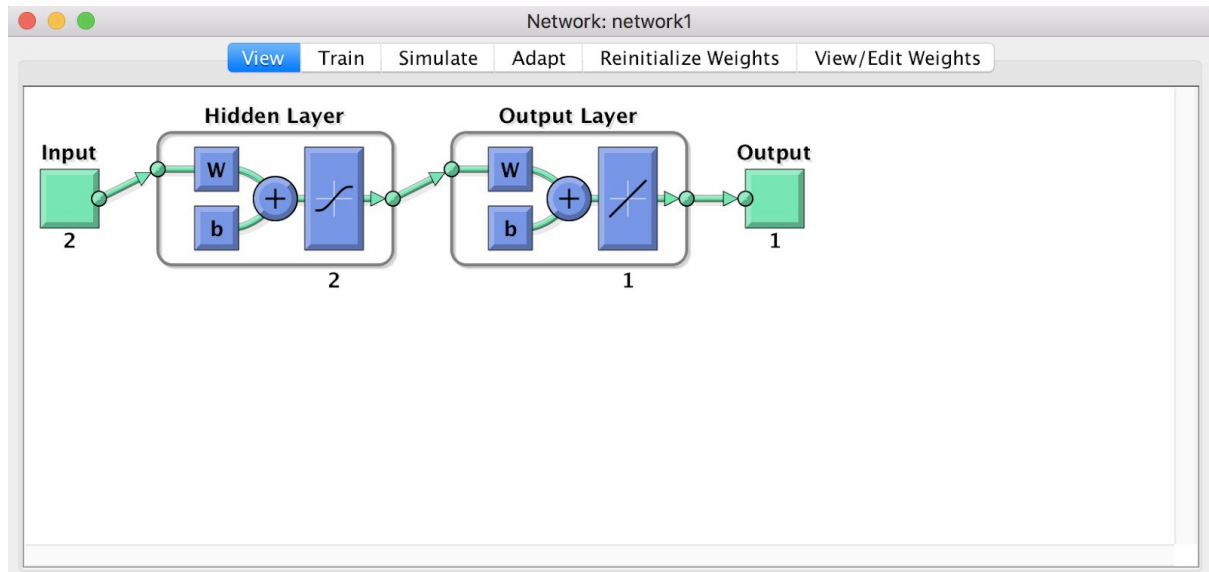
```

## 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:



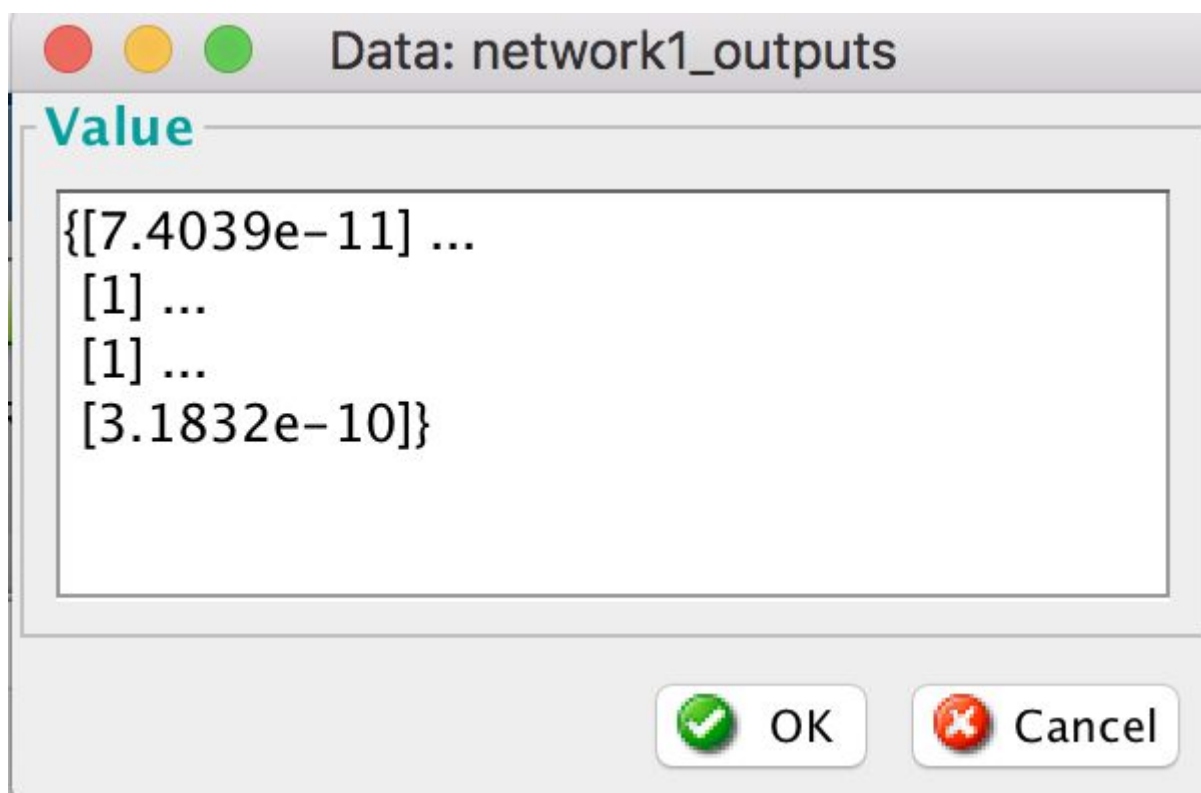
*Создали сеть с двумя скрытыми слоями*

The screenshot shows the "Train" tab selected in the "Network: network1" window. The "Training Info" sub-tab is active. It displays two main sections: "Training Data" and "Training Results".

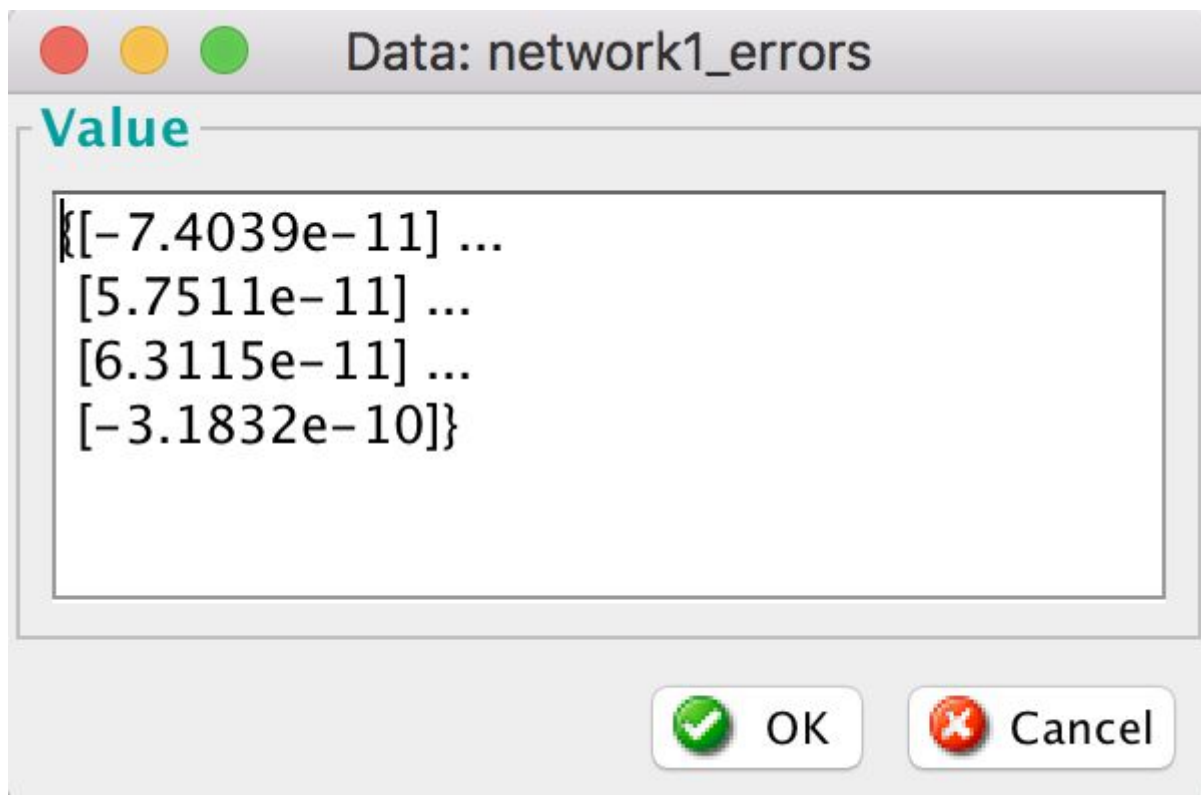
Training Data		Training Results	
Inputs	X_train	Outputs	network1_outputs
Targets	Y_train	Errors	network1_errors
Init Input Delay States	(zeros)	Final Input Delay States	network1_inputStates
Init Layer Delay States	(zeros)	Final Layer Delay States	network1_layerStates

At the bottom right, there is a "Train Network" button with a flame icon.

*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
```

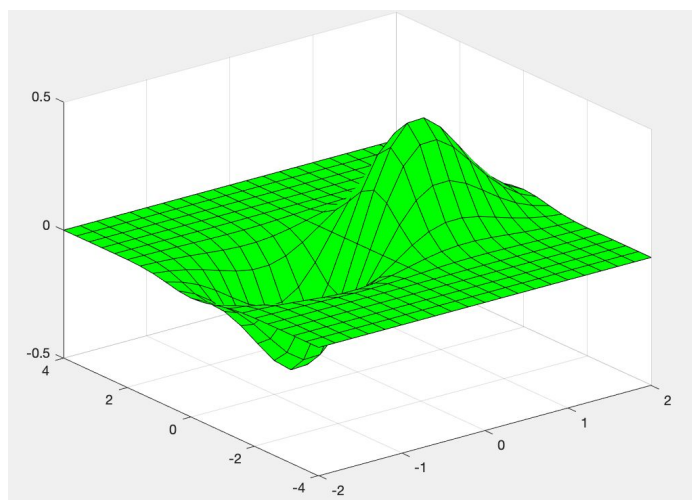
**Функция, замеряющая время выполнения:**

```
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
```

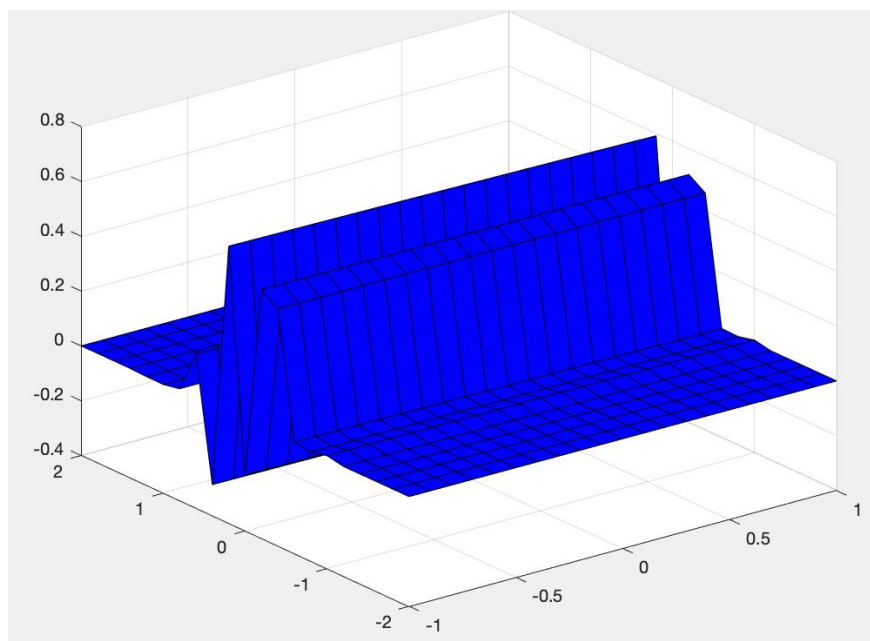
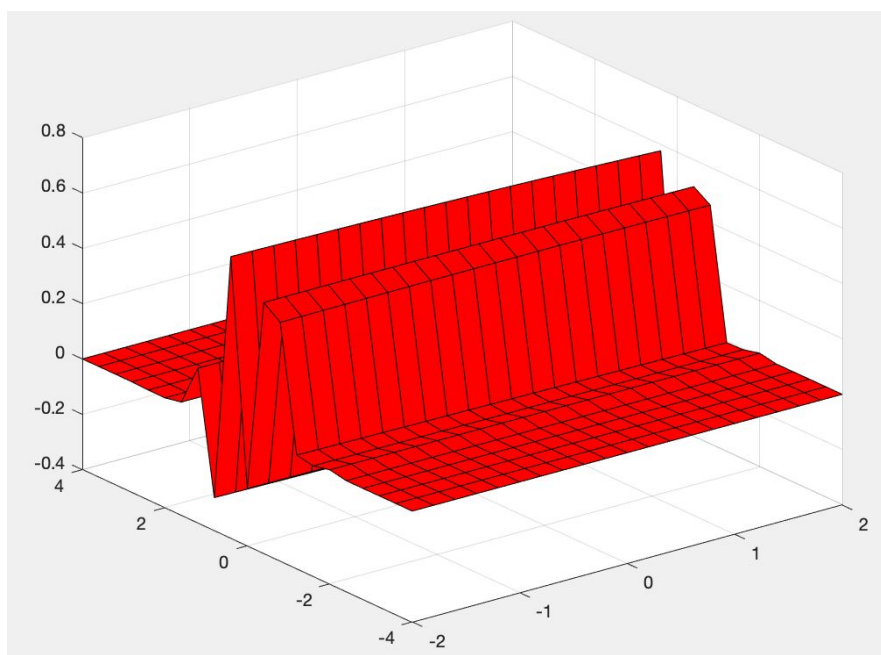
**Отрисованная функция для обучения для всех алгоритмов:**



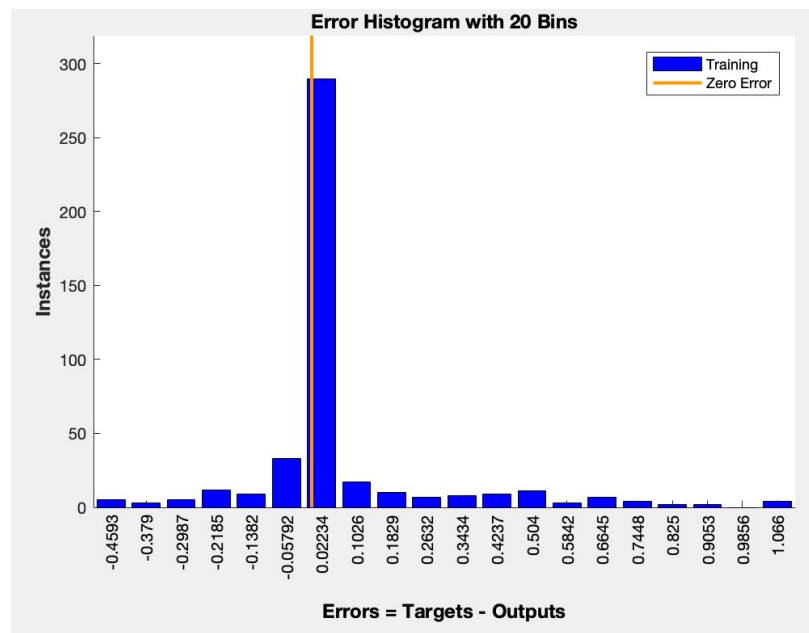
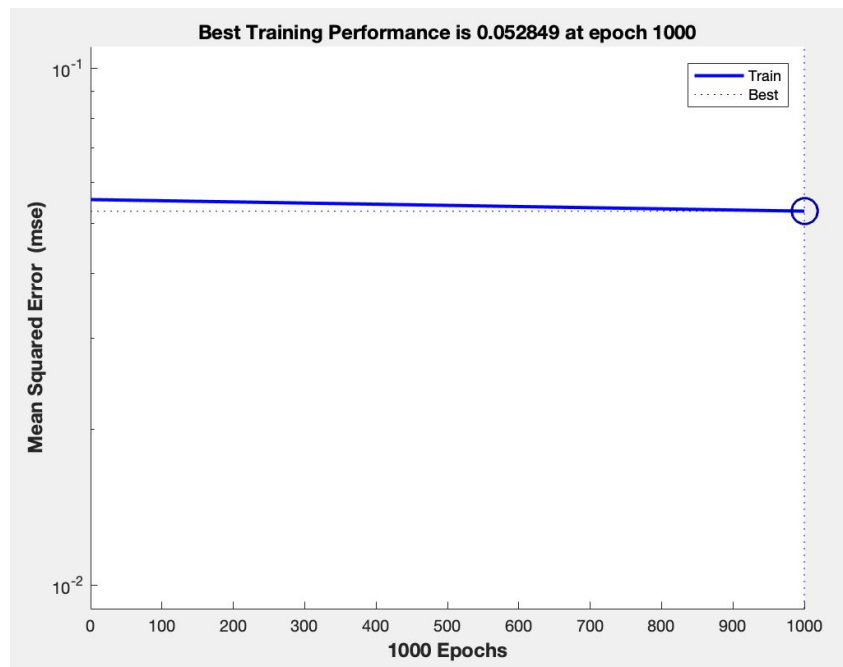
В качестве сравнения выберем алгоритмы `traingd`, `trainrp`, `traingdx`.

## Алгоритм `traingd`

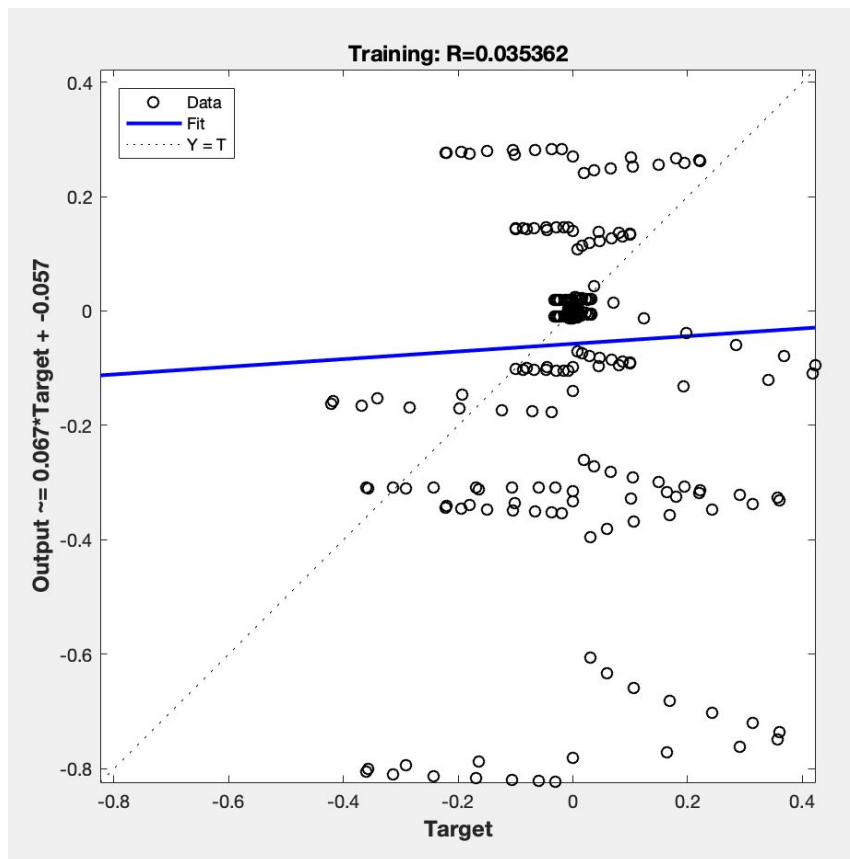
Поверхности при запуске на данных для обучения и тестовых:



Графики, показывающие качество модели:







Ошибка и время работы:

MSE

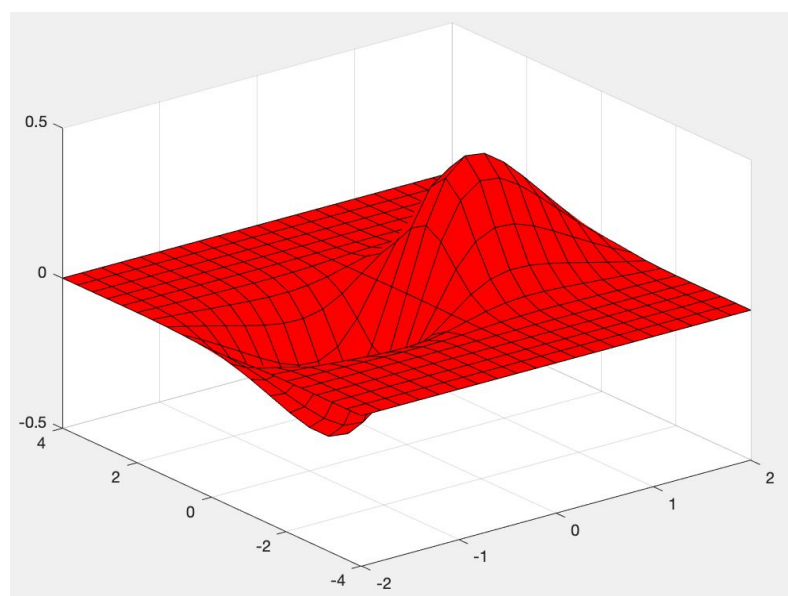
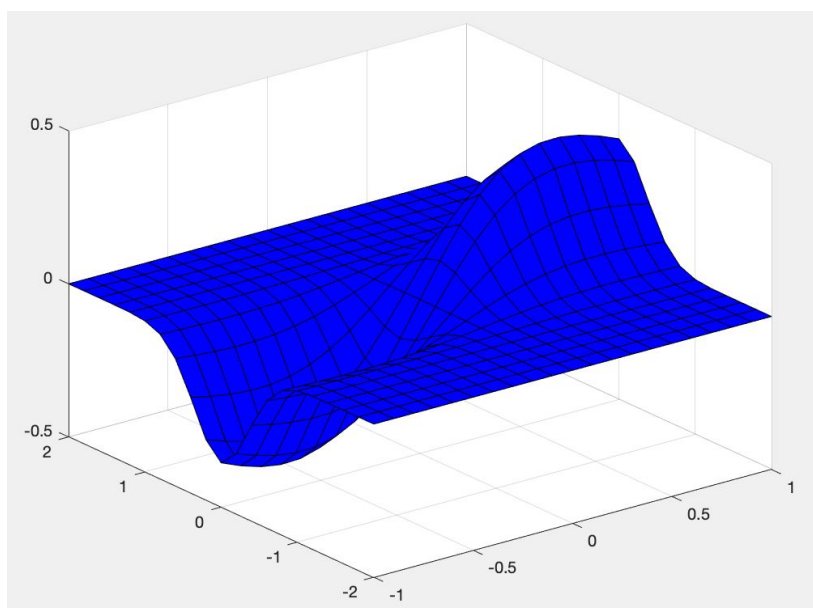
0.0821

Time

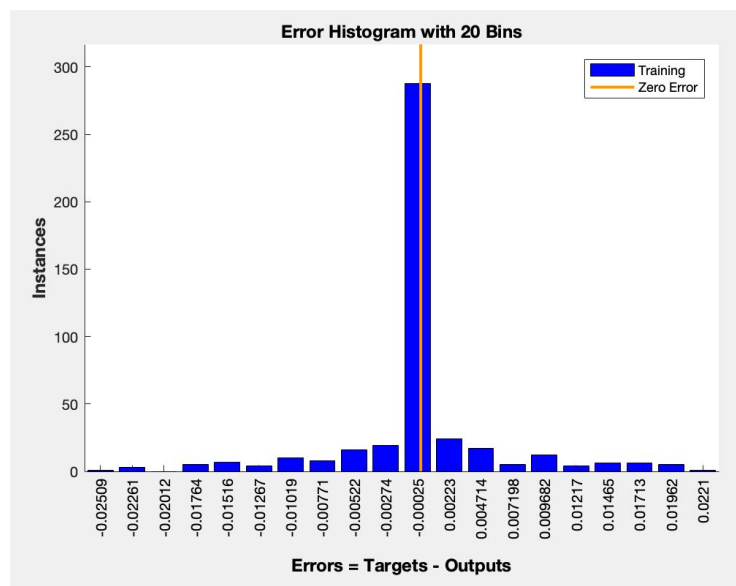
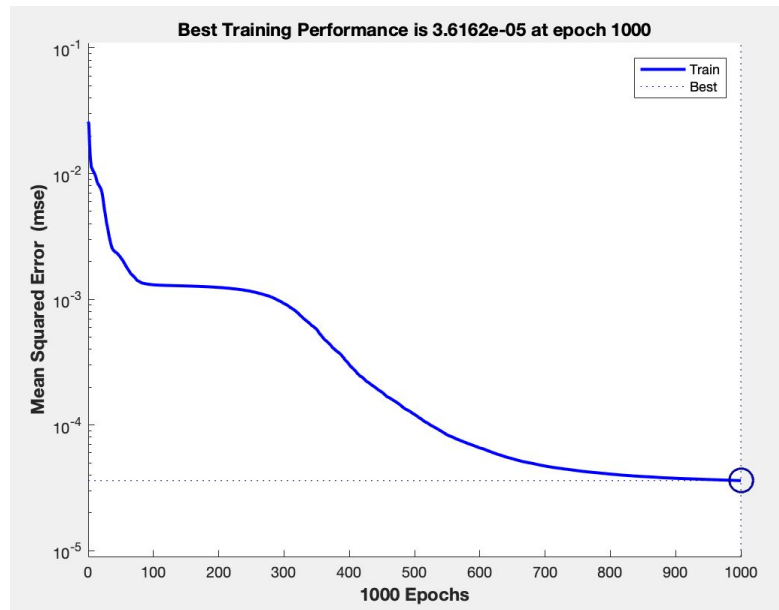
1.9608

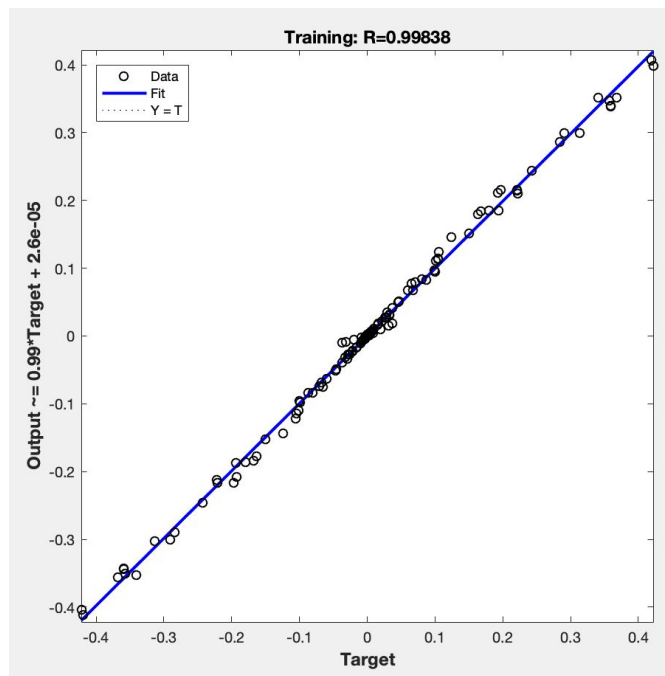
## Алгоритм trainrp

Поверхности при запуске на данных для обучения и тестовых:



Графики, показывающие качество модели:





Ошибка и время работы:

MSE

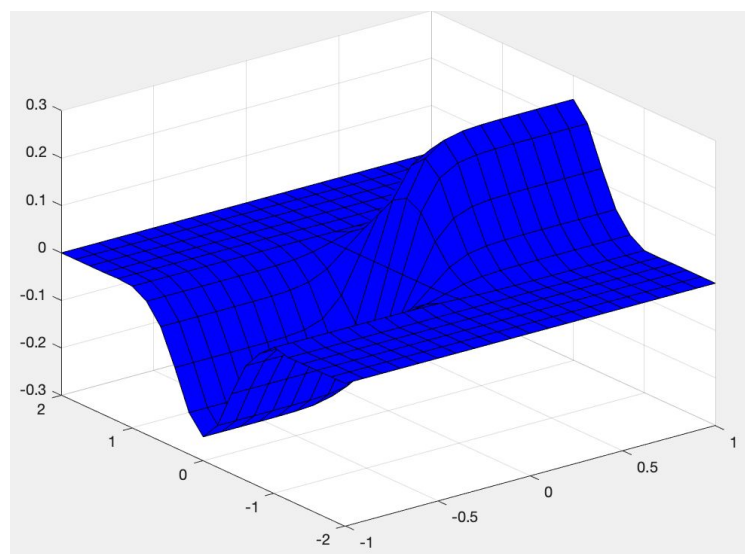
0.0084

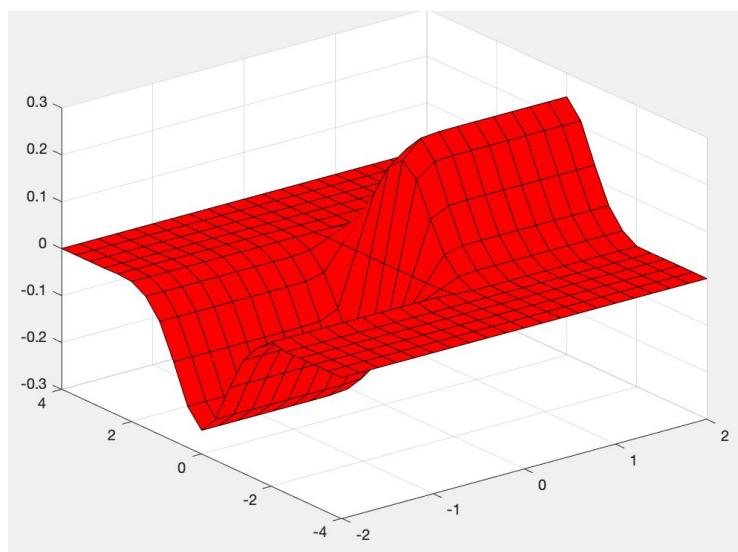
Time

2.5101

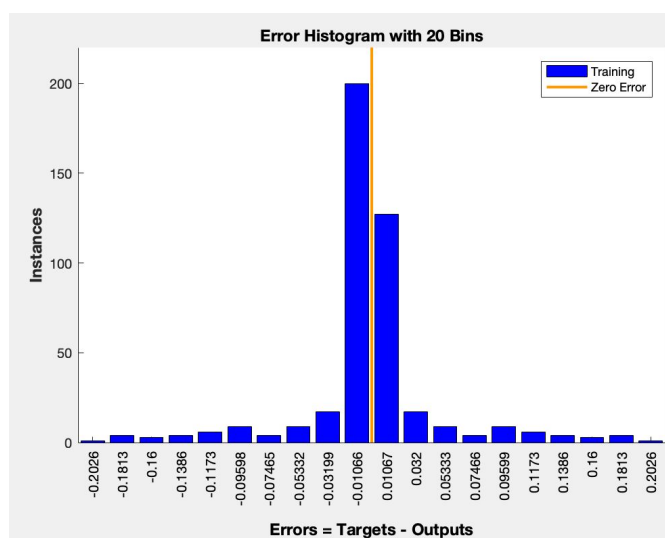
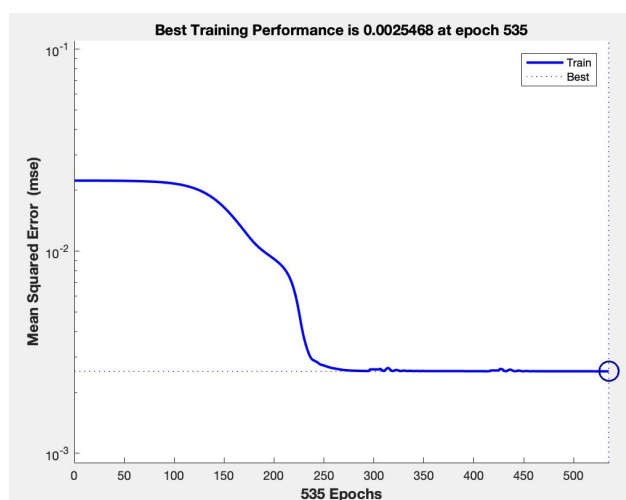
## Алгоритм traingdx

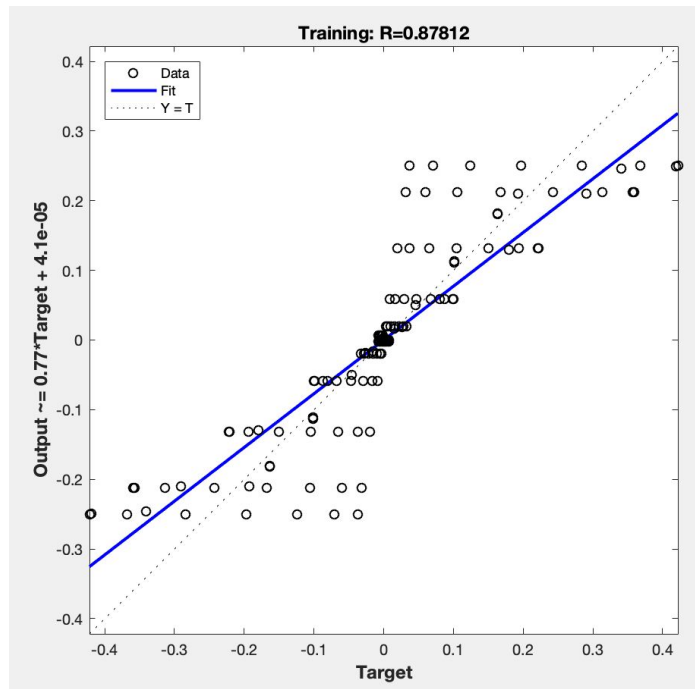
Поверхности при запуске на данных для обучения и тестовых:





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]);

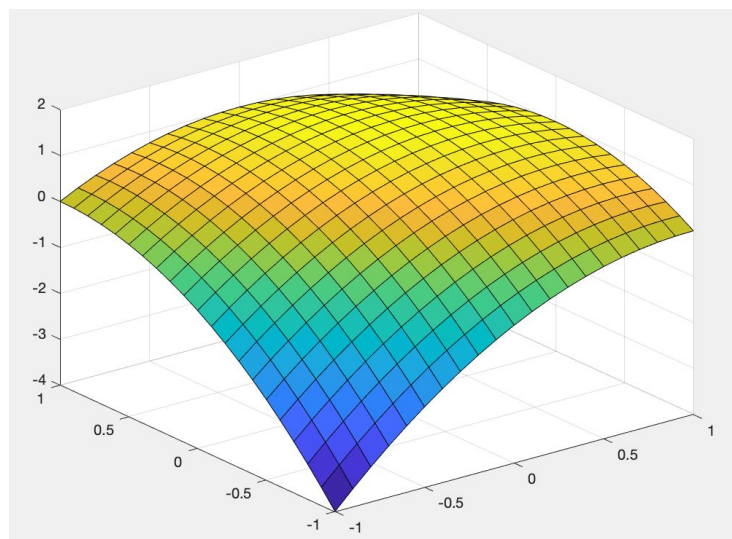
show_surf(2, x1, x2, outputs, N);

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

```

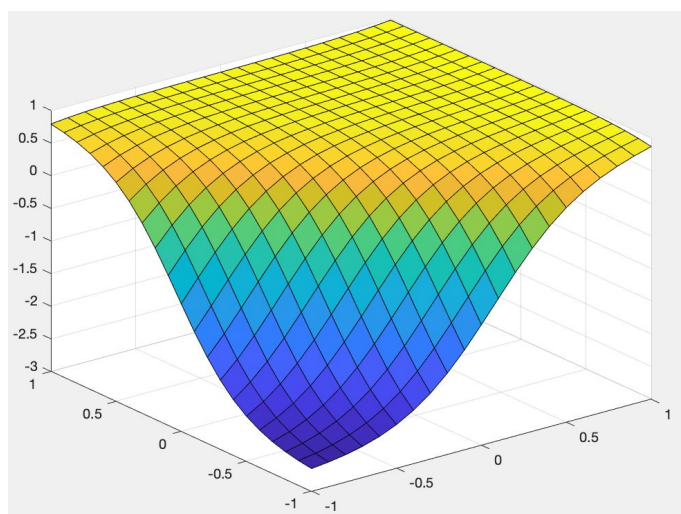
График исходной функции:



## Алгоритм traingdx

График функции на исходных данных, запущенных через обученную сеть:





Сводная информация по обучению:

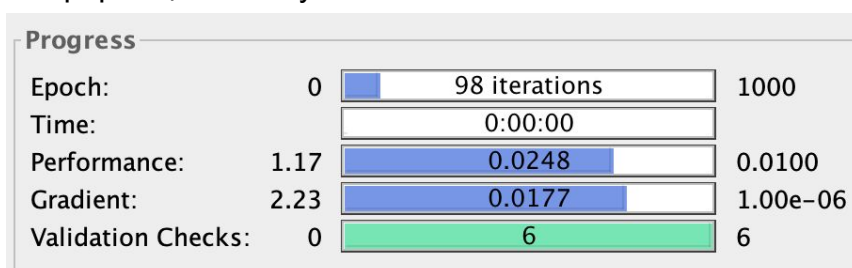


График ошибки:

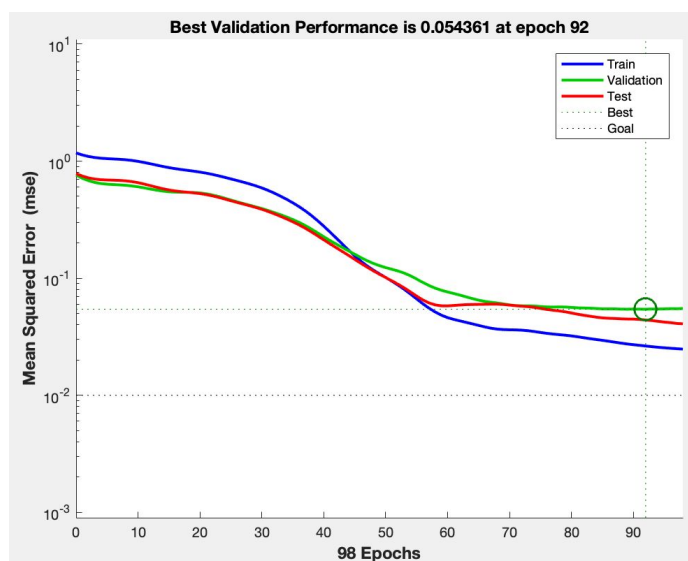


График Training State:

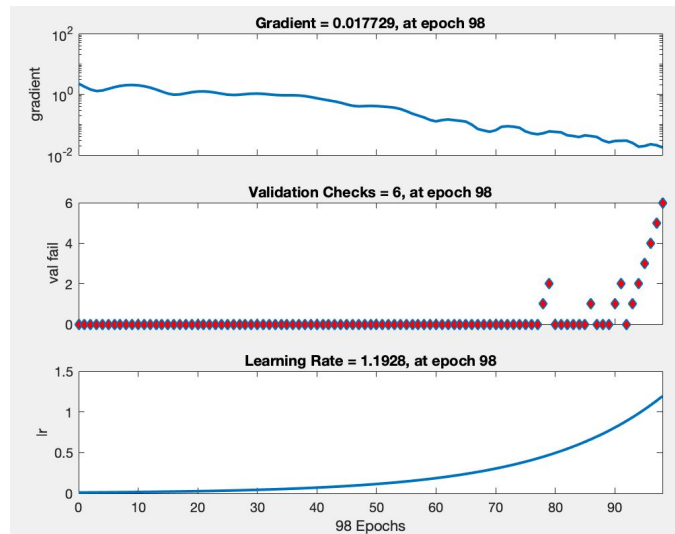
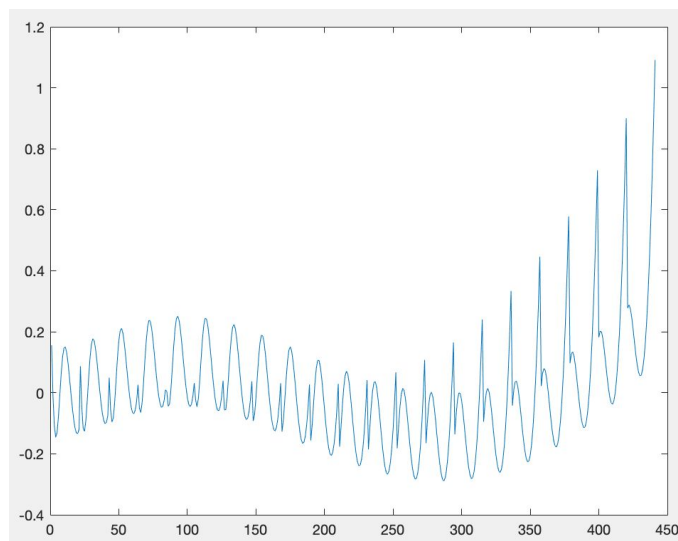
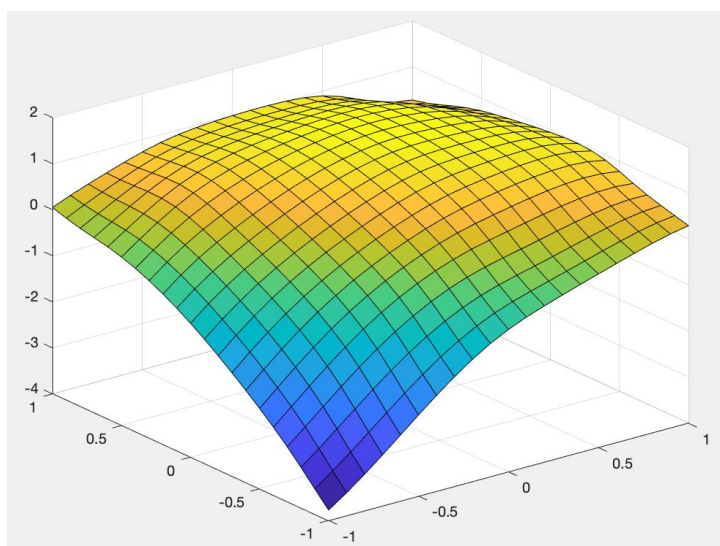


График разности outputs - Y:



## 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:

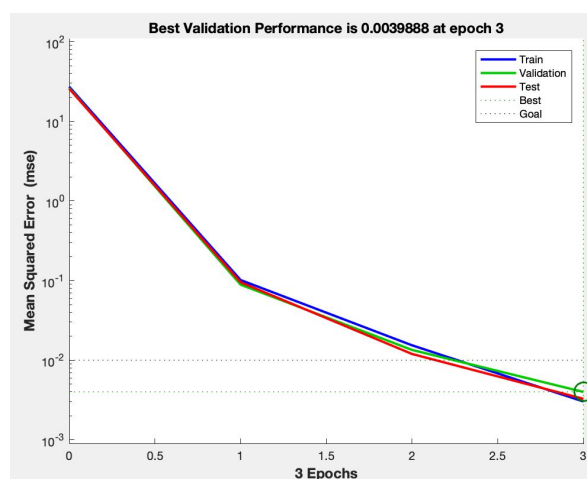


График Training State:

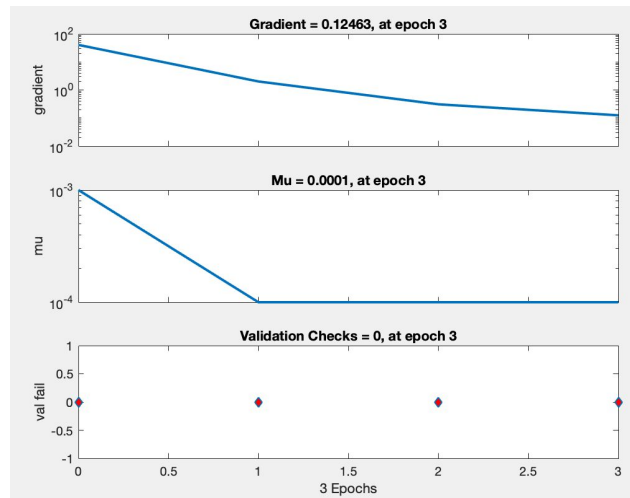
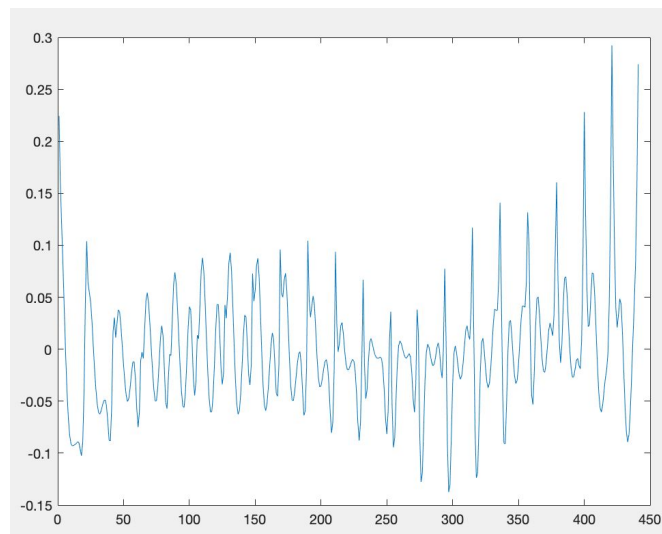


График разности outputs - Y:



## Выводы

Были опробованы два алгоритма: `trainidx` 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 слоев

### 1. 20 слоев:

Исходная функция и аппроксимирующая функция:

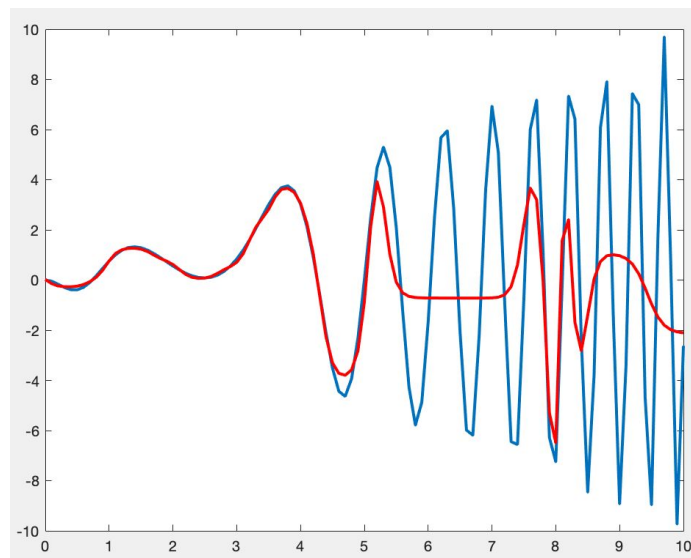
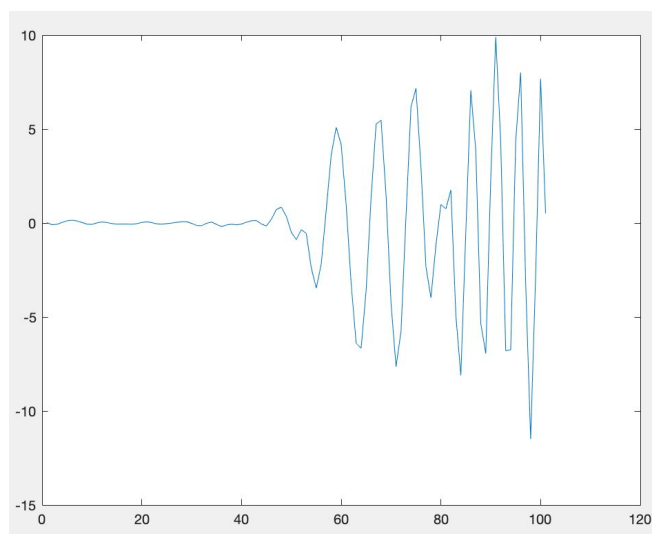


График разности `outputs - Y`:



2. 50 слоев:

Исходная и аппроксимирующая функция:

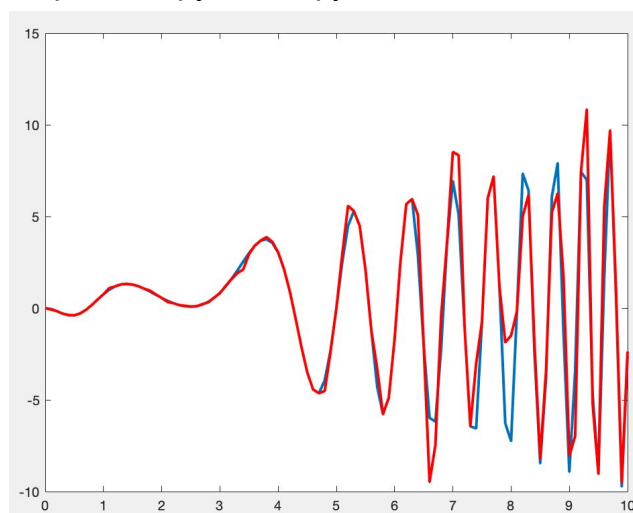
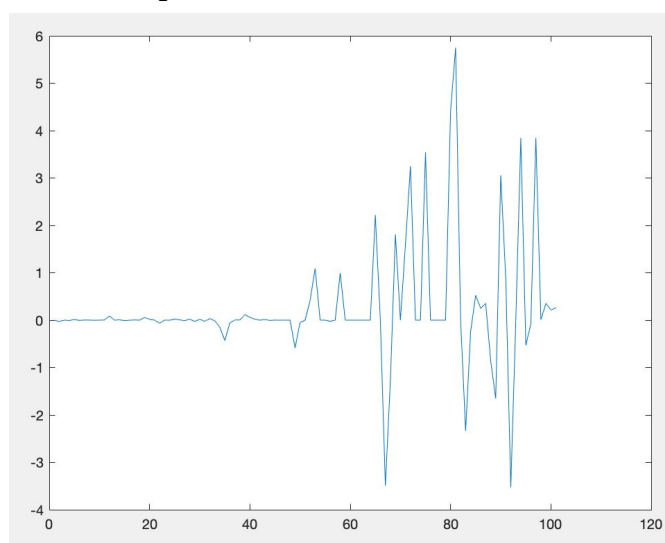


График разности  $\text{outputs} - Y$ :



## **Выводы**

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

## **Лабораторная работа 5**

Цель работы

Содержание

Выводы



# **Лабораторная работа 6**

Цель работы

Содержание

Выводы

# **Лабораторная работа 7**

Цель работы

Содержание

Выводы