Neural Networks: MATLAB examples

Primoz Potocnik
University of Ljubljana
Faculty of Mechanical Engineering
LASIN - Laboratory of Synergetics
www.neural.si | primoz.potocnik@fs.uni-lj.si

Contents

1. nn02_neuron_output - Calculate the output of a simple neuron
2. nn02_custom_nn - Create and view custom neural networks
3. nn03_perceptron - Classification of linearly separable data with a perceptron
4. nn03_perceptron_network - Classification of a 4-class problem with a 2-neuron perceptron
5. nn03_adaline - ADALINE time series prediction with adaptive linear filter
6. nn04_mlp_xor - Classification of an XOR problem with a multilayer perceptron
7. nn04_mlp_4classes - Classification of a 4-class problem with a multilayer perceptron
8. nn04_technical_diagnostic - Industrial diagnostic of compressor connection rod defects [data2.zip]
9. nn05_narnet - Prediction of chaotic time series with NAR neural network
10. nn06_rbfn_func - Radial basis function networks for function approximation
11. nn06_rbfn_xor - Radial basis function networks for classification of XOR problem
12. nn07_som - 1D and 2D Self Organized Map
13. nn08_tech_diag_pca - PCA for industrial diagnostic of compressor connection rod defects [data2.zip]

Neuron output

Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: Calculate the output of a simple neuron

Contents

- Define neuron parameters
- Define input vector
- Calculate neuron output
- Plot neuron output over the range of inputs

Define neuron parameters

```
close all, clear all, clc, format compact

% Neuron weights
w = [4 -2]
% Neuron bias
b = -3
% Activation function
 func = 'tansig'
% func = 'purelin'
% func = 'hardlim'
% func = 'logsig'
```

```
w =
     4    -2
b =
    -3
func =
tansig
```

Define input vector

```
p = [2 3]
```

```
p =
     2     3
```

Calculate neuron output

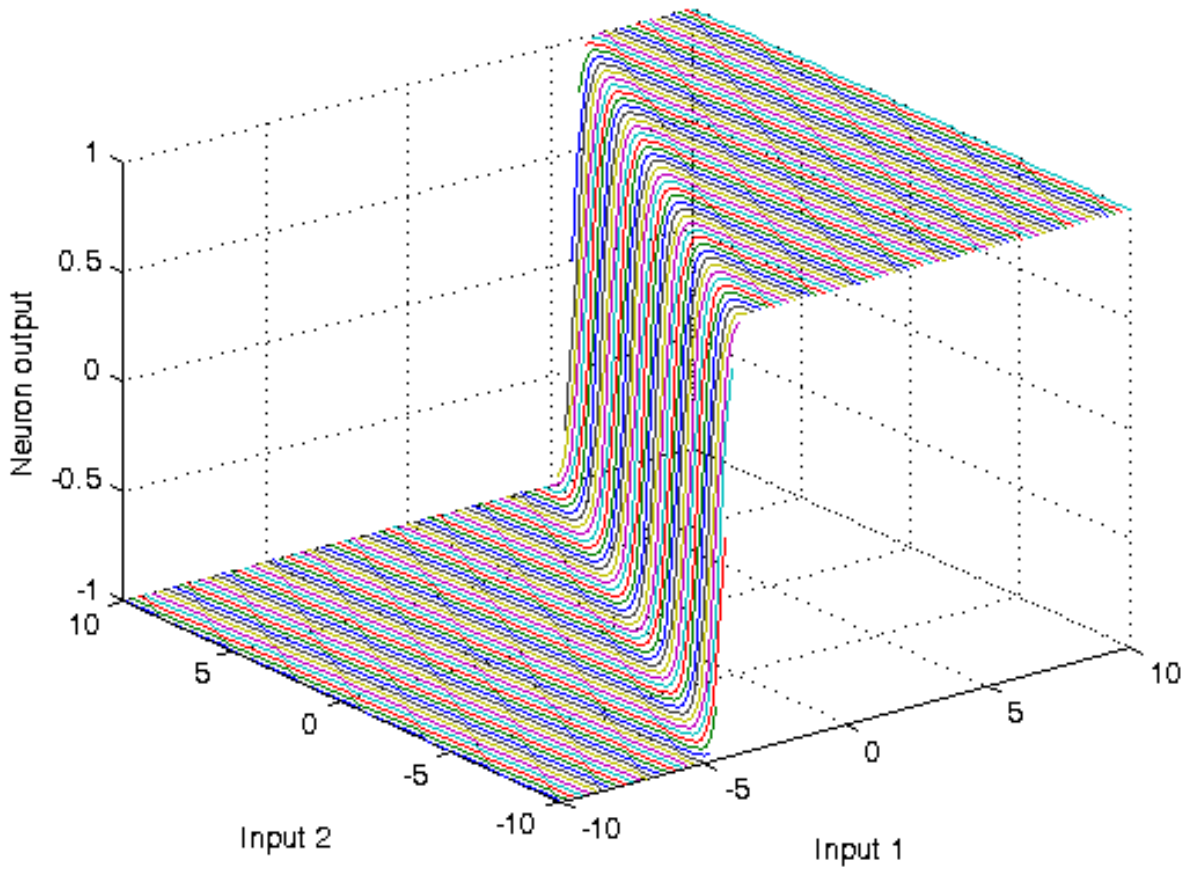
```
activation_potential = p*w'+b
```

```
neuron_output = feval(func, activation_potential)
```

```
activation_potential =
    -1
neuron_output =
   -0.7616
```

Plot neuron output over the range of inputs

```
[p1,p2] = meshgrid(-10:.25:10);
z = feval(func, [p1(:) p2(:)]*w'+b );
z = reshape(z,length(p1),length(p2));
plot3(p1,p2,z)
grid on
xlabel('Input 1')
ylabel('Input 2')
zlabel('Neuron output')
```



Published with MATLAB® 7.14

Custom networks

Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: Create and view custom neural networks

Contents

- Define one sample: inputs and outputs
- Define and custom network
- Define topology and transfer function
- Configure network
- Train net and calculate neuron output

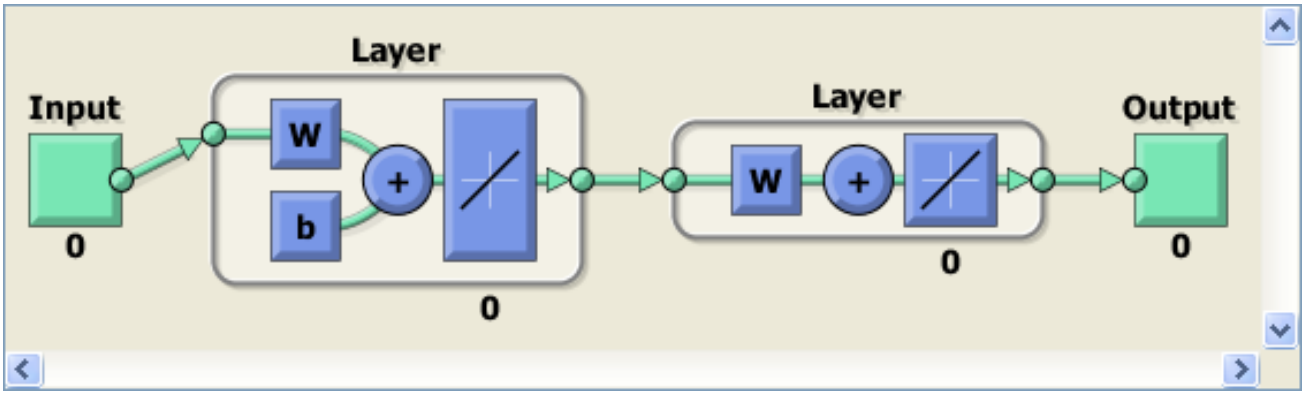
Define one sample: inputs and outputs

```
close all, clear all, clc, format compact

inputs  = [1:6]' % input vector (6-dimensional pattern)
outputs = [1 2]' % corresponding target output vector
```

```
inputs =
     1
     2
     3
     4
     5
     6
outputs =
     1
     2
```

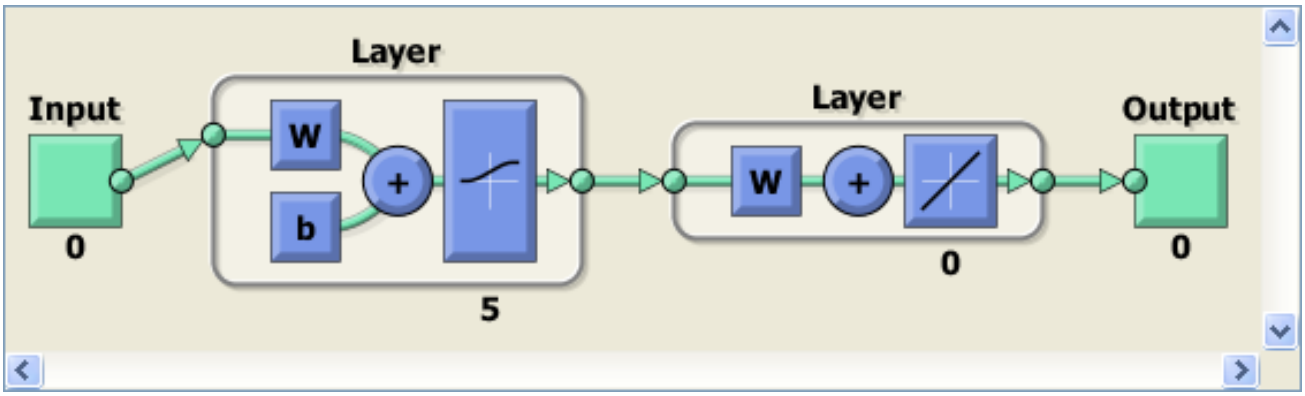
Define and custom network

```
% create network
net = network( ...
1,           ... % numInputs,    number of inputs,
2,           ... % numLayers,    number of layers
[1; 0],      ... % biasConnect,  numLayers-by-1 Boolean vector,
[1; 0],      ... % inputConnect, numLayers-by-numInputs Boolean matrix,
[0 0; 1 0], ... % layerConnect, numLayers-by-numLayers Boolean matrix
[0 1]        ... % outputConnect, 1-by-numLayers Boolean vector
);

% View network structure
view(net);
```



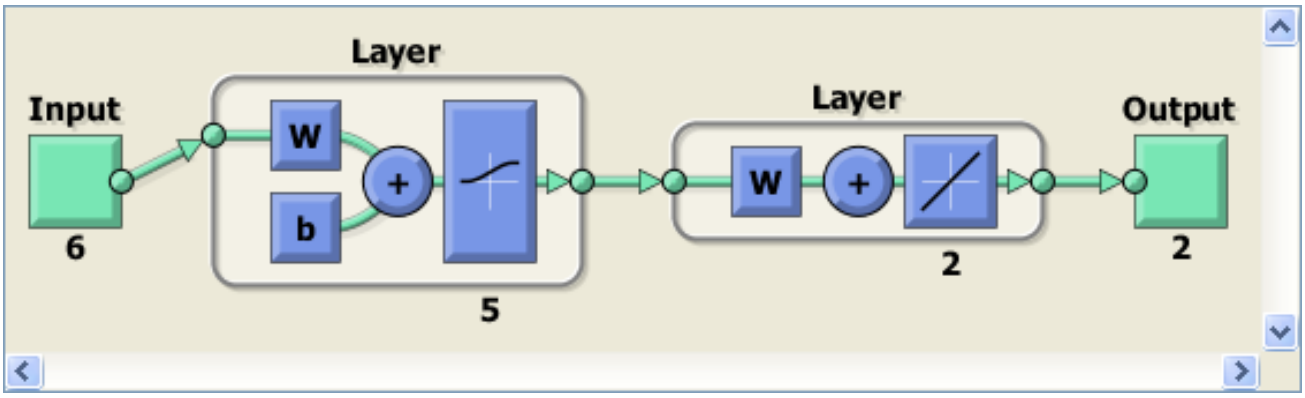
Define topology and transfer function

```
% number of hidden layer neurons
net.layers{1}.size = 5;

% hidden layer transfer function
net.layers{1}.transferFcn = 'logsig';
view(net);
```



Configure network

```
net = configure(net,inputs,outputs);
view(net);
```



Train net and calculate neuron output

```
% initial network response without training
initial_output = net(inputs)

% network training
net.trainFcn = 'trainlm';
net.performFcn = 'mse';
net = train(net,inputs,outputs);

% network response after training
final_output = net(inputs)
```

```
initial_output =
     0
     0
final_output =
    1.0000
    2.0000
```

Published with MATLAB® 7.14

Classification of linearly separable data with a perceptron

Neural Networks course (practical examples) © 2012 Primoz Potocnik

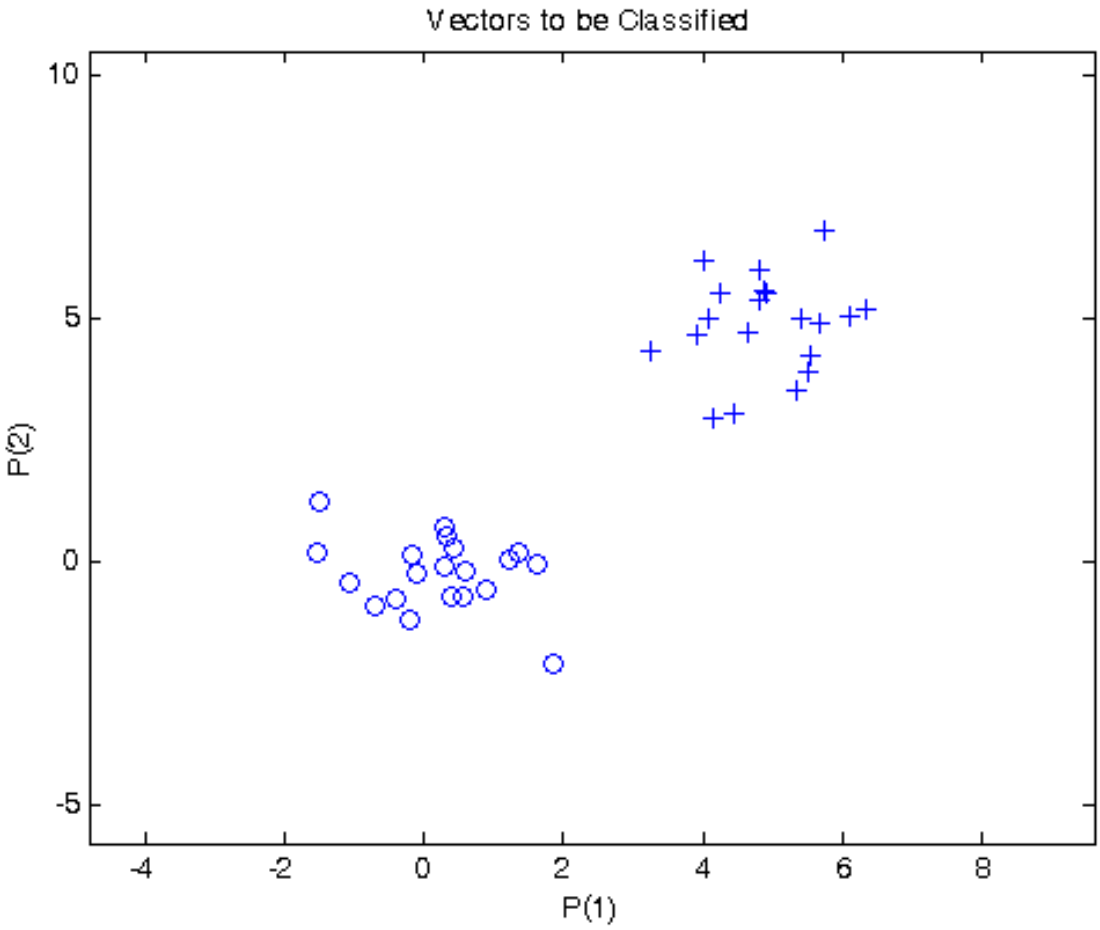
PROBLEM DESCRIPTION: Two clusters of data, belonging to two classes, are defined in a 2-dimensional input space. Classes are linearly separable. The task is to construct a Perceptron for the classification of data.

Contents

- Define input and output data
- Create and train perceptron
- Plot decision boundary

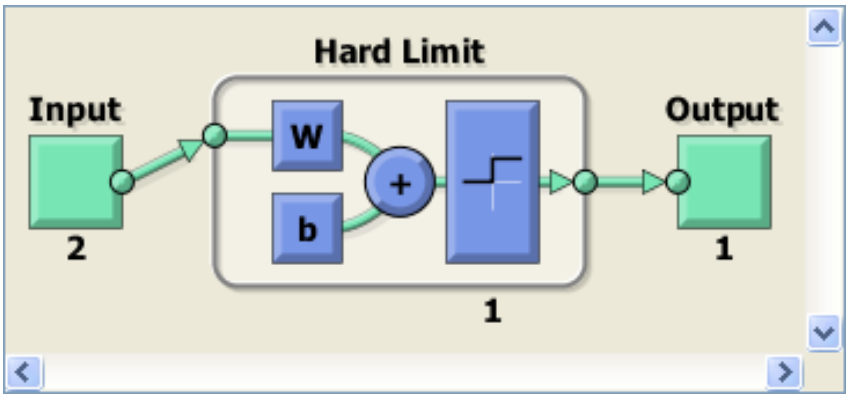
Define input and output data

```
close all, clear all, clc, format compact

% number of samples of each class
N = 20;
% define inputs and outputs
offset = 5; % offset for second class
x = [randn(2,N) randn(2,N)+offset]; % inputs
y = [zeros(1,N) ones(1,N)];         % outputs

% Plot input samples with PLOTPV (Plot perceptron input/target vectors)
figure(1)
plotpv(x,y);
```



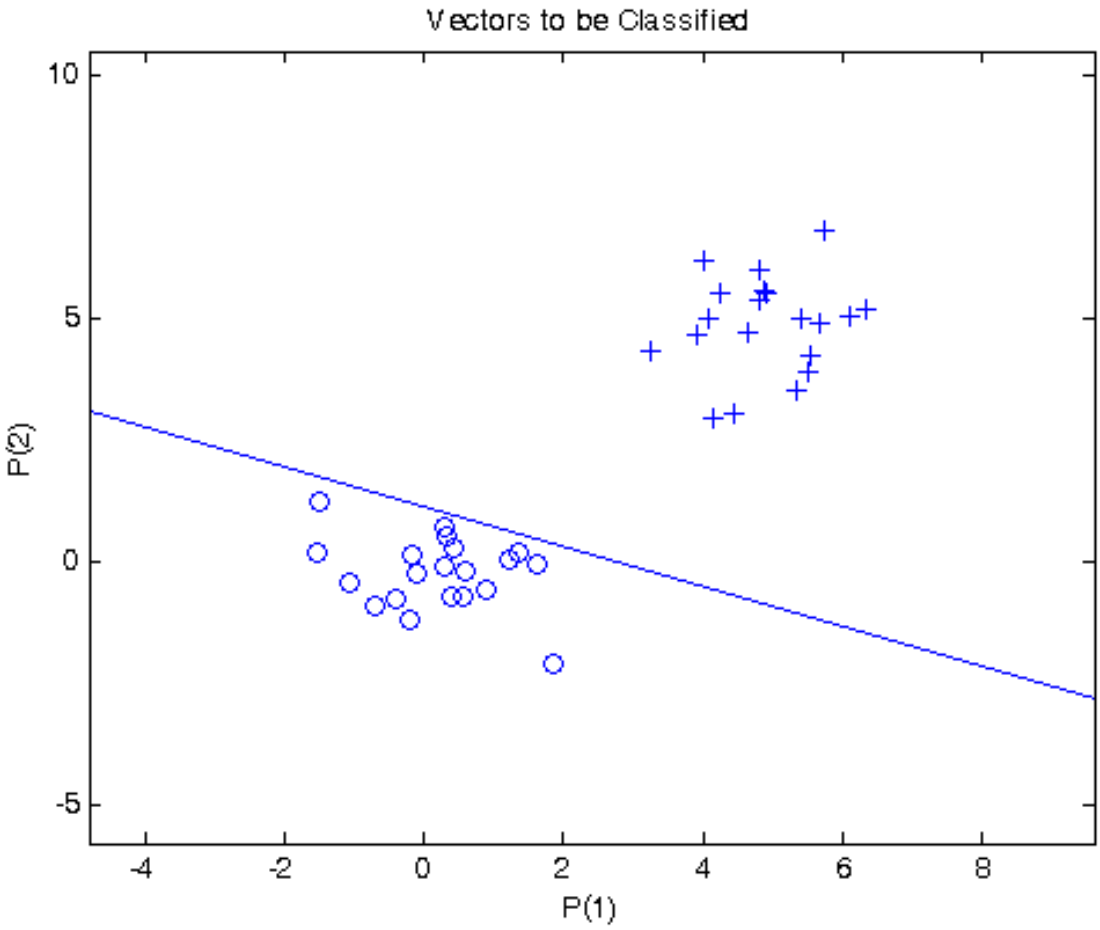
Create and train perceptron

```
net = perceptron;
net = train(net,x,y);
view(net);
```



Plot decision boundary

```
figure(1)
plotpc(net.IW{1},net.b{1});
```



Published with MATLAB® 7.14

Classification of a 4-class problem with a perceptron

Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: Perceptron network with 2-inputs and 2-outputs is trained to classify input vectors into 4 categories

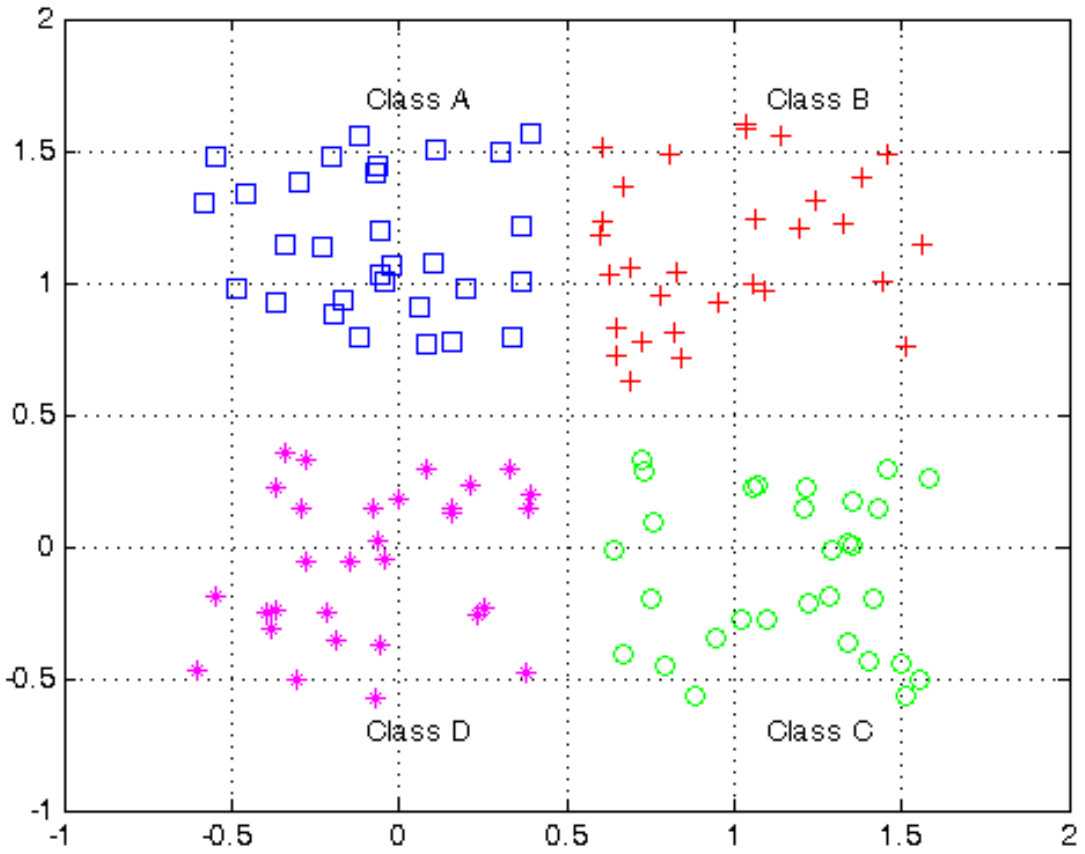
Contents

- Define data
- Prepare inputs & outputs for perceptron training
- Create a perceptron
- Train a perceptron
- How to use trained perceptron

Define data

```
close all, clear all, clc, format compact

% number of samples of each class
K = 30;

% define classes
q = .6; % offset of classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot classes
plot(A(1,:),A(2,:),'bs')
hold on
grid on
plot(B(1,:),B(2,:),'r+')
plot(C(1,:),C(2,:),'go')
plot(D(1,:),D(2,:),'m*')
% text labels for classes
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class C')
text(.5-q,.5-2*q,'Class D')

% define output coding for classes
a = [0 1]';
b = [1 1]';
c = [1 0]';
d = [0 0]';
% % Why this coding doesn't work?
% a = [0 0]';
% b = [1 1]';
% d = [0 1]';
```

```
% c = [1 0]';
% % Why this coding doesn't work?
% a = [0 1]';
% b = [1 1]';
% d = [1 0]';
% c = [0 1]';
```



Prepare inputs & outputs for perceptron training

```
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
     repmat(c,1,length(C)) repmat(d,1,length(D)) ];
%plotpv(P,T);
```

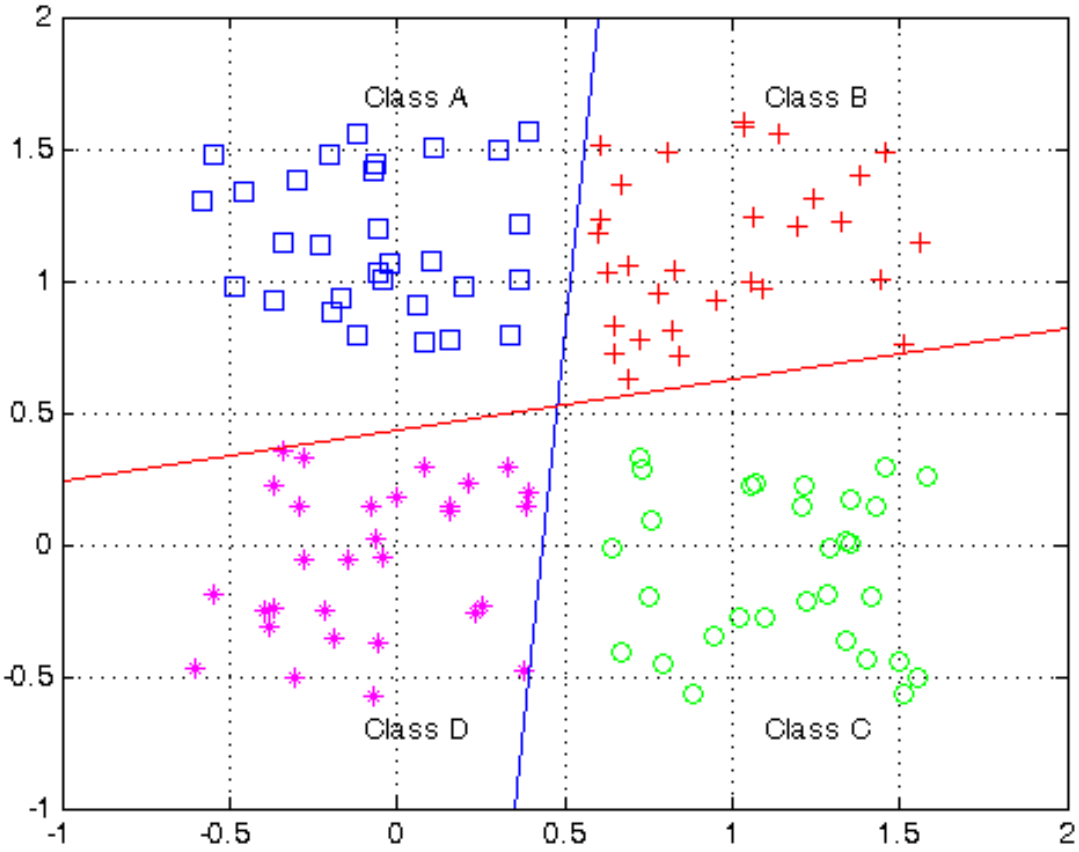
Create a perceptron

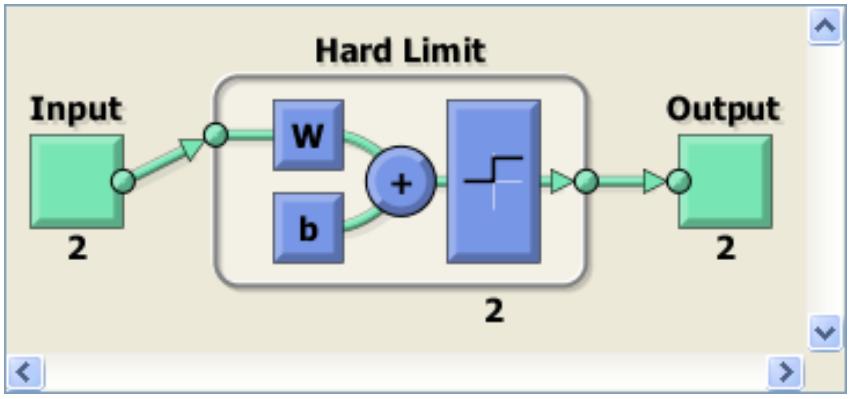
```
net = perceptron;
```

Train a perceptron

ADAPT returns a new network object that performs as a better classifier, the network output, and the error. This loop allows the network to adapt for xx passes, plots the classification line, and continues until the error is zero.

```
E = 1;
net.adaptParam.passes = 1;
linehandle = plotpc(net.IW{1},net.b{1});
n = 0;
while (sse(E) & n<1000)
   n = n+1;
   [net,Y,E] = adapt(net,P,T);
   linehandle = plotpc(net.IW{1},net.b{1},linehandle);
   drawnow;
end
% show perceptron structure
view(net);
```





How to use trained perceptron

```
% For example, classify an input vector of [0.7; 1.2]
p = [0.7; 1.2]
y = net(p)
% compare response with output coding (a,b,c,d)
```

```
p =
    0.7000
    1.2000
y =
     1
     1
```

Published with MATLAB® 7.14

ADALINE time series prediction

Neural Networks course (practical examples) © 2012 Primoz Potocnik

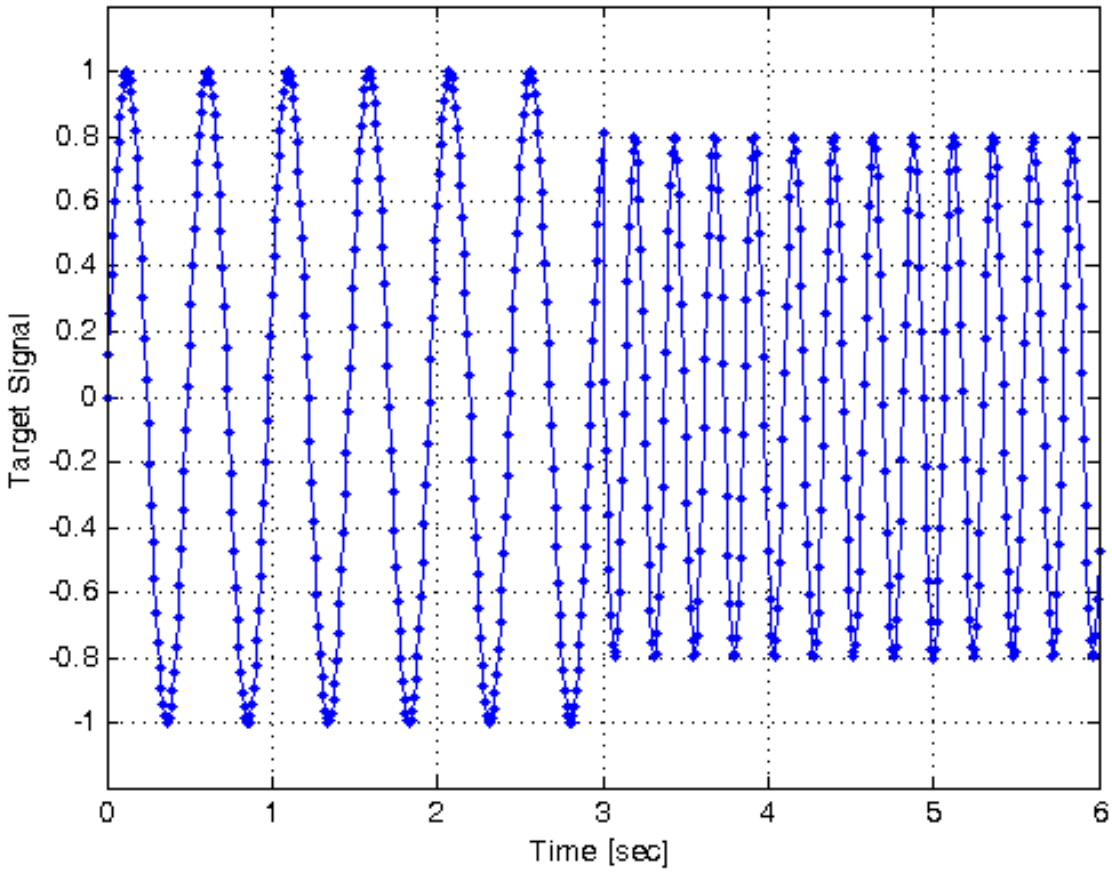
PROBLEM DESCRIPTION: Construct an ADALINE for adaptive prediction of time series based on past time series data

Contents

- Define input and output data
- Prepare data for neural network toolbox
- Define ADALINE neural network
- Adaptive learning of the ADALINE
- Plot results

Define input and output data

```
close all, clear all, clc, format compact

% define segments of time vector
dt = 0.01; % time step [seconds]
t1 = 0    : dt : 3; %    first time vector [seconds]
t2 = 3+dt : dt : 6; %   second time vector [seconds]
t = [t1 t2];        % complete time vector [seconds]
% define signal
y = [sin(4.1*pi*t1) .8*sin(8.3*pi*t2)];
% plot signal
plot(t,y,'.-')
xlabel('Time [sec]');
ylabel('Target Signal');
grid on
ylim([-1.2 1.2])
```



Prepare data for neural network toolbox

```
% There are two basic types of input vectors: those that occur concurrently
% (at the same time, or in no particular time sequence), and those that
% occur sequentially in time. For concurrent vectors, the order is not
% important, and if there were a number of networks running in parallel,
% you could present one input vector to each of the networks. For
% sequential vectors, the order in which the vectors appear is important.
p = con2seq(y);
```

Define ADALINE neural network

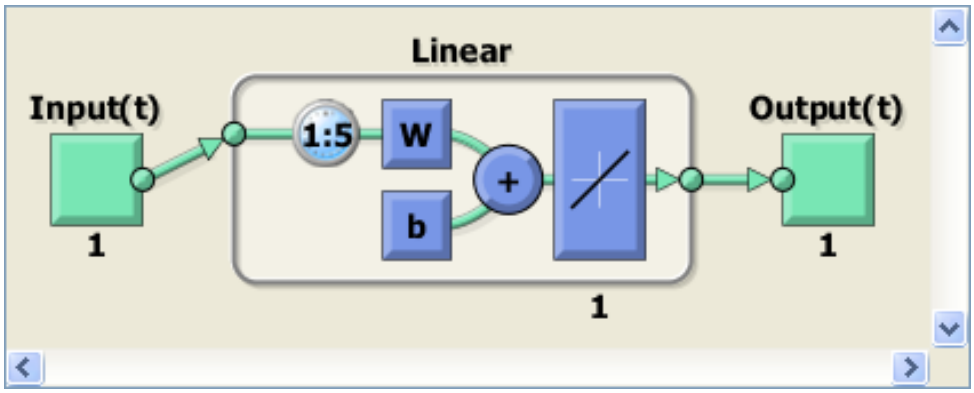
```
% The resulting network will predict the next value of the target signal
% using delayed values of the target.
inputDelays   = 1:5; % delayed inputs to be used
learning_rate = 0.2; % learning rate

% define ADALINE
net = linearlayer(inputDelays,learning_rate);
```

Adaptive learning of the ADALINE

```
% Given an input sequence with N steps the network is updated as follows.
% Each step in the sequence of  inputs is presented to the network one at
% a time.  The network's weight and bias values are updated after each step,
```

```
% before the next step in the sequence is presented. Thus the network is
% updated N times. The output signal and the error signal are returned,
% along with new network.
[net,Y,E] = adapt(net,p,p);

% view network structure
view(net)

% check final network parameters
disp('Weights and bias of the ADALINE after adaptation')
net.IW{1}
net.b{1}
```

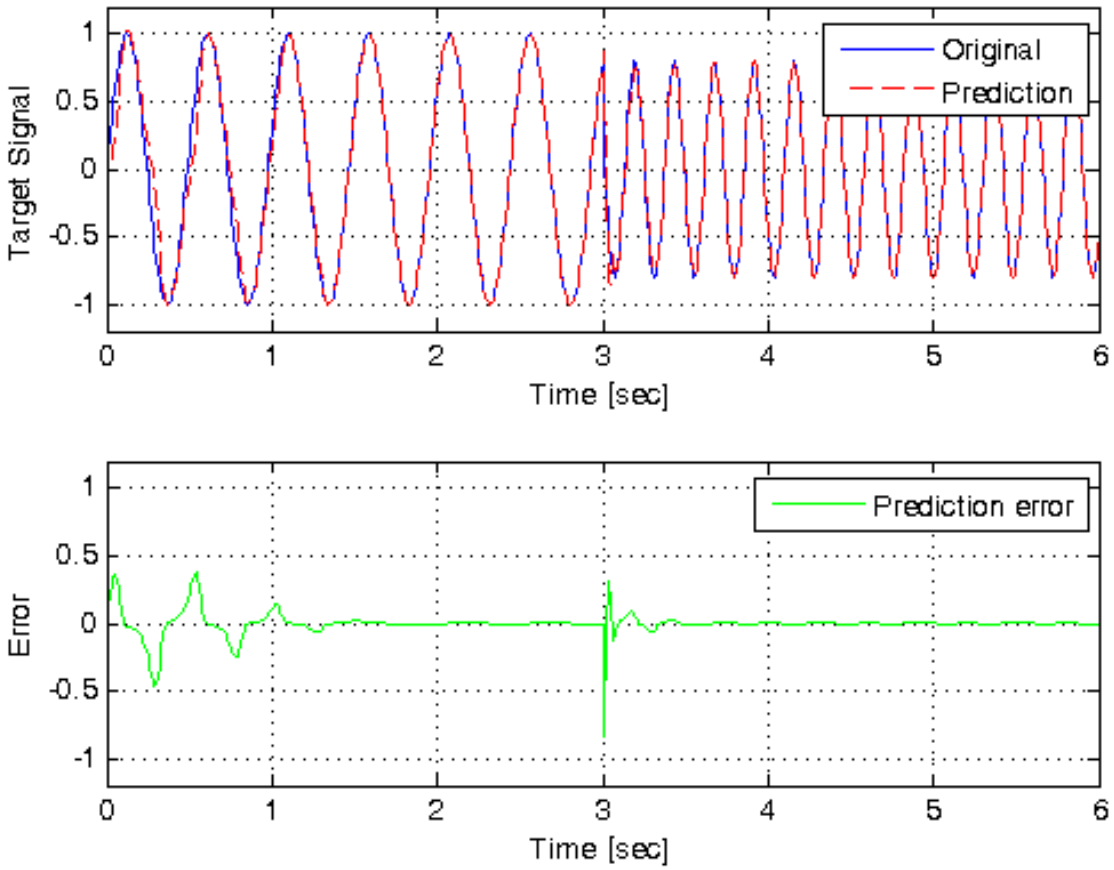
```
Weights and bias of the ADALINE after adaptation
ans =
    0.7179    0.4229    0.1552   -0.1203   -0.4159
ans =
  -1.2520e-08
```



Plot results

```
% transform result vectors
Y = seq2con(Y); Y = Y{1};
E = seq2con(E); E = E{1};
% start a new figure
figure;

% first graph
subplot(211)
plot(t,y,'b', t,Y,'r--');
legend('Original','Prediction')
grid on
xlabel('Time [sec]');
ylabel('Target Signal');
ylim([-1.2 1.2])

% second graph
subplot(212)
plot(t,E,'g');
grid on
```

```
legend('Prediction error')
xlabel('Time [sec]');
ylabel('Error');
ylim([-1.2 1.2])
```



Published with MATLAB® 7.14

Solving XOR problem with a multilayer perceptron

Neural Networks course (practical examples) © 2012 Primoz Potocnik

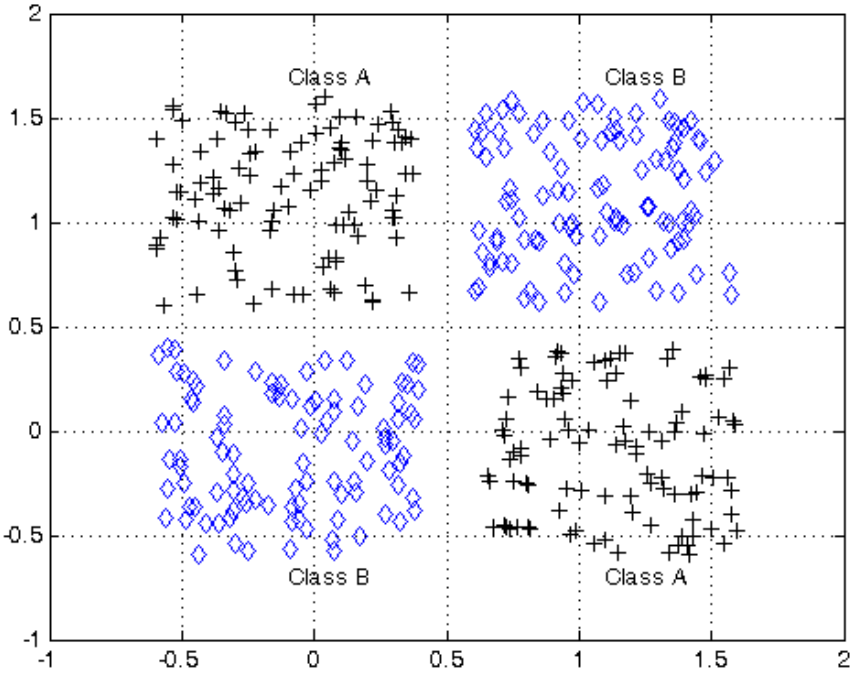
PROBLEM DESCRIPTION: 4 clusters of data (A,B,C,D) are defined in a 2-dimensional input space. (A,C) and (B,D) clusters represent XOR classification problem. The task is to define a neural network for solving the XOR problem.

Contents

- Define 4 clusters of input data
- Define output coding for XOR problem
- Prepare inputs & outputs for network training
- Create and train a multilayer perceptron
- plot targets and network response to see how good the network learns the data
- Plot classification result for the complete input space

Define 4 clusters of input data

```
close all, clear all, clc, format compact

% number of samples of each class
K = 100;
% define 4 clusters of input data
q = .6; % offset of classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot clusters
figure(1)
plot(A(1,:),A(2,:),'k+')
hold on
grid on
plot(B(1,:),B(2,:),'bd')
plot(C(1,:),C(2,:),'k+')
plot(D(1,:),D(2,:),'bd')
% text labels for clusters
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class A')
text(.5-q,.5-2*q,'Class B')
```



Define output coding for XOR problem

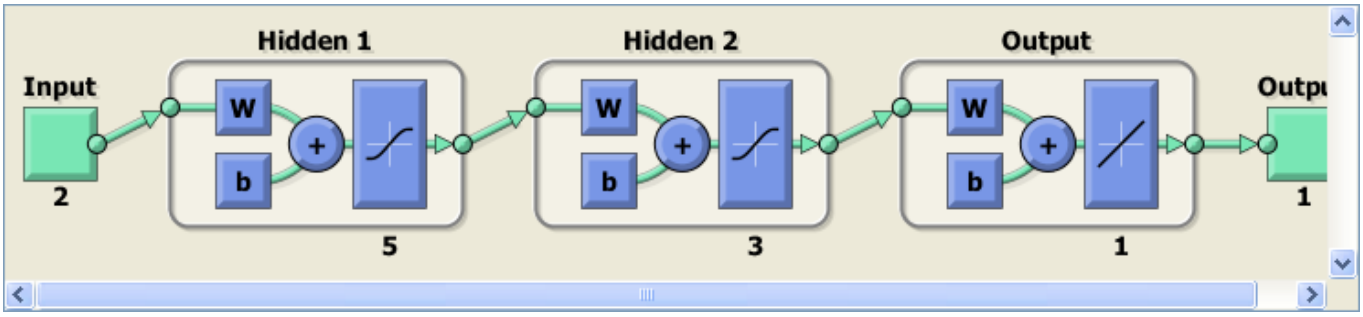
```
% encode clusters a and c as one class, and b and d as another class
a = -1; %  a  |  b
c = -1; %  -------
b =  1; %  d  |  c
d =  1; %
```

Prepare inputs & outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
     repmat(c,1,length(C)) repmat(d,1,length(D)) ];
% view inputs |outputs
%[P' T']
```

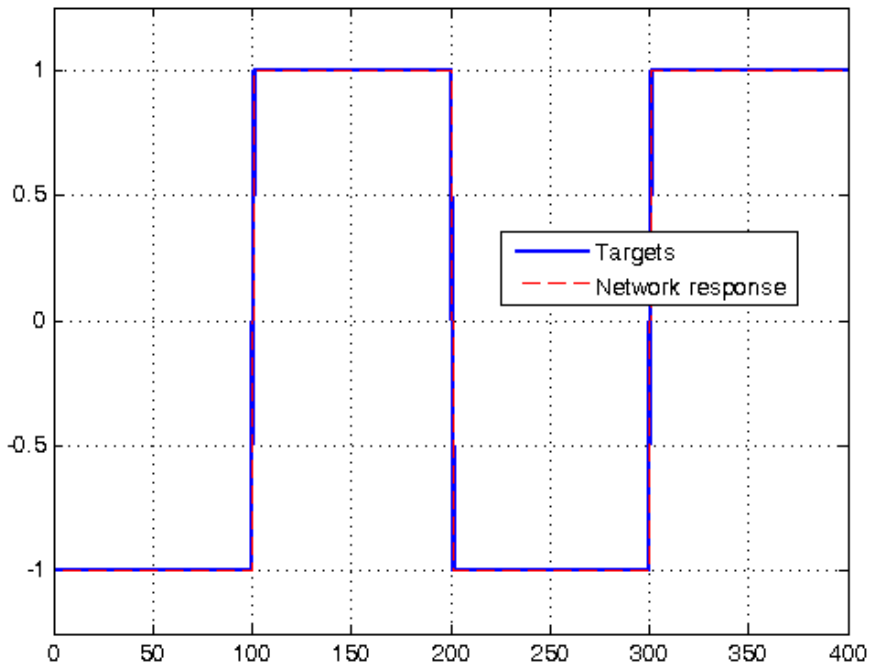
Create and train a multilayer perceptron

```
% create a neural network
net = feedforwardnet([5 3]);

% train net
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio   = 0; % validation set [%]
net.divideParam.testRatio  = 0; % test set [%]

% train a neural network
[net,tr,Y,E] = train(net,P,T);

% show network
view(net)
```



plot targets and network response to see how good the network learns the data

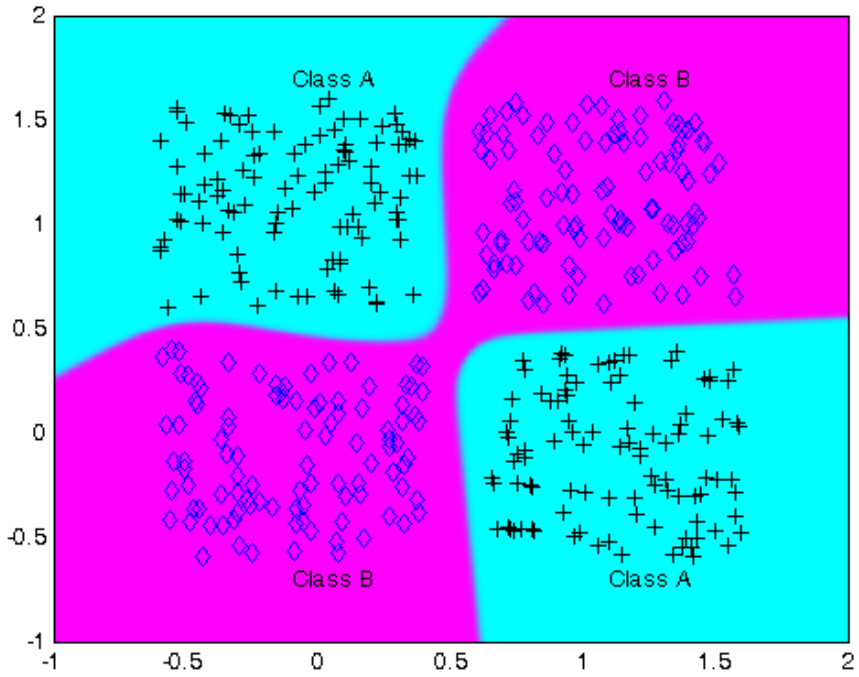
```
figure(2)
plot(T','linewidth',2)
hold on
plot(Y','r--')
grid on
legend('Targets','Network response','location','best')
ylim([-1.25 1.25])
```



Plot classification result for the complete input space

```
% generate a grid
span = -1:.005:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';

% simulate neural network on a grid
aa = net(pp);

% translate output into [-1,1]
%aa = -1 + 2*(aa>0);

% plot classification regions
figure(1)
mesh(P1,P2,reshape(aa,length(span),length(span))-5);
colormap cool
```

```
view(2)
```



Published with MATLAB® 7.14

Classification of a 4-class problem with a multilayer perceptron

Neural Networks course (practical examples) © 2012 Primoz Potocnik

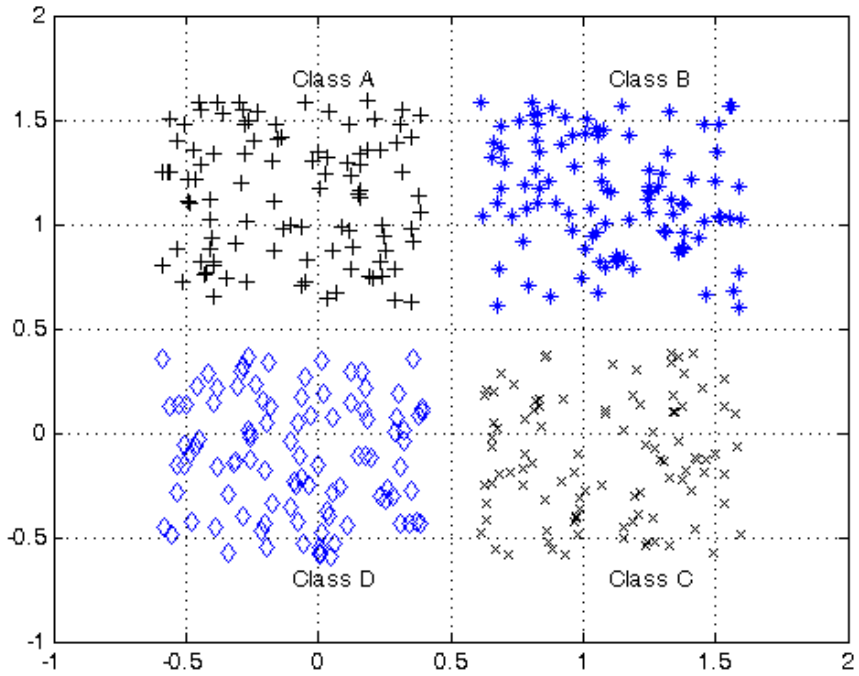
PROBLEM DESCRIPTION: 4 clusters of data (A,B,C,D) are defined in a 2-dimensional input space. The task is to define a neural network for classification of arbitrary point in the 2-dimensional space into one of the classes (A,B,C,D).

Contents

- Define 4 clusters of input data
- Define output coding for all 4 clusters
- Prepare inputs & outputs for network training
- Create and train a multilayer perceptron
- Evaluate network performance and plot results
- Plot classification result for the complete input space

Define 4 clusters of input data

```
close all, clear all, clc, format compact

% number of samples of each class
K = 100;
% define 4 clusters of input data
q = .6; % offset of classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot clusters
figure(1)
plot(A(1,:),A(2,:),'k+')
hold on
grid on
plot(B(1,:),B(2,:),'b*')
plot(C(1,:),C(2,:),'kx')
plot(D(1,:),D(2,:),'bd')
% text labels for clusters
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class C')
text(.5-q,.5-2*q,'Class D')
```



Define output coding for all 4 clusters

```
% coding (+1/-1) of 4 separate classes
a = [-1 -1 -1 +1]';
b = [-1 -1 +1 -1]';
d = [-1 +1 -1 -1]';
c = [+1 -1 -1 -1]';
```

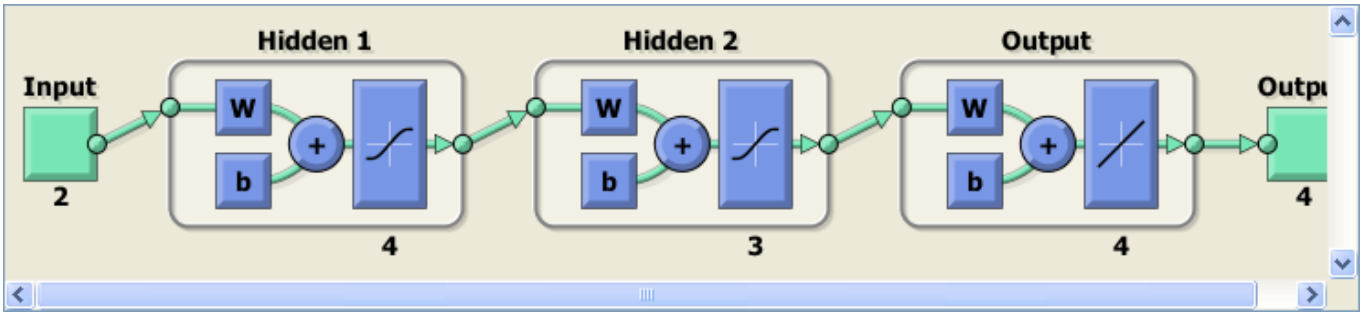
Prepare inputs & outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
     repmat(c,1,length(C)) repmat(d,1,length(D)) ];
```

Create and train a multilayer perceptron

```
% create a neural network
net = feedforwardnet([4 3]);

% train net
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio   = 0; % validation set [%]
net.divideParam.testRatio  = 0; % test set [%]

% train a neural network
[net,tr,Y,E] = train(net,P,T);
```

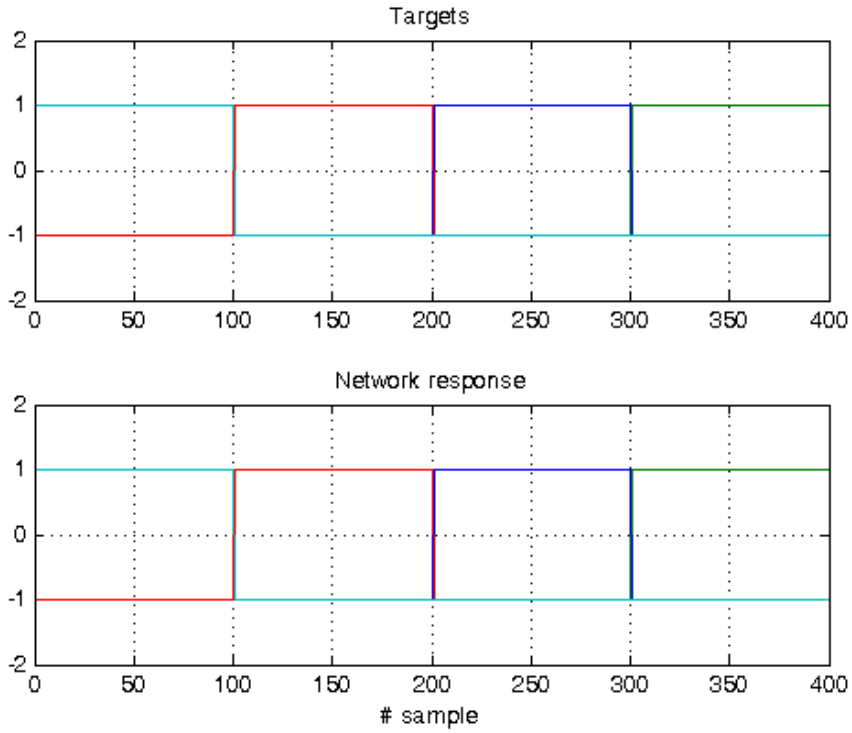
```
% show network
view(net)
```



Evaluate network performance and plot results

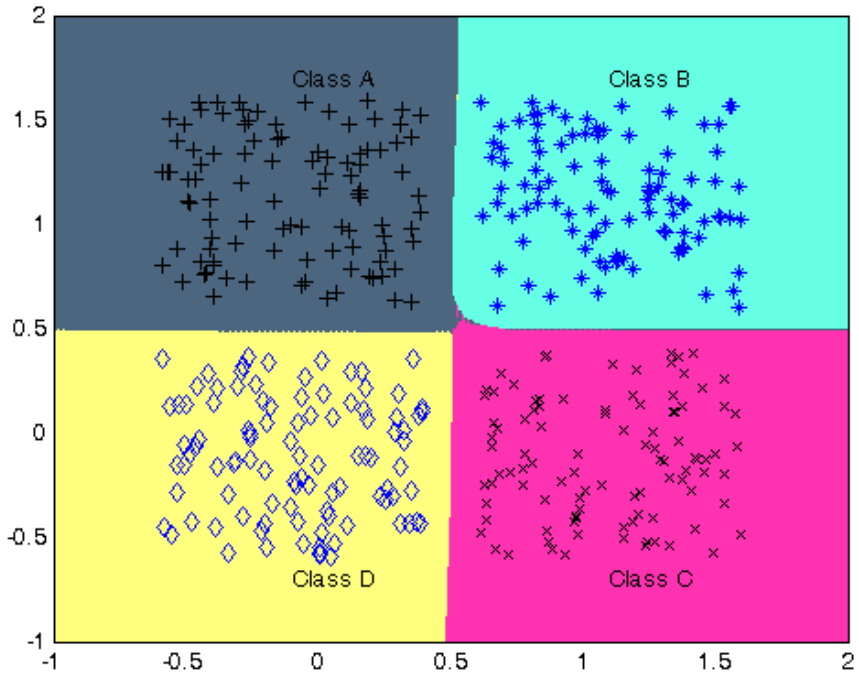
```
% evaluate performance: decoding network response
[m,i] = max(T); % target class
[m,j] = max(Y); % predicted class
N = length(Y);  % number of all samples
k = 0;          % number of missclassified samples
if find(i-j),   % if there exist missclassified samples
  k = length(find(i-j)); % get a number of missclassified samples
end
fprintf('Correct classified samples: %.1f%% samples\n', 100*(N-k)/N)

% plot network output
figure;
subplot(211)
plot(T')
title('Targets')
ylim([-2 2])
grid on
subplot(212)
plot(Y')
title('Network response')
xlabel('# sample')
ylim([-2 2])
grid on
```

```
Correct classified samples: 100.0% samples
```



Plot classification result for the complete input space

```
% generate a grid
span = -1:.01:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';

% simualte neural network on a grid
aa = net(pp);

% plot classification regions based on MAX activation
figure(1)
m = mesh(P1,P2,reshape(aa(1,:),length(span),length(span))-5);
set(m,'facecolor',[1 0.2 .7],'linestyle','none');
hold on
m = mesh(P1,P2,reshape(aa(2,:),length(span),length(span))-5);
set(m,'facecolor',[1 1.0 0.5],'linestyle','none');
m = mesh(P1,P2,reshape(aa(3,:),length(span),length(span))-5);
set(m,'facecolor',[.4 1.0 0.9],'linestyle','none');
m = mesh(P1,P2,reshape(aa(4,:),length(span),length(span))-5);
set(m,'facecolor',[.3 .4 0.5],'linestyle','none');
view(2)
```



Published with MATLAB® 7.14

Industrial diagnostic of compressor connection rod defects

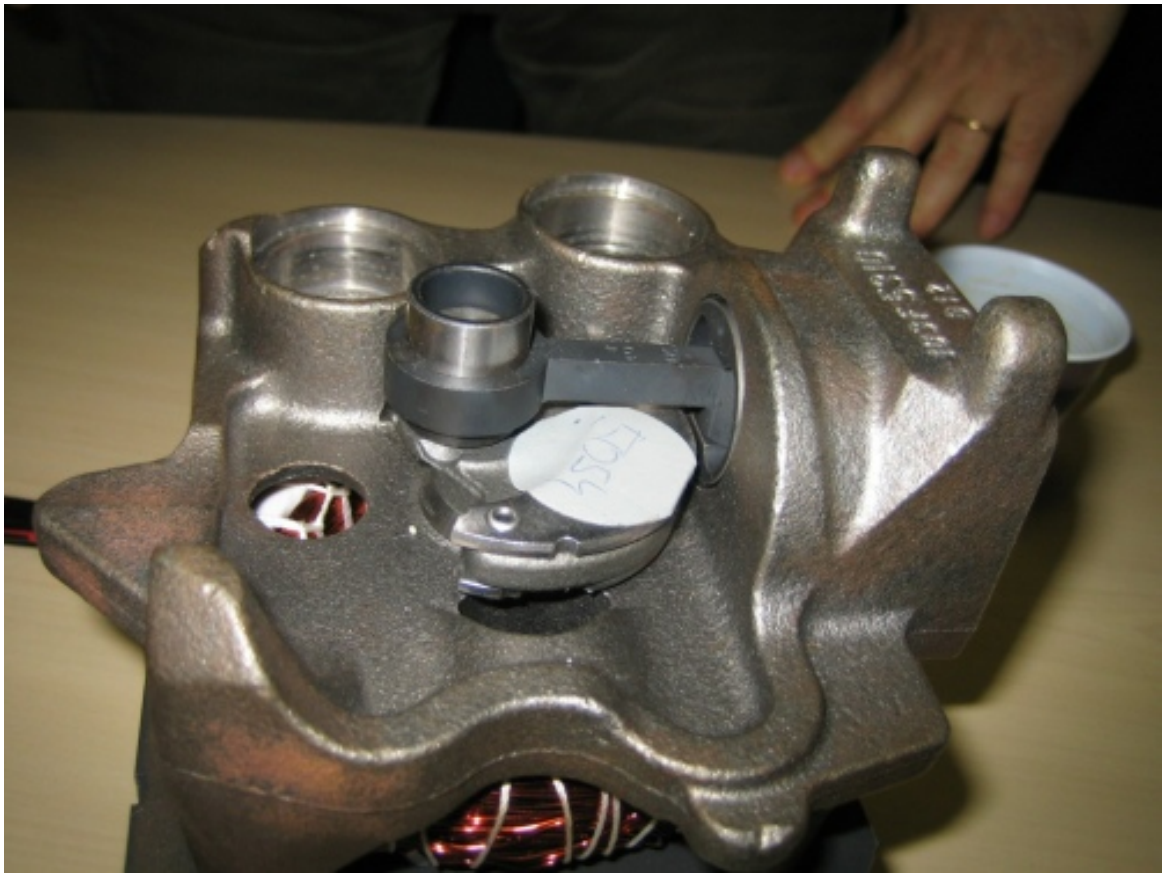
Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: Industrial production of compressors suffers from problems during the imprinting operation where a connection rod is connected with a compressor head. Irregular imprinting can cause damage or crack of the connection rod which results in damaged compressor. Such compressors should be eliminated from the production line but defects of this type are difficult to detect. The task is to detect crack and overload defects from the measurement of the imprinting force.

Contents

- Photos of the broken connection rod
- Load and plot data
- Prepare inputs: Data resampling
- Define binary output coding: 0=OK, 1=Error
- Create and train a multilayer perceptron
- Evaluate network performance
- Application

Photos of the broken connection rod

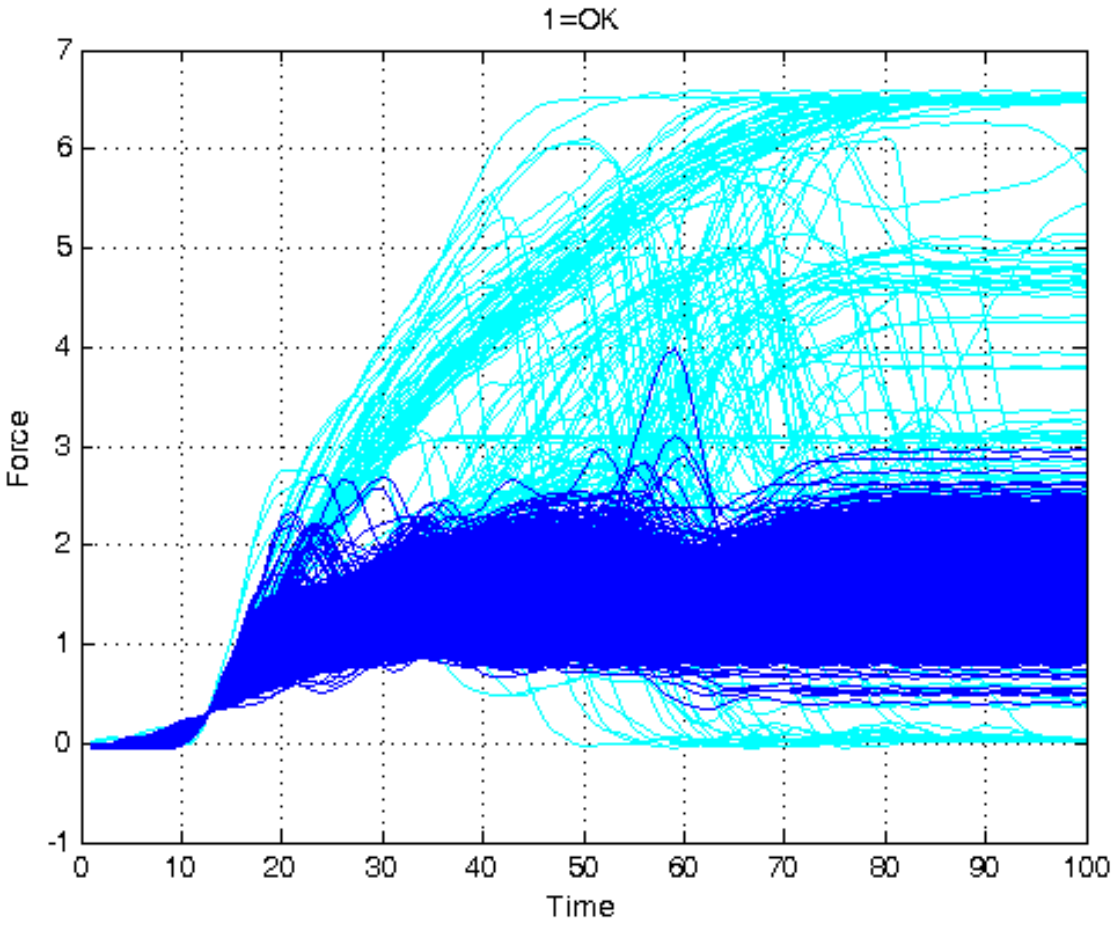


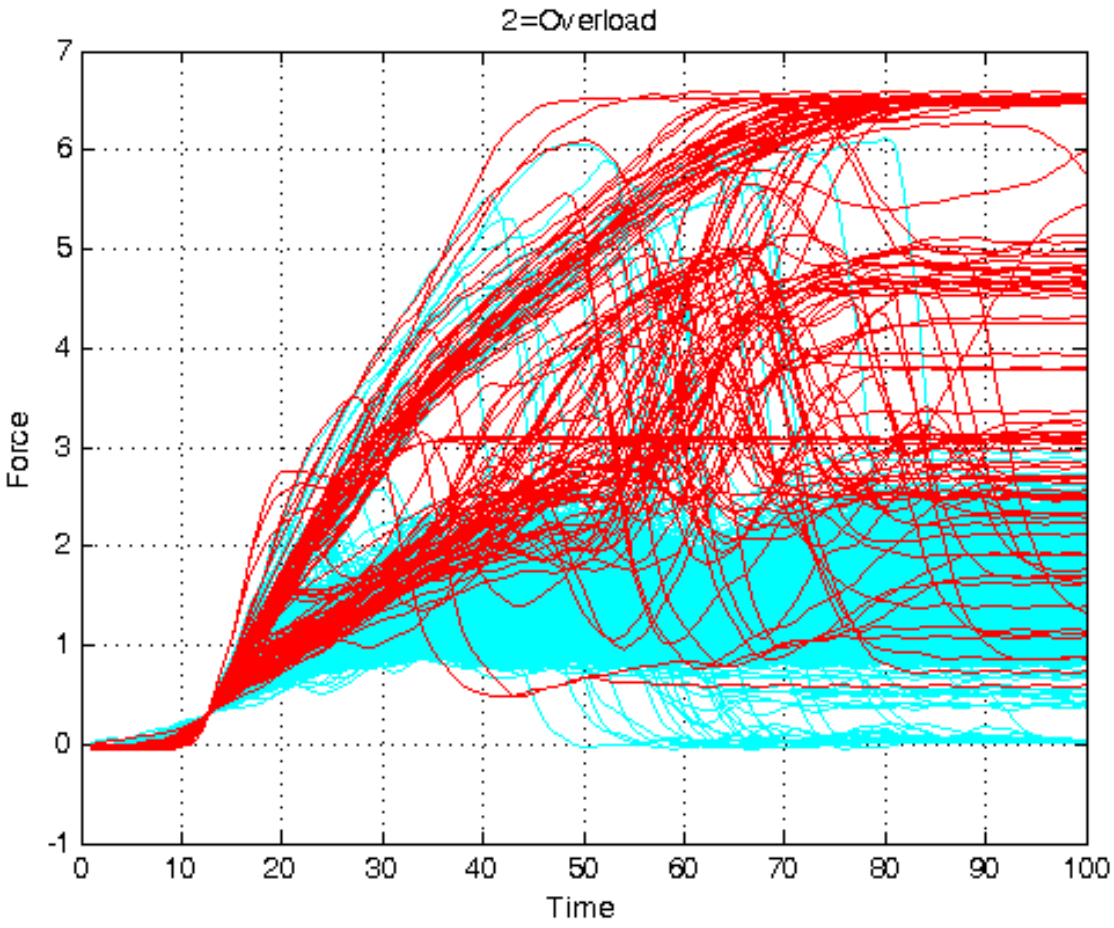
Load and plot data

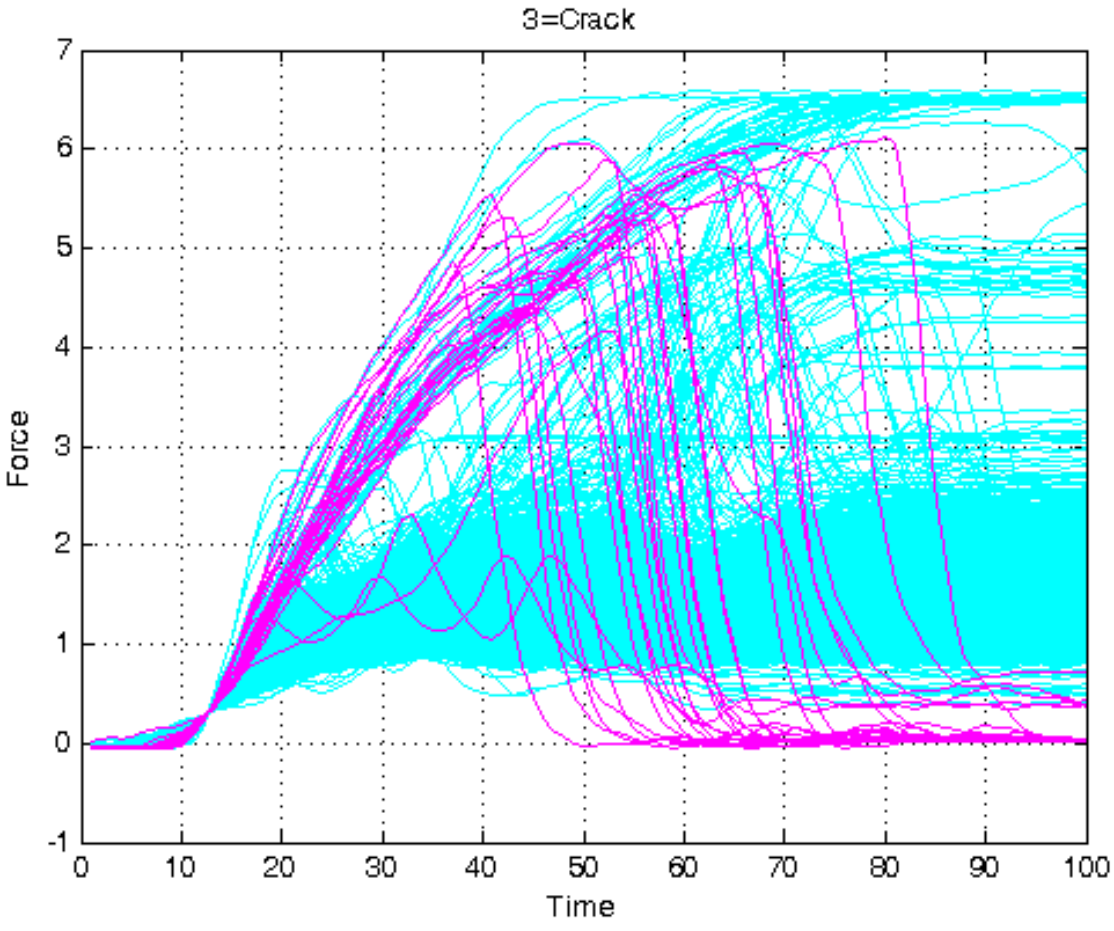
```
close all, clear all, clc, format compact

% industrial data
load data2.mat
whos

% show data for class 1: OK
figure
plot(force','c')
grid on, hold on
plot(force(find(target==1),:)','b')
xlabel('Time')
ylabel('Force')
title(notes{1})

% show data for class 2: Overload
figure
plot(force','c')
grid on, hold on
plot(force(find(target==2),:)','r')
xlabel('Time')
ylabel('Force')
title(notes{2})

% show data for class 3: Crack
figure
plot(force','c')
```

```
grid on, hold on
plot(force(find(target==3),:)','m')
xlabel('Time')
ylabel('Force')
title(notes{3})
```

```
  Name           Size                Bytes  Class     Attributes

  force       2000x100            1600000  double
  notes          1x3                  222  cell
  target      2000x1                 16000  double
```



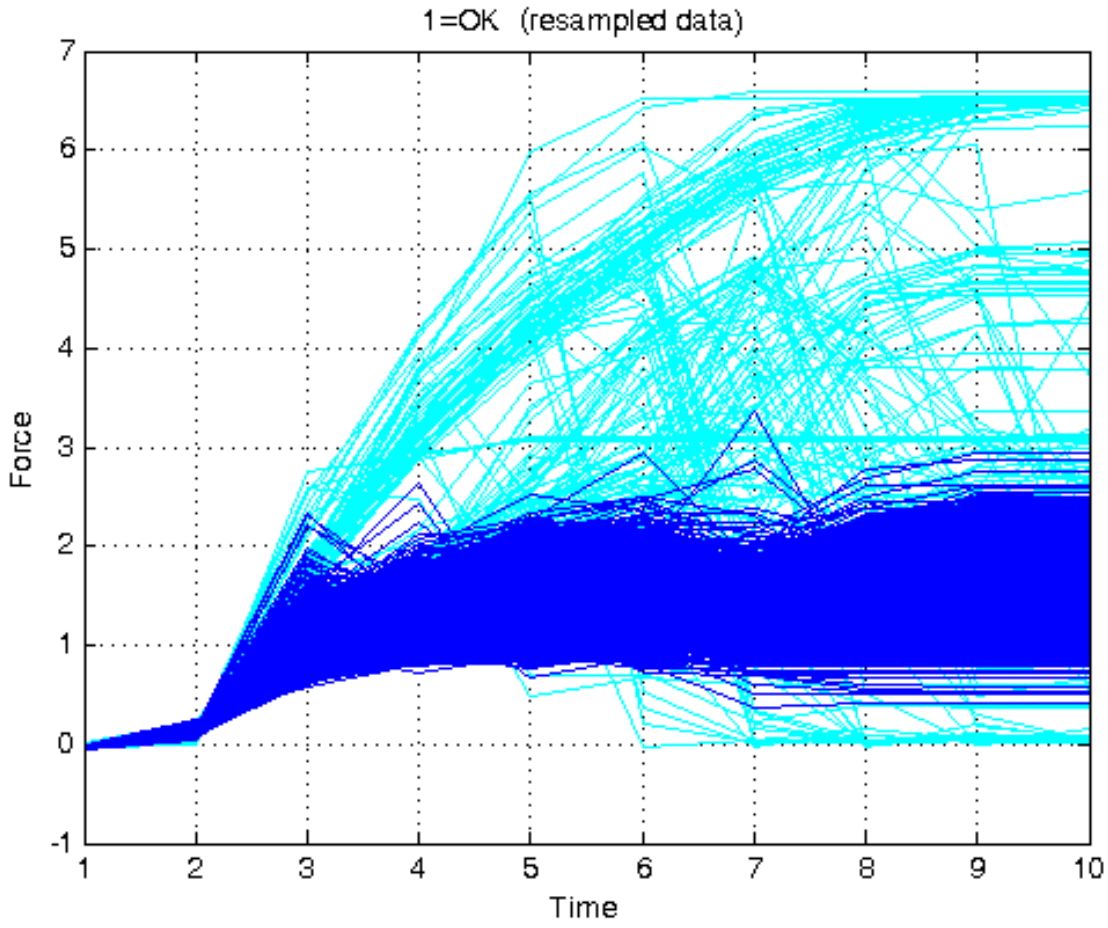


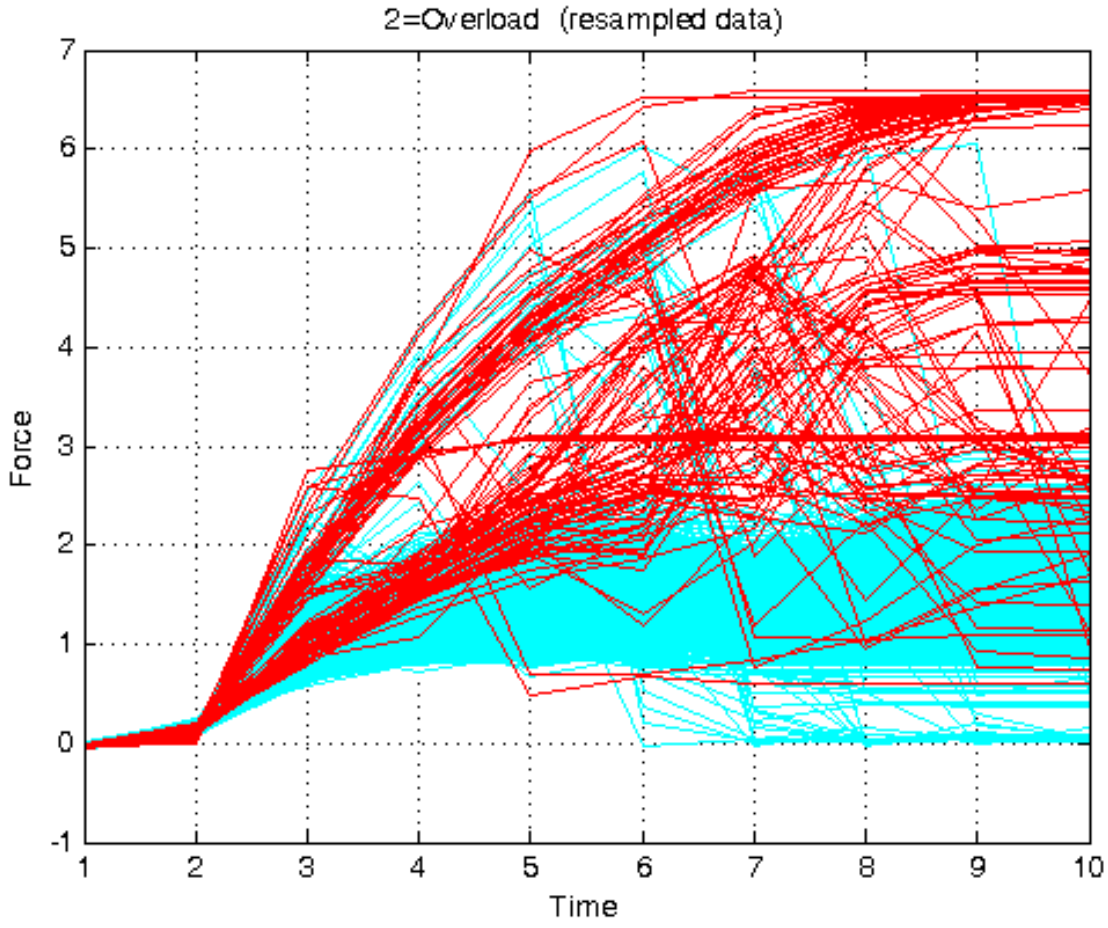


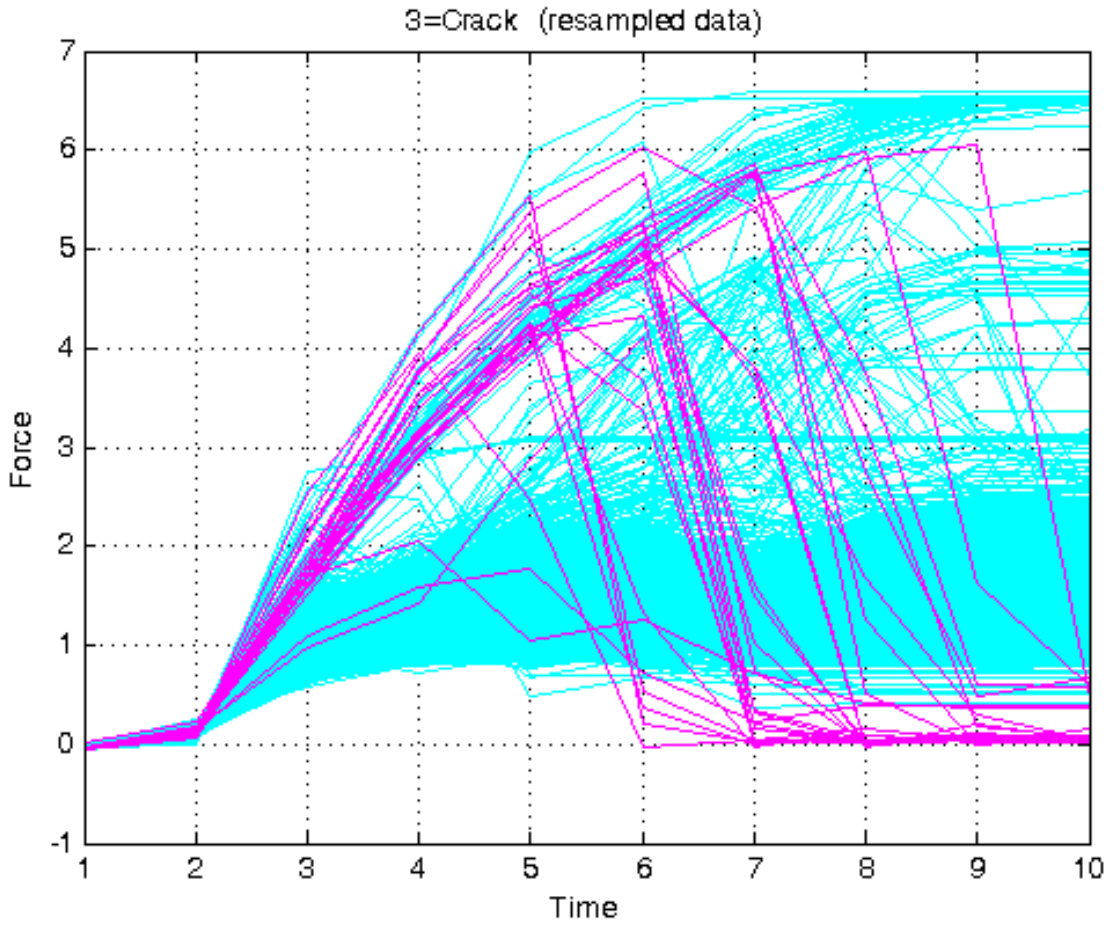
Prepare inputs: Data resampling

```
% include only every step-th data
step = 10;
force = force(:,1:step:size(force,2));
whos

% show resampled data for class 1: OK
figure
plot(force','c')
grid on, hold on
plot(force(find(target==1),:)','b')
xlabel('Time')
ylabel('Force')
title([notes{1} '  (resampled data)'])

% show resampled data for class 2: Overload
figure
plot(force','c')
grid on, hold on
plot(force(find(target==2),:)','r')
xlabel('Time')
ylabel('Force')
title([notes{2} '  (resampled data)'])

% show resampled data for class 3: Crack
figure
plot(force','c')
grid on, hold on
plot(force(find(target==3),:)','m')
xlabel('Time')
ylabel('Force')
title([notes{3} '  (resampled data)'])
```

```
  Name           Size              Bytes  Class     Attributes

  force       2000x10             160000  double
  notes          1x3                 222  cell
  step           1x1                   8  double
  target      2000x1               16000  double
```


1=OK (resampled data)
Force
Time
7
6
5
4
3
2
1
0
-1
1
2
3
4
5
6
7
8
9
10


2=Overload (resampled data)
Force
Time
7
6
5
4
3
2
1
0
-1
1
2
3
4
5
6
7
8
9
10



Define binary output coding: 0=OK, 1=Error

```
% binary coding 0/1
target = double(target > 1);
```

Create and train a multilayer perceptron

```
% create a neural network
net = feedforwardnet([4]);

% set early stopping parameters
net.divideParam.trainRatio = 0.70; % training set [%]
net.divideParam.valRatio   = 0.15; % validation set [%]
net.divideParam.testRatio  = 0.15; % test set [%]

% train a neural network
[net,tr,Y,E] = train(net,force',target');
```

Evaluate network performance

```
% digitize network response
threshold = 0.5;
Y = double(Y > threshold)';
```

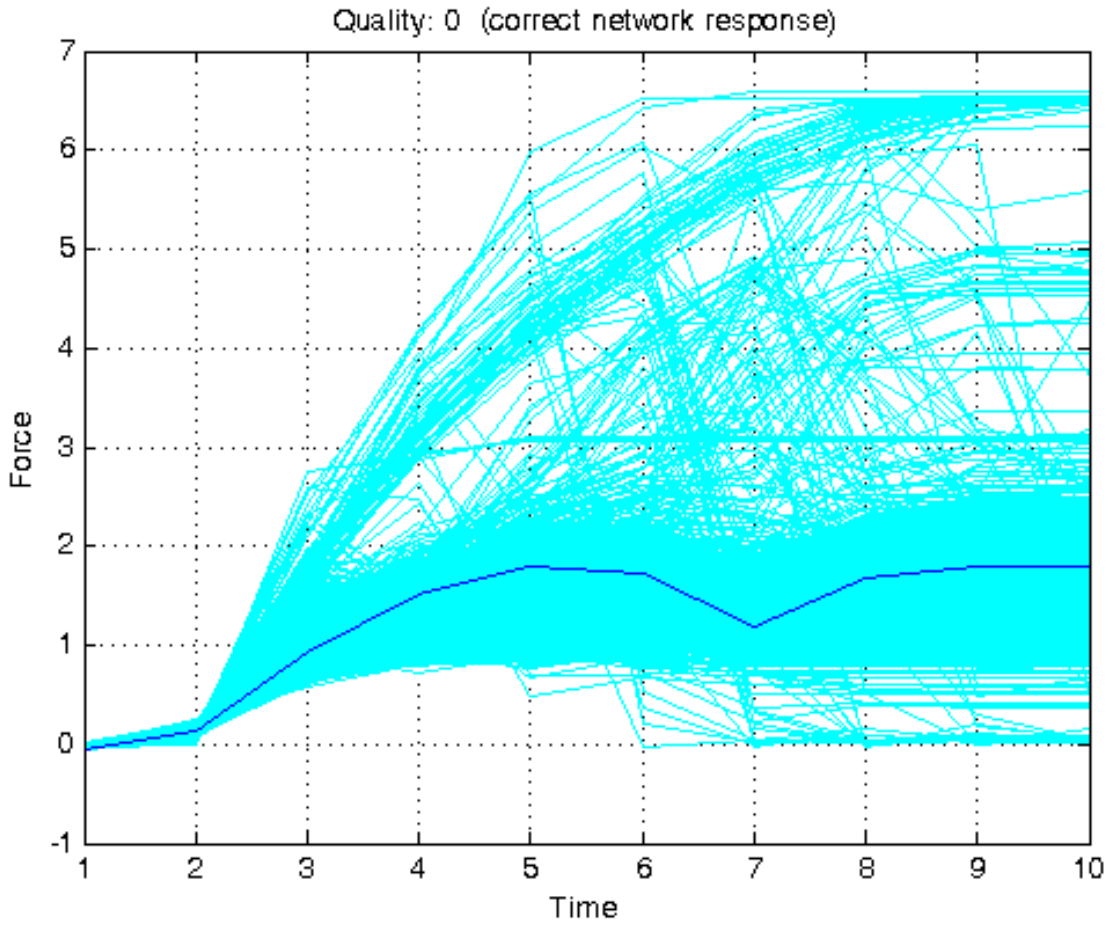
```
% find percentage of correct classifications
correct_classifications = 100*length(find(Y==target))/length(target)
```

```
correct_classifications =
   99.7500
```

Application

```
% get sample
random_index = randi(length(force))
sample = force(random_index,:);

% plot sample
figure
plot(force','c')
grid on, hold on
plot(sample,'b')
xlabel('Time')
ylabel('Force')

% predict quality
q = net(sample');
% digitize network response
q = double(q > threshold)'

% comment network respons
if q==target(random_index)
  title(sprintf('Quality: %d  (correct network response)',q))
else
  title(sprintf('Quality: %d  (wrong network response)',q))
end
```

```
random_index =
        1881
q =
     0
```



Published with MATLAB® 7.14

Prediction of chaotic time series with NAR neural network

Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: Design a neural network for the recursive prediction of chaotic Mackay-Glass time series, try various network architectures and experiment with various delays.

Contents

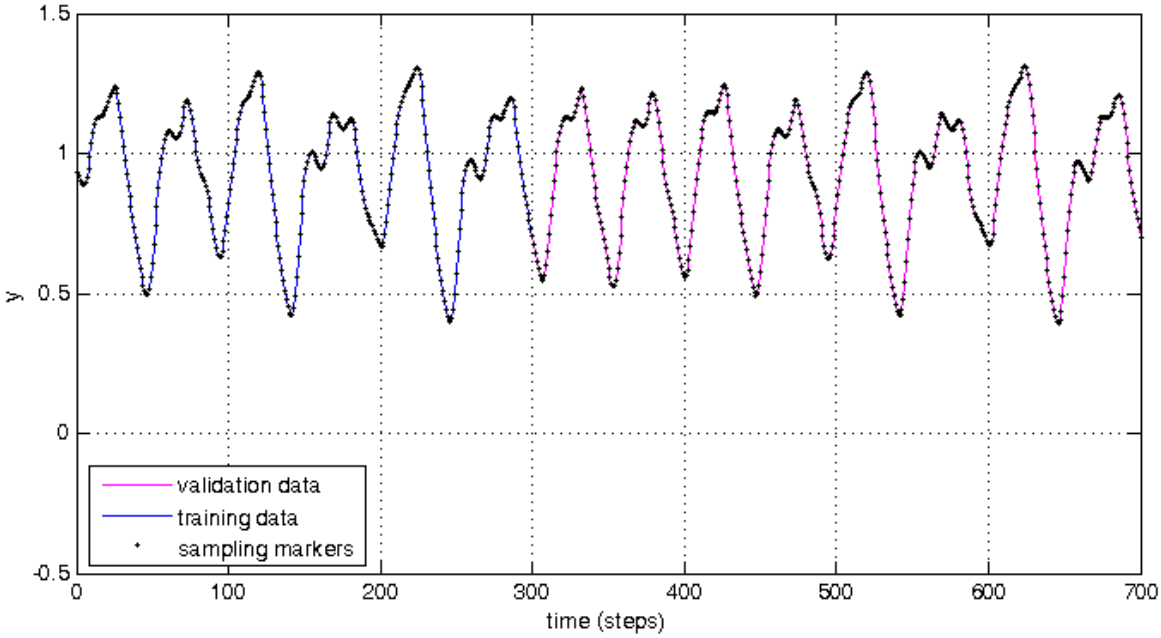
- Generate data (Mackay-Glass time series)
- Define nonlinear autoregressive neural network
- Prepare input and target time series data for network training
- Train net
- Transform network into a closed-loop NAR network
- Recursive prediction on validation data

Generate data (Mackay-Glass time series)

```
close all, clear all, clc, format compact

% data settings
N  = 700; % number of samples
Nu = 300; % number of learning samples

% Mackay-Glass time series
```

$$\dot{y}(t) = -by(t) + \frac{cy(t-\tau)}{1+y^{10}(t-\tau)}$$

```
b   = 0.1;
c   = 0.2;
tau = 17;

% initialization
y = [0.9697 0.9699 0.9794 1.0003 1.0319 1.0703 1.1076 1.1352 1.1485 ...
     1.1482 1.1383 1.1234 1.1072 1.0928 1.0820 1.0756 1.0739 1.0759]';
% generate Mackay-Glass time series
for n=18:N+99
  y(n+1) = y(n) - b*y(n) + c*y(n-tau)/(1+y(n-tau).^10);
end
% remove initial values
y(1:100) = [];

% plot training and validation data
plot(y,'m-')
grid on, hold on
plot(y(1:Nu),'b')
plot(y,'+k','markersize',2)
legend('validation data','training data','sampling markers','location','southwest')
xlabel('time (steps)')
ylabel('y')
ylim([-.5 1.5])
set(gcf,'position',[1 60 800 400])

% prepare training data
yt = con2seq(y(1:Nu)');

% prepare validation data
yv = con2seq(y(Nu+1:end)');
```



Define nonlinear autoregressive neural network

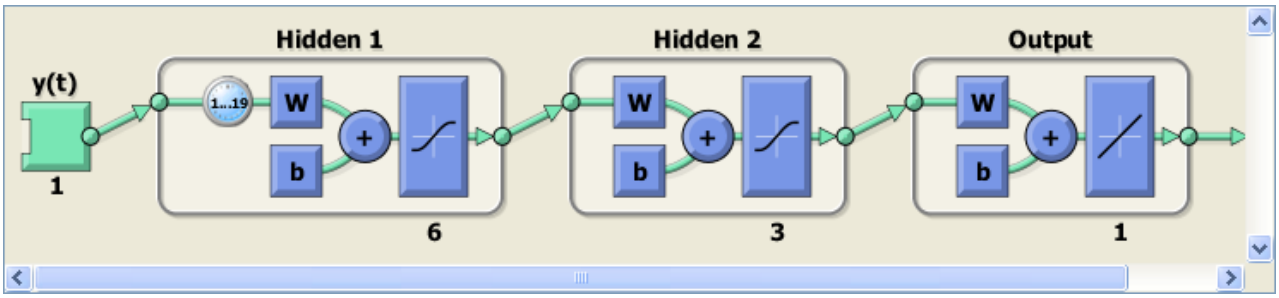
```
%---------- network parameters -------------
% good parameters (you don't know 'tau' for unknown process)
inputDelays = 1:6:19;  % input delay vector
hiddenSizes = [6 3];   % network structure (number of neurons)
%-------------------------------------

% nonlinear autoregressive neural network
net = narnet(inputDelays, hiddenSizes);
```

Prepare input and target time series data for network training

```
% [Xs,Xi,Ai,Ts,EWs,shift] = preparets(net,Xnf,Tnf,Tf,EW)
%
% This function simplifies the normally complex and error prone task of
% reformatting input and target timeseries. It automatically shifts input
% and target time series as many steps as are needed to fill the initial
% input and layer delay states. If the network has open loop feedback,
% then it copies feedback targets into the inputs as needed to define the
% open loop inputs.
%
%  net : Neural network
%  Xnf : Non-feedback inputs
%  Tnf : Non-feedback targets
%   Tf : Feedback targets
%   EW : Error weights (default = {1})
%
%   Xs : Shifted inputs
%   Xi : Initial input delay states
%   Ai : Initial layer delay states
%   Ts : Shifted targets
[Xs,Xi,Ai,Ts] = preparets(net,{},{},yt);
```

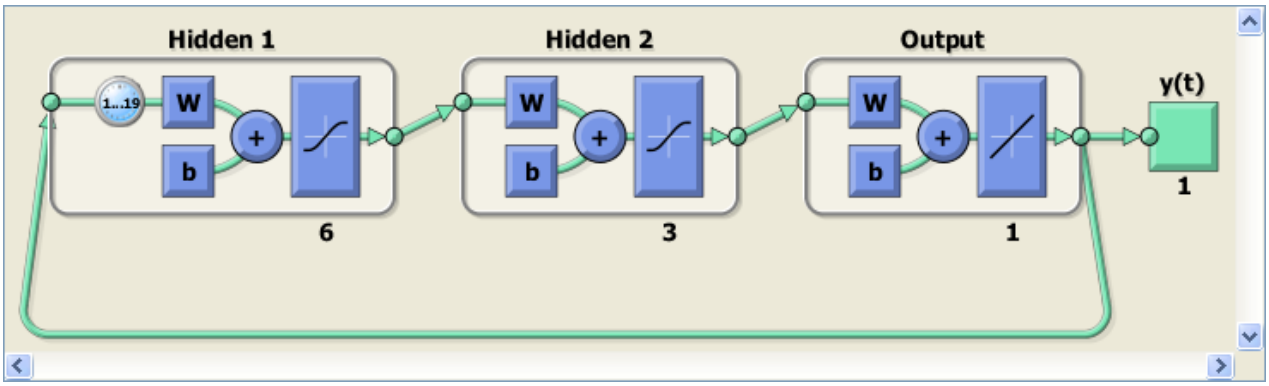
Train net

```
% train net with prepared training data
net = train(net,Xs,Ts,Xi,Ai);
% view trained net
view(net)
```



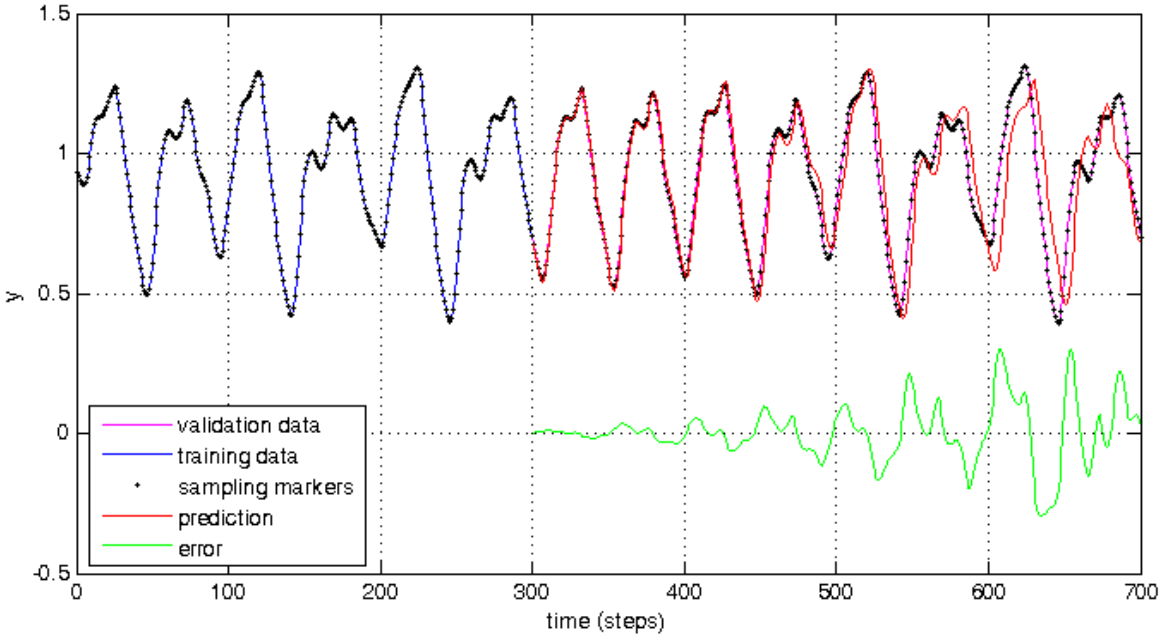
Transform network into a closed-loop NAR network

```
% close feedback for recursive prediction
net = closeloop(net);
% view closeloop version of a net
view(net);
```



Recursive prediction on validation data

```
% prepare validation data for network simulation
yini = yt(end-max(inputDelays)+1:end); % initial values from training data
% combine initial values and validation data 'yv'
[Xs,Xi,Ai] = preparets(net,{},{},[yini yv]);

% predict on validation data
predict = net(Xs,Xi,Ai);

% validation data
Yv = cell2mat(yv);
% prediction
Yp = cell2mat(predict);
% error
e = Yv - Yp;

% plot results of recursive simulation
figure(1)
plot(Nu+1:N,Yp,'r')
plot(Nu+1:N,e,'g')
legend('validation data','training data','sampling markers',...
  'prediction','error','location','southwest')
```



Published with MATLAB® 7.14

Function approximation with RBFN

Neural Networks course (practical examples) © 2012 Primoz Potocnik

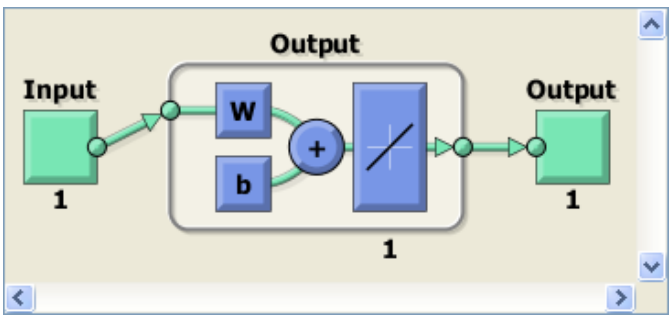
PROBLEM DESCRIPTION: Create a function approximation model based on a measured data set. Apply various Neural Network architectures based on Radial Basis Functions. Compare with Multilayer perceptron and Linear regression models.

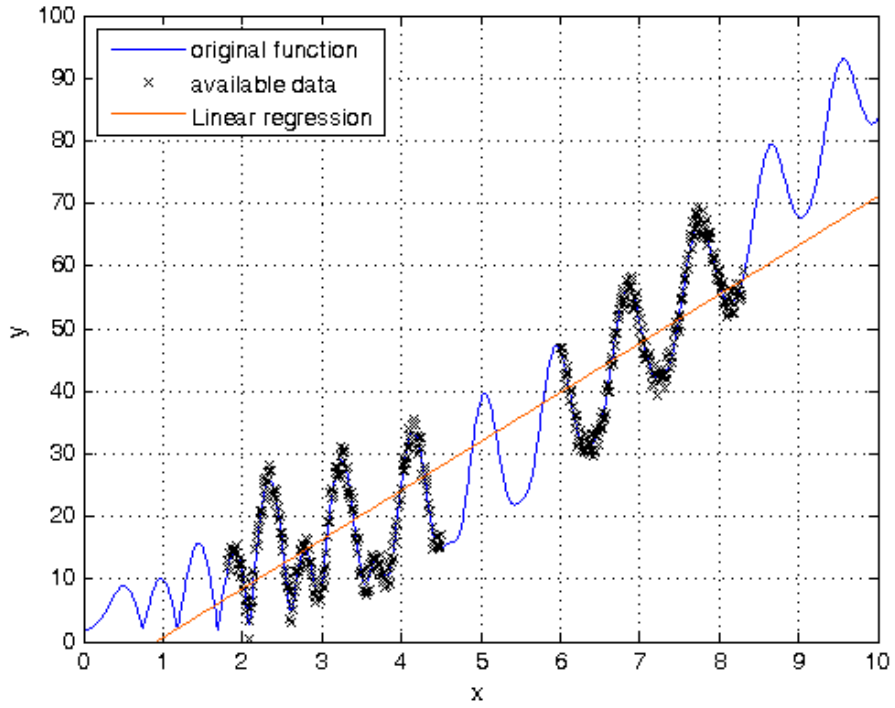
Contents

- Linear Regression
- Exact RBFN
- RBFN
- GRNN
- RBFN trained by Bayesian regularization
- MLP
- Data generator

Linear Regression

```
close all, clear all, clc, format compact

% generate data
[X,Xtrain,Ytrain,fig] = data_generator();

%---------------------------------
% no hidden layers
net = feedforwardnet([]);
% % one hidden layer with linear transfer functions
% net = feedforwardnet([10]);
% net.layers{1}.transferFcn = 'purelin';

% set early stopping parameters
net.divideParam.trainRatio = 1.0; % training set [%]
net.divideParam.valRatio   = 0.0; % validation set [%]
net.divideParam.testRatio  = 0.0; % test set [%]
% train a neural network
net.trainParam.epochs = 200;
net = train(net,Xtrain,Ytrain);
%---------------------------------

% view net
view (net)
% simulate a network over complete input range
Y = net(X);
% plot network response
figure(fig)
plot(X,Y,'color',[1 .4 0])
legend('original function','available data','Linear regression','location','northwest')
```

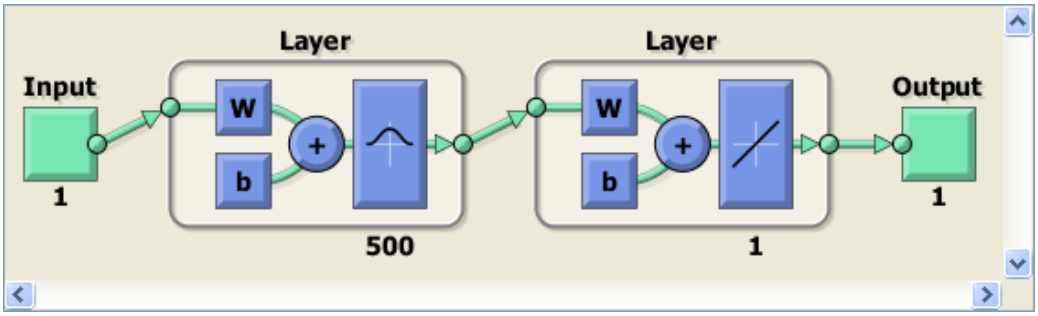


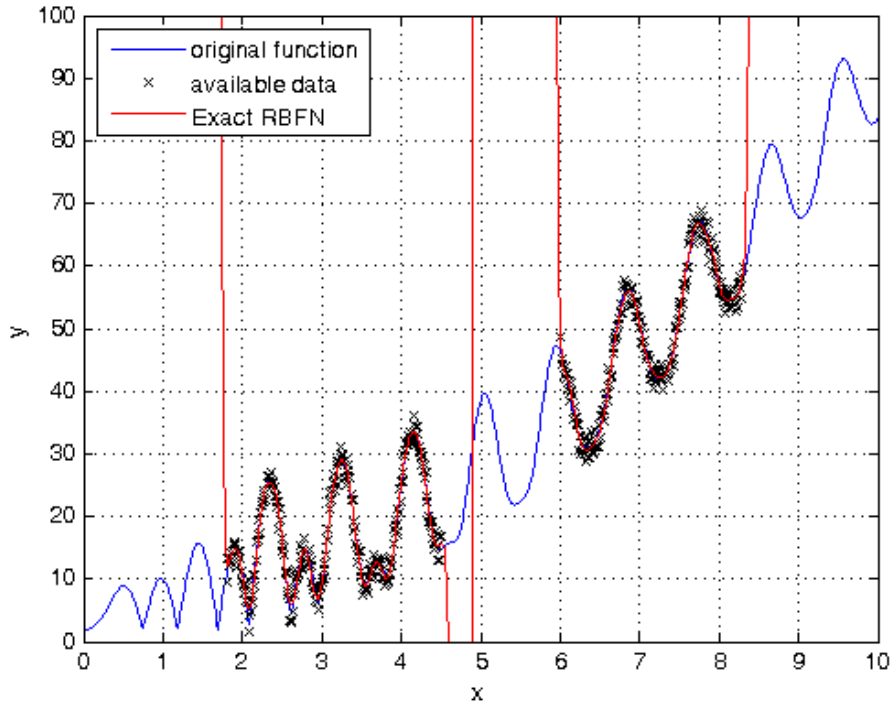


Exact RBFN

```
% generate data
[X,Xtrain,Ytrain,fig] = data_generator();

%---------------------------------
% choose a spread constant
spread = .4;
% create a neural network
net = newrbe(Xtrain,Ytrain,spread);
%---------------------------------

% view net
view (net)
% simulate a network over complete input range
Y = net(X);
% plot network response
figure(fig)
plot(X,Y,'r')
legend('original function','available data','Exact RBFN','location','northwest')
```

```
Warning: Rank deficient, rank = 53, tol =  1.110223e-13.
```

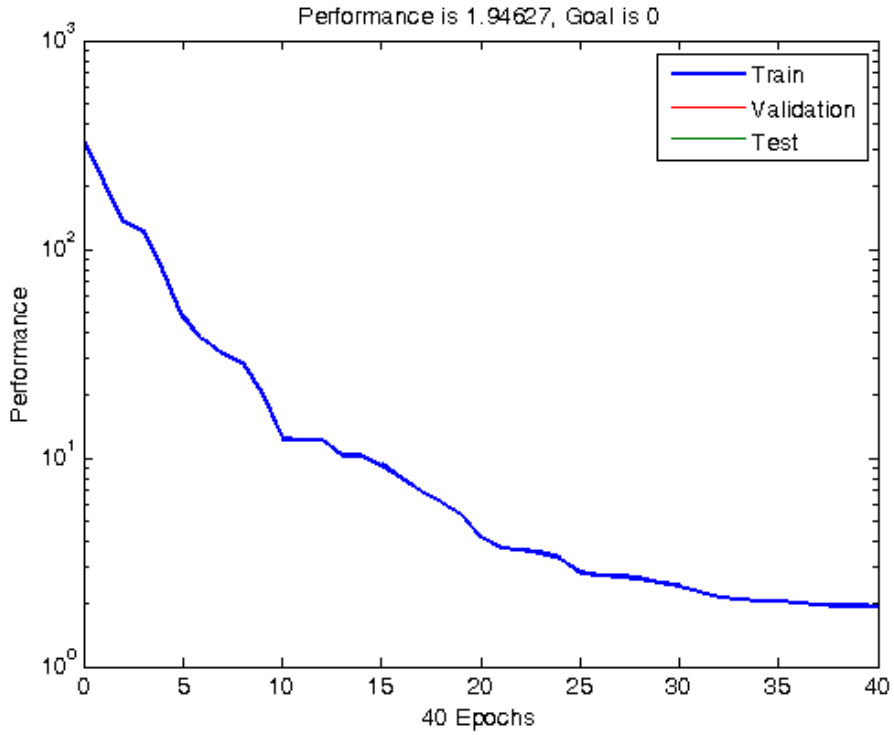


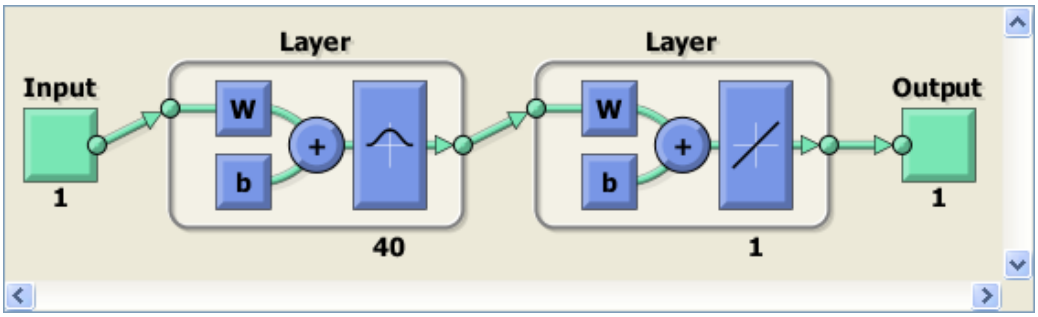


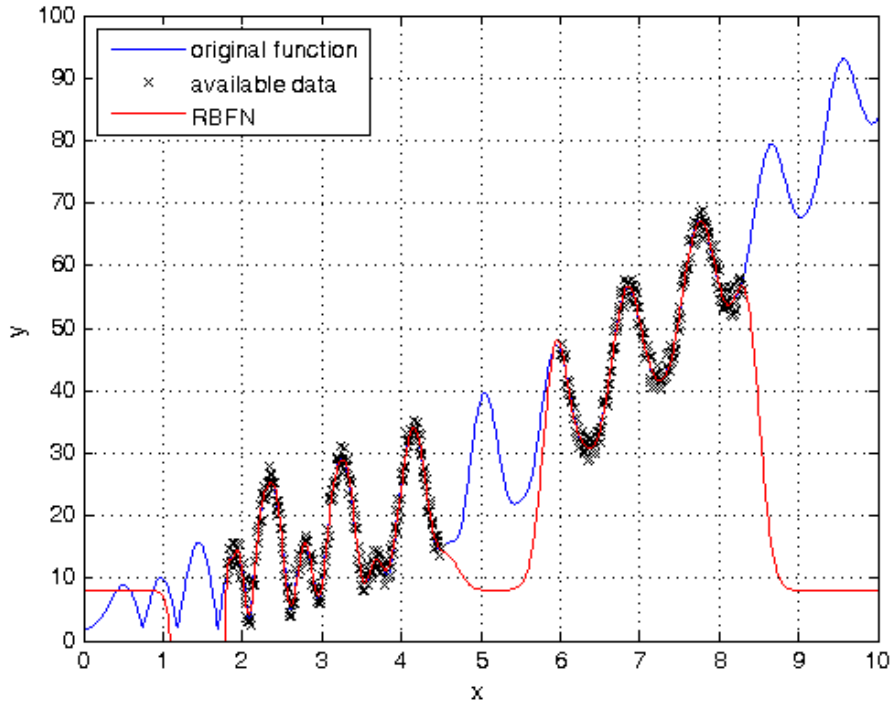
RBFN

```
% generate data
[X,Xtrain,Ytrain,fig] = data_generator();

%---------------------------------
% choose a spread constant
spread = .2;
% choose max number of neurons
K = 40;
% performance goal (SSE)
goal = 0;
% number of neurons to add between displays
Ki = 5;
% create a neural network
net = newrb(Xtrain,Ytrain,goal,spread,K,Ki);
%---------------------------------

% view net
view (net)
% simulate a network over complete input range
Y = net(X);
% plot network response
figure(fig)
plot(X,Y,'r')
legend('original function','available data','RBFN','location','northwest')
```

```
NEWRB, neurons = 0, MSE = 333.938
NEWRB, neurons = 5, MSE = 47.271
NEWRB, neurons = 10, MSE = 12.3371
NEWRB, neurons = 15, MSE = 9.26908
NEWRB, neurons = 20, MSE = 4.16992
NEWRB, neurons = 25, MSE = 2.82444
NEWRB, neurons = 30, MSE = 2.43353
NEWRB, neurons = 35, MSE = 2.06149
NEWRB, neurons = 40, MSE = 1.94627
```



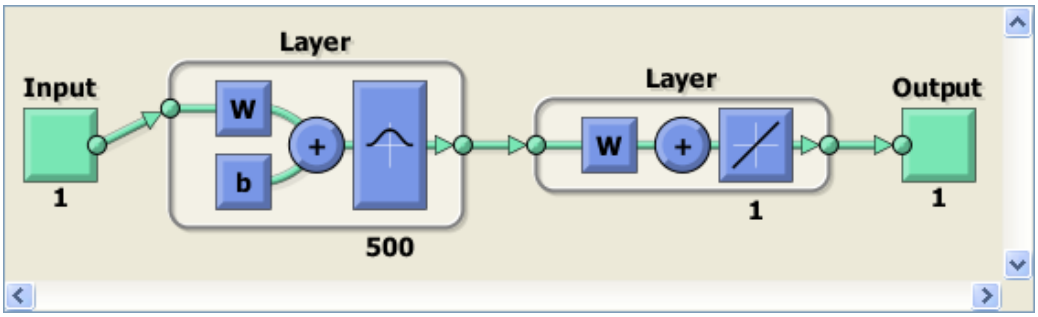


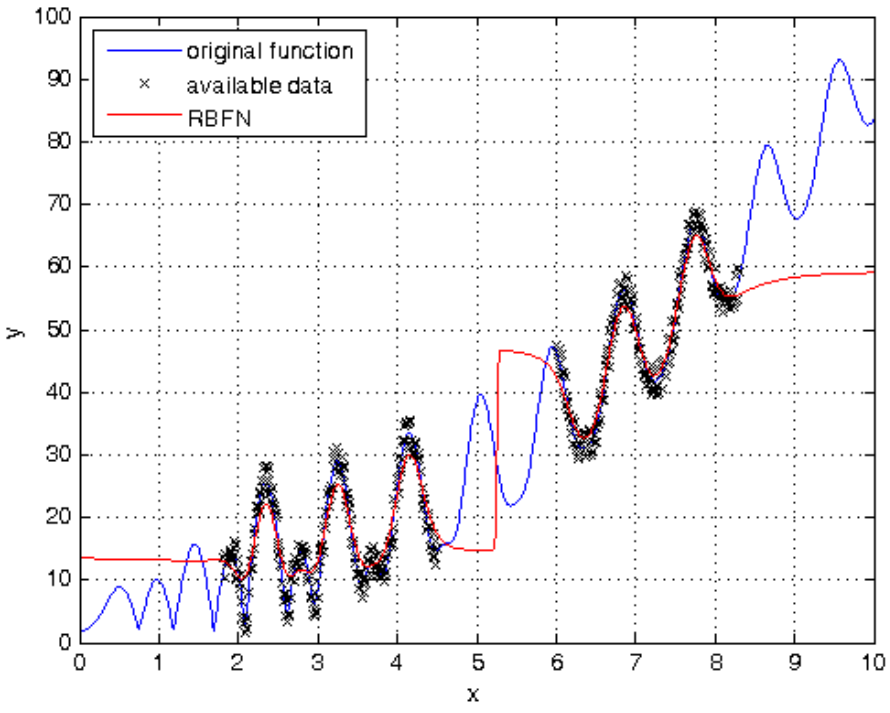


GRNN

```
% generate data
[X,Xtrain,Ytrain,fig] = data_generator();

%---------------------------------
% choose a spread constant
```

```
spread = .12;
% create a neural network
net = newgrnn(Xtrain,Ytrain,spread);
%---------------------------------

% view net
view (net)
% simulate a network over complete input range
Y = net(X);
% plot network response
figure(fig)
plot(X,Y,'r')
legend('original function','available data','RBFN','location','northwest')
```



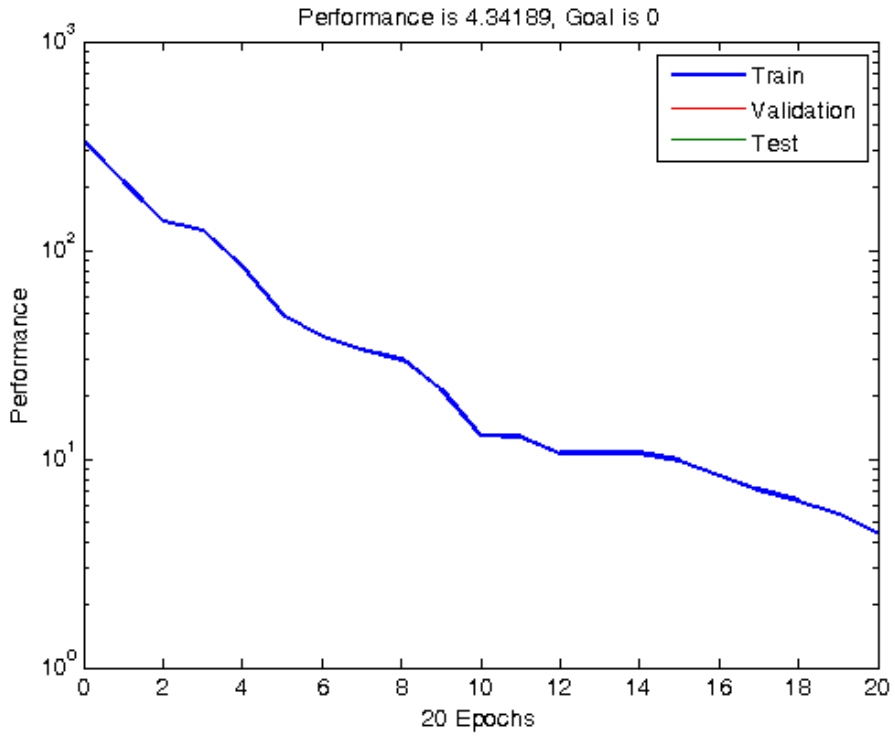


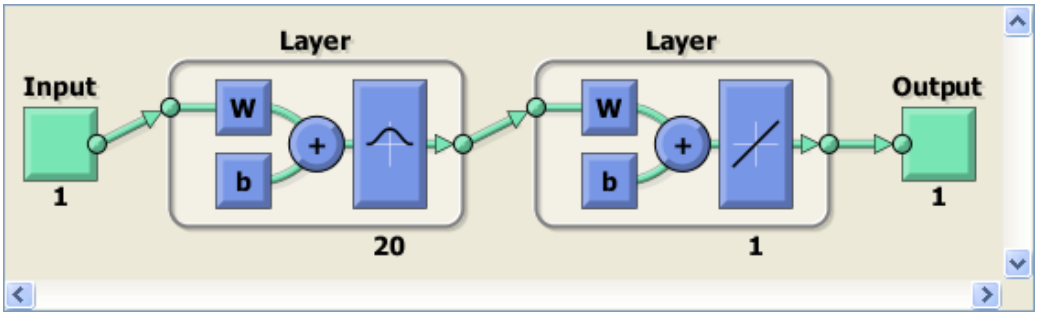
RBFN trained by Bayesian regularization

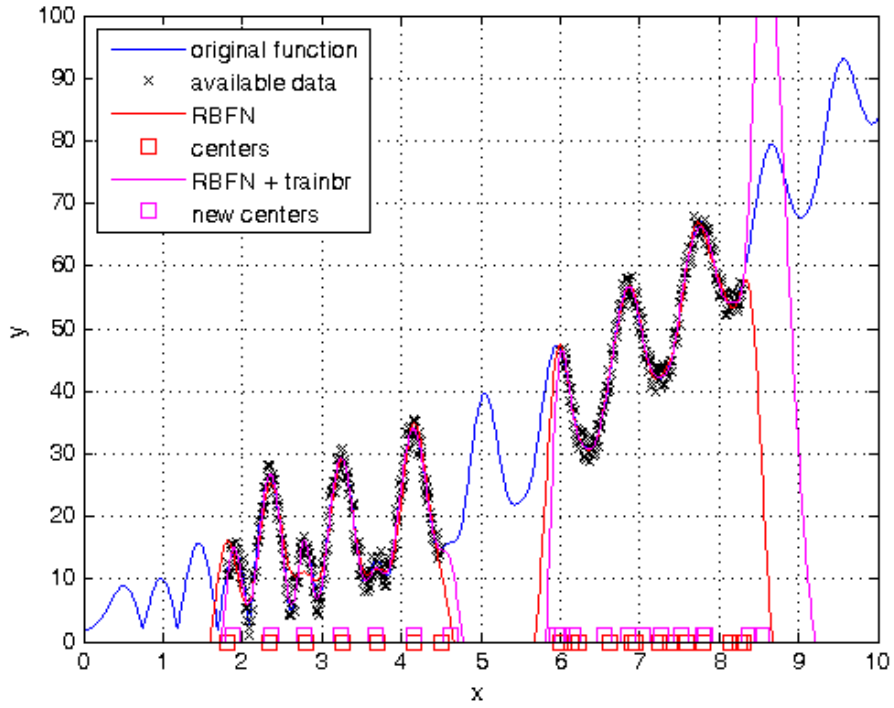
```
% generate data
[X,Xtrain,Ytrain,fig] = data_generator();

%--------- RBFN ------------------
% choose a spread constant
spread = .2;
% choose max number of neurons
K = 20;
% performance goal (SSE)
goal = 0;
% number of neurons to add between displays
Ki = 20;
% create a neural network
net = newrb(Xtrain,Ytrain,goal,spread,K,Ki);
%---------------------------------
```

```
% view net
view (net)
% simulate a network over complete input range
Y = net(X);
% plot network response
figure(fig)
plot(X,Y,'r')
% Show RBFN centers
c = net.iw{1};
plot(c,zeros(size(c)),'rs')
legend('original function','available data','RBFN','centers','location','northwest')

%--------- trainbr ---------------
% Retrain a RBFN using Bayesian regularization backpropagation
net.trainFcn='trainbr';
net.trainParam.epochs = 100;
% perform Levenberg-Marquardt training with Bayesian regularization
net = train(net,Xtrain,Ytrain);
%---------------------------------

% simulate a network over complete input range
Y = net(X);
% plot network response
figure(fig)
plot(X,Y,'m')
% Show RBFN centers
c = net.iw{1};
plot(c,ones(size(c)),'ms')
legend('original function','available data','RBFN','centers','RBFN + trainbr','new
centers','location','northwest')
```

```
NEWRB, neurons = 0, MSE = 334.852
NEWRB, neurons = 20, MSE = 4.34189
```

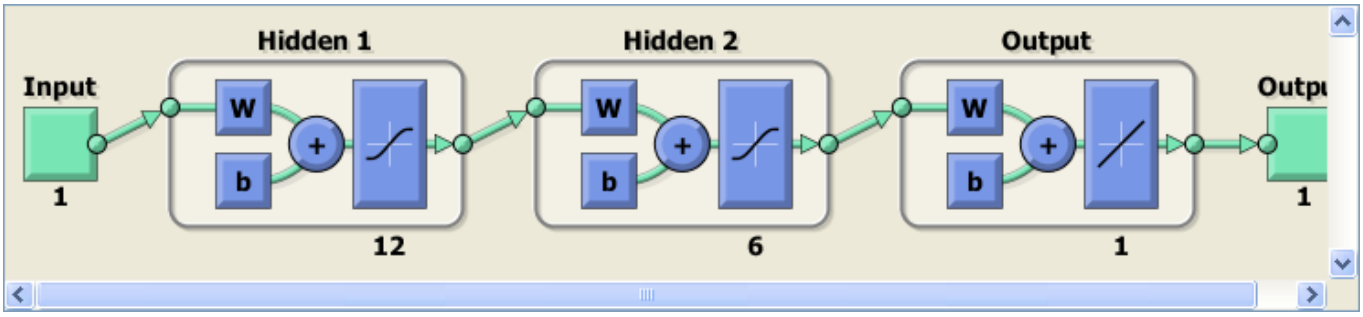


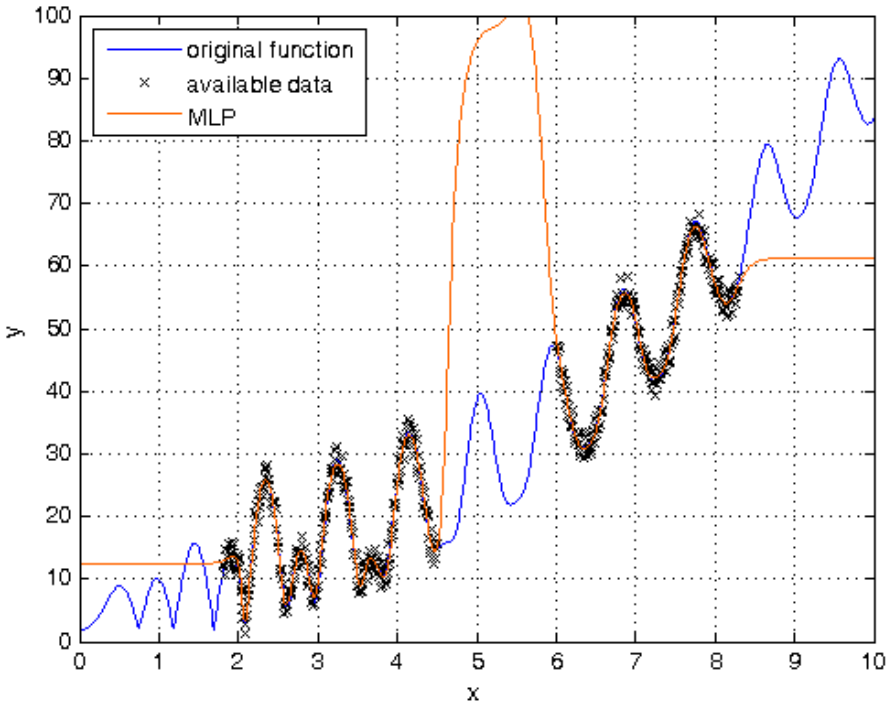




MLP

```
% generate data
[X,Xtrain,Ytrain,fig] = data_generator();

%---------------------------------
% create a neural network
net = feedforwardnet([12 6]);
% set early stopping parameters
net.divideParam.trainRatio = 1.0; % training set [%]
net.divideParam.valRatio   = 0.0; % validation set [%]
net.divideParam.testRatio  = 0.0; % test set [%]
% train a neural network
net.trainParam.epochs = 200;
net = train(net,Xtrain,Ytrain);
%---------------------------------

% view net
view (net)
% simulate a network over complete input range
Y = net(X);
% plot network response
figure(fig)
plot(X,Y,'color',[1 .4 0])
legend('original function','available data','MLP','location','northwest')
```





Data generator

```
type data_generator
```

```
%% Data generator function
function [X,Xtrain,Ytrain,fig] = data_generator()

% data generator
X = 0.01:.01:10;
f = abs(besselj(2,X*7).*asind(X/2) + (X.^1.95)) + 2;
fig = figure;
plot(X,f,'b-')
hold on
grid on

% available data points
Ytrain = f + 5*(rand(1,length(f))-.5);
Xtrain = X([181:450 601:830]);
Ytrain = Ytrain([181:450 601:830]);
plot(Xtrain,Ytrain,'kx')
xlabel('x')
ylabel('y')
ylim([0 100])
legend('original function','available data','location','northwest')
```

Published with MATLAB® 7.14

Radial Basis Function Networks for Classification of XOR problem

Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: 4 clusters of data (A,B,C,D) are defined in a 2-dimensional input space. (A,C) and (B,D) clusters represent XOR classification problem. The task is to define a neural network for solving the XOR problem.

Contents

1. Classification of XOR problem with an exact RBFN
2. Classification of XOR problem with a RBFN
3. Classification of XOR problem with a PNN
4. Classification of XOR problem with a GRNN
5. Bayesian regularization for RBFN

Classification of XOR problem with an exact RBFN

Neural Networks course (practical examples) © 2012 Primoz Potocnik

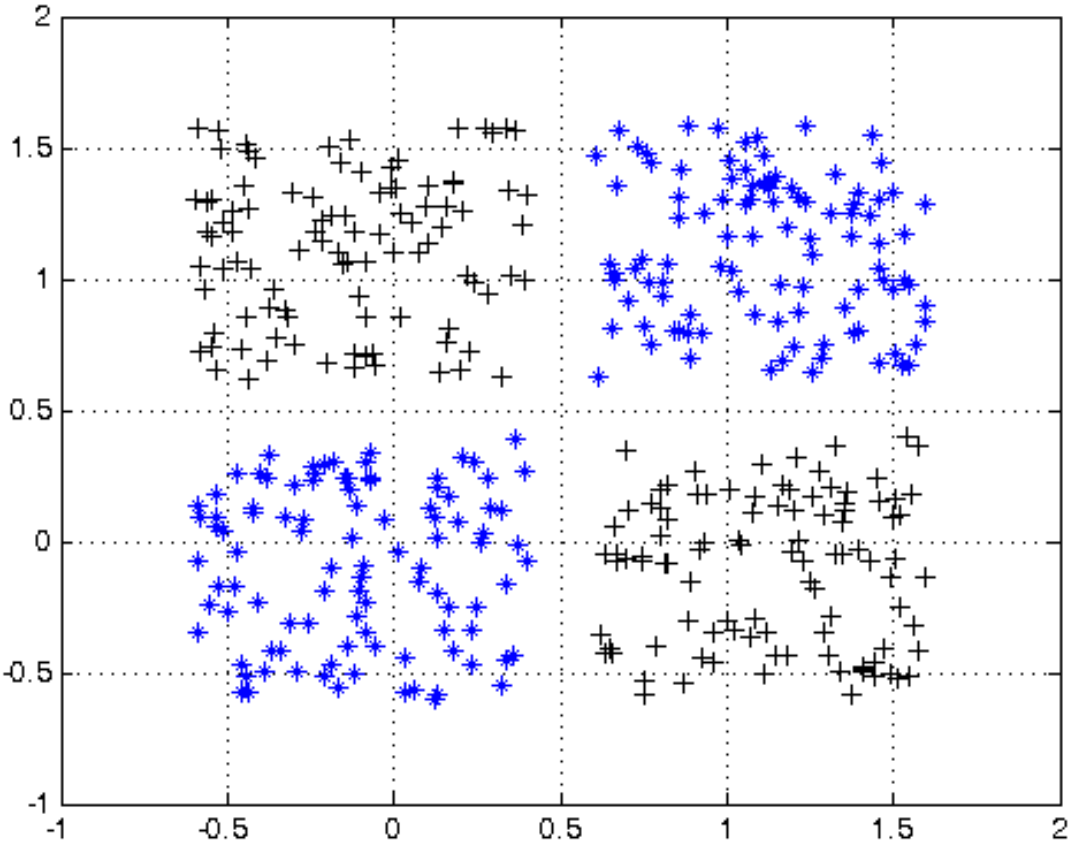
PROBLEM DESCRIPTION: 2 groups of linearly inseparable data (A,B) are defined in a 2-dimensional input space. The task is to define a neural network for solving the XOR classification problem.

Contents

- Create input data
- Define output coding
- Prepare inputs & outputs for network training
- Create an exact RBFN
- Evaluate network performance
- Plot classification result
- Plot RBFN centers

Create input data

```
close all, clear all, clc, format compact

% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A = [rand(1,K)-q rand(1,K)+q;
     rand(1,K)+q rand(1,K)-q];
B = [rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)-q];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```



Define output coding

```
% coding (+1/-1) for 2-class XOR problem
a = -1;
b =  1;
```

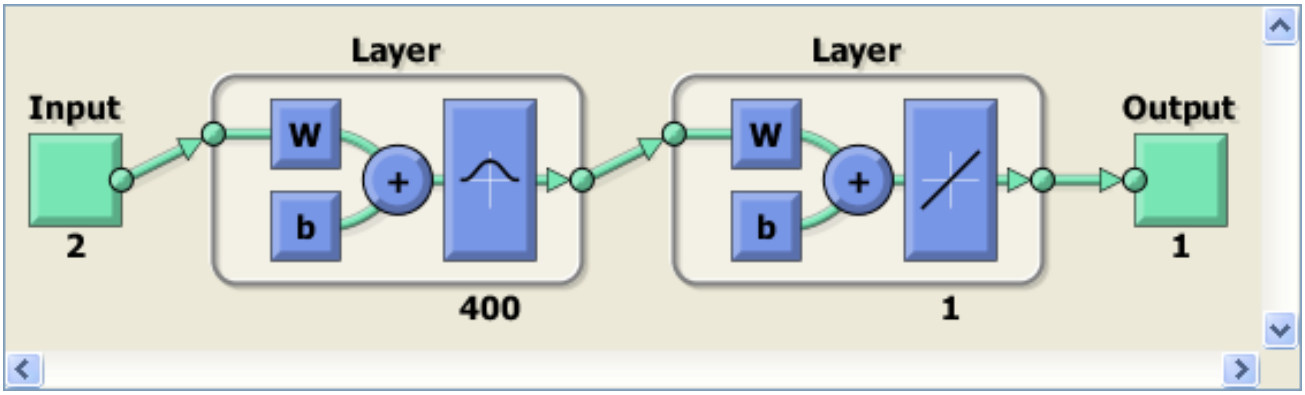
Prepare inputs & outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

Create an exact RBFN

```
% choose a spread constant
spread = 1;
% create a neural network
net    = newrbe(P,T,spread);

% view network
view(net)
```

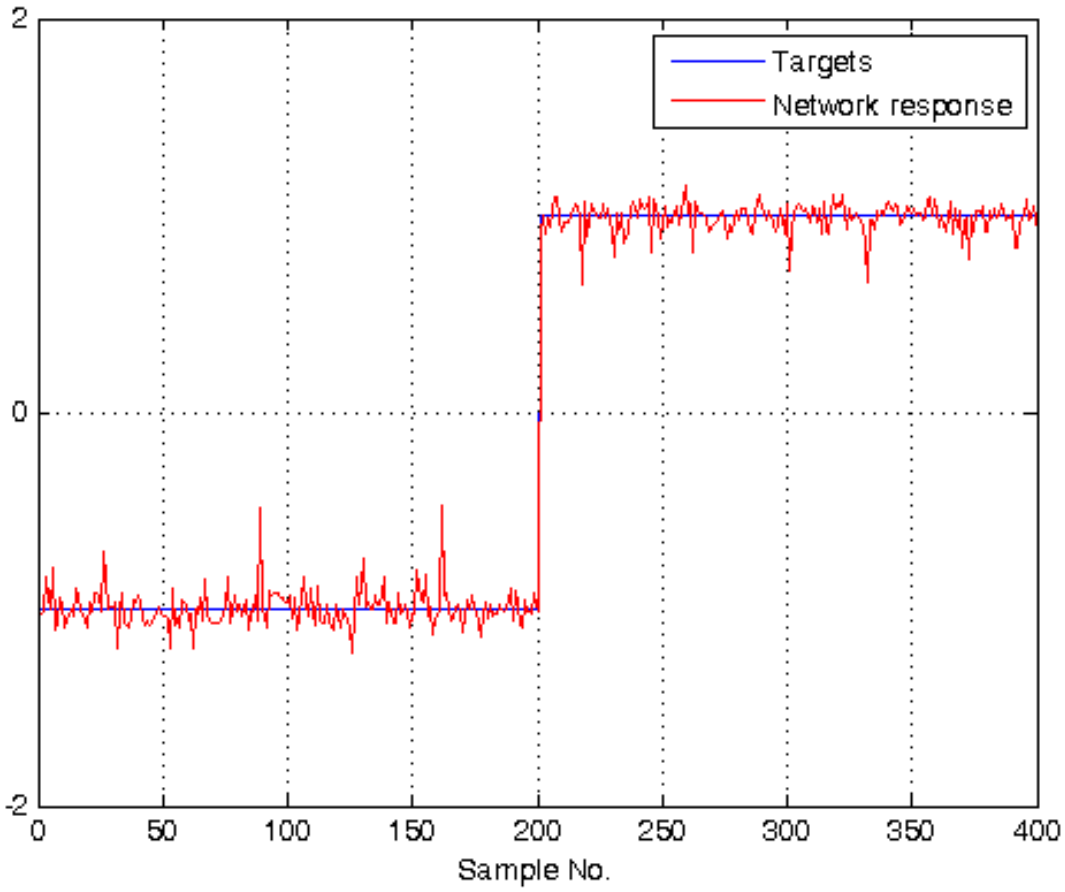
```
Warning: Rank deficient, rank = 124, tol =  8.881784e-14.
```



Evaluate network performance

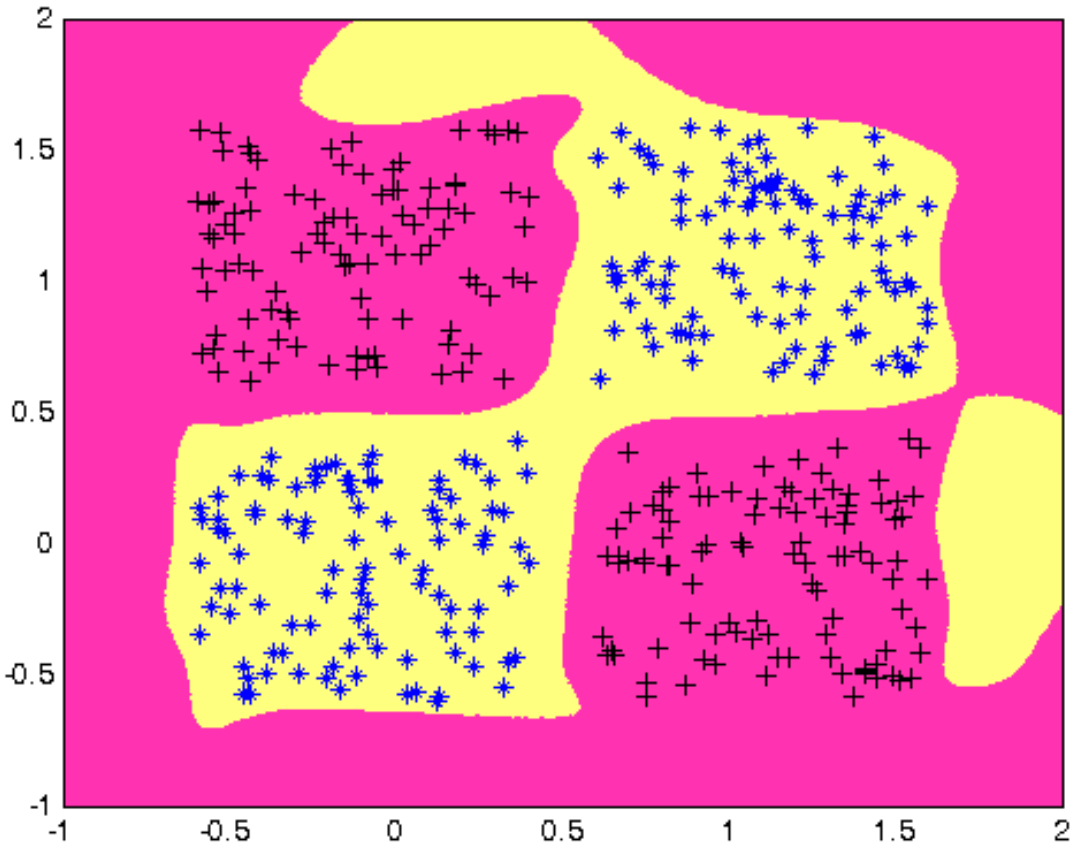
```
% simulate a network on training data
Y = net(P);

% calculate [%] of correct classifications
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('\nSpread          = %.2f\n',spread)
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)

% plot targets and network response
figure;
plot(T')
hold on
grid on
plot(Y','r')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
legend('Targets','Network response')
xlabel('Sample No.')
```

```
Spread          = 1.00
Num of neurons  = 400
Correct class   = 100.00 %
```



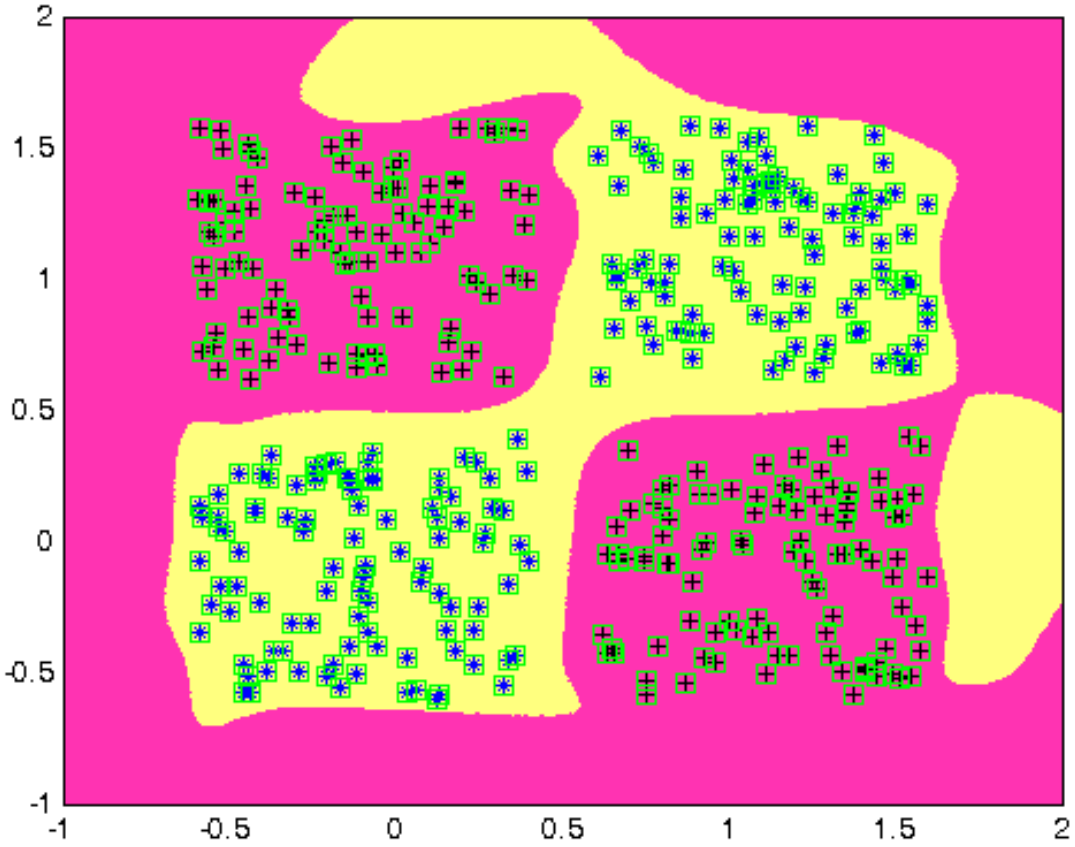
Plot classification result

```
% generate a grid
span    = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp      = [P1(:) P2(:)]';
% simualte neural network on a grid
aa      = sim(net,pp);

% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
```



Plot RBFN centers

```
plot(net.iw{1}(:,1),net.iw{1}(:,2),'gs')
```



Published with MATLAB® 7.14

Classification of XOR problem with a RBFN

Neural Networks course (practical examples) © 2012 Primoz Potocnik

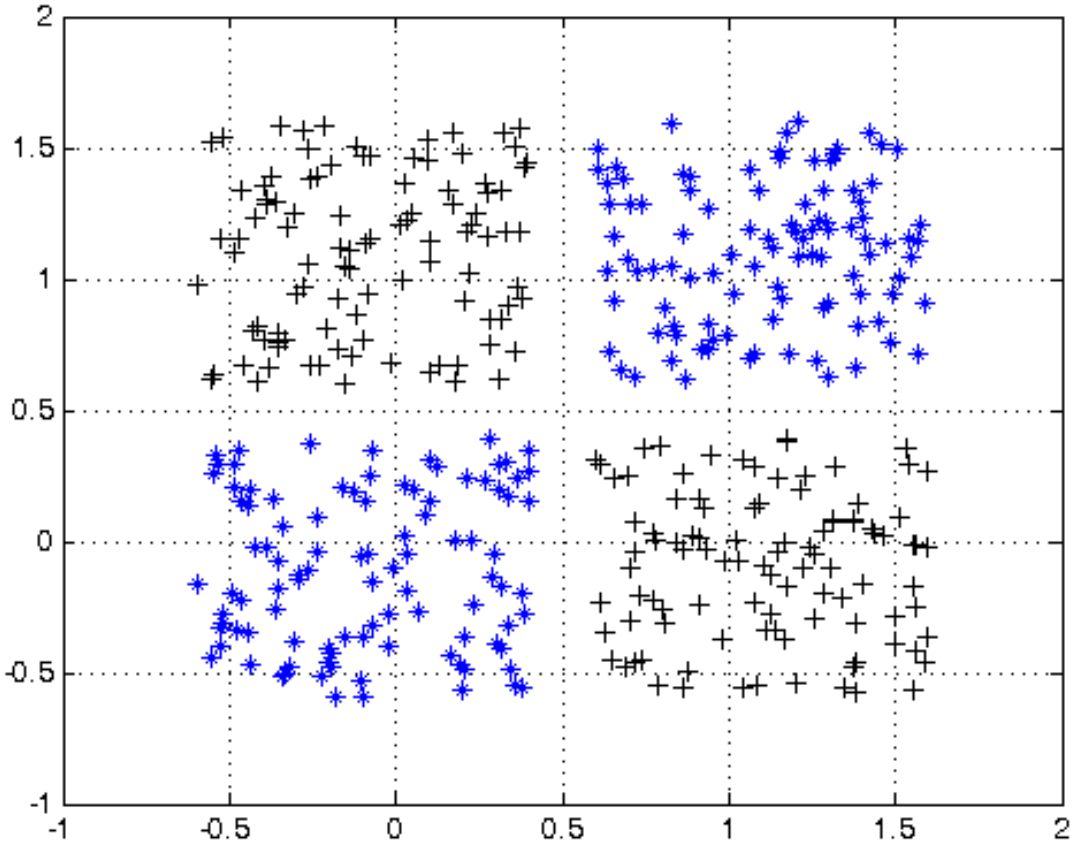
PROBLEM DESCRIPTION: 2 groups of linearly inseparable data (A,B) are defined in a 2-dimensional input space. The task is to define a neural network for solving the XOR classification problem.

Contents

- Create input data
- Define output coding
- Prepare inputs & outputs for network training
- Create a RBFN
- Evaluate network performance
- Plot classification result
- Plot RBFN centers

Create input data

```
close all, clear all, clc, format compact

% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A = [rand(1,K)-q rand(1,K)+q;
     rand(1,K)+q rand(1,K)-q];
B = [rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)-q];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```



Define output coding

```
% coding (+1/-1) for 2-class XOR problem
a = -1;
b =  1;
```

Prepare inputs & outputs for network training

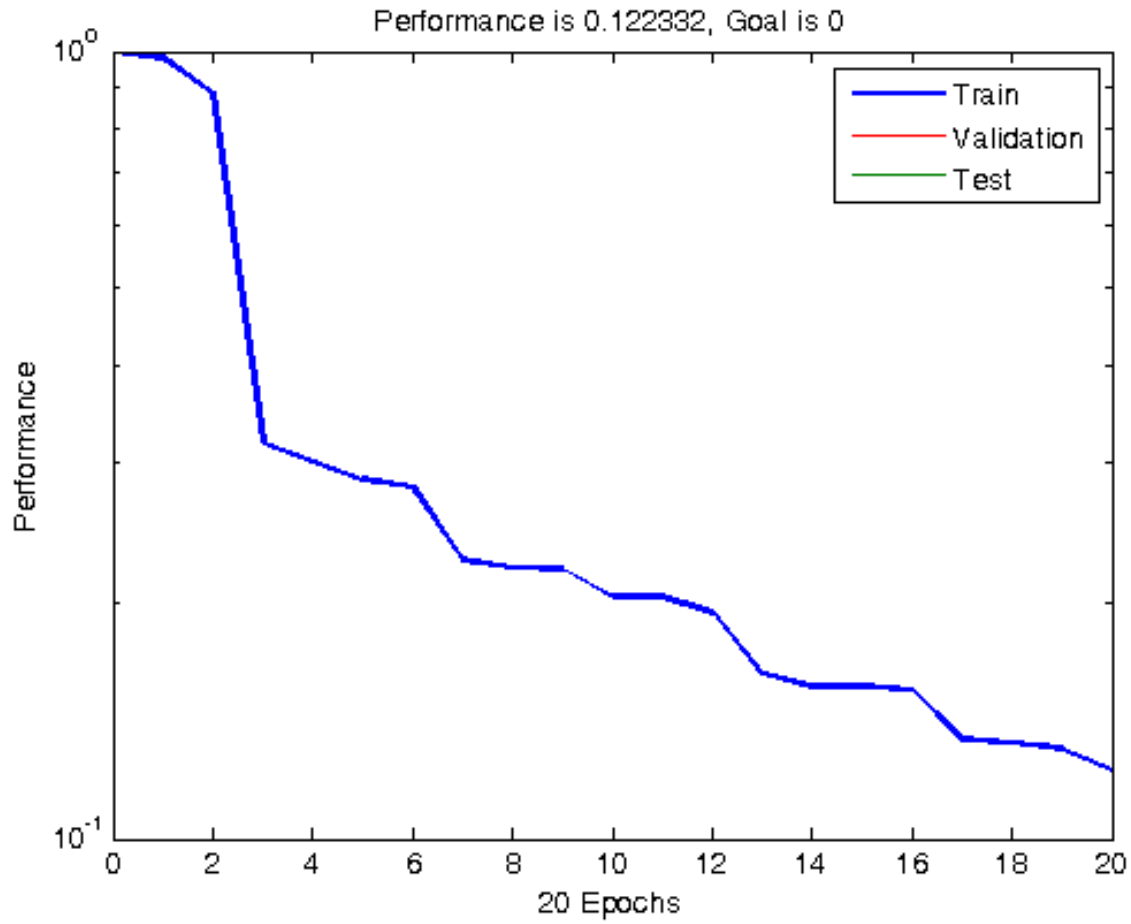
```
% define inputs (combine samples from all four classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

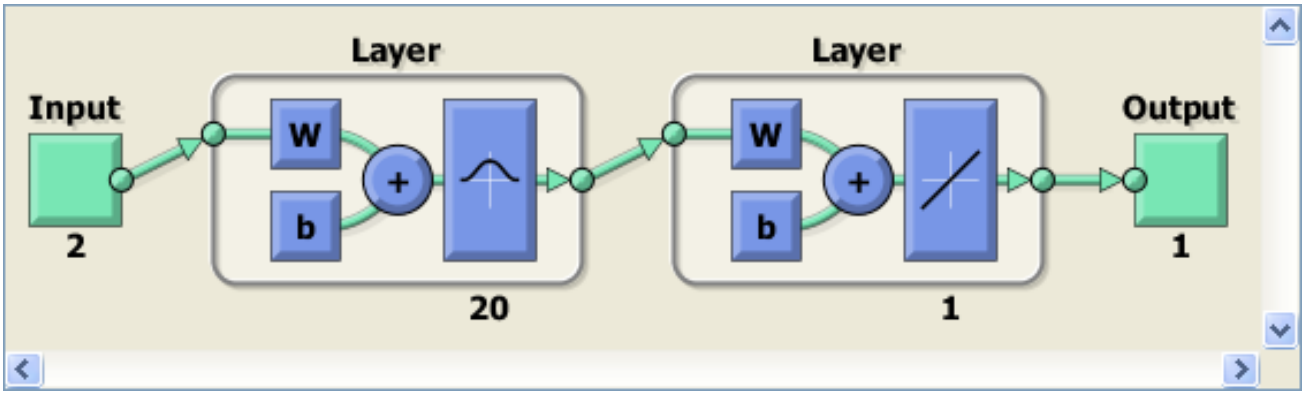
Create a RBFN

```
% NEWRB algorithm
% The following steps are repeated until the network's mean squared error
% falls below goal:
%  1. The network is simulated
%  2. The input vector with the greatest error is found
%  3. A radbas neuron is added with weights equal to that vector
%  4. The purelin layer weights are redesigned to minimize error

% choose a spread constant
```

```
spread = 2;
% choose max number of neurons
K      = 20;
% performance goal (SSE)
goal   = 0;
% number of neurons to add between displays
Ki     = 4;
% create a neural network
net    = newrb(P,T,goal,spread,K,Ki);

% view network
view(net)
```

```
NEWRB, neurons = 0, MSE = 1
NEWRB, neurons = 4, MSE = 0.302296
NEWRB, neurons = 8, MSE = 0.221059
NEWRB, neurons = 12, MSE = 0.193983
NEWRB, neurons = 16, MSE = 0.154859
NEWRB, neurons = 20, MSE = 0.122332
```

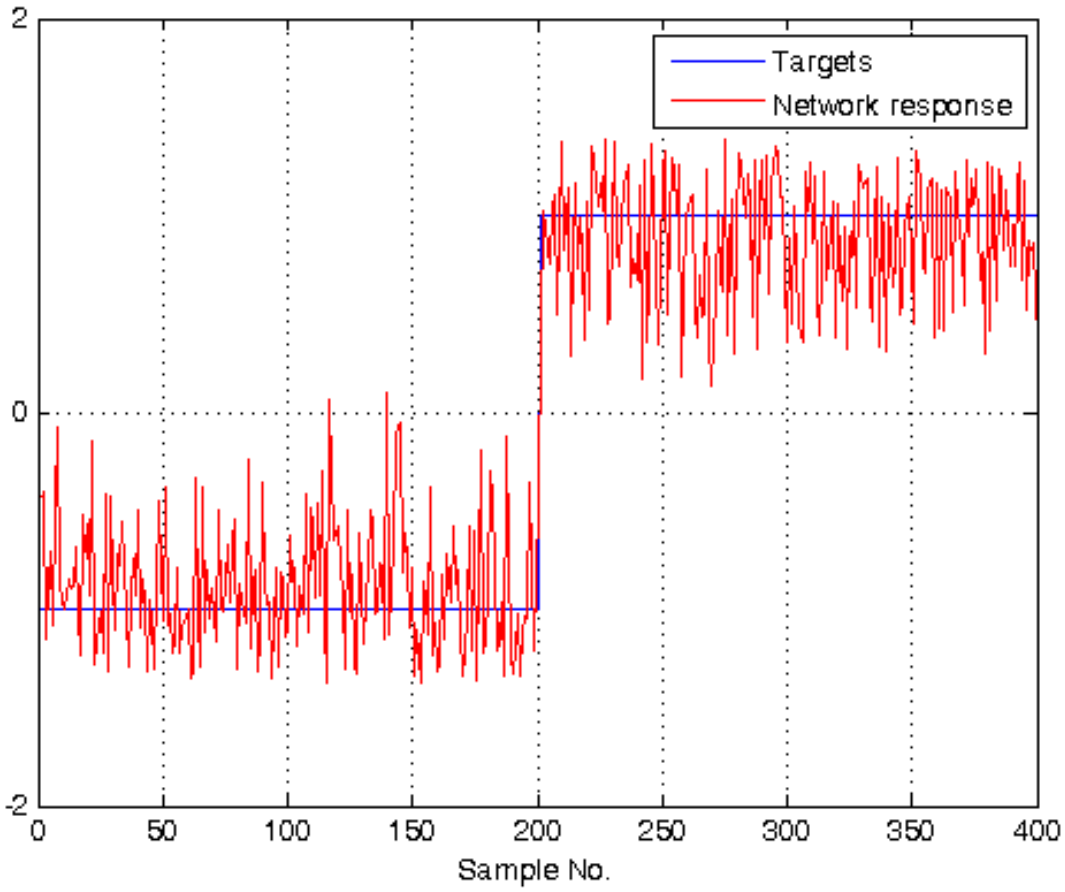




Evaluate network performance

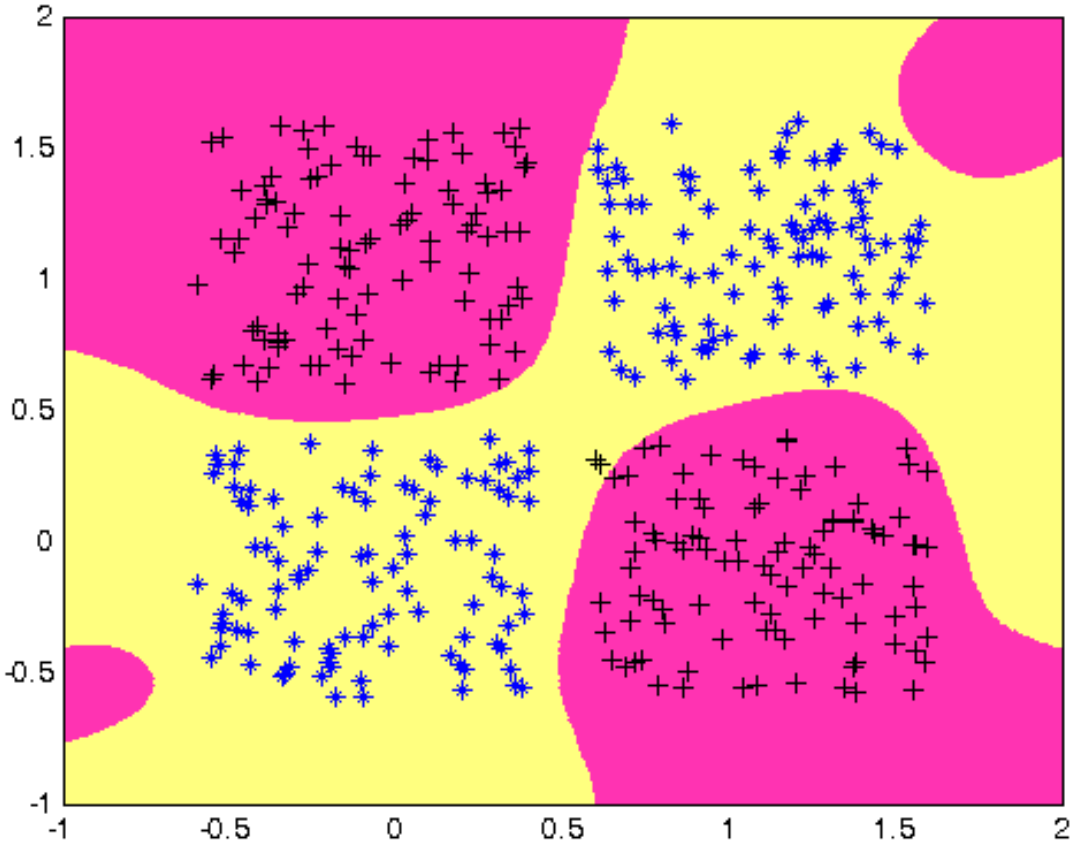
```
% simulate RBFN on training data
Y = net(P);

% calculate [%] of correct classifications
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('\nSpread          = %.2f\n',spread)
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)

% plot targets and network response
figure;
plot(T')
hold on
grid on
plot(Y','r')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
legend('Targets','Network response')
xlabel('Sample No.')
```

```
Spread          = 2.00
Num of neurons  = 20
Correct class   = 99.50 %
```



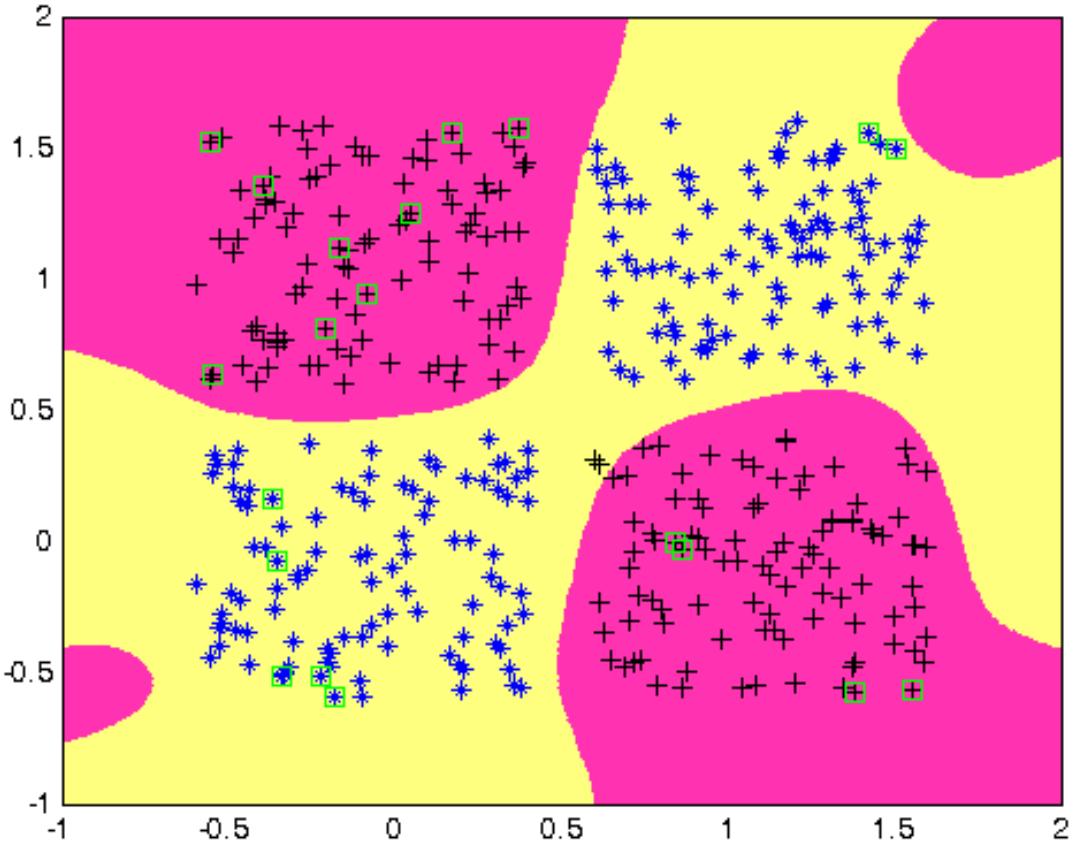
Plot classification result

```
% generate a grid
span    = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp      = [P1(:) P2(:)]';
% simualte neural network on a grid
aa      = sim(net,pp);

% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
```



Plot RBFN centers

```
plot(net.iw{1}(:,1),net.iw{1}(:,2),'gs')
```



Published with MATLAB® 7.14

Classification of XOR problem with a PNN

Neural Networks course (practical examples) © 2012 Primoz Potocnik

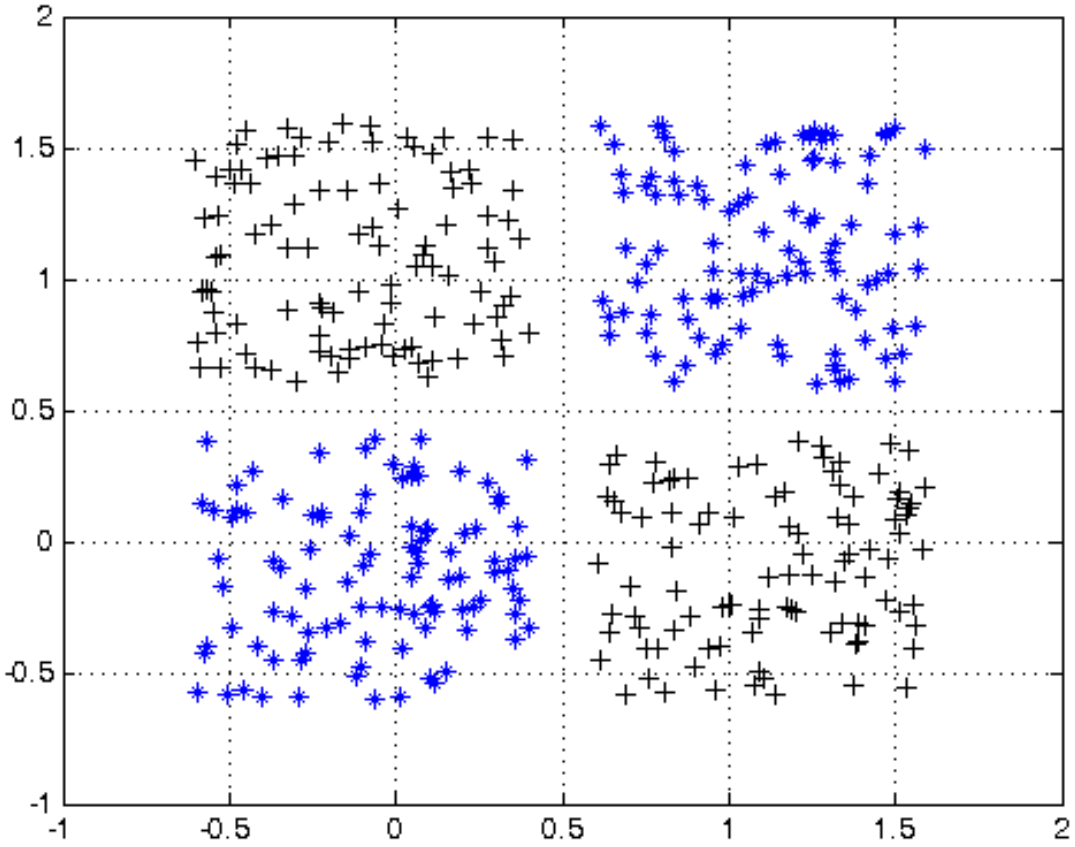
PROBLEM DESCRIPTION: 2 groups of linearly inseparable data (A,B) are defined in a 2-dimensional input space. The task is to define a neural network for solving the XOR classification problem.

Contents

- Create input data
- Define output coding
- Prepare inputs & outputs for network training
- Create a PNN
- Evaluate network performance
- Plot classification result for the complete input space
- plot PNN centers

Create input data

```
close all, clear all, clc, format compact

% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A = [rand(1,K)-q rand(1,K)+q;
     rand(1,K)+q rand(1,K)-q];
B = [rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)-q];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```



Define output coding

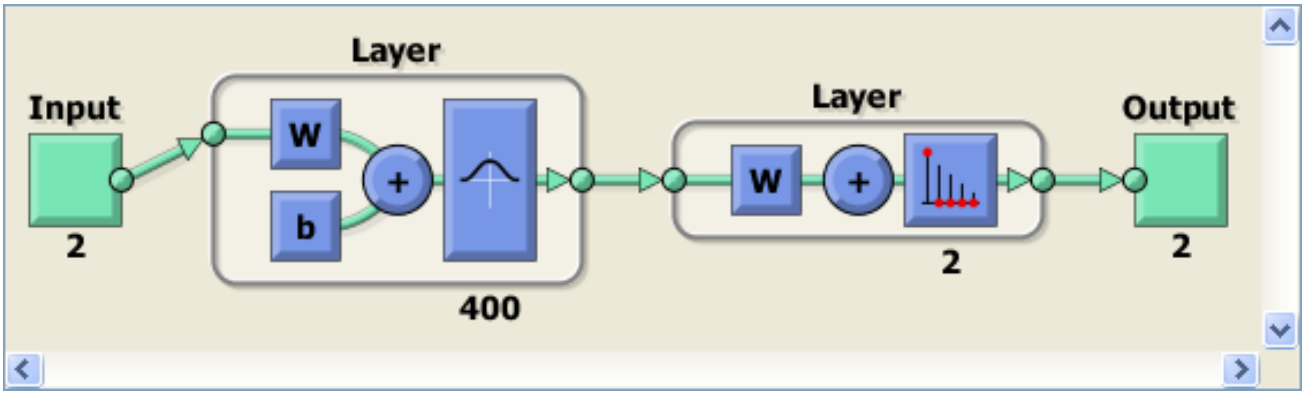
```
% coding (+1/-1) for 2-class XOR problem
a = 1;
b = 2;
```

Prepare inputs & outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

Create a PNN

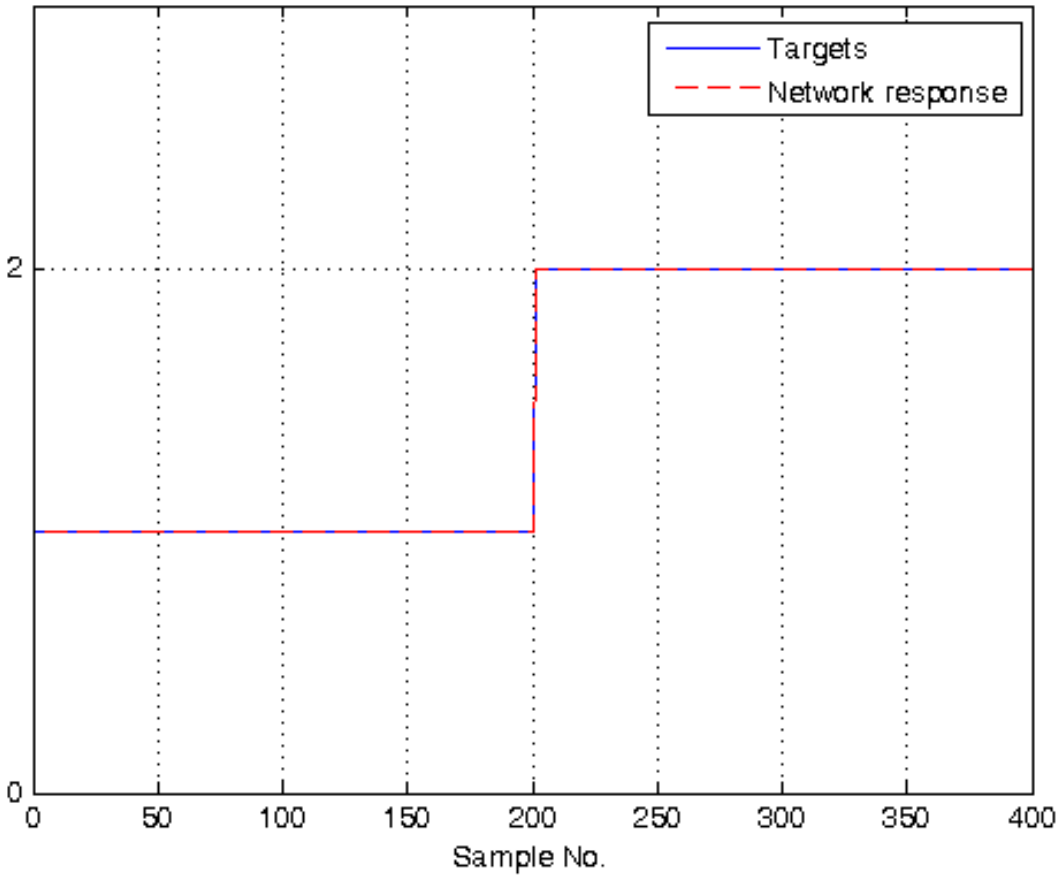
```
% choose a spread constant
spread = .5;
% create a neural network
net    = newpnn(P,ind2vec(T),spread);

% view network
view(net)
```



Evaluate network performance

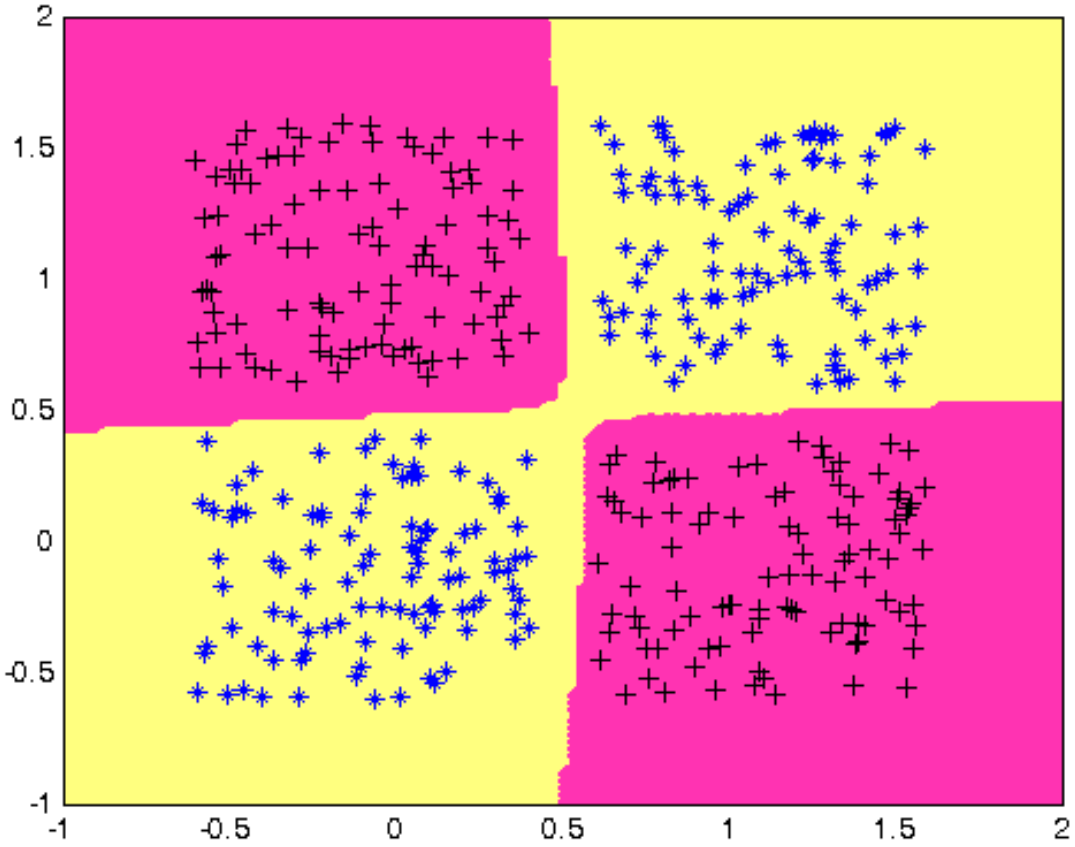
```
% simulate RBFN on training data
Y = net(P);
Y = vec2ind(Y);

% calculate [%] of correct classifications
correct = 100 * length(find(T==Y)) / length(T);
fprintf('\nSpread          = %.2f\n',spread)
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)

% plot targets and network response
figure;
plot(T')
hold on
grid on
plot(Y','r--')
ylim([0 3])
set(gca,'ytick',[-2 0 2])
legend('Targets','Network response')
xlabel('Sample No.')
```

```
Spread          = 0.50
Num of neurons  = 400
Correct class   = 100.00 %
```



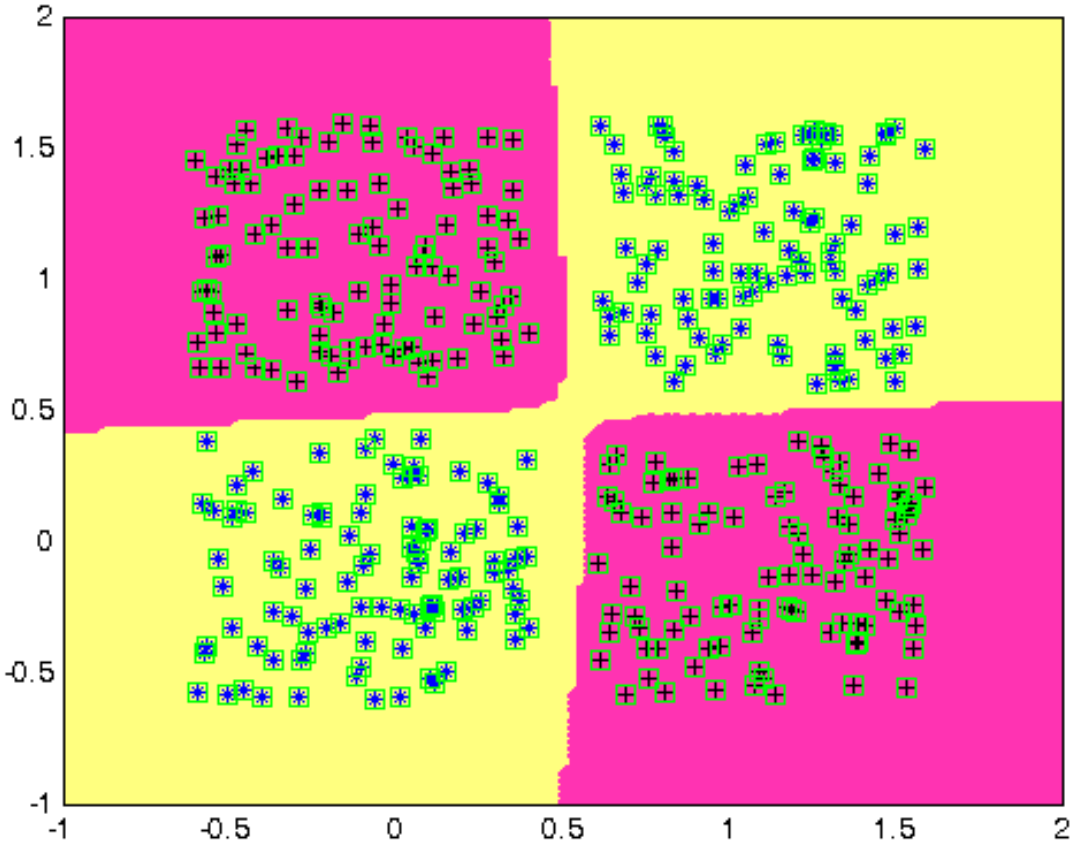
Plot classification result for the complete input space

```
% generate a grid
span    = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp      = [P1(:) P2(:)]';
% simualte neural network on a grid
aa      = sim(net,pp);
aa      = vec2ind(aa)-1.5; % convert

% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
```



plot PNN centers

```
plot(net.iw{1}(:,1),net.iw{1}(:,2),'gs')
```



Published with MATLAB® 7.14

Classification of XOR problem with a GRNN

Neural Networks course (practical examples) © 2012 Primoz Potocnik

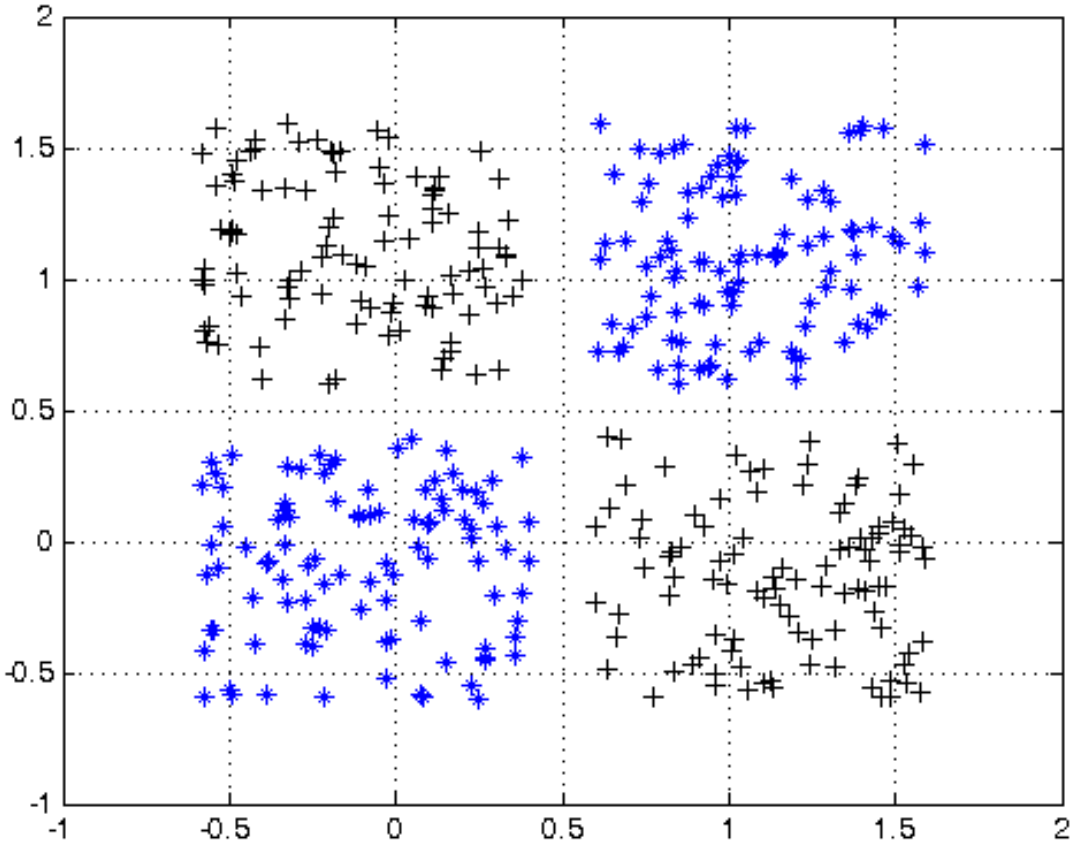
PROBLEM DESCRIPTION: 2 groups of linearly inseparable data (A,B) are defined in a 2-dimensional input space. The task is to define a neural network for solving the XOR classification problem.

Contents

- Create input data
- Define output coding
- Prepare inputs & outputs for network training
- Create a GRNN
- Evaluate network performance
- Plot classification result
- plot GRNN centers

Create input data

```
close all, clear all, clc, format compact

% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A = [rand(1,K)-q rand(1,K)+q;
     rand(1,K)+q rand(1,K)-q];
B = [rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)-q];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```



Define output coding

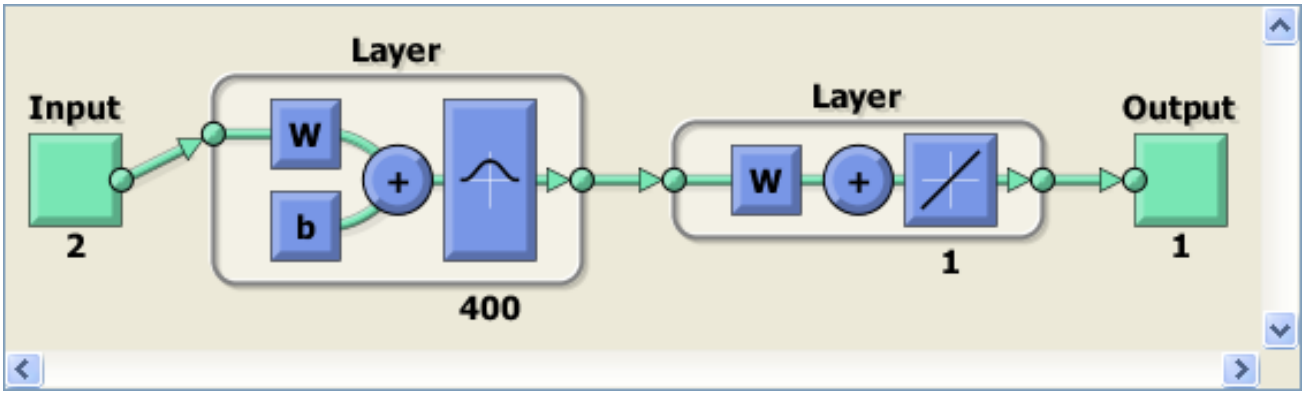
```
% coding (+1/-1) for 2-class XOR problem
a = -1;
b =  1;
```

Prepare inputs & outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

Create a GRNN

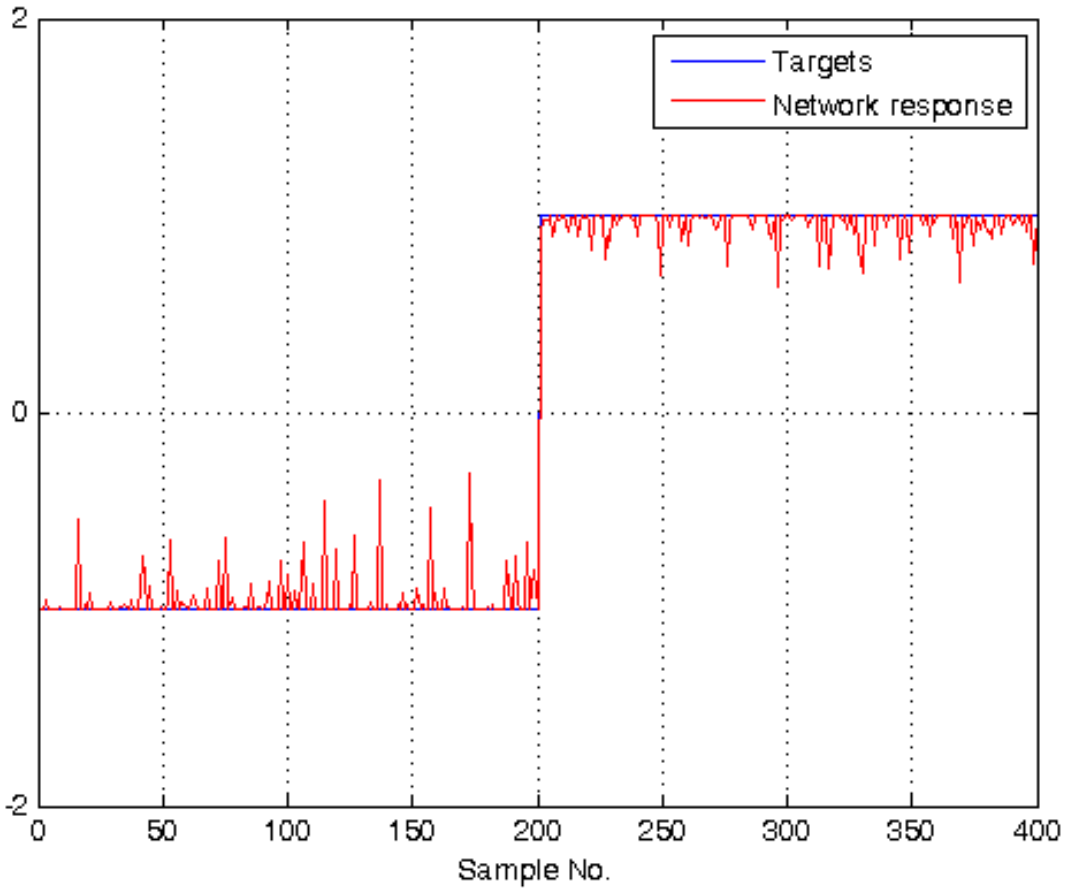
```
% choose a spread constant
spread = .2;
% create a neural network
net    = newgrnn(P,T,spread);

% view network
view(net)
```



Evaluate network performance

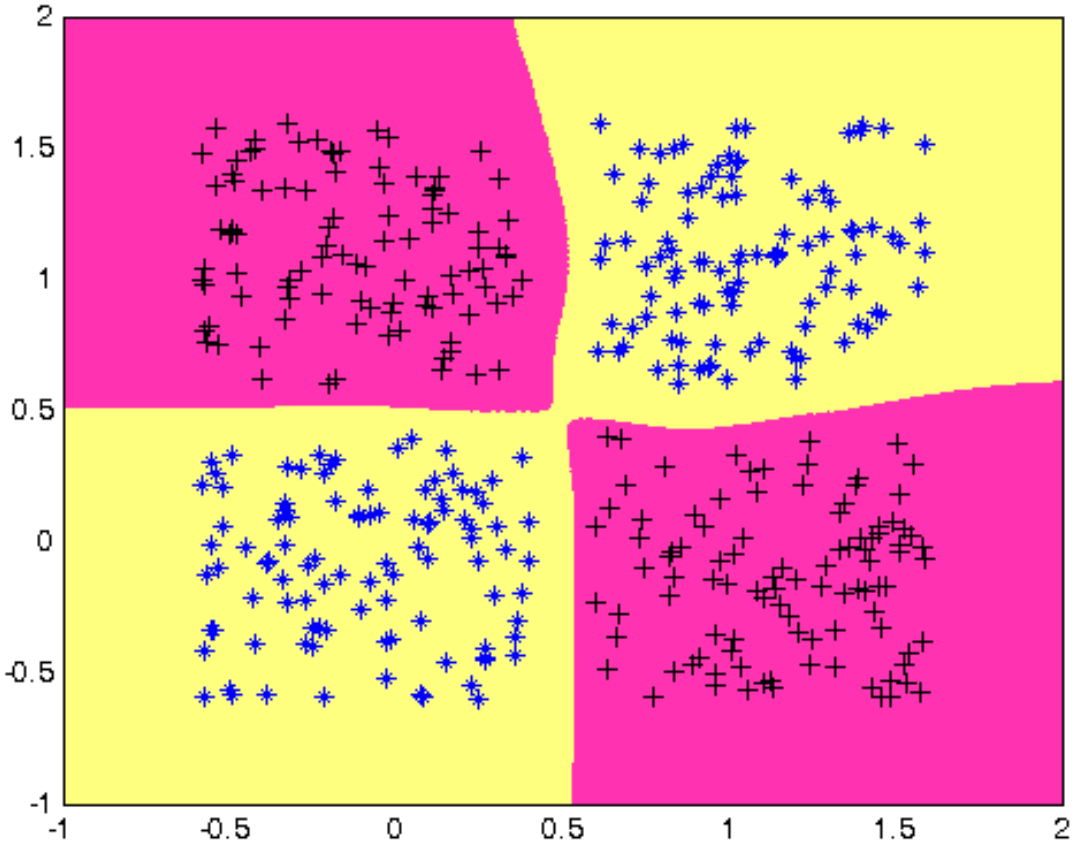
```
% simulate GRNN on training data
Y = net(P);

% calculate [%] of correct classifications
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('\nSpread          = %.2f\n',spread)
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)

% plot targets and network response
figure;
plot(T')
hold on
grid on
plot(Y','r')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
legend('Targets','Network response')
xlabel('Sample No.')
```

```
Spread          = 0.20
Num of neurons  = 400
Correct class   = 100.00 %
```



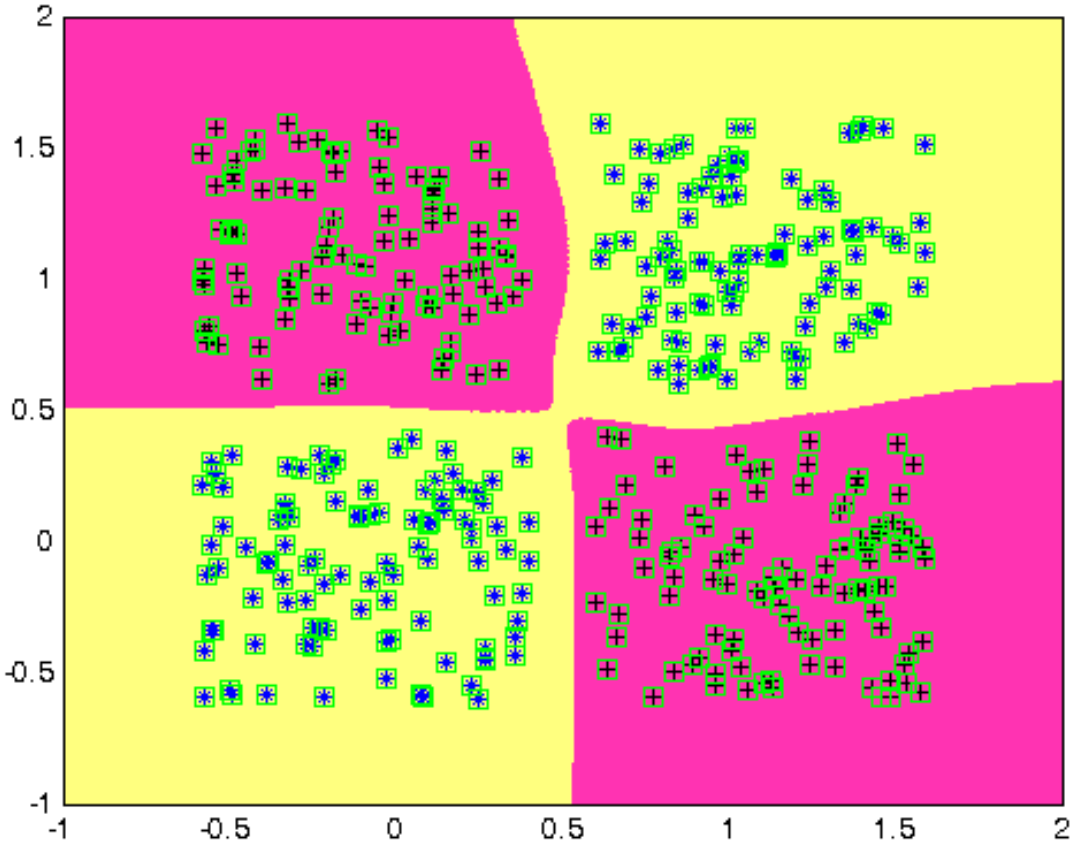
Plot classification result

```
% generate a grid
span    = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp      = [P1(:) P2(:)]';
% simualte neural network on a grid
aa      = sim(net,pp);

% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
```



plot GRNN centers

```
plot(net.iw{1}(:,1),net.iw{1}(:,2),'gs')
```



Published with MATLAB® 7.14

Bayesian regularization for RBFN

Neural Networks course (practical examples) © 2012 Primoz Potocnik

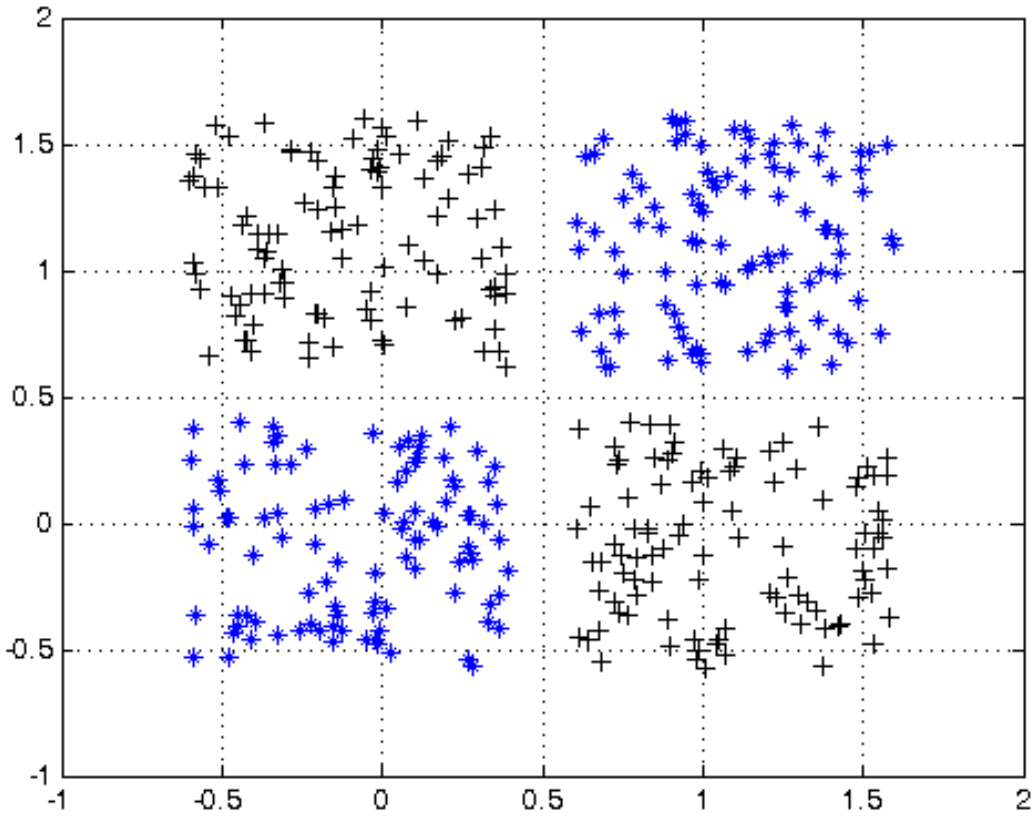
PROBLEM DESCRIPTION: 2 groups of linearly inseparable data (A,B) are defined in a 2-dimensional input space. The task is to define a neural network for solving the XOR classification problem.

Contents

- Create input data
- Define output coding
- Prepare inputs & outputs for network training
- Create a RBFN
- Evaluate network performance
- Plot classification result
- Retrain a RBFN using Bayesian regularization backpropagation
- Evaluate network performance after Bayesian regularization training
- Plot classification result after Bayesian regularization training

Create input data

```
close all, clear all, clc, format compact

% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A = [rand(1,K)-q rand(1,K)+q;
     rand(1,K)+q rand(1,K)-q];
B = [rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)-q];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```



Define output coding

```
% coding (+1/-1) for 2-class XOR problem
a = -1;
b =  1;
```

Prepare inputs & outputs for network training

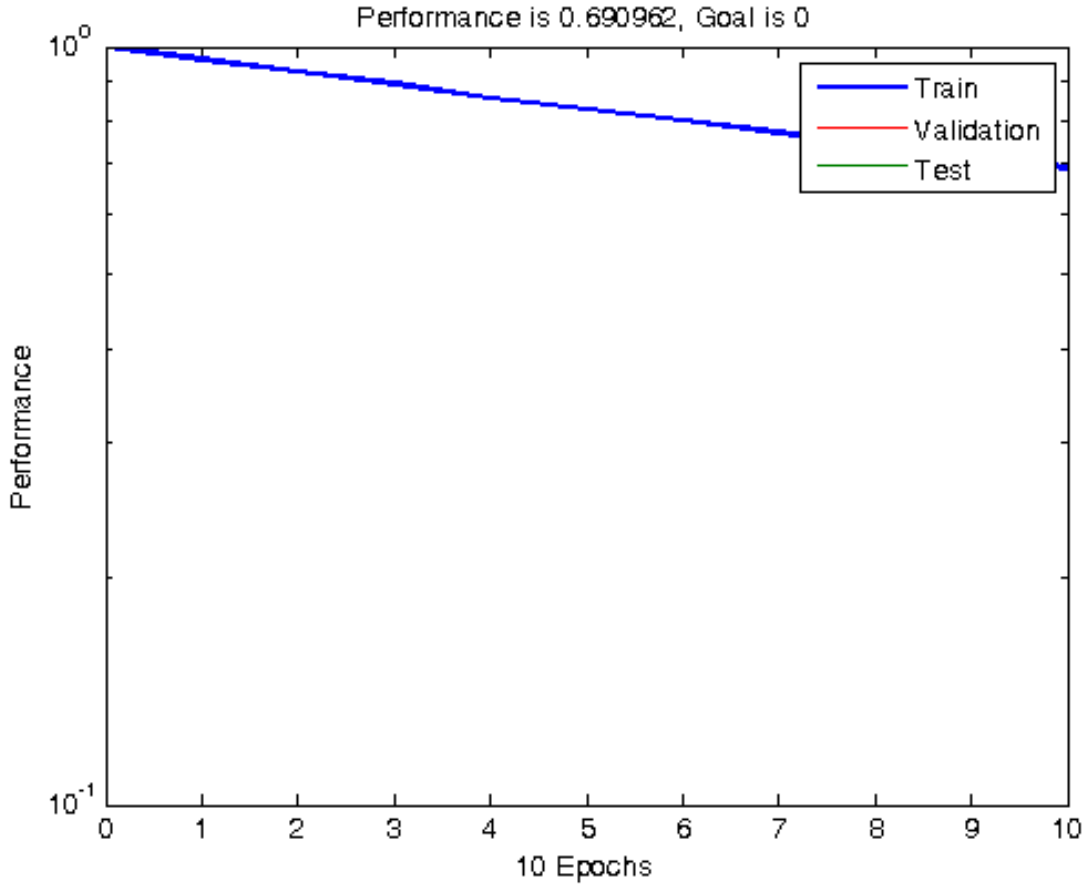
```
% define inputs (combine samples from all four classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

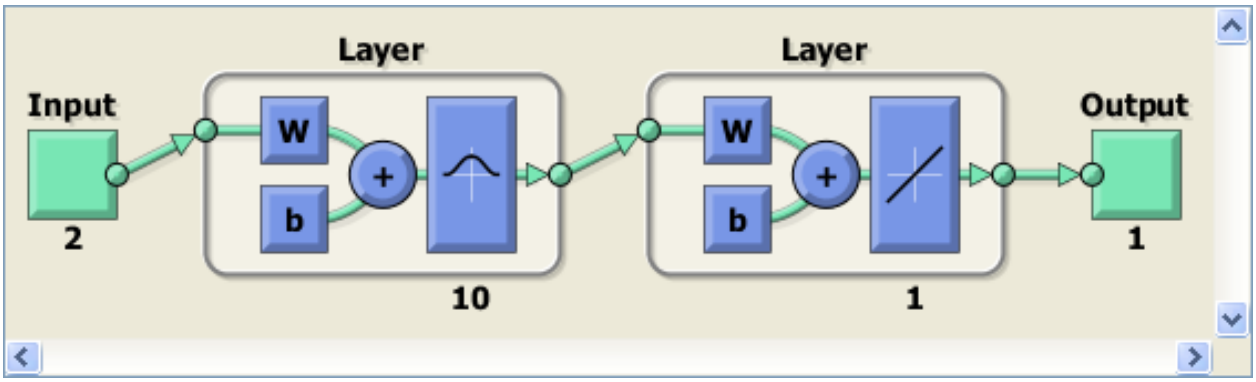
Create a RBFN

```
% choose a spread constant
spread = .1;
% choose max number of neurons
K      = 10;
% performance goal (SSE)
goal   = 0;
% number of neurons to add between displays
Ki     = 2;
% create a neural network
net    = newrb(P,T,goal,spread,K,Ki);

% view network
```

```
view(net)
```

```
NEWRB, neurons = 0, MSE = 1
NEWRB, neurons = 2, MSE = 0.928277
NEWRB, neurons = 4, MSE = 0.855829
NEWRB, neurons = 6, MSE = 0.798564
NEWRB, neurons = 8, MSE = 0.742854
NEWRB, neurons = 10, MSE = 0.690962
```



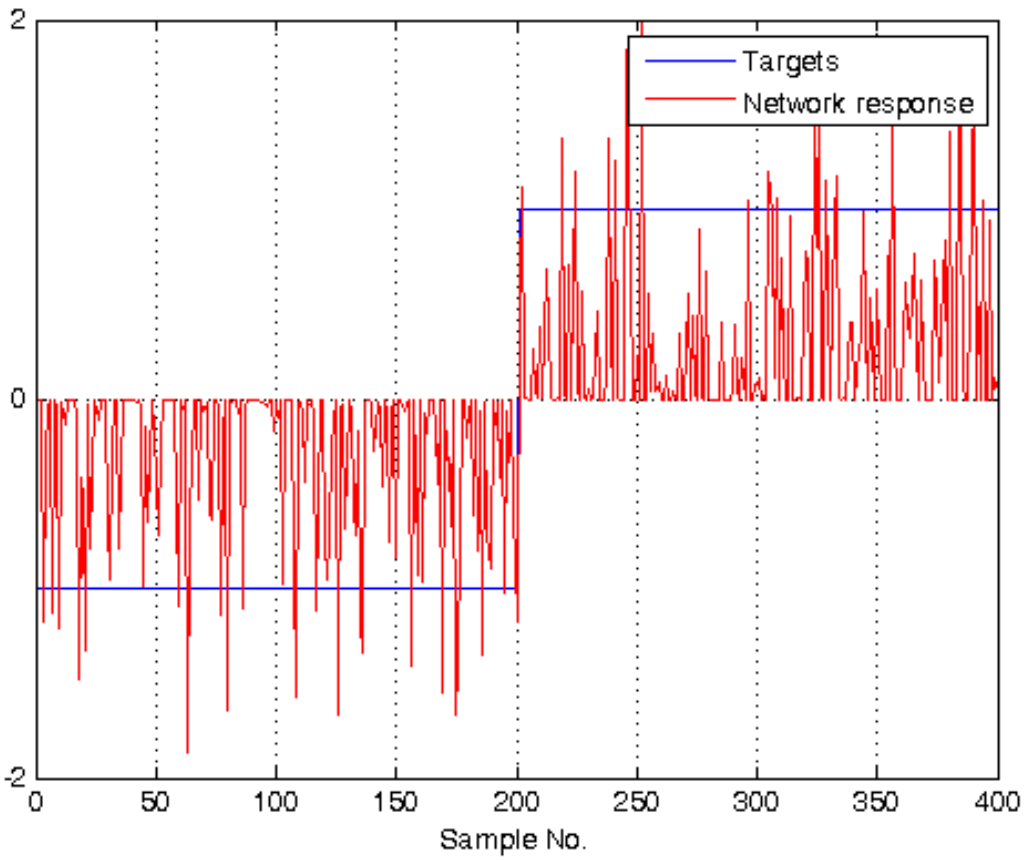


Evaluate network performance

```
% check RBFN spread
actual_spread = net.b{1}

% simulate RBFN on training data
Y = net(P);
```

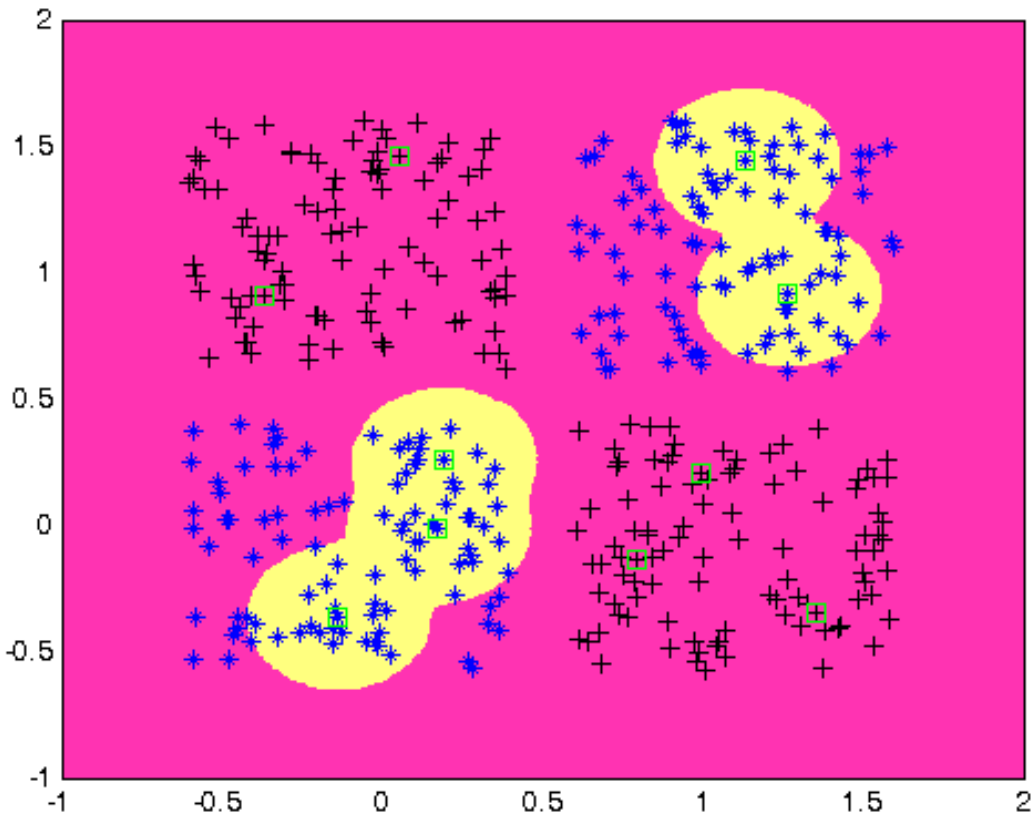
```
% calculate [%] of correct classifications
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('\nSpread          = %.2f\n',spread)
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)

% plot targets and network response to see how good the network learns the data
figure;
plot(T')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
hold on
grid on
plot(Y','r')
legend('Targets','Network response')
xlabel('Sample No.')
```

```
actual_spread =
    8.3255
    8.3255
    8.3255
    8.3255
    8.3255
    8.3255
    8.3255
    8.3255
    8.3255
    8.3255

Spread          = 0.10
Num of neurons  = 10
Correct class   = 79.50 %
```



Plot classification result

```
% generate a grid
span    = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp      = [P1(:) P2(:)]';
% simualte neural network on a grid
aa      = sim(net,pp);

% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)

% plot RBFN centers
plot(net.iw{1}(:,1),net.iw{1}(:,2),'gs')
```



Retrain a RBFN using Bayesian regularization backpropagation

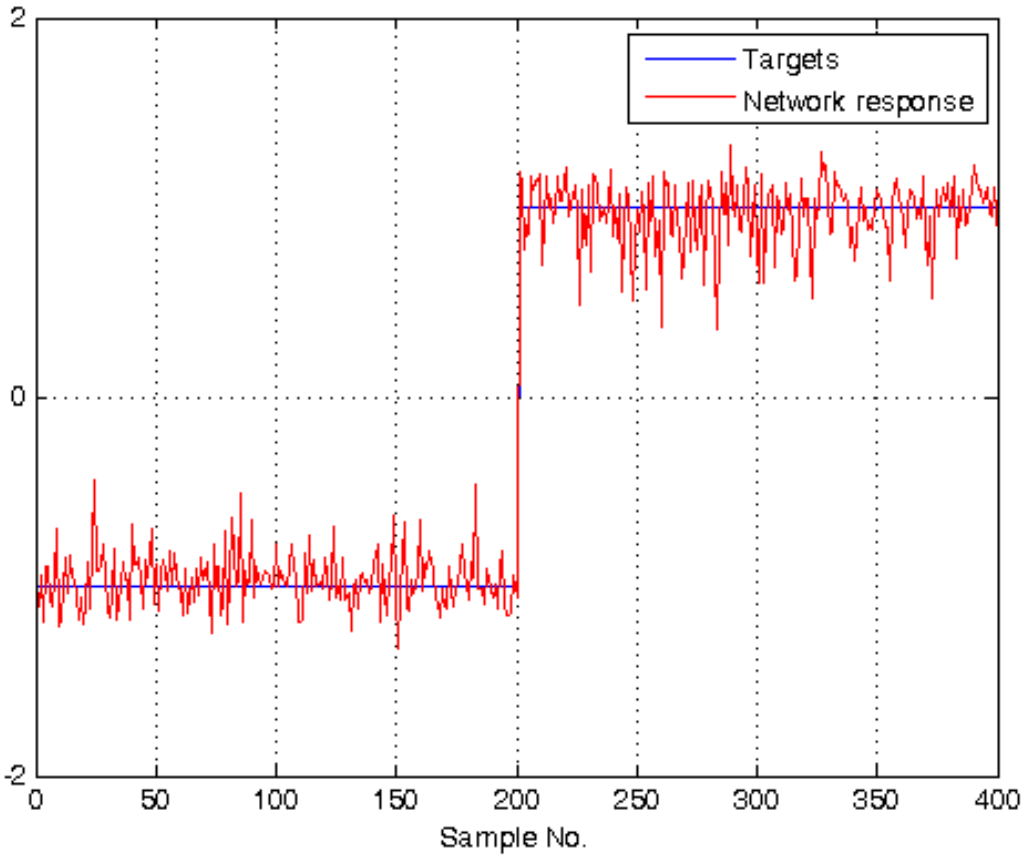
```
% define custom training function: Bayesian regularization backpropagation
net.trainFcn='trainbr';
% perform Levenberg-Marquardt training with Bayesian regularization
net = train(net,P,T);
```

Evaluate network performance after Bayesian regularization training

```
% check new RBFN spread
spread_after_training = net.b{1}
% simulate RBFN on training data
Y = net(P);

% calculate [%] of correct classifications
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)

% plot targets and network response
figure;
plot(T')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
hold on
grid on
plot(Y','r')
legend('Targets','Network response')
```

```
xlabel('Sample No.')
```

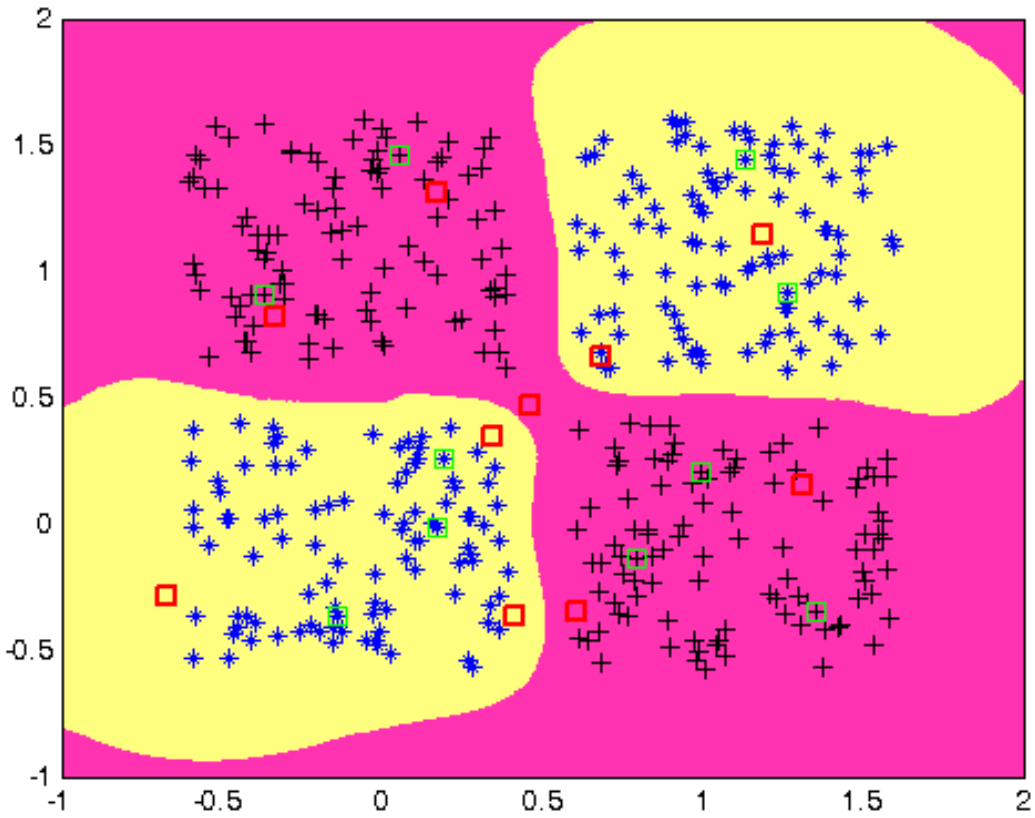
```
spread_after_training =
    2.9924
    3.0201
    0.7809
    0.5933
    2.6968
    2.8934
    2.2121
    2.9748
    2.7584
    3.5739
Num of neurons  = 10
Correct class   = 100.00 %
```



Plot classification result after Bayesian regularization training

```
% simulate neural network on a grid
aa = sim(net,pp);
% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
```

```
% Plot modified RBFN centers
plot(net.iw{1}(:,1),net.iw{1}(:,2),'rs','linewidth',2)
```



Published with MATLAB® 7.14

1D and 2D Self Organized Map

Neural Networks course (practical examples) © 2012 Primoz Potocnik

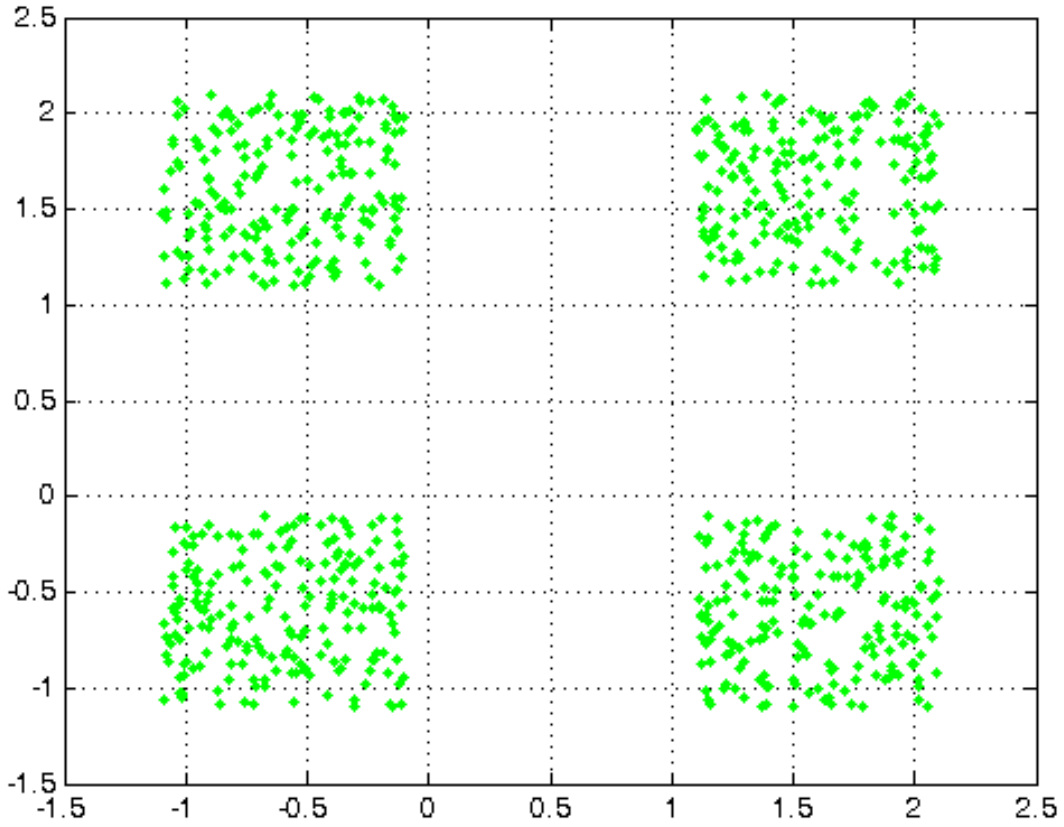
PROBLEM DESCRIPTION: Define 1-dimensional and 2-dimensional SOM networks to represent the 2-dimensional input space.

Contents

- Define 4 clusters of input data
- Create and train 1D-SOM
- plot 1D-SOM results
- Create and train 2D-SOM
- plot 2D-SOM results

Define 4 clusters of input data

```
close all, clear all, clc, format compact

% number of samples of each cluster
K = 200;
% offset of classes
q = 1.1;
% define 4 clusters of input data
P = [rand(1,K)-q rand(1,K)+q rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)+q rand(1,K)-q rand(1,K)-q];
% plot clusters
plot(P(1,:),P(2,:),'g.')
hold on
grid on
```

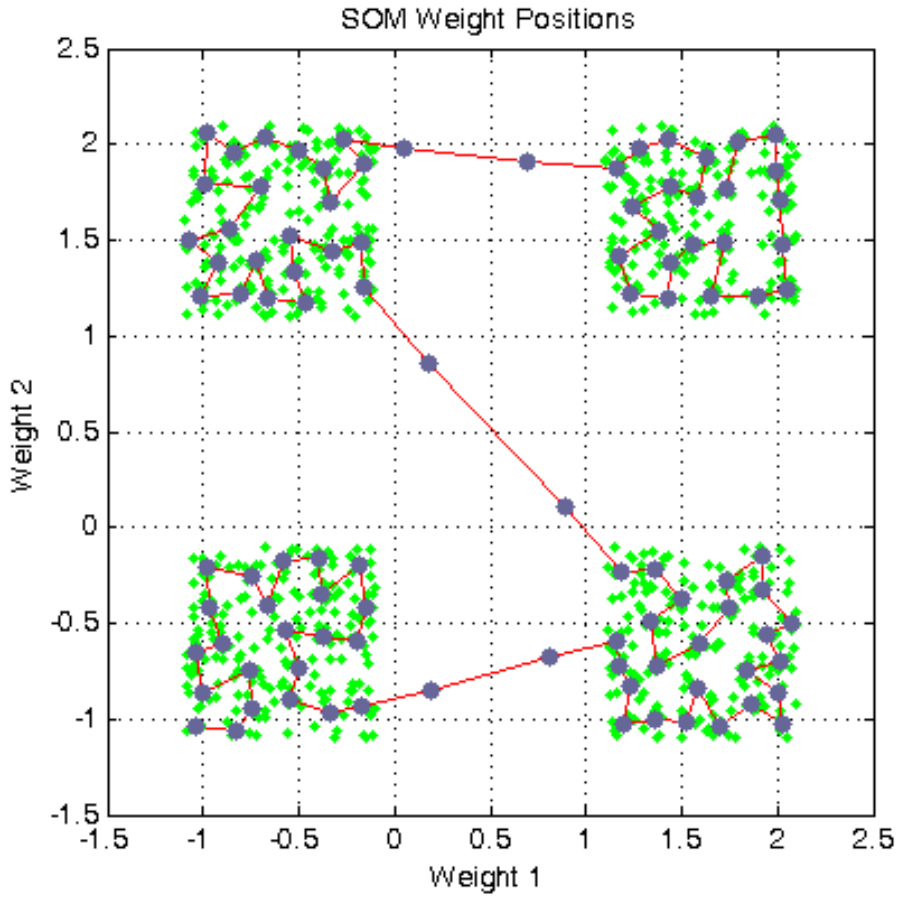


Create and train 1D-SOM

```
% SOM parameters
dimensions   = [100];
coverSteps   = 100;
initNeighbor = 10;
topologyFcn  = 'gridtop';
distanceFcn  = 'linkdist';

% define net
net1 = selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);

% train
[net1,Y] = train(net1,P);
```

plot 1D-SOM results

```
% plot input data and SOM weight positions
plotsompos(net1,P);
grid on
```

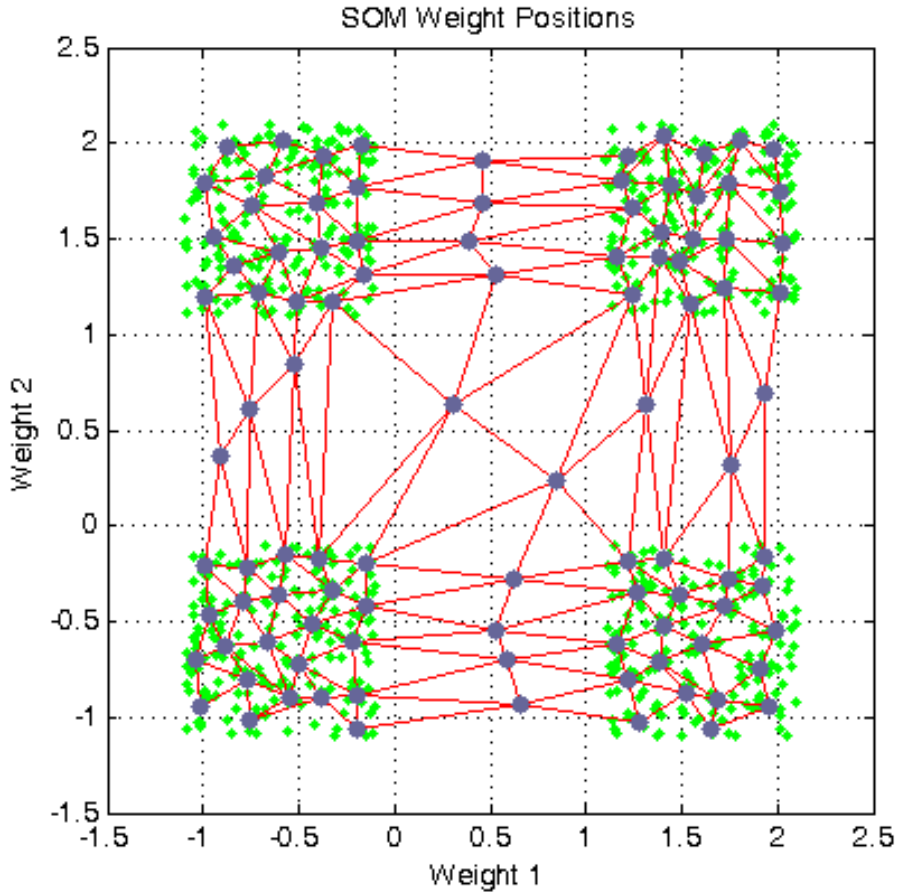


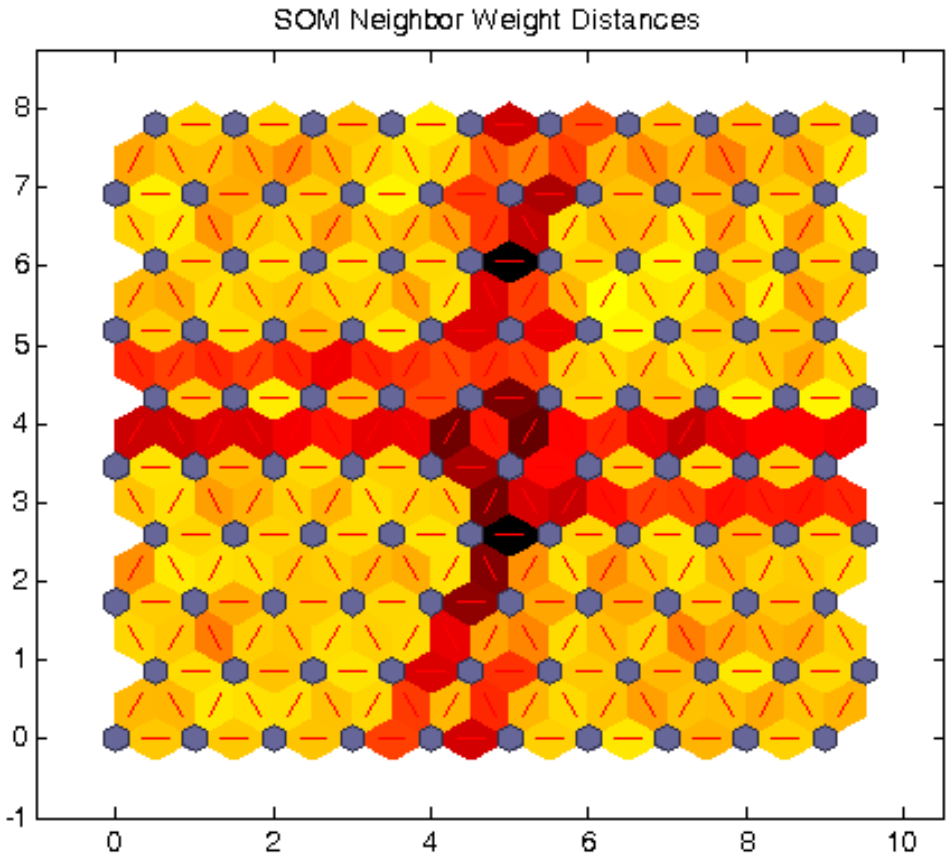
Create and train 2D-SOM

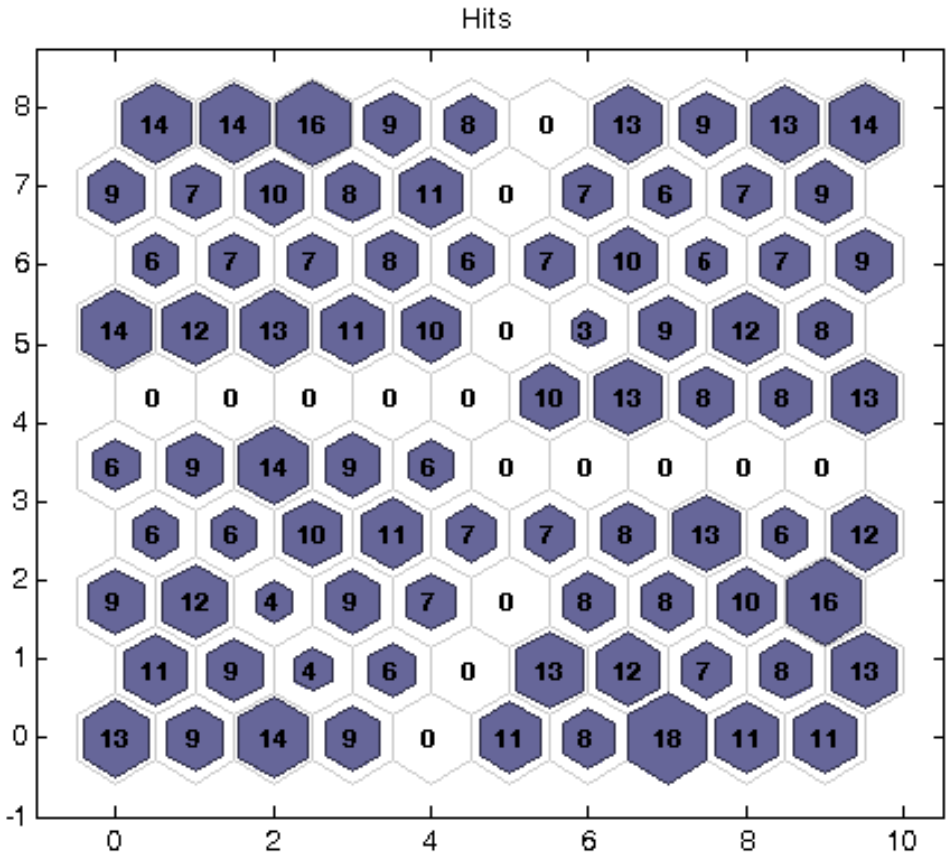
```
% SOM parameters
dimensions   = [10 10];
coverSteps   = 100;
initNeighbor = 4;
topologyFcn  = 'hextop';
distanceFcn  = 'linkdist';

% define net
net2 = selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);

% train
[net2,Y] = train(net2,P);
```

plot 2D-SOM results

```
% plot input data and SOM weight positions
plotsompos(net2,P);
grid on

% plot SOM neighbor distances
plotsomnd(net2)

% plot for each SOM neuron the number of input vectors that it classifies
figure
plotsomhits(net2,P)
```


SOM Weight Positions
Weight 2
Weight 1
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5


SOM Neighbor Weight Distances
8
7
6
5
4
3
2
1
0
-1
0
2
4
6
8
10



Published with MATLAB® 7.14

PCA for industrial diagnostic of compressor connection rod defects

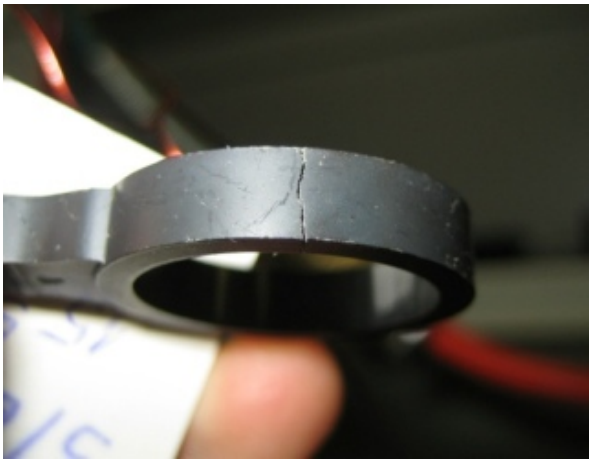
Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: Industrial production of compressors suffers from problems during the imprinting operation where a connection rod is connected with a compressor head. Irregular imprinting can cause damage or crack of the connection rod which results in damaged compressor. Such compressors should be eliminated from the production line but defects of this type are difficult to detect. The task is to detect crack and overload defects from the measurement of the imprinting force.

Contents

- Photos of the broken connection rod
- Load and plot data
- Prepare inputs by PCA
- Define output coding: 0=OK, 1=Error
- Create and train a multilayer perceptron
- Evaluate network performance
- Plot classification result

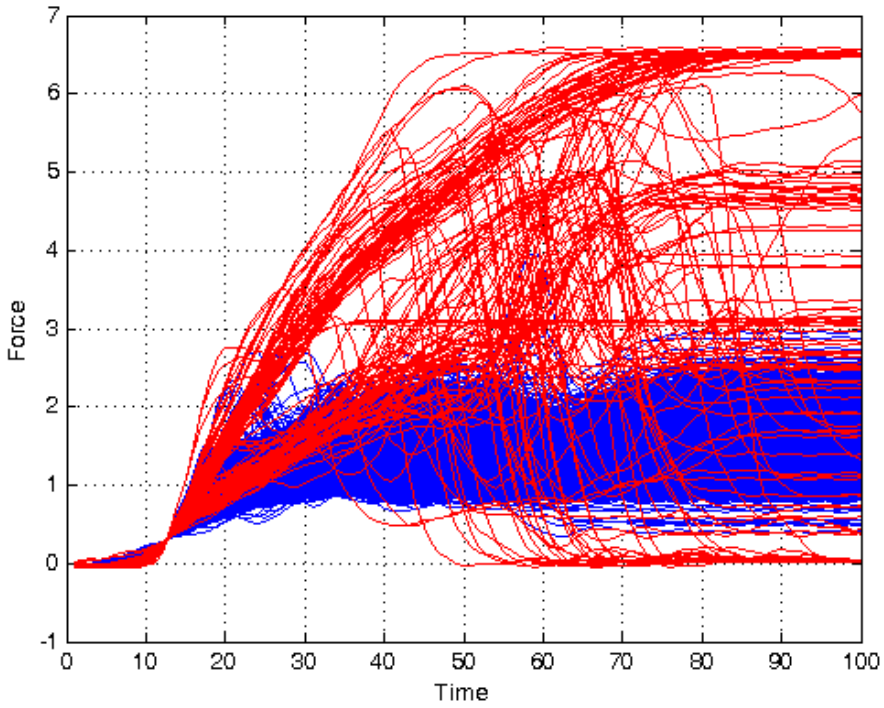
Photos of the broken connection rod



Load and plot data

```
close all, clear all, clc, format compact

% industrial data
load data2.mat
whos

% show data
figure
plot(force(find(target==1),:)','b') % OK (class 1)
grid on, hold on
plot(force(find(target>1),:)','r') % NOT OK (class 2 & 3)
xlabel('Time')
ylabel('Force')
```

```
  Name           Size              Bytes  Class     Attributes

  force       2000x100            1600000  double
  notes          1x3                  222  cell
  target      2000x1                 16000  double
```

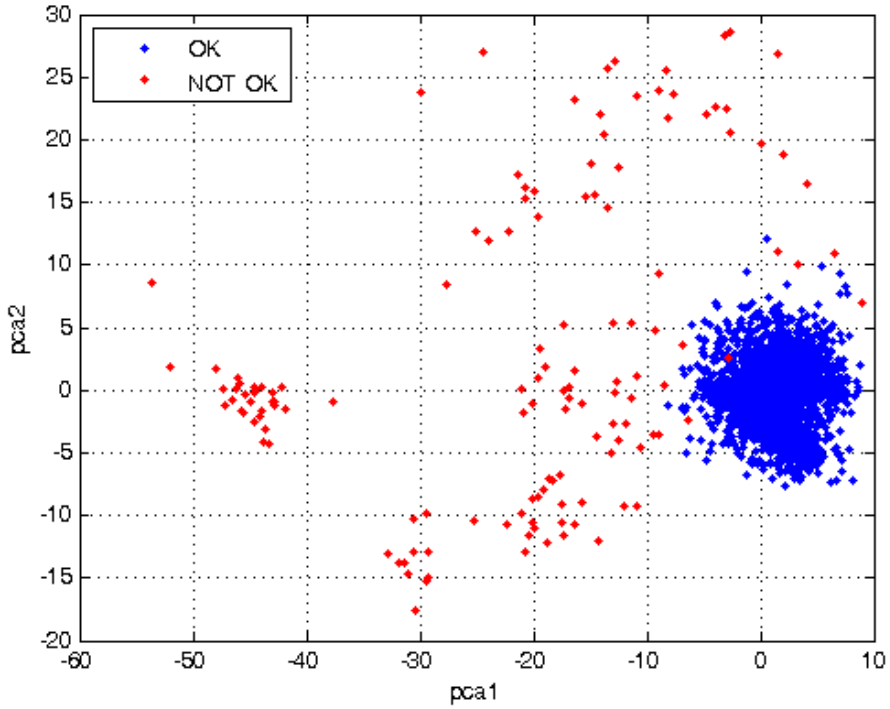


Prepare inputs by PCA

```
% 1. Standardize inputs to zero mean, variance one
[pn,ps1] = mapstd(force');

% 2. Apply Principal Compoments Analysis
% inputs whose contribution to total variation are less than maxfrac are removed
FP.maxfrac = 0.1;
% process inputs with principal component analysis
[ptrans,ps2] = processpca(pn, FP);
ps2
% transformed inputs
force2 = ptrans';
whos force force2

% plot data in the space of first 2 PCA components
figure
plot(force2(:,1),force2(:,2),'.') % OK
grid on, hold on
plot(force2(find(target>1),1),force2(find(target>1),2),'r.') % NOT_OK
xlabel('pca1')
ylabel('pca2')
legend('OK','NOT OK','location','nw')

% % plot data in the space of first 3 PCA components
% figure
% plot3(force2(find(target==1),1),force2(find(target==1),2),force2(find(target==1),3),'b.')
% grid on, hold on
% plot3(force2(find(target>1),1),force2(find(target>1),2),force2(find(target>1),3),'r.')
```

```
ps2 =
         name: 'processpca'
        xrows: 100
      maxfrac: 0.1000
        yrows: 2
    transform: [2x100 double]
    no_change: 0
  Name           Size                Bytes  Class     Attributes

  force       2000x100            1600000  double
```

```
force2      2000x2                32000  double
```

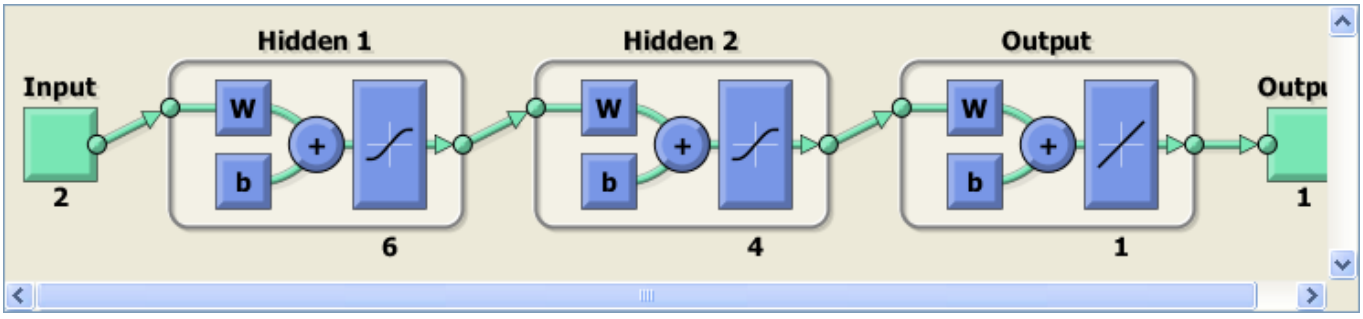


Define output coding: 0=OK, 1=Error

```
% binary coding 0/1
target = double(target > 1);
```

Create and train a multilayer perceptron

```
% create a neural network
net = feedforwardnet([6 4]);

% set early stopping parameters
net.divideParam.trainRatio = 0.70; % training set [%]
net.divideParam.valRatio   = 0.15; % validation set [%]
net.divideParam.testRatio  = 0.15; % test set [%]

% train a neural network
[net,tr,Y,E] = train(net,force2',target');

% show net
view(net)
```



Evaluate network performance

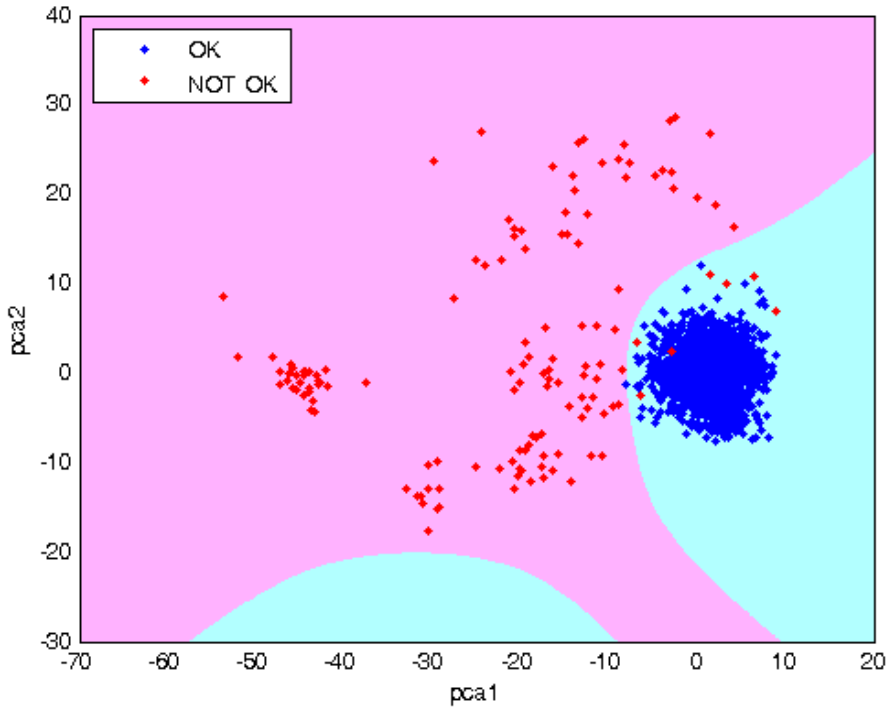
```
% digitize network response
```

```
threshold = 0.5;
Y = double(Y > threshold)';

% find percentage of correct classifications
cc = 100*length(find(Y==target))/length(target);
fprintf('Correct classifications: %.1f [%%]\n', cc)
```

```
Correct classifications: 99.6 [%]
```

Plot classification result

```
figure(2)
a = axis;
% generate a grid, expand input space
xspan = a(1)-10 : .1 : a(2)+10;
yspan = a(3)-10 : .1 : a(4)+10;
[P1,P2] = meshgrid(xspan,yspan);
pp      = [P1(:) P2(:)]';
% simualte neural network on a grid
aa      = sim(net,pp);
aa      = double(aa > threshold);

% plot classification regions based on MAX activation
ma = mesh(P1,P2,reshape(-aa,length(yspan),length(xspan))-4);
mb = mesh(P1,P2,reshape( aa,length(yspan),length(xspan))-5);
set(ma,'facecolor',[.7 1.0 1],'linestyle','none');
set(mb,'facecolor',[1 0.7  1],'linestyle','none');
view(2)
```



Published with MATLAB® 7.14