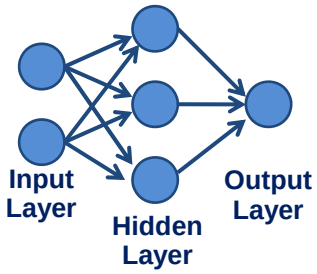
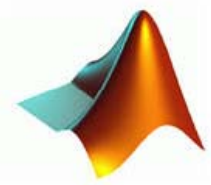
Artificial Neural Network II
MATLAB Neural Network Toolbox

Werapon Chiracharit
Department of Electronic and Telecommunication Engineering
King Mongkut's University of Technology Thonburi





MATrixLABoratory

```
>> A=[1 2 3; 4 5 6]
>> B=[1 2; 3 4; 5 6]
>> C=A'+2*B
>> D=A*B
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times3}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3\times2}$$

$$\begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}_{2\times2}$$

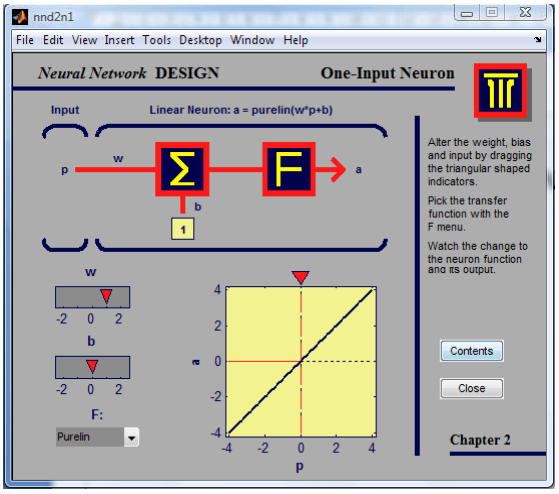
$$\begin{bmatrix} 3 & 8 \\ 8 & 13 \\ 13 & 18 \end{bmatrix}_{3\times2}$$

MATLAB Demos (1)

>>demos

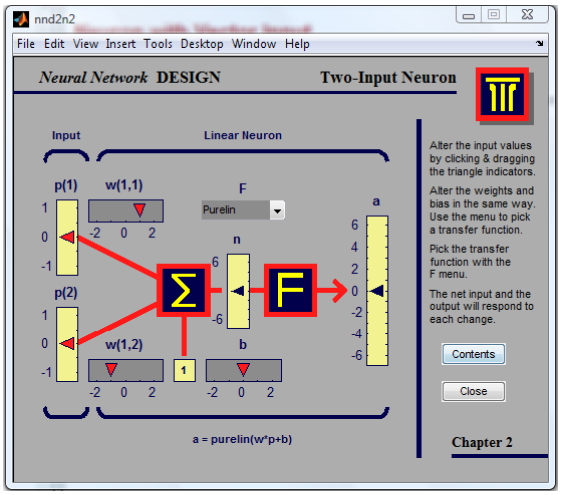
Toolboxes\Neural Network\Simple Neuron and Transfer Function

- Run this demo
  nnd2n1 (M-GUI)
- To understand
  - Weight (slope)
  - Bias (y-intercept)
  - Transfer function



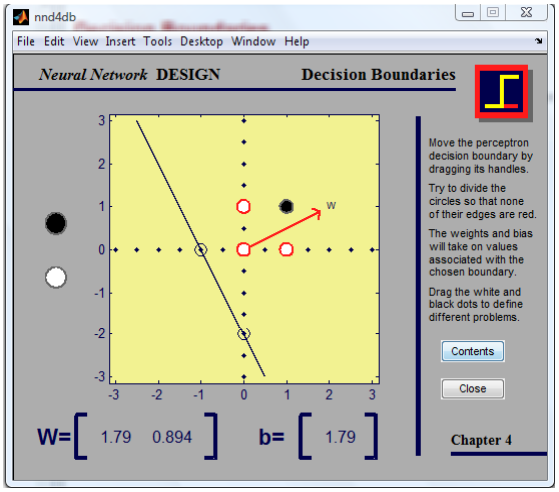
MATLAB Demos (2)

- Run nnd2n2
- To understand
  2-input neuron



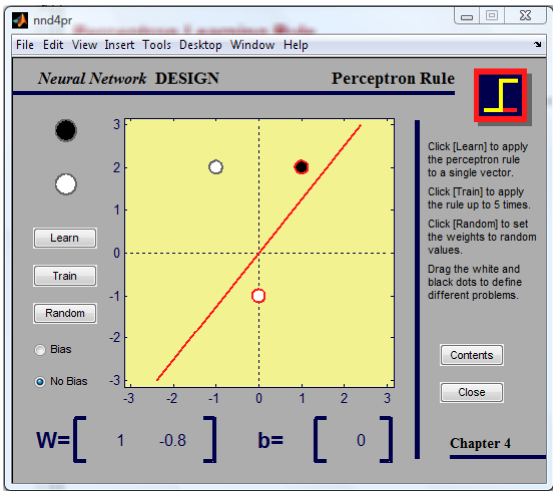
MATLAB Demos (3)

- Run nnd4db
- To understand Decision boundary



MATLAB Demos (4)

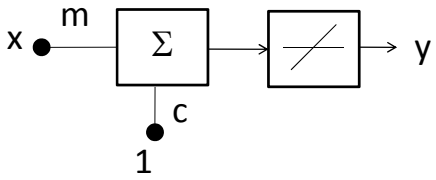
- Run nnd4pr
- To understand Perceptron

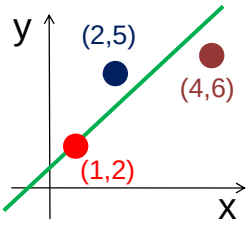


Neural Network Design

- Collect data
- Create a network
- Configure the network
- Initialize the weights and biases
- Train the network
- Validate the network
- Test the network

Line Fitting (1)





Known: x, y (training set)

Unknown: m, c

No.	x	y_{target}	$y_{calculated}$ = purelin(mx+c)
1	1	2	m+c
2	2	5	2m+c
3	4	6	4m+c

Line Fitting (2)

- Define cost function or error to be minimized,

  $E = \sum_{points} [y_{target} - y_{calculated}]^2$

  $= [2 - (m+c)]^2 + [5 - (2m+c)]^2 + [6 - (4m+c)]^2$

  $= [2 - m - c]^2 + [5 - 2m - c]^2 + [6 - 4m - c]^2$

- From gradient descent,

  $\partial E/\partial m = -2[2-m-c] - 4[5-2m-c] - 8[6-4m-c]$

  $= -72 + 42m + 14c$

  $\partial E/\partial c = -2[2-m-c] - 2[5-2m-c] - 2[6-4m-c]$

  $= -26 + 14m + 6c$

Line Fitting (3)

- Adaption with learning rate $0 < \alpha < 1$

  $m(t+1) = m(t) - \alpha\, \partial E/\partial m$

  $c(t+1) = c(t) - \alpha\, \partial E/\partial c$

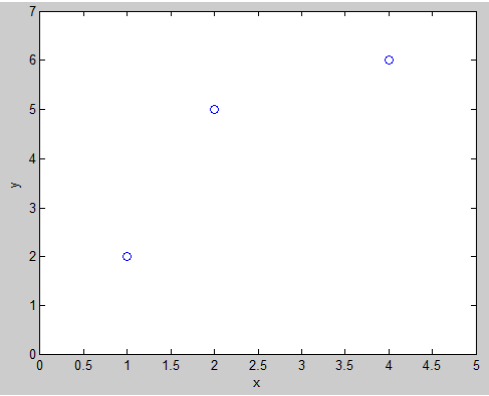
- Initialize m(0) [-1,1] and c(0) [-1,1] randomly and repeat adaption for N iterations

Line Fitting (4)

```
% Initialize randomly slope and y-intercept
>> m=unifrnd(–1,1)
m = –0.1565
>> c=unifrnd(–1,1)
c = 0.8315
% Learning rate
>> alpha=0.01;
% Number of iterations
>> iteration=10;
```


Line Fitting (5)

```
>> x=[1 2 4]; y=[2 5 6]; plot( x, y, 'o'), hold on
>> xlabel('x'), ylabel('y'), axis([0 5 0 7])
```

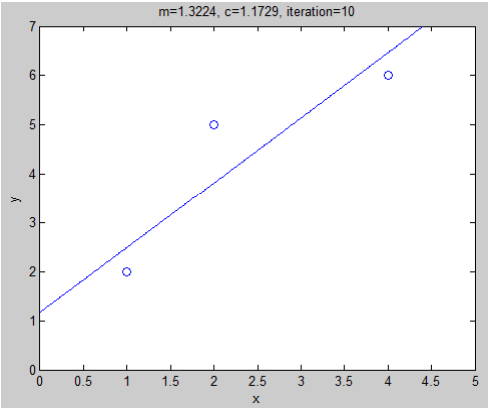


Line Fitting (6)

```
% Adaption with gradient descent
>> for i=1:iteration
        m=m–alpha*(–72+42*m+14*c);
        c=c–alpha*(–26+14*m+6*c);
   end
% Final slope and y=intercept
>> m, c
m = 1.3224
c = 1.1729
```

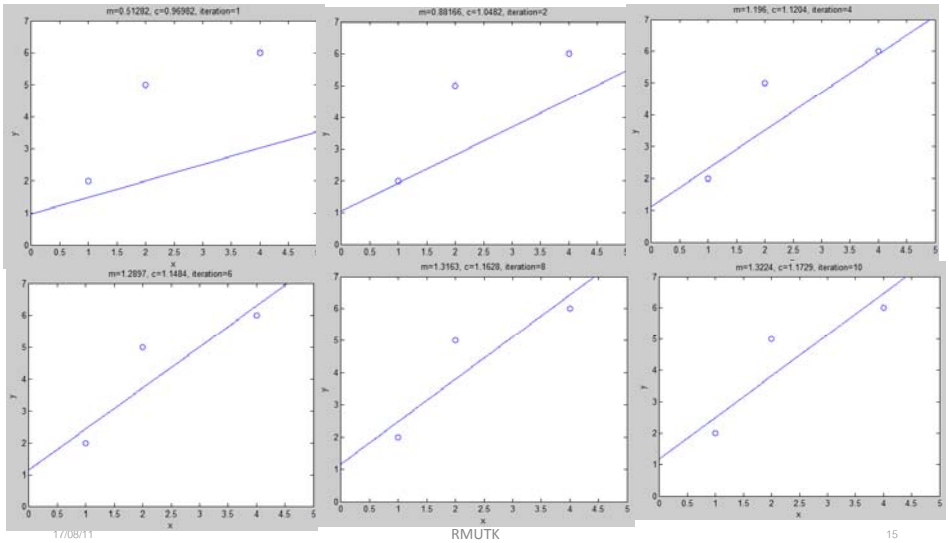

Line Fitting (7)

```
>> x2=0:0.01:5; y2=m*x2+c; plot(x2, y2)
>> title(strcat('m=', num2str(m), ', c=', ...
   num2str(c), ', iteration=', num2str(iteration)))
```

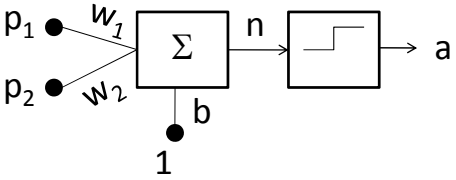


Line Fitting (8)

- Convergence for 1, 2, 4, 6, 8 and 10 iterations

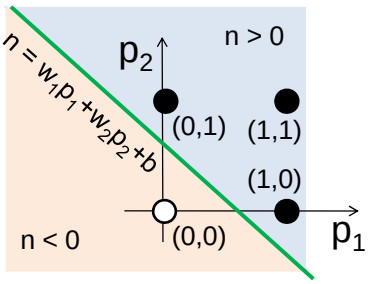


Perceptron for OR-Problem (1)



Known: p_1, p_2, t

Unknown: w_1, w_2, b



p_1	p_2	t	a = hardlim($w_1p_1+w_2p_2+b$)
0	0	0	hardlim(b)
0	1	1	hardlim(w_2+b)
1	0	1	hardlim(w_1+b)
1	1	1	hardlim(w_1+w_2+b)

Perceptron for OR-Problem (2)

- Define error function,

  $E = \sum_{i=1\rightarrow4} (t_i - a_i)^2$

  $= [\ 0 - \text{hardlim}(b)\ ]^2 + [\ 1 - \text{hardlim}(w_2+b)\ ]^2$

  $+ [1 - \text{hardlim}(w_1+b)\ ]^2 + [\ 1 - \text{hardlim}(w_1+w_2+b)\ ]^2$

- Gradient descent,

  $\partial E/\partial w_1 = 0 + 0 - 2\ [\ 1 - \text{hardlim}(w_1+b)\ ]$

  $- 2\ [\ 1 - \text{hardlim}(w_1+w_2+b)\ ]$

  $= 2\ \text{hardlim}(w_1+b) + 2\ \text{hardlim}(w_1+w_2+b) - 4$

  $\partial E/\partial w_2 = 2\ \text{hardlim}(w_2+b) + 2\ \text{hardlim}(w_1+w_2+b) - 4$

Perceptron for OR-Problem (3)

$\partial E/\partial b = 2\ \text{hardlim}(b) + 2\ \text{hardlim}(w_2+b)$

$+ 2\ \text{hardlim}(w_1+b) + 2\ \text{hardlim}(w_1+w_2+b) - 6$

- Adaption with $0 < \alpha < 1$

  $w_1(t+1) = w_1(t) - \alpha\ \partial E/\partial w_1$

  $w_2(t+1) = w_2(t) - \alpha\ \partial E/\partial w_2$

  $b(t+1) = b(t) - \alpha\ \partial E/\partial b$

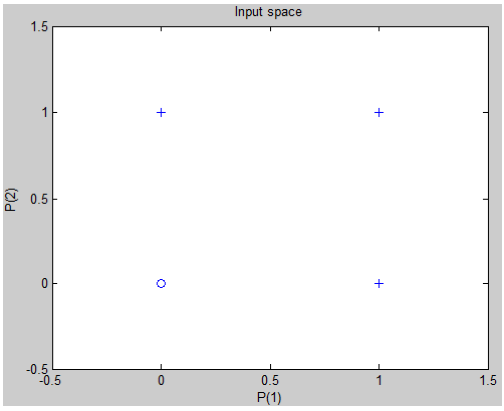
- Initialize weights [-1,1] and bias [-1,1] randomly and repeat adaption until $|t - a| \leq \gamma$

Perceptron for OR-Problem (4)

```
% Initialize weights and bias
>> w1=unifrnd(–1,1), w2=unifrnd(–1,1)
w1 = 0.5075
w2 = –0.2391
>> b=unifrnd(–1,1)
b = 0.1356
>> alpha=0.1;   % Learning rate
>> gamma=0;   % Threshold
>> plot( [0 1 1], [1 0 1], '+'), hold on
```


Perceptron for OR-Problem (5)

```
>> plot(0, 0, 'o')
>> axis([–0.5 1.5 –0.5 1.5]), title('Input space')
>> xlabel('P(1)')
>> ylabel('P(2)')
% Define target and
output
>> T=[0 1 1 1];
>> y=[0 0 0 0];
>> iteration=1;
```



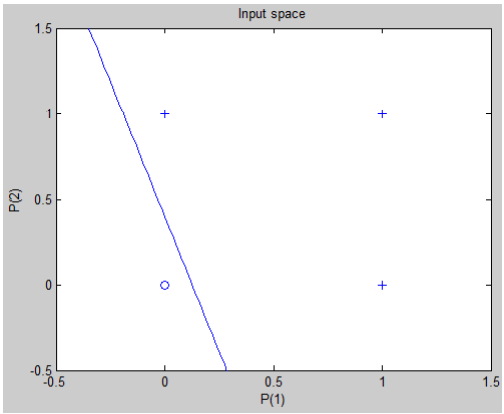
Perceptron for OR-Problem (6)

```
% Adaption
>> while sum(abs(T–y)) > gamma
        a1=hardlim(b);  a2=hardlim(w2+b);
        a3=hardlim(w1+b);  a4=hardlim(w1+w2+b);
        y=[a1 a2 a3 a4];  E(iteration)=sum((T–y).^2);
        w1=w1–alpha*(a3+a4–2);
        w2=w2–alpha*(a2+a4–2);
        b=b–alpha*(a1+a2+a3+a4–3);
        iteration=iteration+1;
end
```


Perceptron for OR-Problem (7)

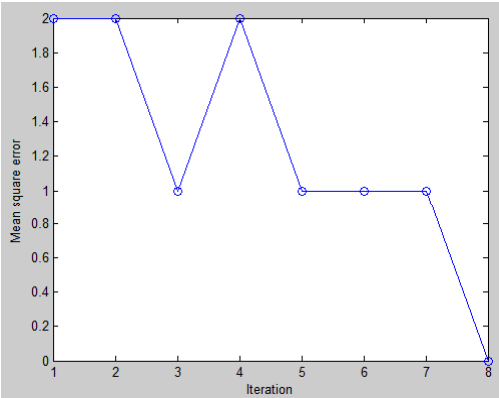
```
>> w1, w2, b, iteration
w1 = 0.5075
w2 = 0.1609
b = –0.0644
iteration = 8

>> p1=–0.5:0.1:1.5;
>> p2=–(w1*p1+b)/w2;
>> plot(p1, p2), hold off
```



Perceptron for OR-Problem (8)

```
% Error
>> figure, plot(E, 'o-')
>> xlabel('Iteration'), ylabel('Mean square error')
```

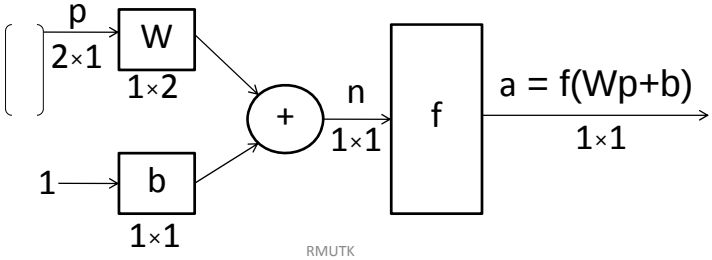


Data Structures (1)

Input/target data format for training and simulation

e.g. 2-input neuron and 4 elements for each input

$$p_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}_{4\times1} \quad p_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}_{4\times1} \quad t = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}_{4\times1}$$



Data Structures (2)

- Batch training is to update the weights after each presenting the complete data set

  >> p=[0 0 1 1; 0 1 0 1];

  >> T=[0 1 1 1];

- Incremental training is to update the weights after the presentation of each single sample

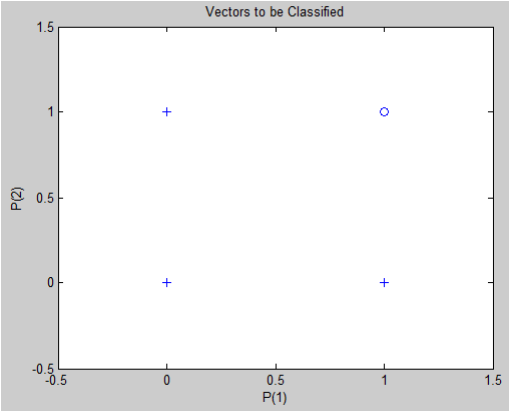
  >> p={[0;0] [0;1] [1;0] [1;1]};

  >> T={[0] [1] [1] [1]};

Neural Network Function (1)

e.g. NAND problem (command line in m-file)

```
% Define input P (4 input-vectors) and target T
>> P=[0 0 1 1; …
       0 1 0 1];
>> T=[1 1 1 0];
```




```
>> plotpv(P, T) % Plot perceptron vector
```

Neural Network Function (2)

% Create a perceptron network, input ranges N×2, number of neuron, transfer function

```
>> net=newp([0 1; 0 1], 1, 'hardlim');

>> net.IW{1}    % Initial weight
ans = 0 0
>> net.b{1}   % Initial bias
ans = 0
```


Neural Network Function (3)

% Number of adaption passing through the entire sequence

```
>> net.adaptParam.passes=5;
```

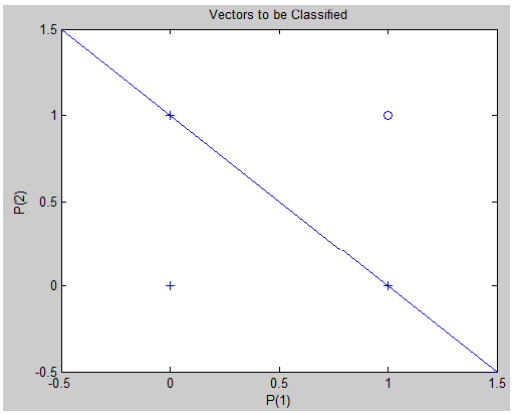
% Training with adaption algorithm, output, error

```
>> [net, y, e]=adapt(net, P, T);
>> y, e, mse(e)   % percent of training
y = 1 1 1 0
e = 0 0 0 0
ans = 0
```

Neural Network Function (4)

% Plot perceptron classification by –p(1)–p(2)+1

>> plotpc(net.IW{1}, net.b{1})



>> net.IW{1}

ans = –1 –1 % Final weight

>> net.b{1}

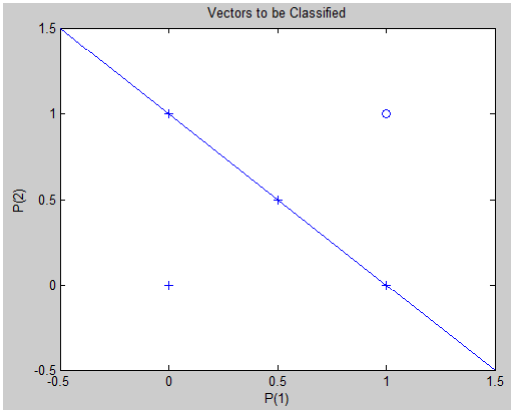
ans = 1 % Final bias

Neural Network Function (5)

% Simulation

>> P2=[0.5; 0.5];

>> y2=sim(net, P2);



>> plotpv(P2, y2)

>> hold on

>> plotpv(P, T)

>> plotpc(net.IW{1}, net.b{1})

References

- MathWorks Neural Network Toolbox webpage, http://www.mathworks.com/products/neuralnet/demos.html

Thank you for attention

Q & A