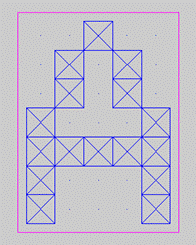
Распознавание букв в Матлабе

**alphabet = prprob();** задаем буквы алфавита в переменную alphabet

**plotchar (alphabet(: ,1) );** Выводим первую букву алфавита, чтобы убедиться, что все работает



Создаем сеть ( многослойный персептрон с обратной связью) со входным вектором, диапазон значений задается функцией **minmax** и выходным вектором с тем же диапазоном**,** количество нейронов в каждом слое сети (входной слой вектор 35, но мы его не пишем, она сама поймет, скрытый слой, например 20 с передаточной функцией **logsig** и выходной слой – 1 с передаточной функцией **logsig**), и в качестве обучающего алгоритма выбран алгоритм Levenberg-Marquardt (trainlm).

**net=newff(minmax(alphabet), minmax(alphabet) , [20,1],****{'logsig' , 'logsig' },'****trainlm');****net.performFcn='sse';** Задаем функцию оценки функционирования sse

Критерий SSE (Sum of squares due to error) - сумма квадратов ошибок.

http://matlab.exponenta.ru/curvefitting/images_3/065.gifКритерий SSE вычисляется по формуле:

где *wk* - веса, *yk* - данные выходные, а *http://matlab.exponenta.ru/curvefitting/images_3/y.gifk* – данные эталонные. Близость SSE к нулю говорит о хорошем качестве приближения данных параметрической моделью.

В нашем случае в качестве оценки вычисляется сумма квадратичных отклонений выходов сети от эталонов.

Задаем критерий окончания обучения – значение отклонения, при котором обучение будет считаться законченным:

**net.trainParam.goal=0.01**

Задаем максимальное количество циклов обучения. После того, как будет выполнено это количество циклов, обучение будет завершено:  
  
**net.trainParam.epochs=1000;**

Видим, что сеть создана

*Neural Network object:*

*architecture:*

*numInputs: 1*

*numLayers: 3*

*biasConnect: [1; 1; 1]*

*inputConnect: [1; 0; 0]*

*layerConnect: [0 0 0; 1 0 0; 0 1 0]*

*outputConnect: [0 0 1]*

*numOutputs: 1 (read-only)*

*numInputDelays: 0 (read-only)*

*numLayerDelays: 0 (read-only)*

*subobject structures:*

*inputs: {1x1 cell} of inputs*

*layers: {3x1 cell} of layers*

*outputs: {1x3 cell} containing 1 output*

*biases: {3x1 cell} containing 3 biases*

*inputWeights: {3x1 cell} containing 1 input weight*

*layerWeights: {3x3 cell} containing 2 layer weights*

*functions:*

*adaptFcn: 'trains'*

*divideFcn: 'dividerand'*

*gradientFcn: 'gdefaults'*

*initFcn: 'initlay'*

*performFcn: 'sse'*

*plotFcns: {'plotperform','plottrainstate','plotregression'}*

*trainFcn: 'trainlm'*

*parameters:*

*adaptParam: .passes*

*divideParam: .trainRatio, .valRatio, .testRatio*

*gradientParam: (none)*

*initParam: (none)*

*performParam: (none)*

*trainParam: .show, .showWindow, .showCommandLine, .epochs,*

*.time, .goal, .max\_fail, .mem\_reduc,*

*.min\_grad, .mu, .mu\_dec, .mu\_inc,*

*.mu\_max*

*weight and bias values:*

*IW: {3x1 cell} containing 1 input weight matrix*

*LW: {3x3 cell} containing 2 layer weight matrices*

*b: {3x1 cell} containing 3 bias vectors*

*other:*

*name: ''*

*userdata: (user information)*  
Теперь можно начинать обучение:

**[net,tr]=train(net,alphabet,alphabet)**

Состояние тренировки

*tr =*

*trainFcn: 'trainlm'*

*trainParam: [1x1 struct]*

*performFcn: 'sse'*

*performParam: [1x1 struct]*

*divideFcn: 'dividerand'*

*divideParam: [1x1 struct]*

*trainInd: [1 2 4 5 7 8 9 10 13 16 17 18 19 22 23 26]*

*valInd: [11 12 14 15 24]*

*testInd: [3 6 20 21 25]*

*stop: 'Validation stop.'*

*num\_epochs: 8*

*best\_epoch: 2*

*goal: 0.0100*

*states: {'epoch' 'time' 'perf' 'vperf' 'tperf' 'mu' 'gradient' 'val\_fail'}*

*epoch: [0 1 2 3 4 5 6 7 8]*

*time: [0.3540 0.4670 0.5770 0.7180 0.8640 0.9720 1.0920 1.2310 1.3730]*

*perf: [108.7332 96.9928 86.9276 82.0871 78.6760 76.2830 75.2448 74.7827 74.6757]*

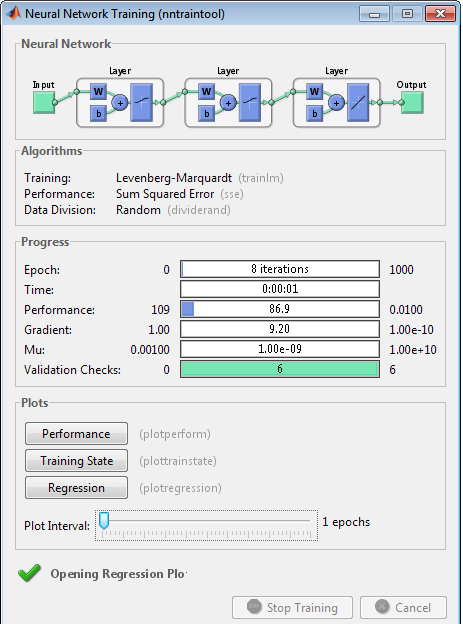
*vperf: [36.7501 37.1895 35.2946 35.7739 38.0309 39.1381 39.3440 40.7067 41.0557]*

*tperf: [26.4803 32.2162 27.6741 39.8570 32.7447 37.6924 34.8741 36.9248 27.6741]*

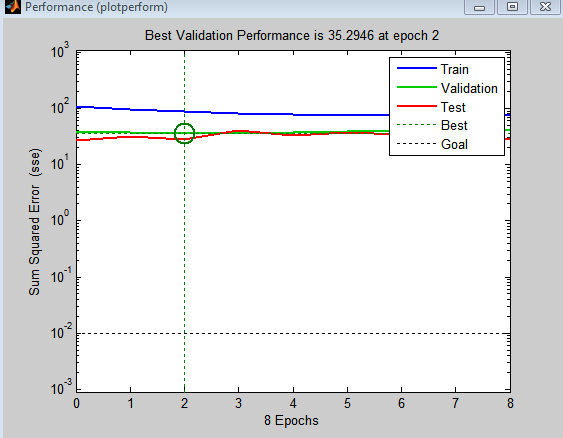
*mu: [1.0000e-003 1.0000e-004 1.0000e-005 1.0000e-005 1.0000e-005 1.0000e-006 1.0000e-007 1.0000e-008 1.0000e-009]*

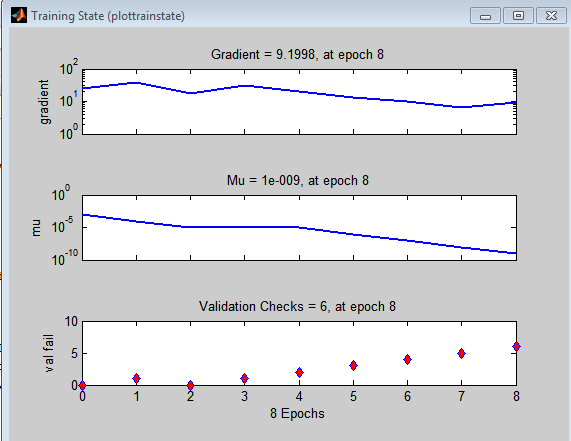
*gradient: [24.8736 38.3698 17.0197 31.7821 20.6245 13.1163 10.1158 6.7452 9.1998]*

*val\_fail: [0 1 0 1 2 3 4 5 6]*

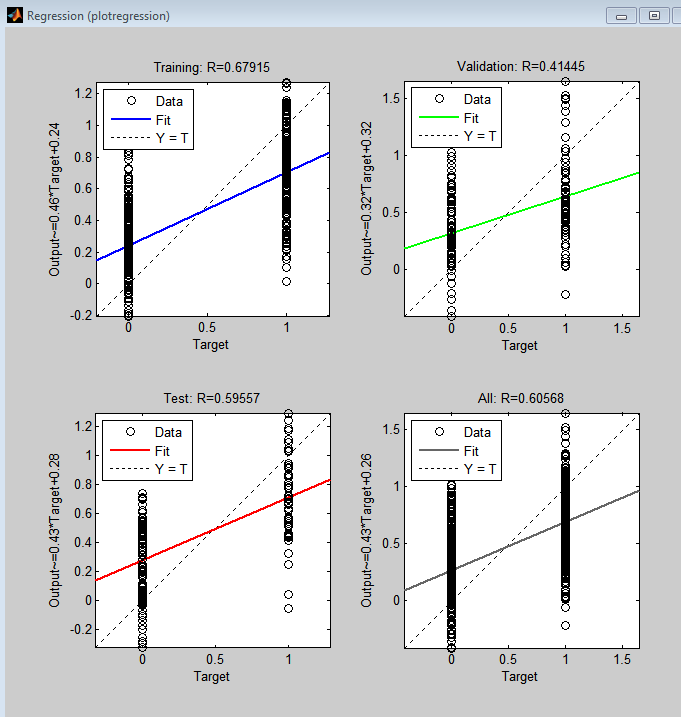
**

Процесс обучения иллюстрируется графиком зависимости оценки функционирования от номера цикла обучения.

****



Состояние обучения можно видеть на графике



Данные регрессии

Теперь, когда сеть обучилась, предъявим ей несколько букв для опознания, начнем с буквы **B**

**P=alphabet(: ,2)**

**P =**

**1**

**1**

**1**

**1**

**0**

**1**

**0**

**0**

**0**

**1**

**1**

**0**

**0**

**0**

**1**

**1**

**1**

**1**

**1**

**0**

**1**

**0**

**0**

**0**

**1**

**1**

**0**

**0**

**0**

**1**

**1**

**1**

**1**

**1**

**0**

**>>** **y=sim(net,P)**

**y =**

**0.5942**

**0.5630**

**0.6199**

**0.5660**

**0.5475**

**0.8243**

**0.1106**

**0.1201**

**0.0987**

**0.6928**

**0.6268**

**0.3143**

**0.1588**

**0.1502**

**0.4140**

**0.6810**

**0.2634**

**0.6077**

**0.2260**

**0.4214**

**0.6823**

**0.0530**

**0.3746**

**0.2849**

**0.4885**

**0.8079**

**0.0001**

**0.2187**

**0.1508**

**0.5408**

**0.3822**

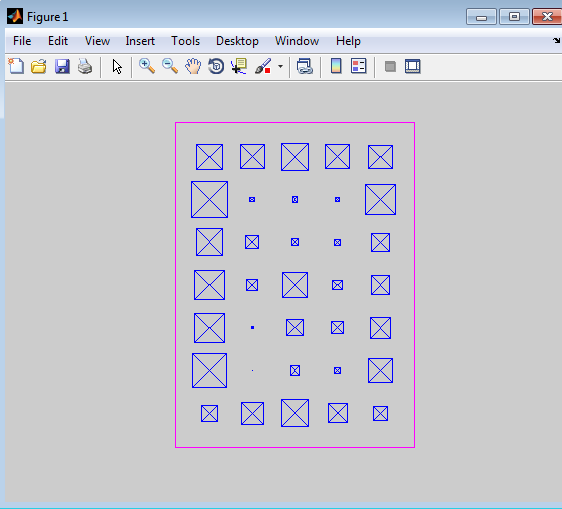
**0.5198**

**0.6408**

**0.4428**

**0.3307**

**>>** **plotchar (y);**

****

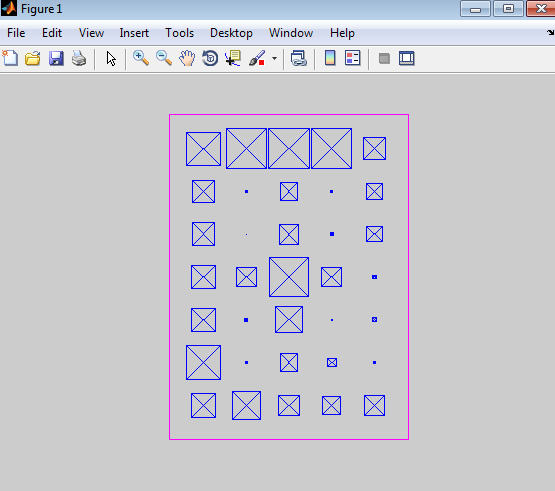
В принципе похоже на букву B

Попробуем с другими буквами, например F

P=alphabet(: ,6);

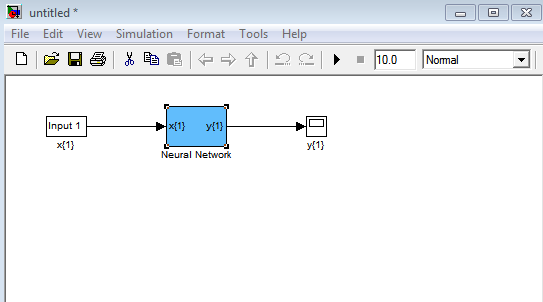
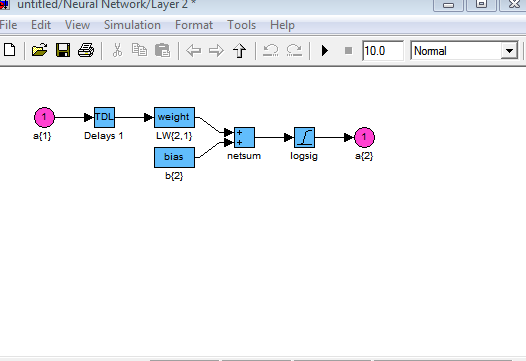
>> y=sim(net,P);

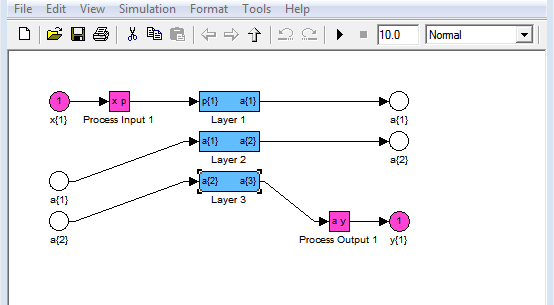
>> plotchar (y)

Вот что получилось:

Теперь можно построить другую сеть и посмотреть какие она даст результаты.

Если щелкнуть по сети в окне тренировки – можно получить более подробную структуру сети



и по слоя м и в каждом слое

Демонстрация респознавания букв с шумом

>> demo

>> appcr1

% NEWFF - Creates feed-forward networks.

% SIM - Simulates feed-forward networks.

% CHARACTER RECOGNITION:

% Using the above functions a feed-forward network is trained

% to recognize character bit maps, in the presence of noise.

pause % Strike any key to continue...

% DEFINING THE MODEL PROBLEM

% ==========================

% The script file PRPROB defines a matrix ALPHABET

% which contains the bit maps of the 26 letters of the

% alphabet.

% This file also defines target vectors TARGETS for

% each letter. Each target vector has 26 elements with

% all zeros, except for a single 1. A has a 1 in the

% first element, B in the second, etc.

[alphabet,targets] = prprob;

pause % Strike any key to define the network...

% DEFINING THE NETWORK

% ====================

% The character recognition network will have 25 TANSIG

% neurons in its hidden layer.

net = newff(alphabet,targets,25);

pause % Strike any key to train the network...

% TRAINING THE NETWORK WITHOUT NOISE

% ==================================

% The network will be trained without dividing data up into

% training and validation sets, because this is a small problem

% with only 26 samples.

%

% Training begins...please wait...

net1 = net;

net1.divideFcn = '';

[net1,tr] = train(net1,alphabet,targets);

net=newff(minmax(alphabet), minmax(alphabet) , [20,1],{'logsig' , 'logsig' },'trainlm');

% ...and finally finishes.

pause % Strike any key to train the network with noise...

% TRAINING THE NETWORK WITH NOISE

% ===============================

% The network will be trained on the original letters

% along with 10 sets of noisy letters.

numNoisy = 10;

alphabet2 = [alphabet repmat(alphabet,1,numNoisy)+randn(35,26\*numNoisy)\*0.2];

targets2 = [targets repmat(targets,1,numNoisy)];

net2 = train(net,alphabet2,targets2);

% ...and finally finishes.

pause % Strike any key to finish training the network...

% SET TESTING PARAMETERS

noise\_range = 0:.05:.5;

max\_test = 100;

network1 = [];

network2 = [];

% PERFORM THE TEST

for noiselevel = noise\_range

fprintf('Testing networks with noise level of %.2f.\n',noiselevel);

Testing networks with noise level of 0.00.

errors1 = 0;

errors2 = 0;

for i=1:max\_test

x = alphabet + randn(35,26)\*noiselevel;

% TEST NETWORK 1

y = sim(net1,x);

yy = compet(y);

errors1 = errors1 + sum(sum(abs(yy-targets)))/2;

% TEST NETWORK 2

yn = sim(net2,x);

yyn = compet(yn);

errors2 = errors2 + sum(sum(abs(yyn-targets)))/2;

echo off

Testing networks with noise level of 0.05.

Testing networks with noise level of 0.10.

Testing networks with noise level of 0.15.

Testing networks with noise level of 0.20.

Testing networks with noise level of 0.25.

Testing networks with noise level of 0.30.

Testing networks with noise level of 0.35.

Testing networks with noise level of 0.40.

Testing networks with noise level of 0.45.

Testing networks with noise level of 0.50.

pause % Strike any key to display the test results...

% DISPLAY RESULTS

% ===============

% Here is a plot showing the percentage of errors for

% the two networks for varying levels of noise.

clf

plot(noise\_range,network1\*100,'--',noise\_range,network2\*100);

title('Percentage of Recognition Errors');

xlabel('Noise Level');

ylabel('Network 1 \_ \_ Network 2 \_\_\_');

% Network 1, trained without noise, has more errors due

% to noise than does Network 2, which was trained with noise.

echo off

End of APPCR1

Warning: incomplete command in TeX text string:

'