Neuron output

Neural Networks course (practical examples) © 2012 Primoz Potocnik

PROBLEM DESCRIPTION: Calculate the output of a simple neuron

Contents

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

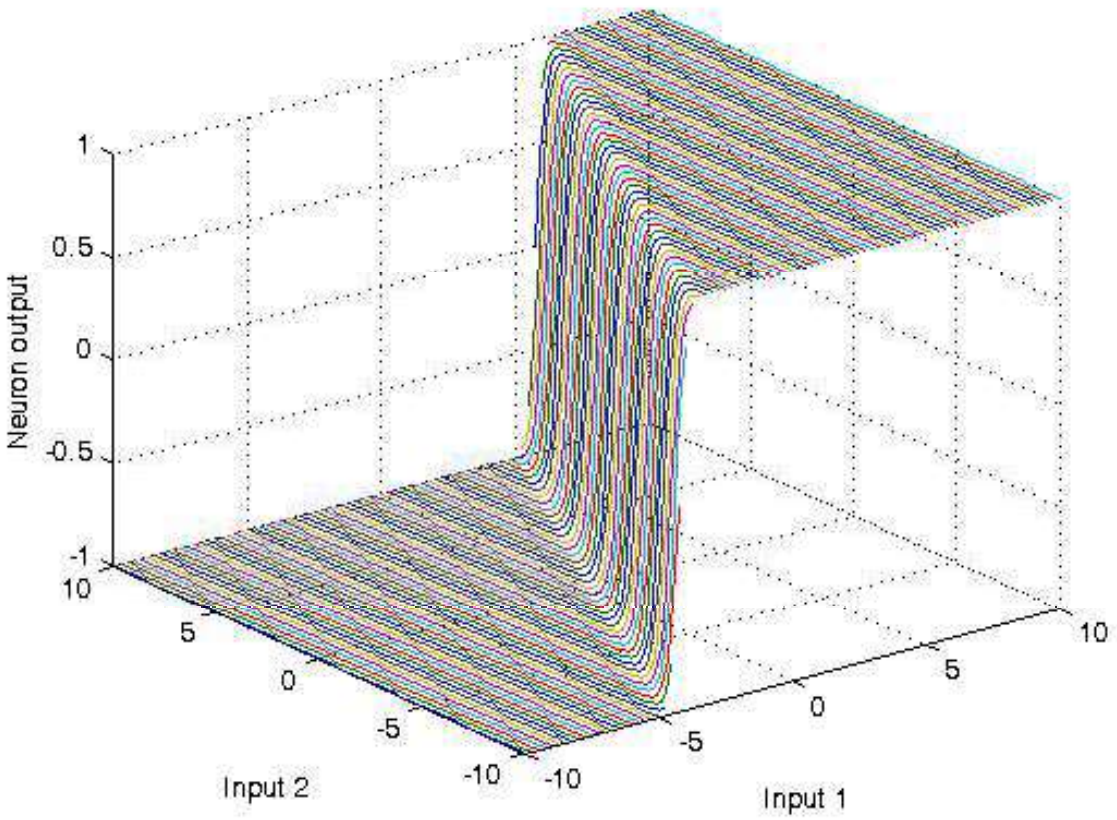
```
activation_potential = p*w'+b
```

```
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

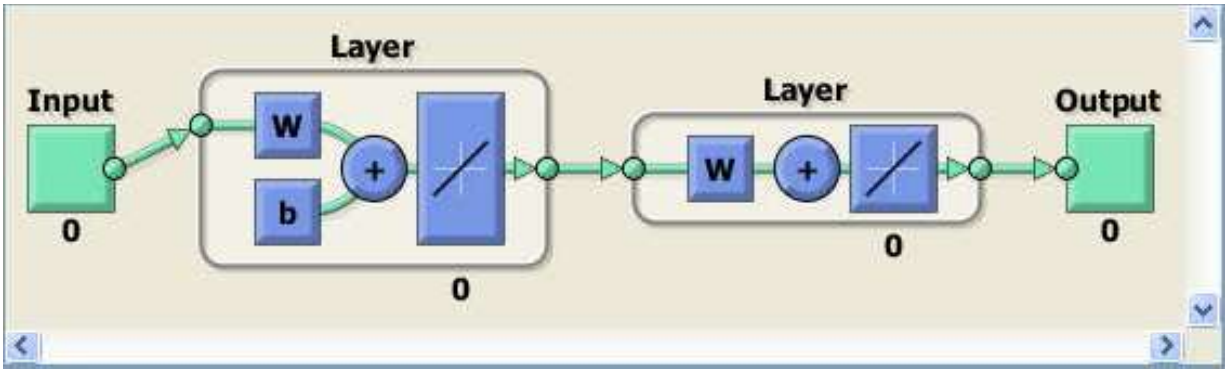
Define one sample: inputs and outputs

```
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
```

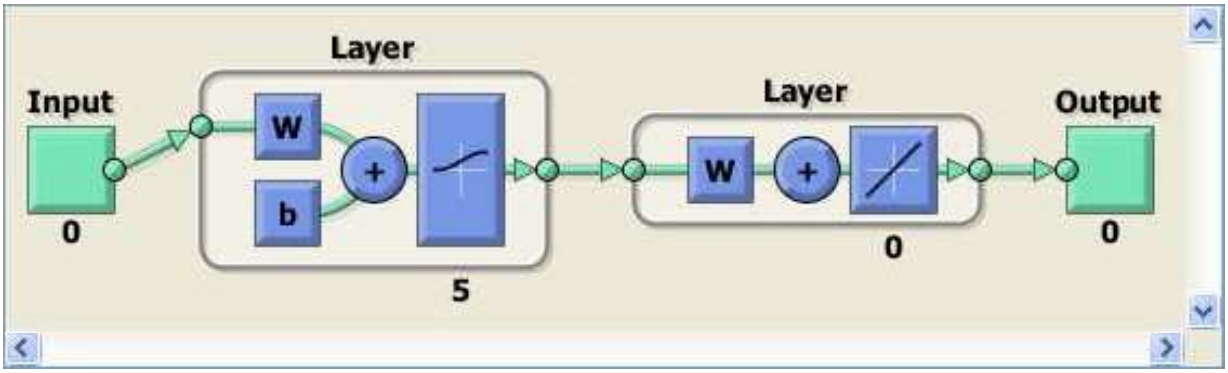
Define and custom network

```
% 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);
```



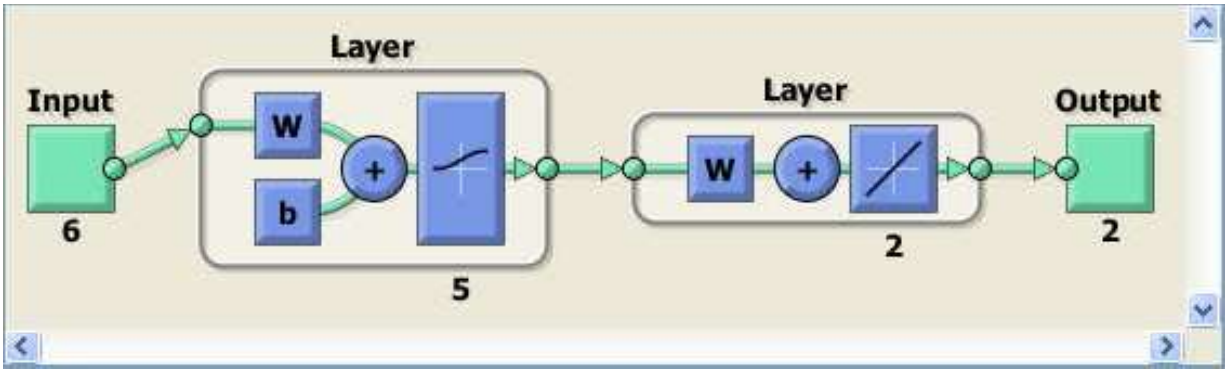
Define topology and transfer function

```
% 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

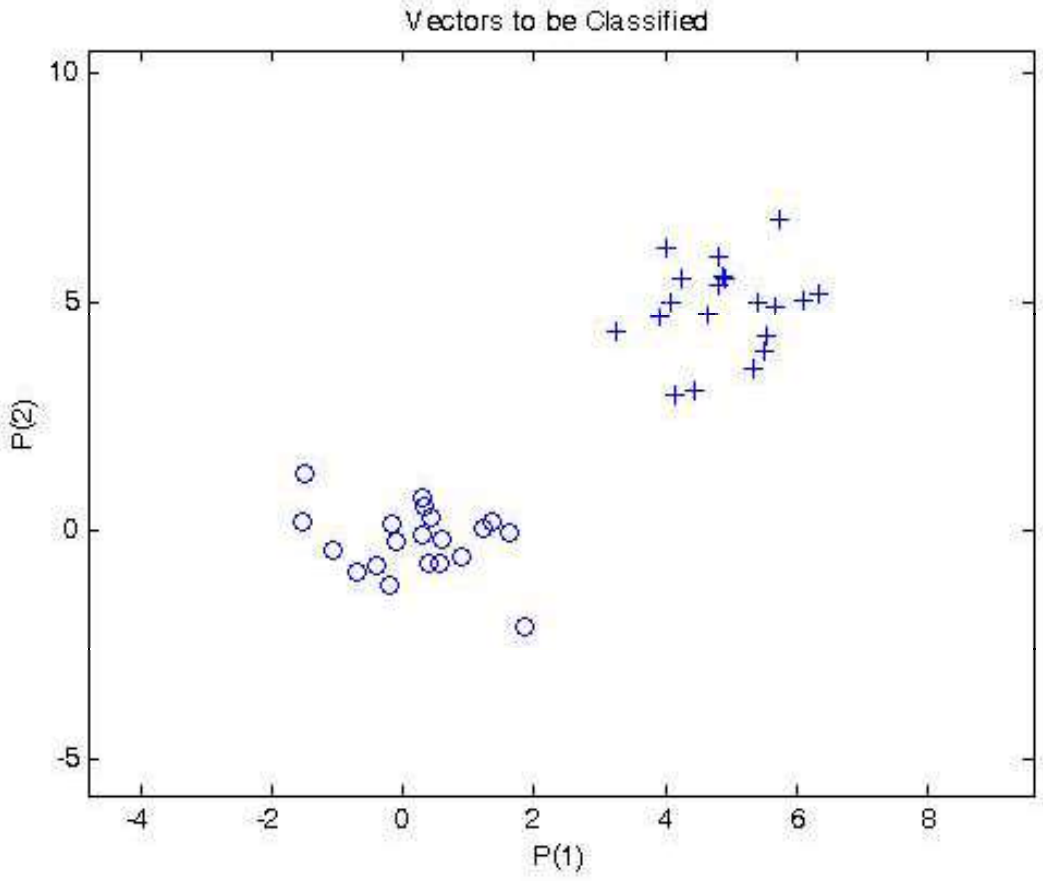
Neural Networks course (practical examples) © 2012 Primoz Potocnik

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

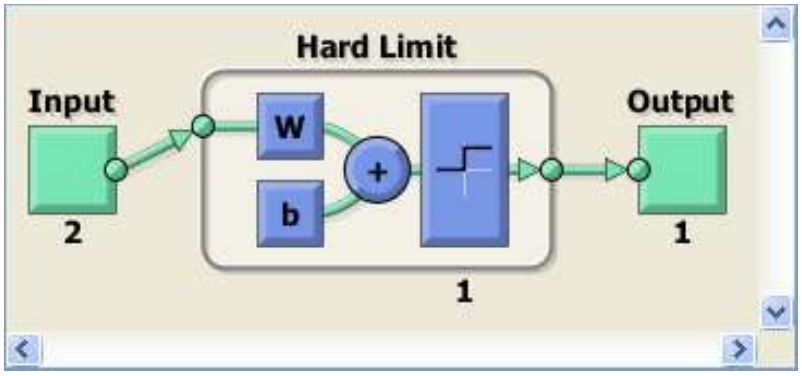
Define input and output data

```
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);
```



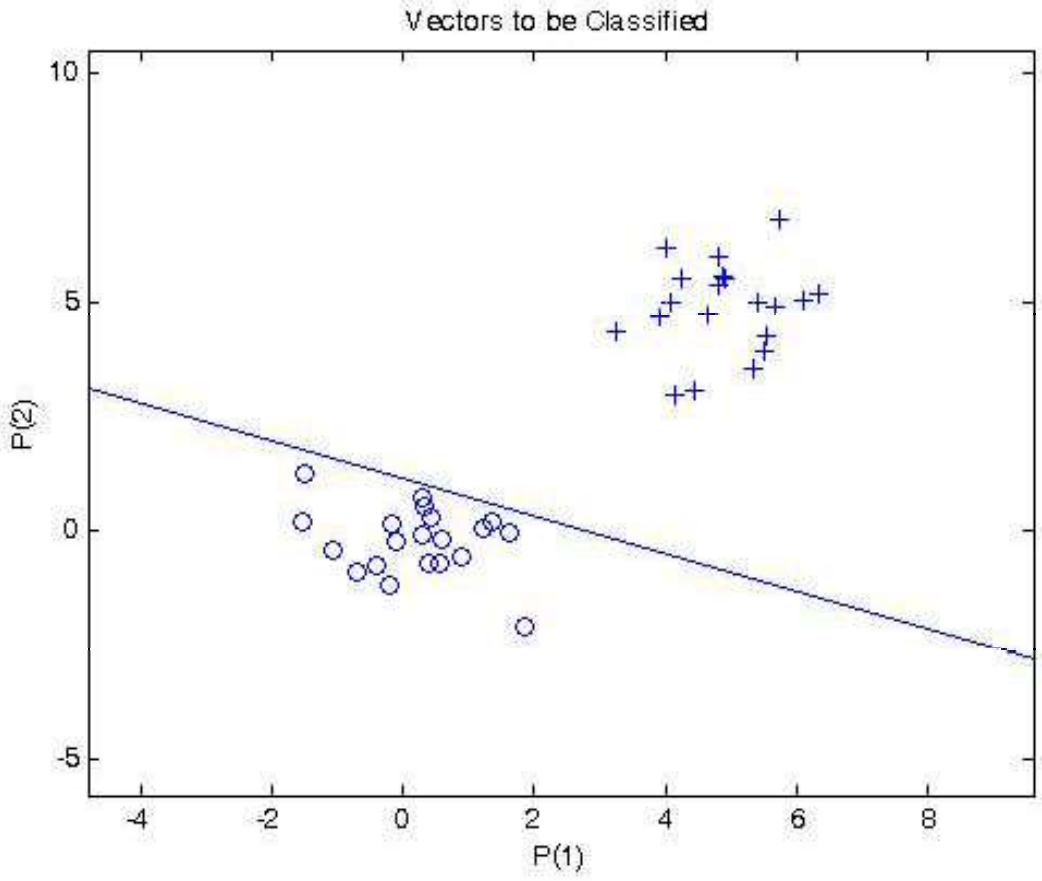
Create and train perceptron

```
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

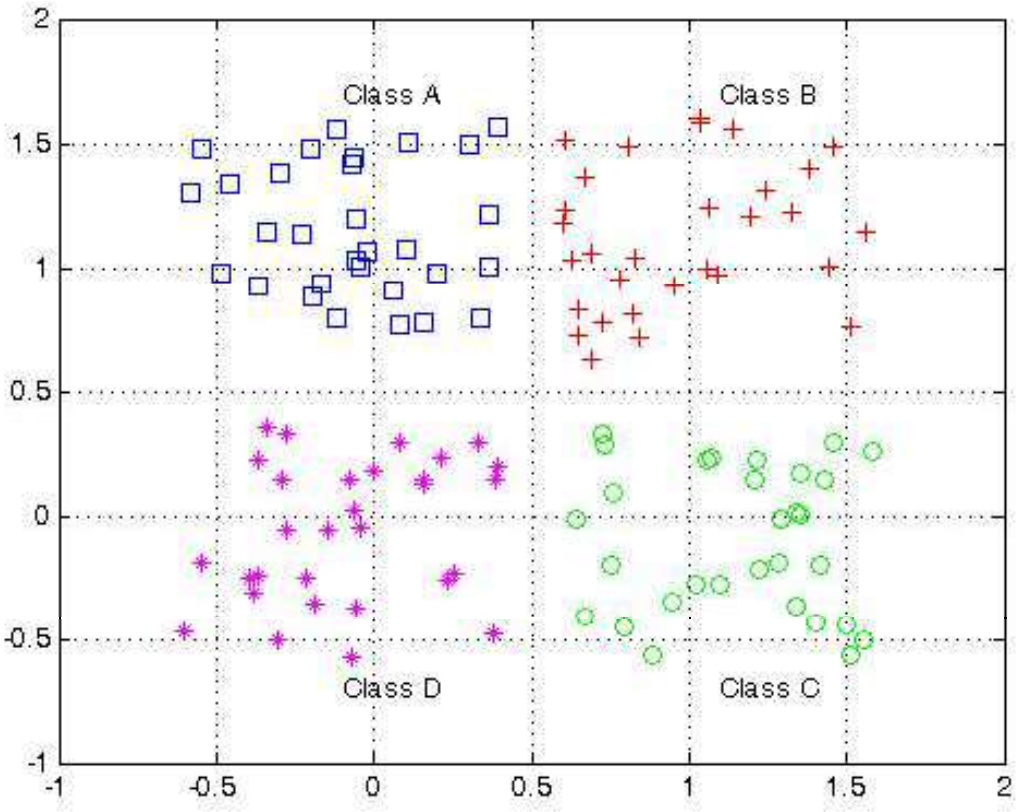
PROBLEM DESCRIPTION: Perceptron network with 2-inputs and 2-outputs is trained to classify input vectors into 4 categories

Contents

Define data

```
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);
```

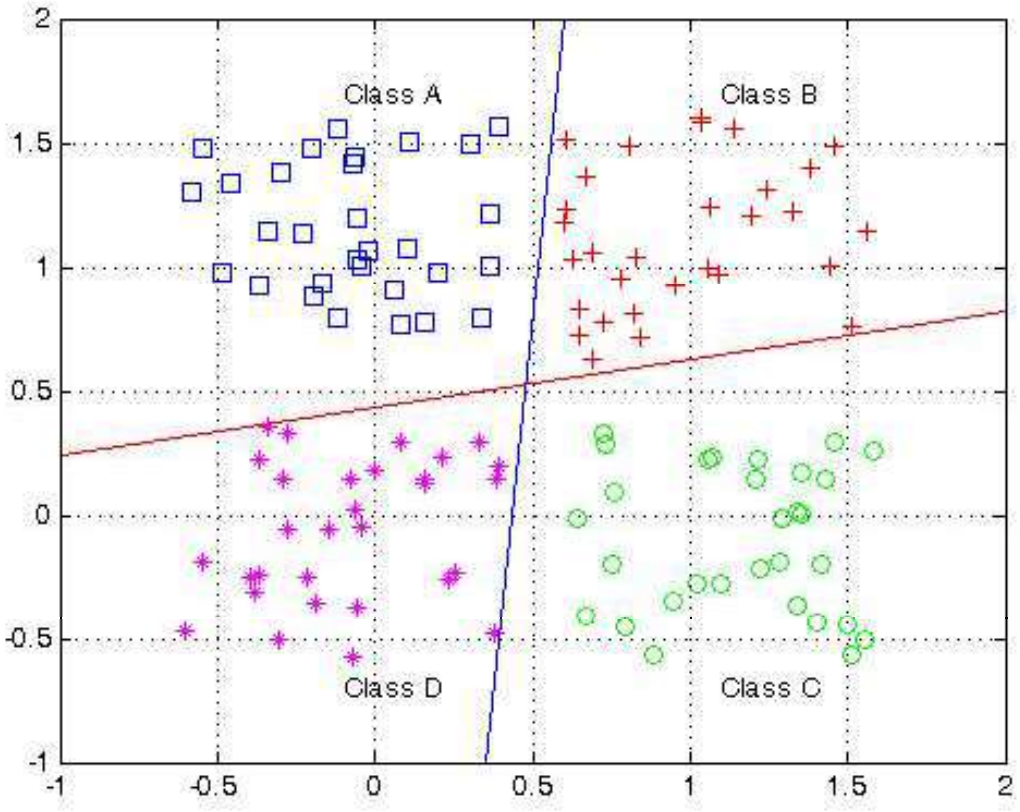
Create a perceptron

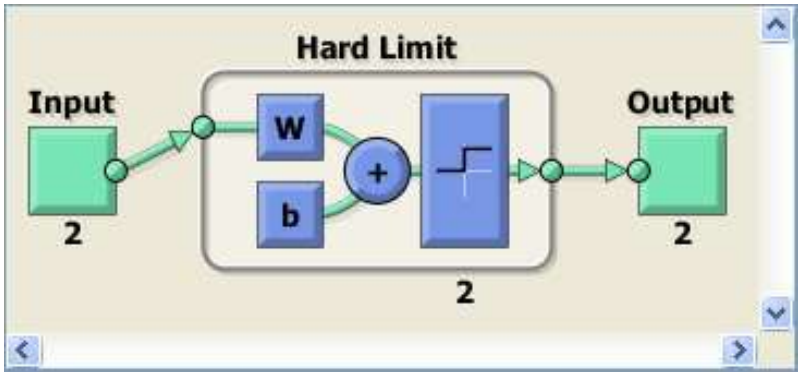
```
net = perceptron;
```

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

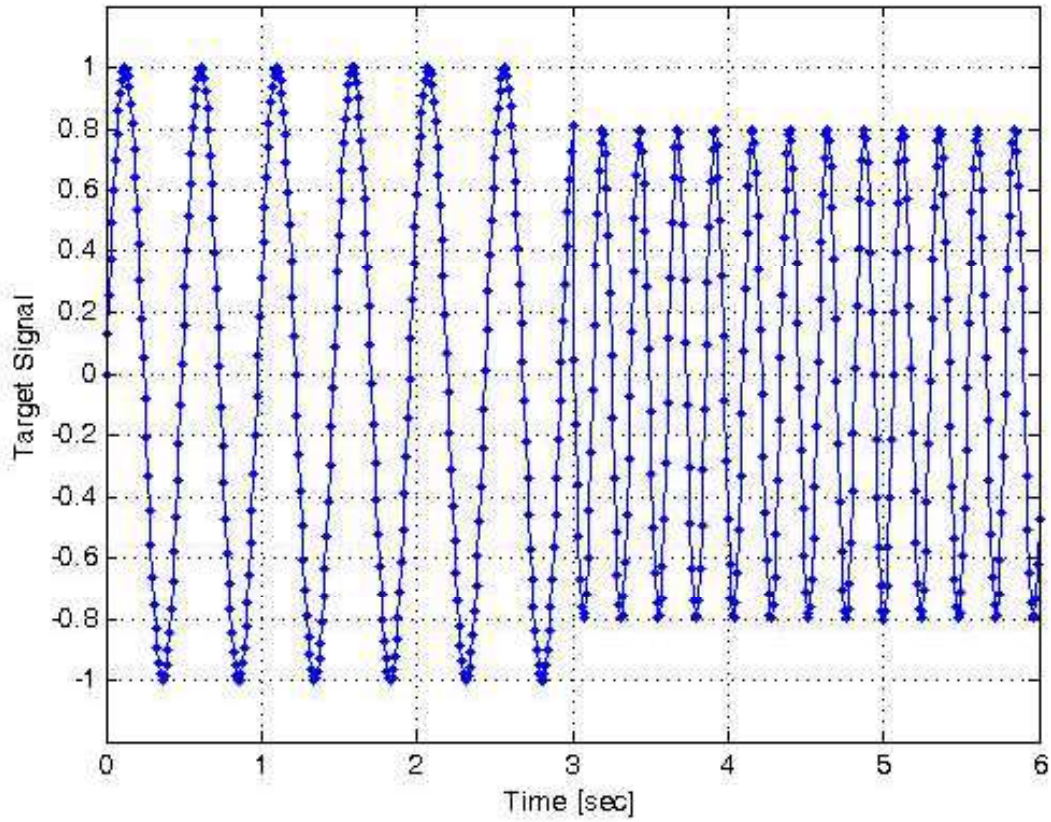
Neural Networks course (practical examples) © 2012 Primoz Potocnik

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

Contents

Define input and output data

```
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

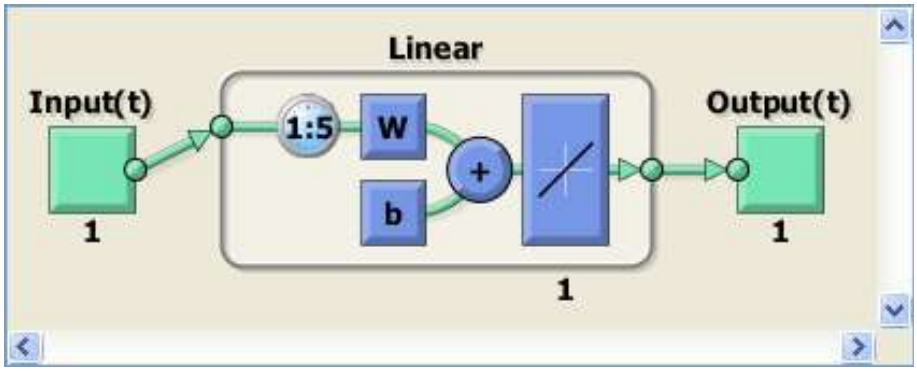
```
% The resulting network will predict the next value of the target signal
% using delayed values of the target.
inputDelays   = 1:5; % delayed inputs to be used
learning_rate = 0.2; % learning rate

% 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}
```

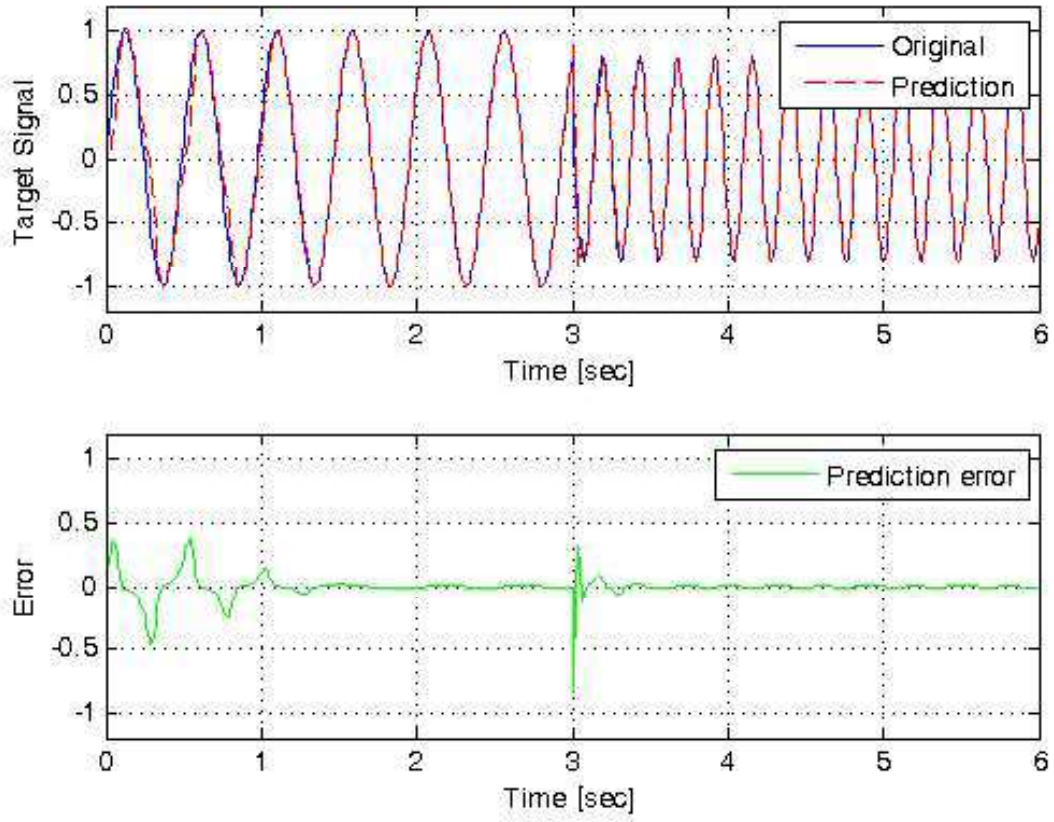
```
Weights and bias of the ADALINE after adaptation
ans =
    0.7179    0.4229    0.1552   -0.1203   -0.4159
ans =
  -1.2520e-08
```



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

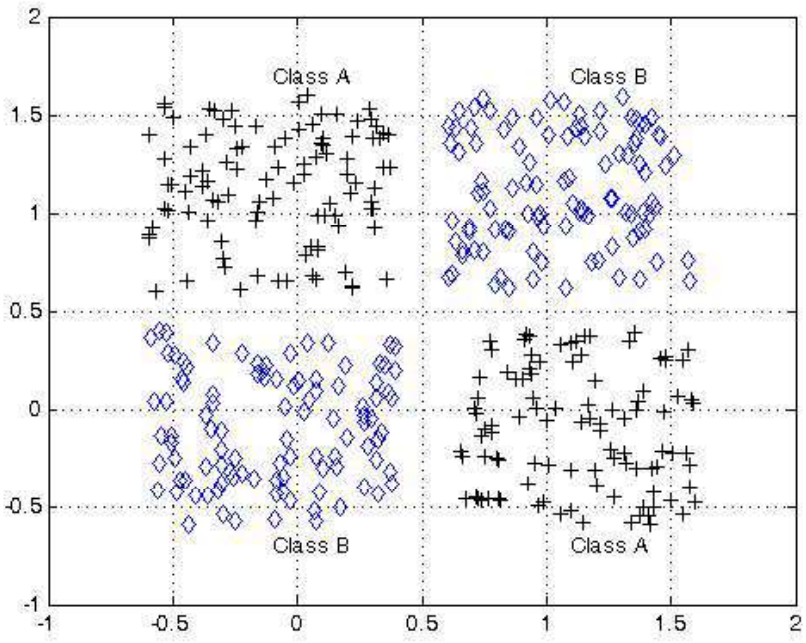
Neural Networks course (practical examples) © 2012 Primoz Potocnik

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

```
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

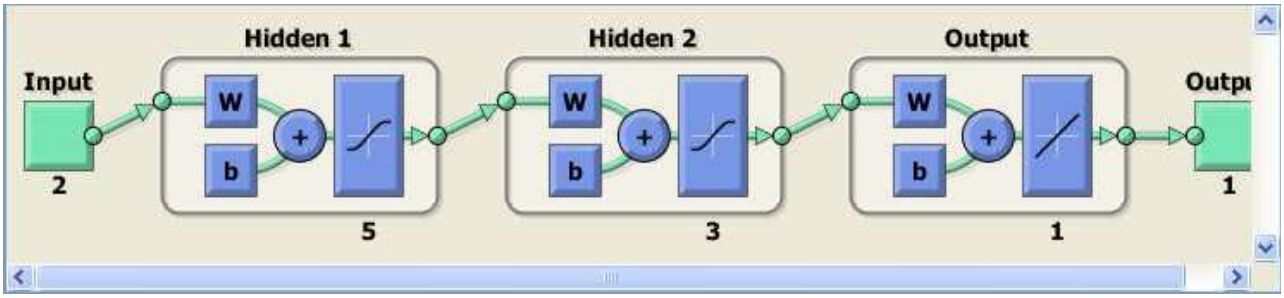
```
% encode clusters a and c as one class, and b and d as another class
a = -1; %  a  |  b
c = -1; %  -------
b =  1; %  d  |  c
d =  1; %
```

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']
```

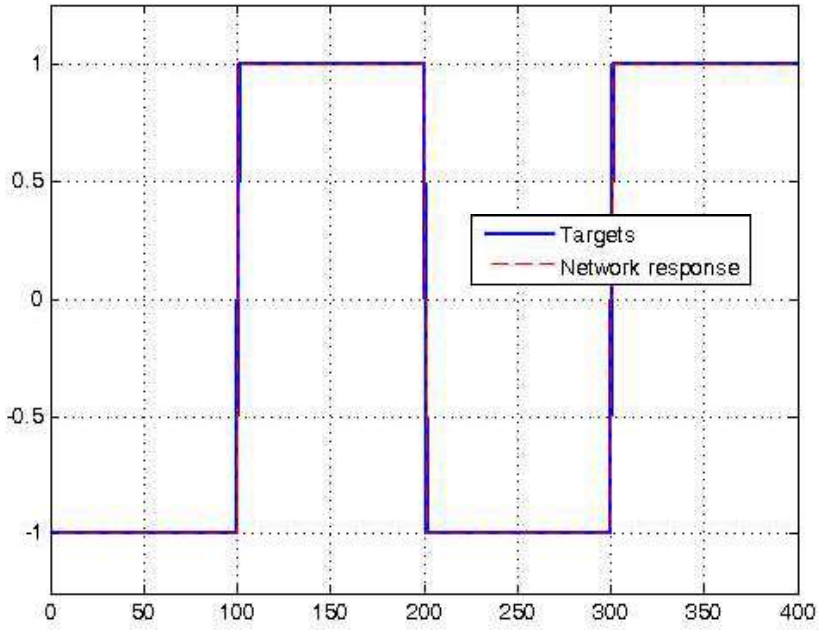
Create and train a multilayer perceptron

```
% 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

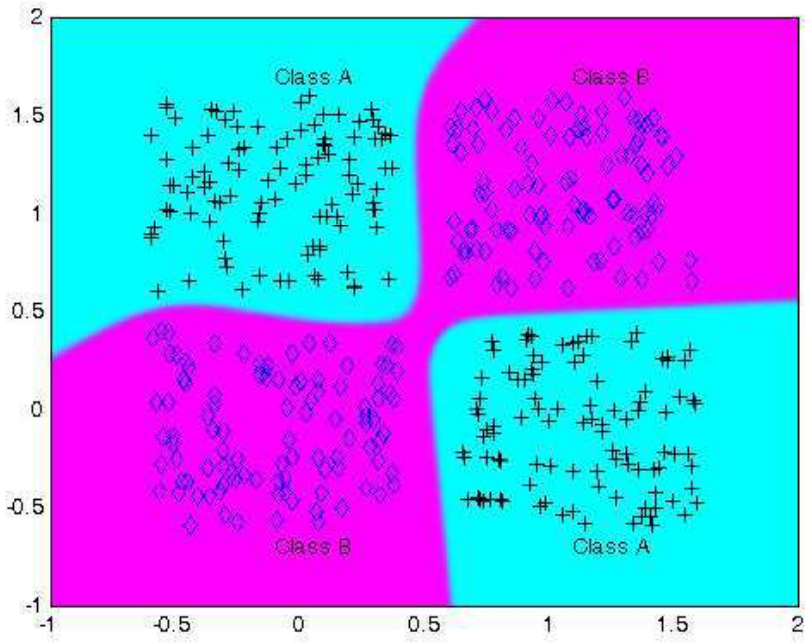
```
figure(2)
plot(T','linewidth',2)
hold on
plot(Y','r--')
grid on
legend('Targets','Network response','location','best')
ylim([-1.25 1.25])
```



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

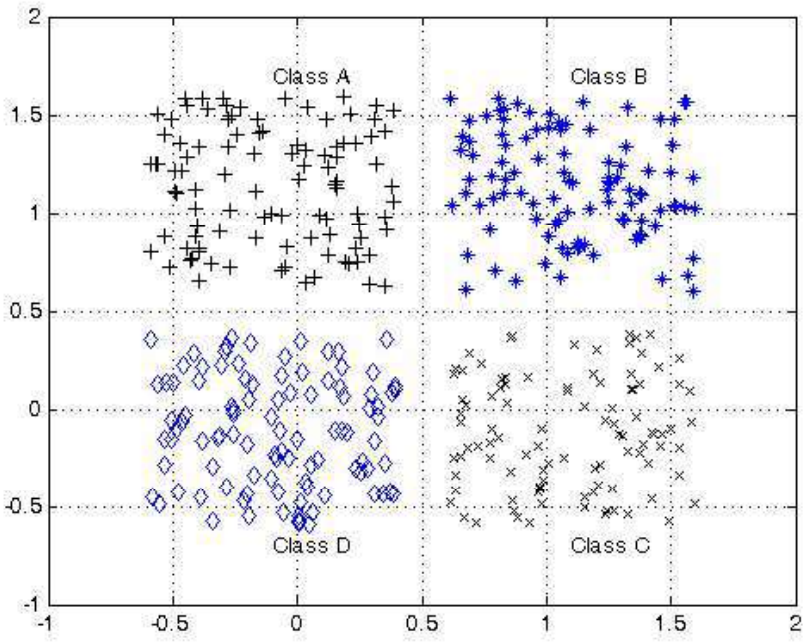
Neural Networks course (practical examples) © 2012 Primoz Potocnik

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

```
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]';
```

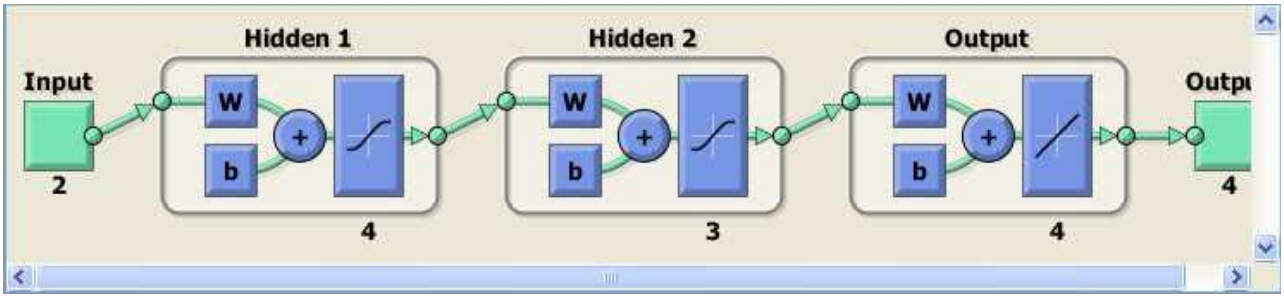
Prepare inputs & outputs for network training

```
% 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);
```

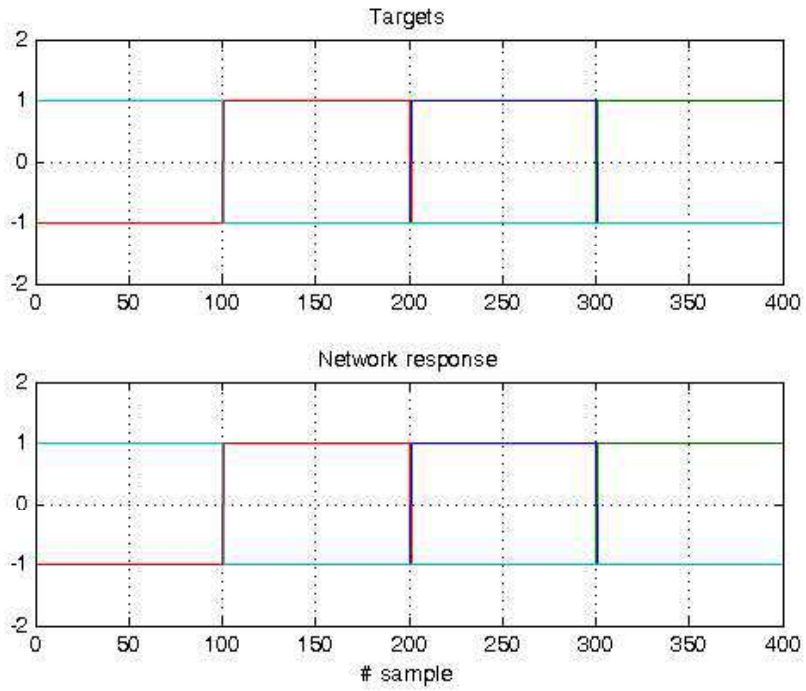
```
% show network
view(net)
```



Evaluate network performance and plot results

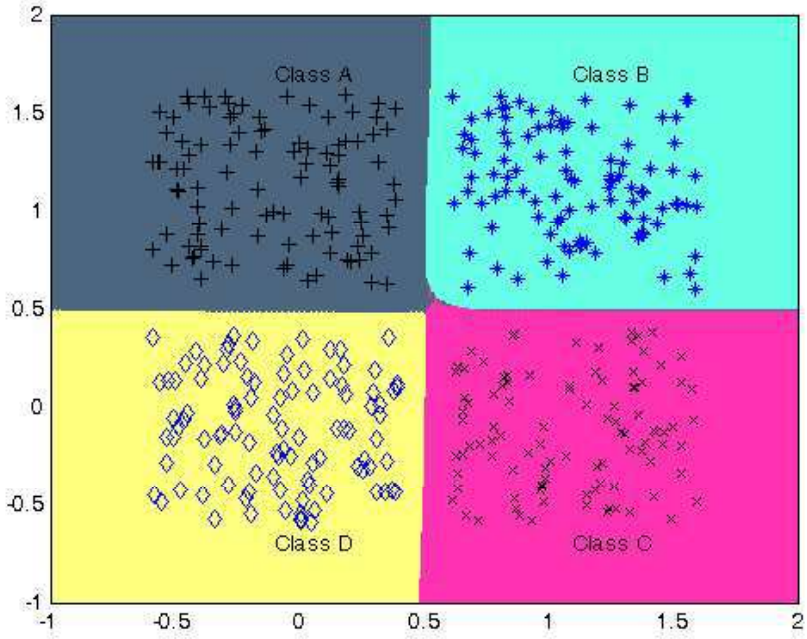
```
% evaluate performance: decoding network response
[m,i] = max(T); % target class
[m,j] = max(Y); % predicted class
N = length(Y);  % number of all samples
k = 0;          % number of missclassified samples
if find(i-j),   % if there exist missclassified samples
  k = length(find(i-j)); % get a number 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

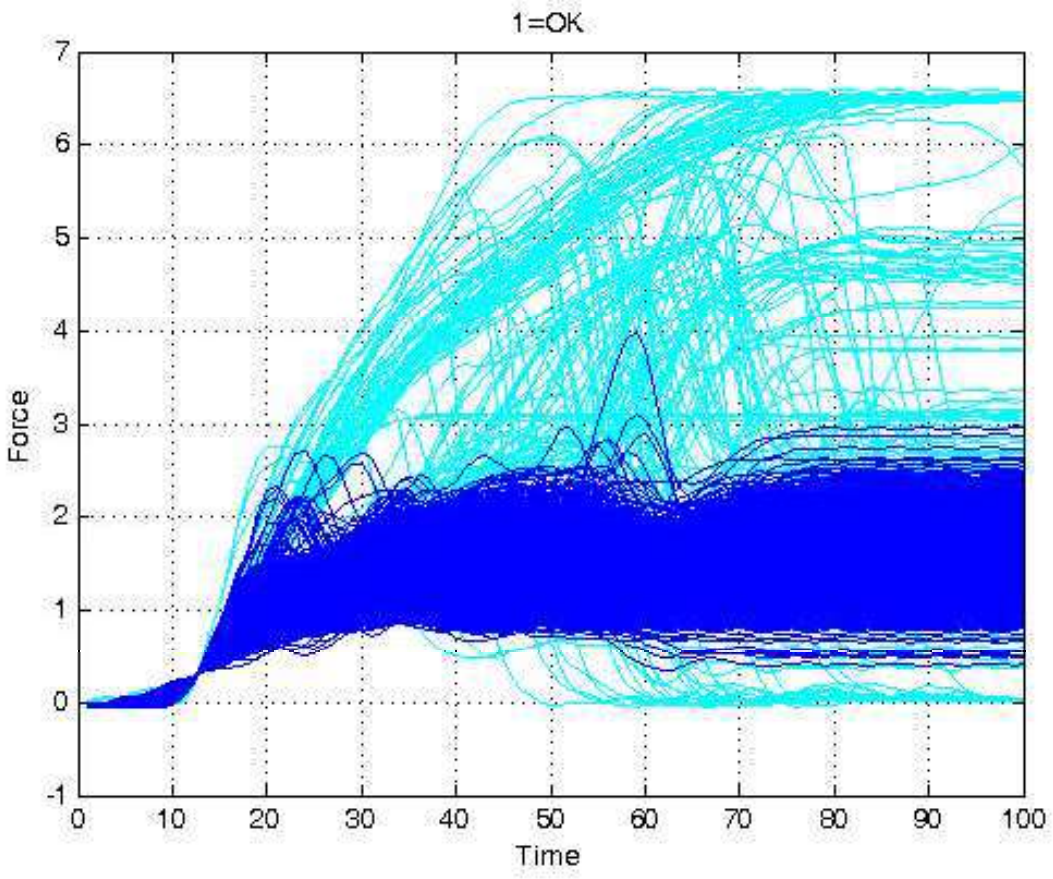
Photos of the broken connection rod

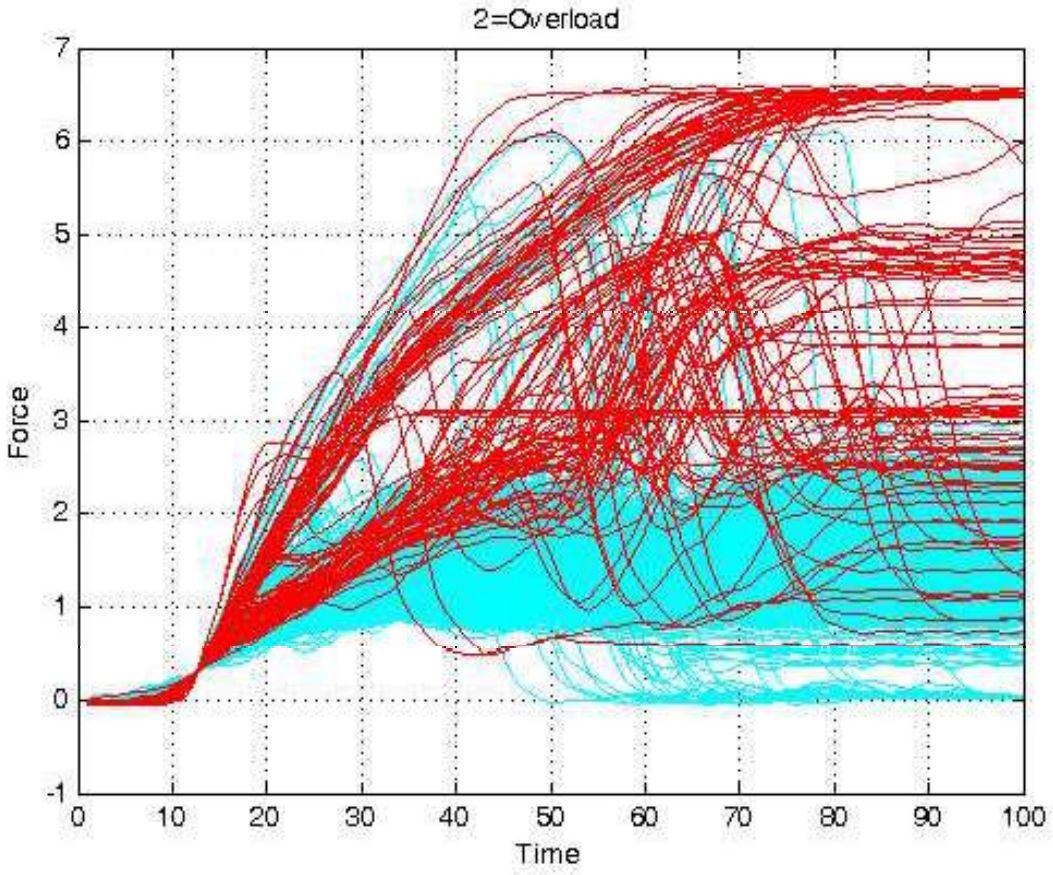
Load and plot data

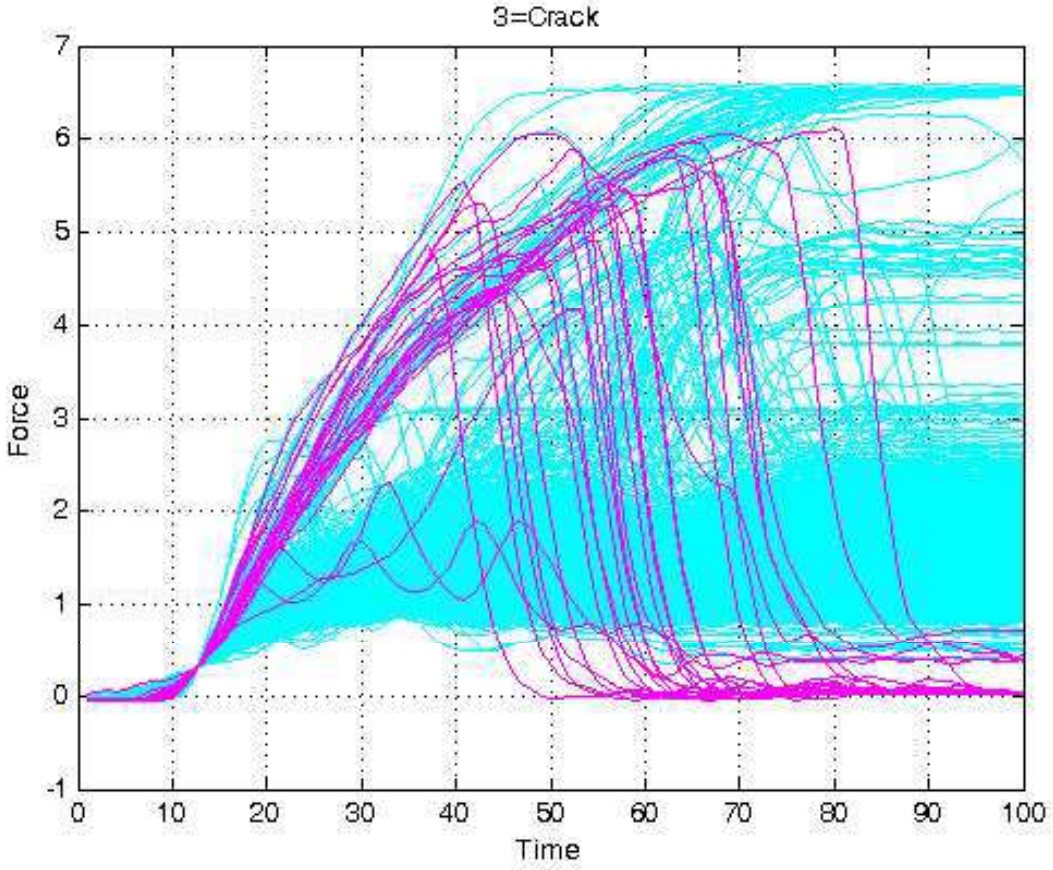
```
close all, clear all, clc, format compact

% 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
```



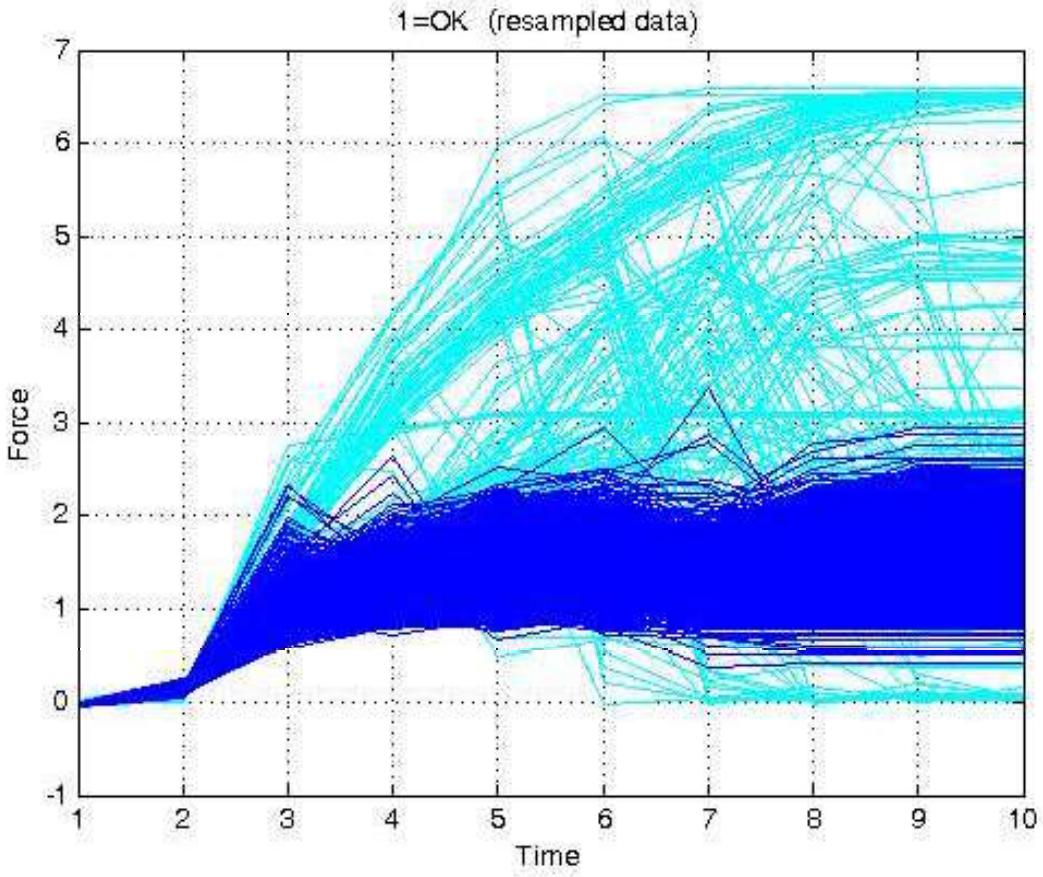


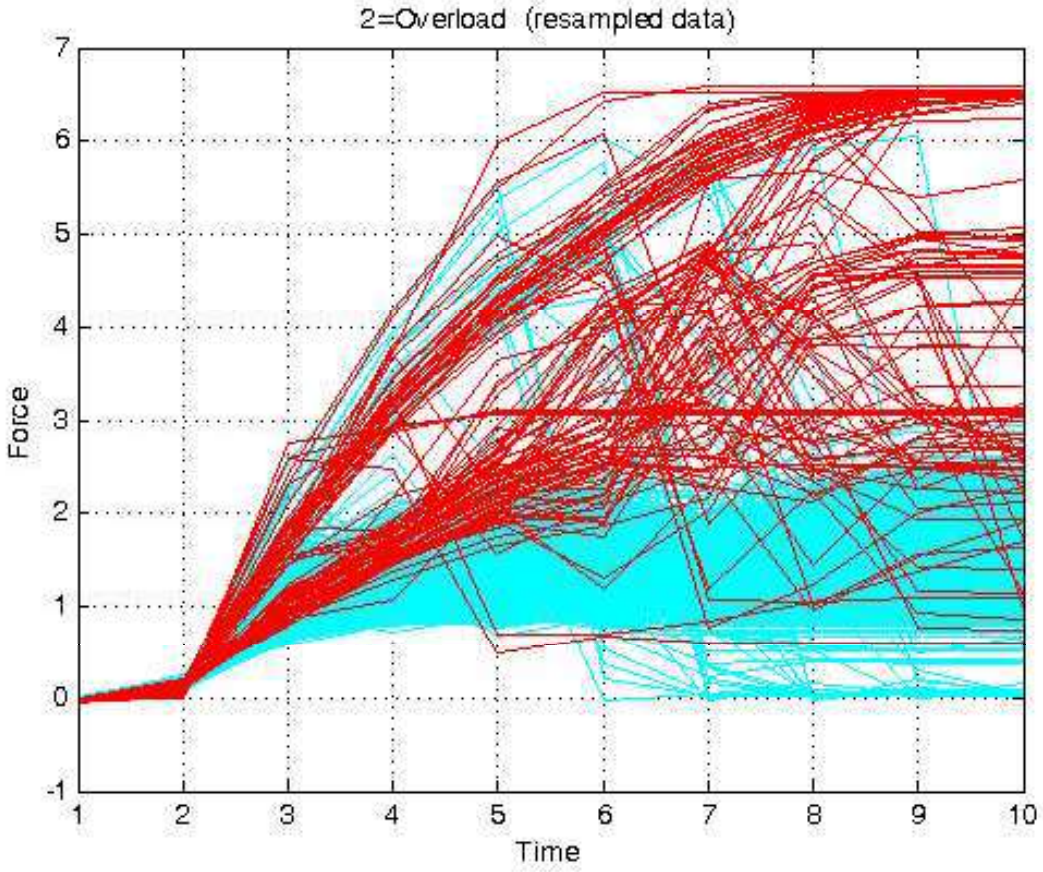


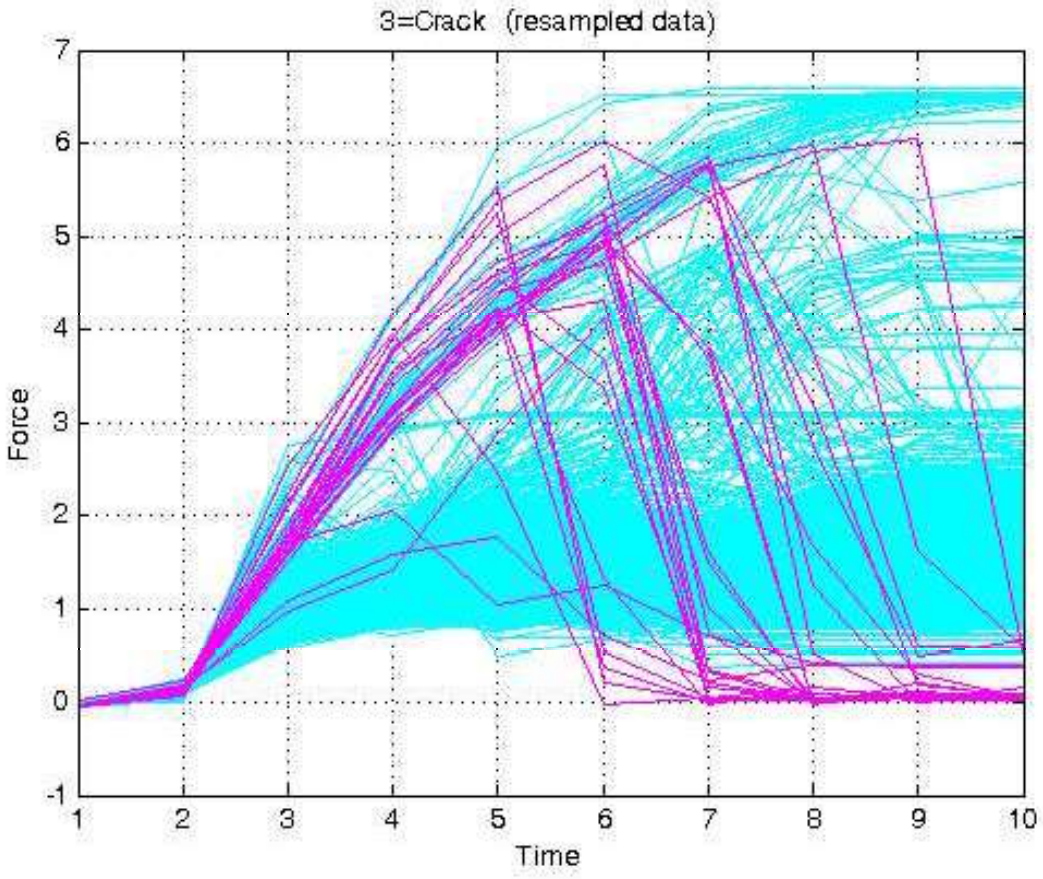
Prepare inputs: Data resampling

```
% 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)';
```

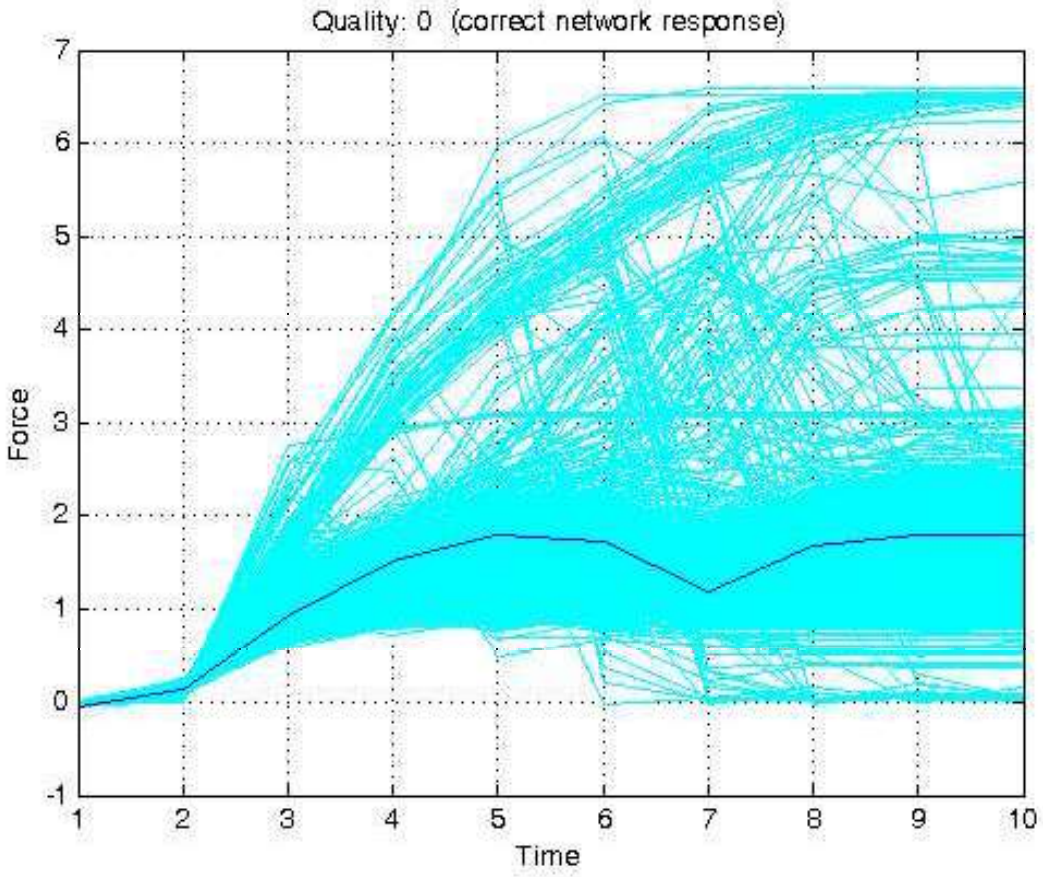
```
% find percentage of correct classifications
correct_classifications = 100*length(find(Y==target))/length(target)
```

```
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