Introduction to Artificial Neural Networks

Lecture 4:

Perceptron and ADALINE

By: Ali Motie Nasrabadi

Outline

- **Introduction**
- **Perceptron**
  - **Selection of weights for the Perceptron**
  - **Perceptron Learning Theorem**
  - **Implementation of Logical Gate**
  - **Finding Weights by MSE Method: off-line**
  - **Perceptron learning law: the geometric interpretation**
  - **Convergence of the Perceptron learning law**
  - **Limitation of Perceptron**
  - **Representation of Perceptron in MATLAB**
- **ADALINE — The Adaptive Linear Element**
  - **Applications of Adaline**
  - **Error concept**
  - **Method of steepest descent**
  - **The LMS (Widrow- Hoff) Learning Law**
  - **network training**
  - **Some general comments on the learning process**
  - **The effect of learning Rate**

Introduction

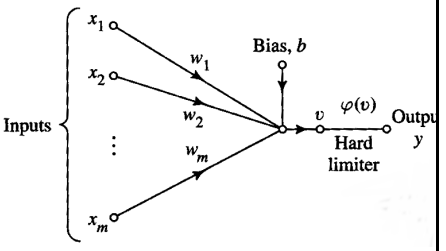
- **The Rosenblatt's *LMS algorithm* for Perceptron (1958) is built around a linear neuron (a neuron with a linear activation function).**
- **However, the *Perceptron* is built around a nonlinear neuron, namely the *McCulloch-Pitts model* of a neuron.**
  - **This neuron has a hard- limiting activation function (performing the *signum* function).**
- **Recently the term *multilayer Perceptron* has often been used as a synonym for the term *multilayer feedforward neural networ*k. In this section we will be referring to the former meaning.**

Perceptron(1)

- **Goal**
  - **classifying applied Input $x_1, x_2, \ldots, x_m$ into one of two classes**
- **Procedure**
  - **if output of hard limiter is +1, to class C_1 if it is -1, to class C_2**
    - **input of hard limiter : weighted sum of input**

$$v = \sum_{i=1}^{m} w_i x_i + b$$

  - **effect of bias b is merely to shift decision boundary away from origin**
  - **synaptic weights adapted on iteration by iteration basis**

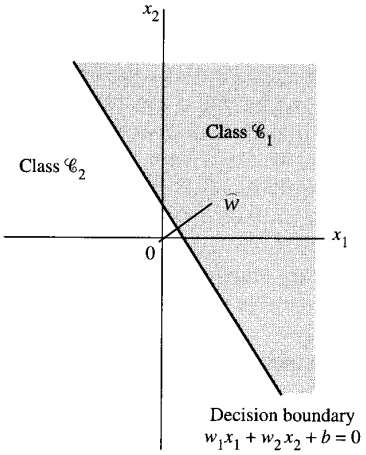


Perceptron(2)

- **Decision regions separated by a hyperplane**

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

  - **point (x1,x2) above boundary line is assigned to** C_1
  - **point (y1,y2) below boundary line to class** C_2



Lecture 4-5

Selection of weights for the Perceptron

- **In general two basic methods can be employed to select a suitable weight vector:**
  - **By** off- line calculation **of weights.**
    - **If the problem is relatively simple it is often possible to calculate the weight vector from the specification of the problem.**
  - **By** learning **procedure.**
    - **The weight vector is determined from a given (training) set of input- output vectors (exemplars) in such a way to achieve the best classification of the training vectors.**

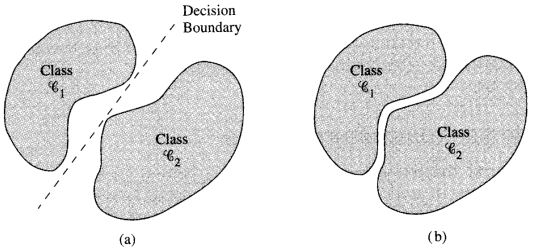
Lecture 4-6

Perceptron Learning Theorem (1)

- **Linearly separable**
  - **if two classes are linearly separable, there exists decision surface consisting of hyperplane.**
  - **If so, there exists weight vector w**

$$\mathbf{w}^T\mathbf{x} > 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } C_1$$
$$\mathbf{w}^T\mathbf{x} \le 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } C_2$$

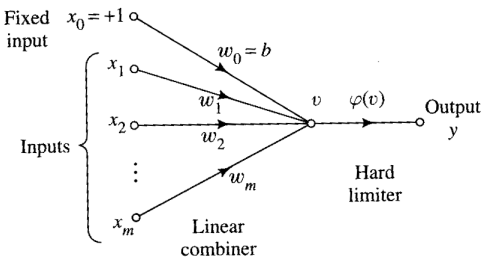
  - **for only linearly separable classes, perceptron works well**



Perceptron Learning Theorem (2)

- **Using modified signal-flow graph**
  - **bias b(n) is treated as synaptic weight driven by fixed input +1**
  - $w_0(n)$ **is** $b(n)$
  - **linear combiner output**

$$v(n) = \sum_{i=1}^{m} w_i(n)x_i(n)$$
$$= \mathbf{w}^T(n)\mathbf{x}(n)$$



Perceptron Learning Theorem (3)

- **Weight adjustment**
  - **if x(n) is correctly classified**

$$\mathbf{w}(n+1) = \mathbf{w}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_1$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) \le 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_2$$

  - **otherwise**

$$\mathbf{w}(n+1) = \mathbf{w}(n) - h(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_2$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + h(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) \le 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_1$$

  - **learning rate parameter $h(n)$ controls adjustment applied to weight vector**

Summary of Learning

1. **Initialization**
   1. **set w(0)=0**
2. **Activation**
   1. **at time step n, activate perceptron by applying continuous valued input vector x(n) and desired response d(n)**
3. **Computation of actual response**

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

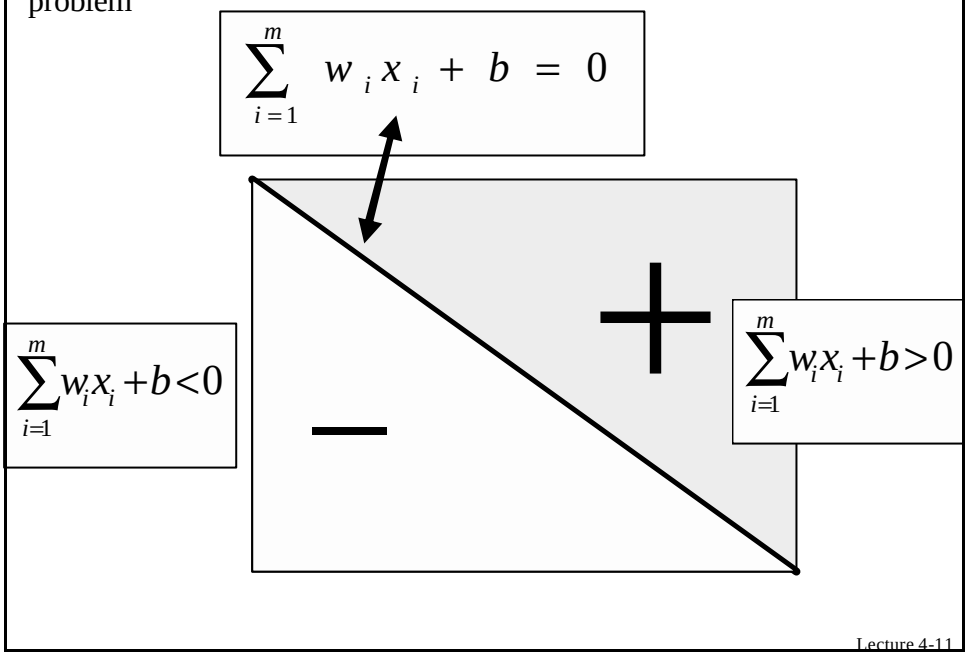
4. **Adaptation of Weight Vector**

$$\mathbf{w}(n+1) = \mathbf{w}(n) + h[d(n) - y(n)]\mathbf{x}(n)$$

$$e(n) = d(n) - y(n) : error \qquad d(n) = \begin{cases} +1 & \text{if x(n) belongs to class } C_1 \\ -1 & \text{if x(n) belongs to class } C_2 \end{cases}$$

5. **Continuation**
   1. **inclement time step n and go back to step 2**

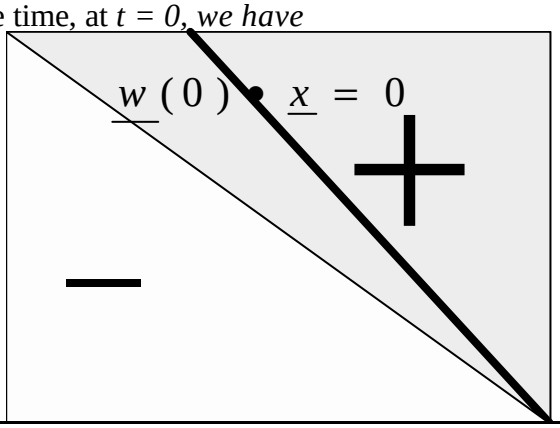
The network is capable of solving linearly separable problem



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

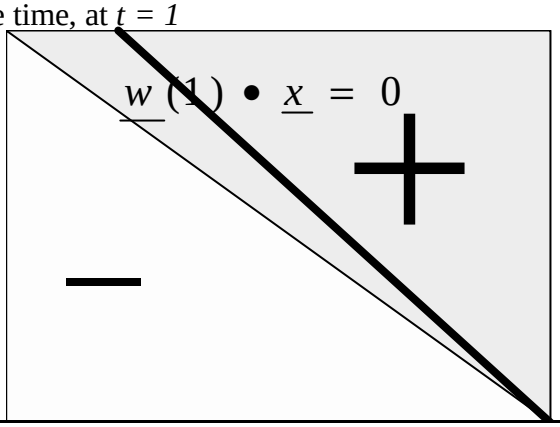
Let *t* be the time, at *t = 0, we have*



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

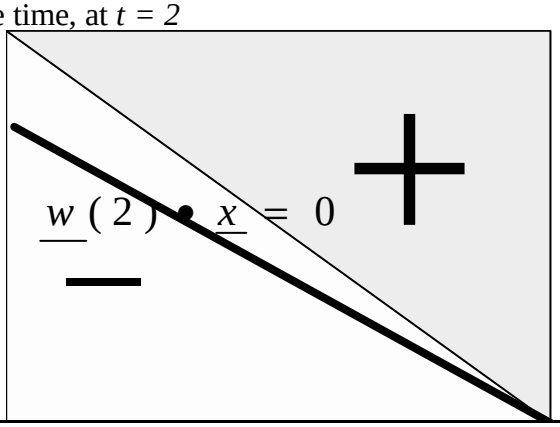
Let t be the time, at $t = 1$



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

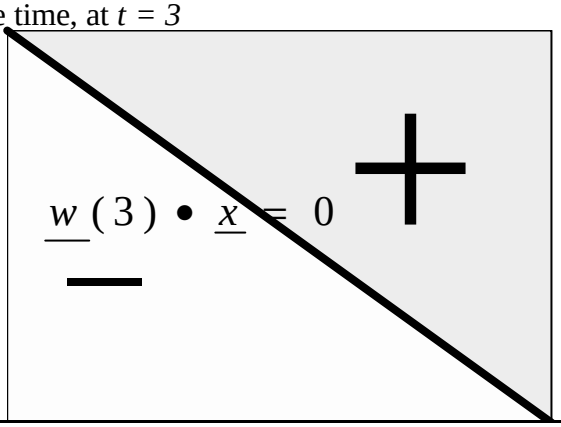
Let t be the time, at $t = 2$



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

Let t be the time, at $t = 3$



Lecture 4-15

Implementation of Logical NOT, AND, and OR

NOT

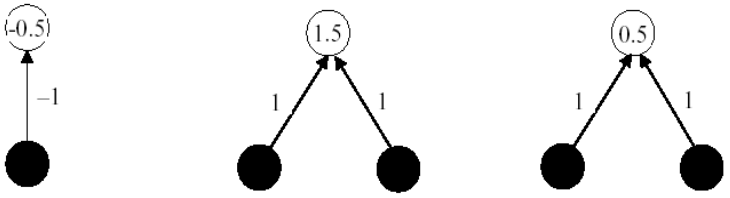
in	*out*
0	1
1	0

AND

in_1	in_2	*out*
0	0	0
0	1	0
1	0	0
1	1	1

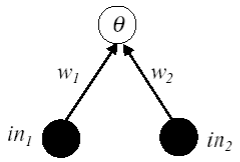
OR

in_1	in_2	*out*
0	0	0
0	1	1
1	0	1
1	1	1



Lecture 4-16

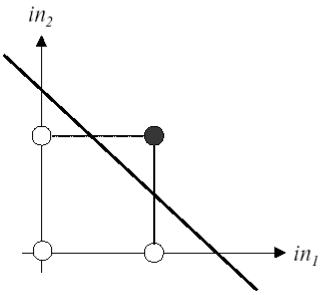
Implementation of Logical Gate



$$out = \mathrm{sgn}(w_1 in_1 + w_2 in_2 - \theta)$$

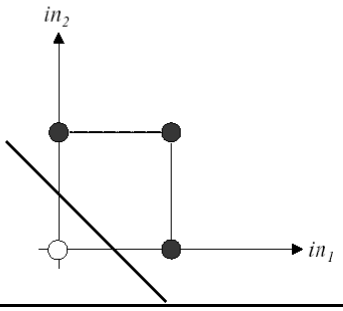
AND

$w_1 = 1,\ w_2 = 1,\ \theta = 1.5$



OR

$w_1 = 1,\ w_2 = 1,\ \theta = 0.5$



cture 4-17

Finding Weights Analytically for the AND Network: off-line

We have two weights w_1 and w_2 and the threshold θ, and for each training pattern we need to satisfy

$$out = \mathrm{sgn}(w_1 in_1 + w_2 in_2 - \theta)$$

So the training data lead to four inequalities:

in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$\Rightarrow$

$$\begin{aligned} w_1 0 + w_2 0 - \theta &< 0 \\ w_1 0 + w_2 1 - \theta &< 0 \\ w_1 1 + w_2 0 - \theta &< 0 \\ w_1 1 + w_2 1 - \theta &\geq 0 \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} \theta &> 0 \\ w_2 &< \theta \\ w_1 &< \theta \\ w_1 + w_2 &\geq \theta \end{aligned}$$

Lecture 4-18

Finding Weights by MSE Method: off-line

- **Write a equation for each training data**
- **Output for first class is +1 and for second class is -1(or 0)**
- **Apply the MSE method to solve the problem**
- **Example: Implementation of AND gate**

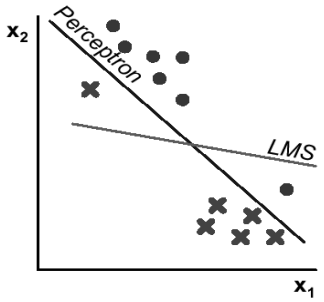
$$\sum_{i=1}^{m} w_i x_i + b = 0$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \quad \Rightarrow \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1.5 \end{bmatrix}$$

Lecture 4-19

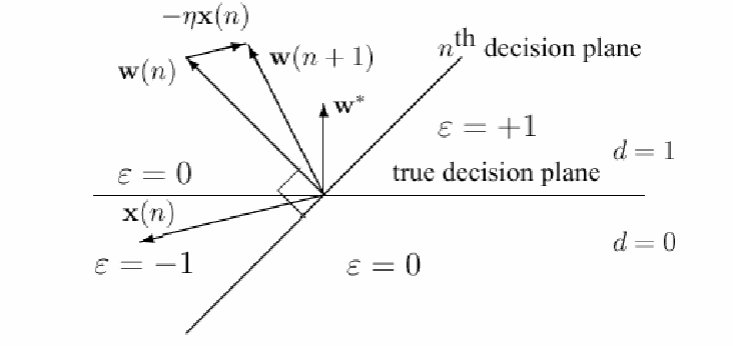
Summary: Perceptron vs. MSE procedures

- **Perceptron rule**
  - The perceptron rule always finds a solution if the classes are linearly separable, but does not converge if the classes are non-separable
- **MSE criterion**
  - The MSE solution has guaranteed convergence, but it may not find a separating hyperplane if classes are linearly separable
    - Notice that MSE tries to minimize the sum of the squares of the distances of the training data to the separating hyperplane, as opposed to finding this hyperplane



Lecture 4-20

Perceptron learning law: the geometric interpretation



w(*n*) : current weight vector
w(*n*+1) : next weight vector
w* : correct (desired) weight vector

Perceptron learning law: the geometric interpretation *(cont.)*

- **During the learning process the current weight vector** w(***n***) **is modified in the direction of the current input vector** x(***n***)**, if the input pattern is misclassified, that is, if the error is non-zero.**
- **Presenting the Perceptron with enough training vectors, the weight vector** w(***n***) **will tend to the correct value** w***.**
- **Rosenblatt proved that if input patterns are *linearly separable*, then the Perceptron learning law *converges*, and the hyperplane separating two classes of input patterns can be determined.**

Convergence of the Perceptron learning law (1)

- **Fixed increment convergence theorem**
  - **for linearly separable vectors X1 and X2 , perceptron converges after some n_0 iterations**

$$\mathbf{w}(n_0) = \mathbf{w}(n_0 + 1) = \mathbf{w}(n_0 + 2) = \ldots$$

is solution vector for $n_0 \le n_{\max}$

  - **proof in case of** $h(n) = 1$

Lecture 4-23

Convergence of the Perceptron learning law (2)

- **Assume:**
  - **Coorect weight vector :** $|W^*| = 1$ **and** $|X| = 1$
  - **A small positive fixed number** d **such that** $W^*.X > d \quad \forall X$
  - **Define** $G(W) = \dfrac{W^*.W}{|W|} \le 1$
  - **G(w) is the cosine of the angle between W and W***
  - **Consider the behavior of G(w) through adaptation (step4 :slide 11)**

$$\begin{aligned} W^*.W(n+1) &= W^*.(W(n) + X) \\ &= W^*.W(n) + W^*.X \\ &\ge W^*.W(n) + d \end{aligned}$$

Lecture 4-24

Convergence of the Perceptron learning law (3)

- **After the nth application**

$$W^*.W(n) \geq nd$$

- **Denominator of G(W) is:**

$$\begin{aligned} |W(n+1)|^2 &= W(n+1).W(n+1) \\ &= (W(n)+X).(W(n)+X) \\ &= |W(n)|^2 + 2W(n).X + |X| \\ &\leq |W(n)|^2 + 1 \\ \Rightarrow |W(n)|^2 &< n \end{aligned}$$

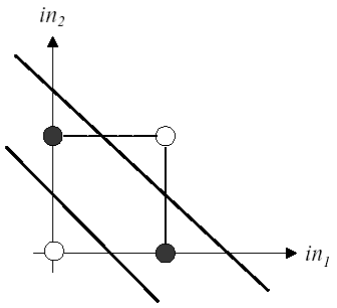
$$\left.\begin{aligned} G(W(n)) &= \frac{W^*.W(n)}{|W(n)|} > \frac{nd}{\sqrt{n}} \\ G(W) &\leq 1 \end{aligned}\right| \Rightarrow n \leq \frac{1}{d^2}$$

Convergence of the Perceptron learning law (4)

- **The number of times ,n, that we go to "adaptation step" will still be finite and will be** $\frac{1}{d^2}$

Limitation of Perceptron

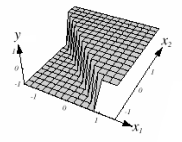
- **The XOR problem (Minsky): nonlinear separability**



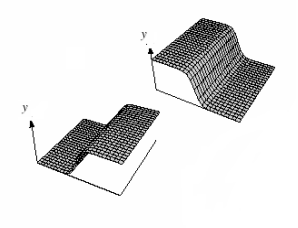
Lecture 4-27

Perceptron with sigmoid activation function

- **For single neuron with step activation function:**



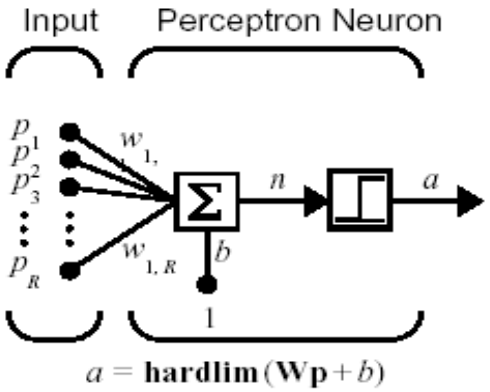
- **For single neuron with Sigmoid activation function:**



Lecture 4-28

Representation of Perceptron in MATLAB

Perceptron Neuron



$a = \mathbf{hardlim}(\mathbf{W}\mathbf{p} + b)$

Where...

R = number of elements in input vector

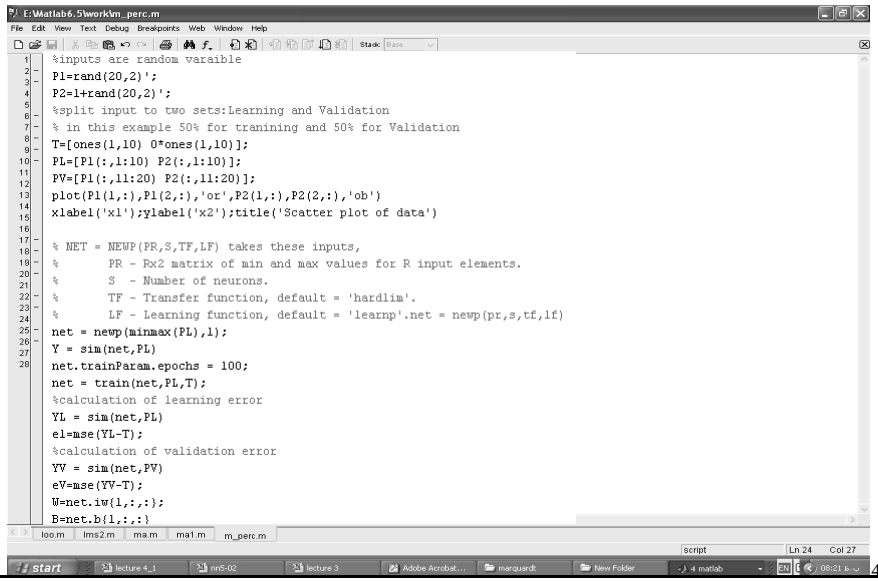
MATLAB TOOLBOX

- **net = newp(pr,s,tf,lf)**
  - **Description of function**
    - **Perceptrons are used to solve simple (i.e. linearly separable) classification problems.**
  - **NET = NEWP(PR,S,TF,LF) takes these inputs,**
    - **PR - Rx2 matrix of min and max values for R input elements.**
    - **S - Number of neurons.**
    - **TF - Transfer function, default = 'hardlim'.**
    - **LF - Learning function, default = 'learnp'.**

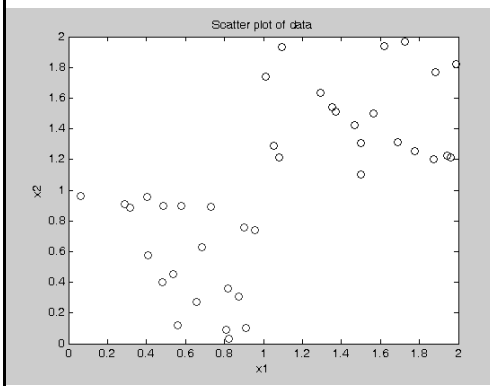
    Returns a new perceptron.

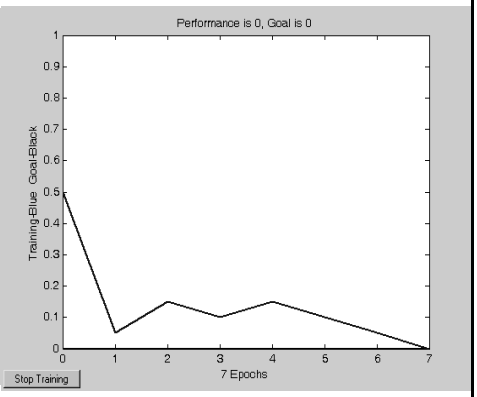
Classification example: Linear separability

- See the M_file

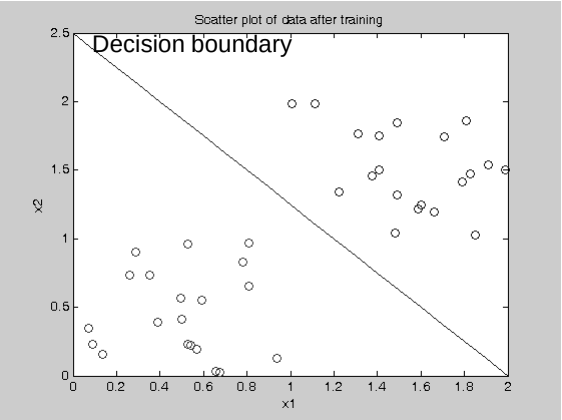


Scatter plot of data **learning curve**

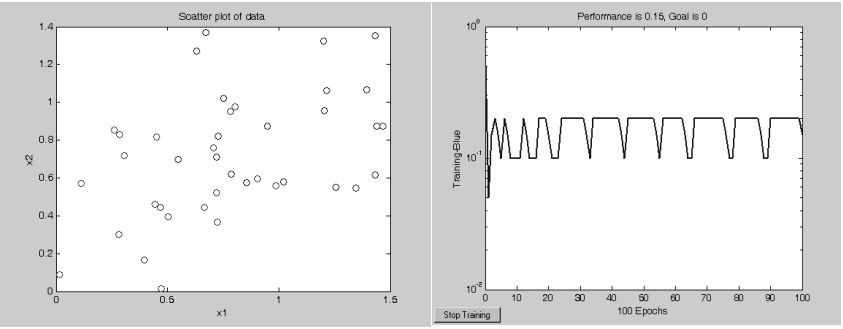




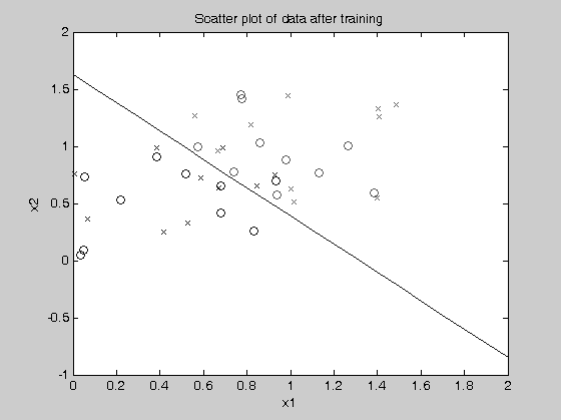
scatter plot of data after training



Classification of data :nonlinear separability

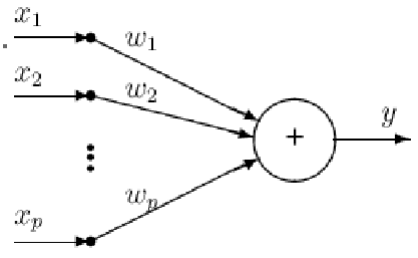


Classification of data :nonlinear separability



ADALINE — The Adaptive Linear Element

- **ADALINE is Perceptron with linear activation function**
- **This is proposed by Widrow**



$$y = \sum w_i x_i = X^T.W$$

Applications of Adaline

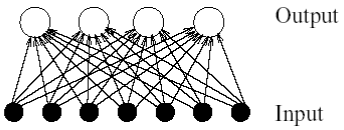
- **In general, the Adaline is used to perform**
  - Linear approximation **of a "small" segment of a nonlinear hyper-surface, which is generated by a p– variable function, y = f(x). In this case, the bias is usually needed.**
  - Linear filtering and prediction **of data (signals);**
  - Pattern association, **that is, generation of m–element output vectors associated with respective p–element input vectors.**

Error concept

- **For single neuron**

$$e = d - y$$

- **For multi neuron**
  - **m is number of output neuron**



$$e_i = d_i - y_i \quad i = 1:m$$

$$e_{m\times 1} = d_{m\times 1} - y_{m\times 1}$$

- **The total measure of the goodness of approximation, or the** performance index**, can be specified by the** mean- squared error **over *m* neurons and *N* training vectors:**

$$J(W) = \frac{1}{2mN}\sum_{i=1}^{N}\sum_{j=1}^{m} e_j^2(i)$$

- **Input data**

$$X_{N\times p}=\begin{vmatrix} x_{11}....x_{1p} \\ x_{21}....x_{2p} \\ ..... \\ x_{N1}....x_{Np} \end{vmatrix}$$

- **Desired output or Target**

$$D_{N\times m}=\begin{vmatrix} d_{11}....x_{1m} \\ x_{21}....x_{2m} \\ ..... \\ x_{N1}....x_{Nm} \end{vmatrix}$$

- **Weight**

$$W_{p\times m}=\begin{vmatrix} w_{11}....w_{1m} \\ w_{21}....w_{2m} \\ ..... \\ w_{N1}....w_{Nm} \end{vmatrix}$$

- **Equitation** $X_{N\times p}W_{p\times m}=Y_{N\times m}$

Lecture 4-39

- **The MSE solution is:**

$$W_{p\times m}=(X^{T}_{p\times N}X_{N\times p})^{-1}X^{T}_{p\times N}D_{N\times m}$$

- **The Error epuiation is:**

$$J(W)=\frac{1}{2N}\sum_{m}E^{T}_{m\times N}E_{N\times m}$$

Lecture 4-40

- **For single neuron m=1:**

$$J(W) = \frac{1}{2N} E_{1\times N}^{T} E_{N\times 1}$$

- **Replacing error in equation:**

$$\begin{aligned} J(W) &= \frac{1}{2N}[(D - XW)^T (D - XW)] \\ &= \frac{1}{2N}[(D^T - W^T X^T)((D - XW)] \\ &= \frac{1}{2N}[D^T D - D^T XW - W^T X^T D + W^T X^T XW] \\ &= \frac{1}{2N}[D^T D - 2D^T XW + W^T X^T XW] \end{aligned}$$

Lecture 4-41

- **Example 1**

$$W = \begin{vmatrix} w_1 \\ w_2 \end{vmatrix}$$

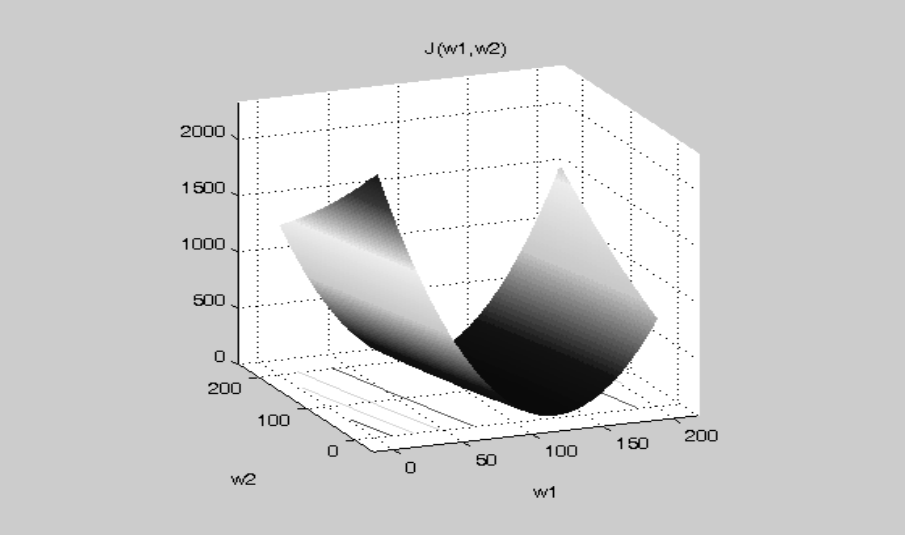
$$X = \begin{vmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \\ 9 & 1 \end{vmatrix} \quad D = \begin{vmatrix} 1.1 \\ 1.8 \\ 3.2 \\ 4.1 \\ 4.8 \\ 5.7 \\ 7.3 \\ 7.9 \\ 9.2 \\ 9.9 \end{vmatrix}$$

$$J(w_1, w_2) = \frac{1}{20}[385.38 - 2\begin{vmatrix} 330 & 55 \end{vmatrix}\begin{vmatrix} w_1 \\ w_2 \end{vmatrix} + \begin{vmatrix} w_1 & w_2 \end{vmatrix}\begin{vmatrix} 285 & 45 \\ 45 & 10 \end{vmatrix}\begin{vmatrix} w_1 \\ w_2 \end{vmatrix}$$

$$J(w_1, w_2) = \frac{1}{20}[385.38 - 660 w_1 - 110 w_2 + 285 w_1^2 + 90 w_1 w_2 + 10 w_2^2]$$

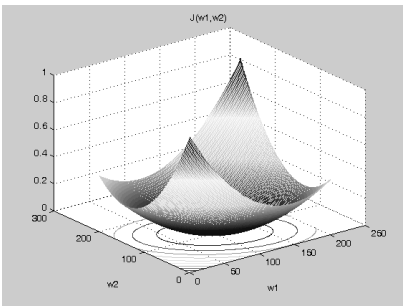
Lecture 4-42

The plot of performance index J(w1,w2) of example

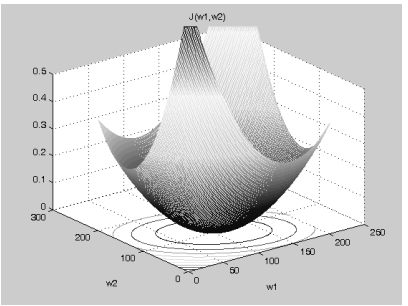


Lecture 4-43

- **Example 2:the performance index in general case**



$$\begin{vmatrix} w_1 \\ w_2 \end{vmatrix} = \begin{vmatrix} 3.19 \\ 8.24 \end{vmatrix}$$



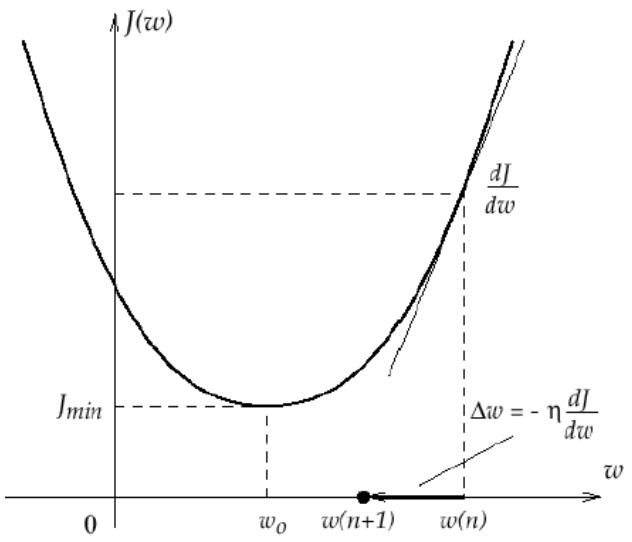
Lecture 4-44

Method of steepest descent

- **If N is large the order of calculation will be high**
- **In order to avoid this problem, we can find the optimal weight vector for which the mean- squared error, *J*(**w**), attains minimum by iterative modification of the weight vector for each training exemplar in the** direction opposite to the gradient **of the performance index, *J*(**w**), as Illustrated in Figure 4– 5 for a single weight situation.**

Lecture 4-45

Illustration of the steepest descent method



Lecture 4-46

- **When the weight vector attains the optimal value for which the gradient is zero (*w*0 in Figure 4–5), the iterations are stopped. More precisely, the iterations are specified as**

$$W(n+1) = W(n) + \Delta W(n)$$

- **where the weight adjustment, .w(*n*), is proportional to the** gradient **of the mean-squared error**

$$\Delta W(n) = -h\nabla J(W(n))$$

- **where *?* is a learning gain.**

Lecture 4-47

- **The gradient of performance index is**

$$J(W) = \frac{1}{2N}[D^T D - 2D^T XW + W^T X^T XW]$$

$$\frac{\partial J(W)}{\partial W} = \frac{1}{2N}[-2D^T X + 2X^T XW]$$

$$= \frac{1}{N}[-D^T X + X^T XW]$$

$$Q = D^T X : Cross\ \ Correlation$$

$$R = X^T X : Input\ \ Correlation$$

The second derivative of J which is known as the Hessian matrix :

$$\mathrm{H(w)} = \frac{\partial^2 J}{\partial W^2} = \frac{\partial}{\partial W}(\nabla J(J(W)) = R$$

- **And must be calculated in each iteration ,n=1:N.**
- **This is complex but there is a method for applying pervious calculation to next calculation (recursive)**

Lecture 4-48

The LMS (Widrow- Hoff) Learning Law

- **The** Least- Mean- Square **learning law replaces the gradient of the mean- squared error with the** gradient update **and can be written in following form:**

$$\Delta W_{p\times m}(n) = hx^{T}_{p\times 1}(n)e_{1\times m}(n)$$

$$e_i = d_i - y_i \quad i = 1:m$$
$$e_{m\times 1} = d_{m\times 1} - y_{m\times 1}$$

$$W(n+1) = W(n) + hx^{T}(n)e(n)$$

- **For single neuron**

For linear neuron

$$y = \sum w_i x_i$$
$$e = (d - y)$$
$$J = \frac{1}{2}e^2 = \frac{1}{2}(d - \sum w_i x_i)^2$$
$$\frac{\partial J}{\partial w_i} = -e\frac{\partial e}{\partial w_i} = -(d - \sum w_i x_i)x_i$$
$$\frac{\partial J}{\partial w_i} = -e\frac{\partial e}{\partial w_i} = -(d - \sum w_i x_i)x_i$$

for nonlinear neuron

$$net = \sum w_i x_i$$
$$y = f(net)$$
$$e = (d - y)$$
$$J = \frac{1}{2}e^2 = \frac{1}{2}(d - f(net))^2$$
$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial net}\frac{\partial net}{\partial w_i} = -(d - f(net))f'(net)\frac{\partial net}{\partial w_i}$$
$$\frac{\partial J}{\partial w_i} = -(d - f(net))f'(net)x_i$$

network training

- **Two types of network training:**
- **Sequential mode or incremental (on-line, stochastic, or per-pattern):**
  - ***Weights updated after each pattern is presented***
- **Batch mode (off-line or per-epoch) :**
  - ***Weights updated after all pattern is presented***

Some general comments on the learning process

- **Computationally, the learning process goes through all training examples (an epoch) number of times, until a** stopping criterion **is reached.**
- **The convergence process can be monitored with the plot of the mean- squared error function *J*(*W*(*n*)).**
- **The popular stopping criteria are:**
  - **the mean- squared error is sufficiently small:**

$$J(W(n)) < \in$$

  - **The rate of change of the mean- squared error is sufficiently small:**

$$\frac{\Delta J(W(n))}{\Delta n} < \in$$

The effect of learning Rate h

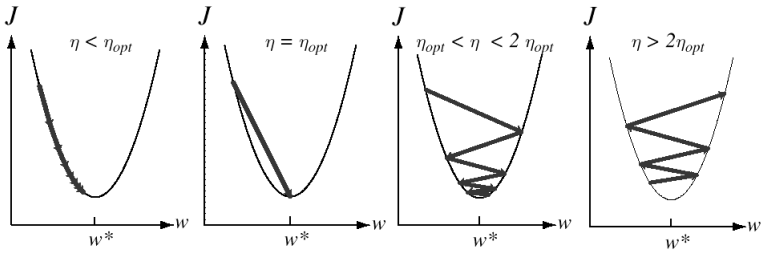


FIGURE 6.16. Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Applications (1)

- **MA (Moving average) modeling (filtering)**

$$y(n) = \sum_{i=0}^{M} b_i x(n-i) \quad M : Order\ of\ \ Model$$

$$y : [y(0)\ \ y(1)\ \ y(2)\ \ .....\ \ y(N)]$$
$$x : [x(0)\ \ x(1)\ \ x(2)\ \ .....\ \ x(N)]$$

  - **For M=2 :**

$$w = \begin{vmatrix} b_0 \\ b_1 \\ b_2 \end{vmatrix} \qquad X = \begin{vmatrix} x_2 & x_1 & x_0 \\ x_3 & x_2 & x_1 \\ ..... & & \\ x_N & x_{N-1} & x_{N-2} \end{vmatrix}_{(N-1)\times 3} \qquad D = \begin{vmatrix} y_2 \\ y_3 \\ ... \\ y_N \end{vmatrix}_{(N-1)\times 3}$$

Applications (2)

- **AR (auto regressive) modeling:**

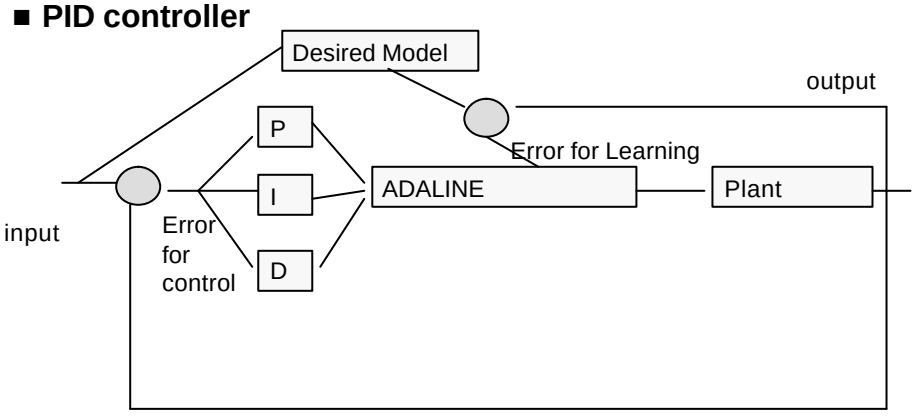
$$y(n)=\sum_{i=1}^{N} a_i\, y(n-i)+bx(n) \quad N: Order \;\; of \;\; Model$$

- **For N=2 :**

$$w=\begin{vmatrix} a_1 \\ a_2 \\ b \end{vmatrix} \qquad X=\begin{vmatrix} y_1 & y_0 & x_2 \\ y_2 & y_1 & x_3 \\ \cdots\cdots & & \\ y_{N-1} & y_{N-2} & x_N \end{vmatrix}_{(N-1)\times 3} \qquad D=\begin{vmatrix} y_2 \\ y_3 \\ \cdots \\ y_N \end{vmatrix}_{(N-1)\times 3}$$

Applications (3)

- **PID controller**

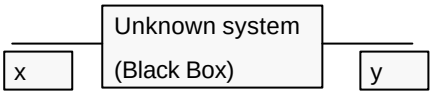


Simulation of MA modeling

- **Supose the MA model as:**

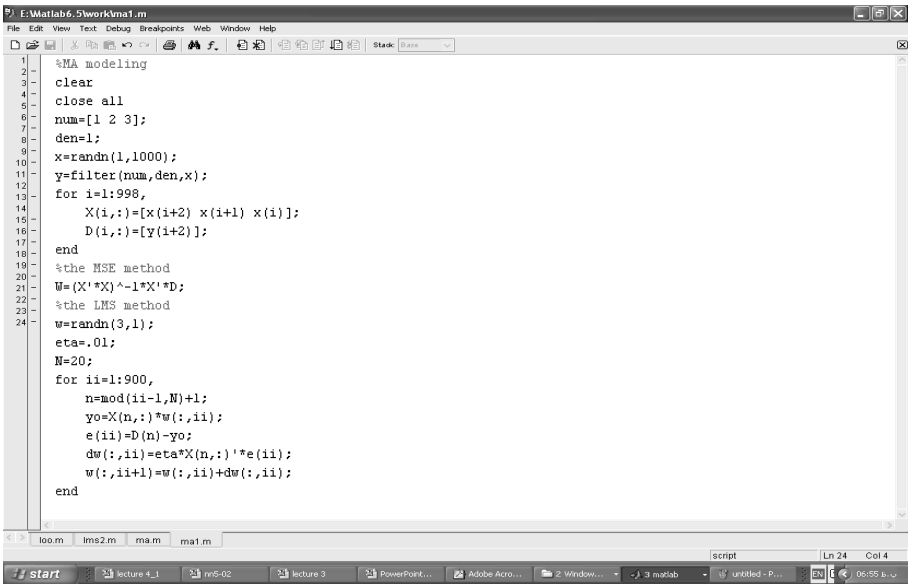
$$b = \begin{vmatrix} 1 \\ 2 \\ 3 \end{vmatrix} \quad M = 3;$$

- **Input is Gaussian noise with mean=0 and var=1**
- **y is Calculated by recursive equation**
- **Please see the M_file**



Lecture 4-57

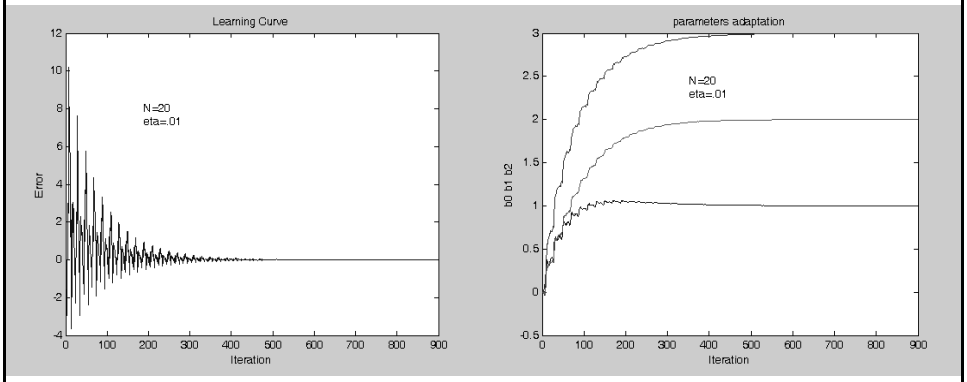
M_file of MA Modeling



Lecture 4-58

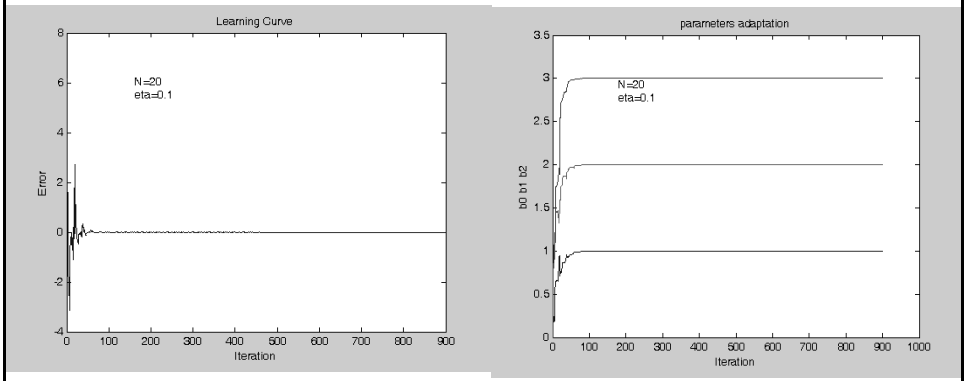
MA Modeling

- **Weight initial : zeros and** $h = 0.01$
- **N=20; data training set**



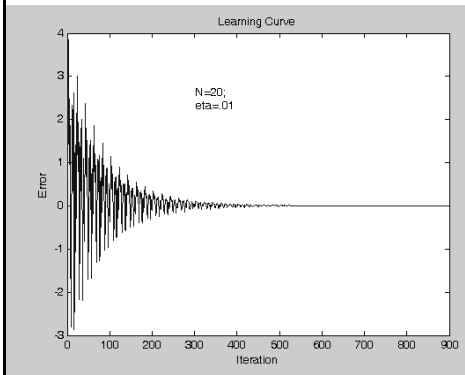
MA Modeling

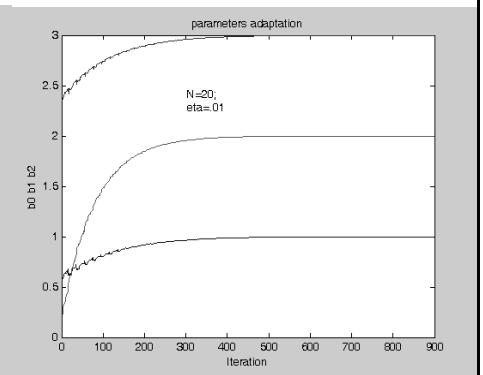
- **Weight initial : zeros and** $h = 0.1$
- **N=20; data training set**



MA Modeling

- **Weight initial : random and** $h = 0.01$
- **N=20; data training set**

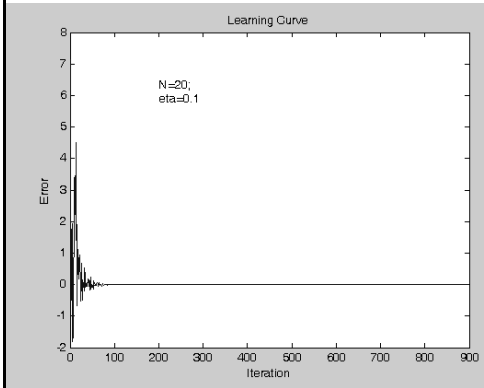


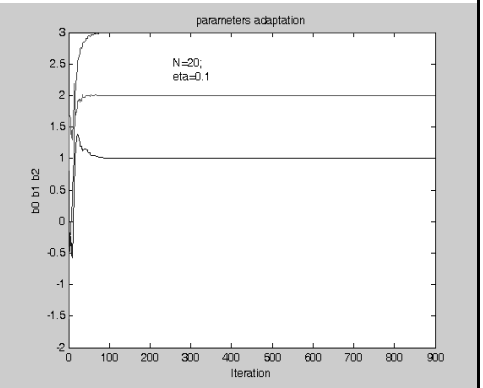


Lecture 4-61

MA Modeling

- **Weight initial : random and** $h = 0.1$
- **N=20; data training set**

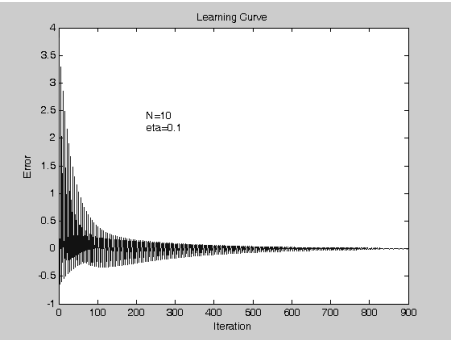


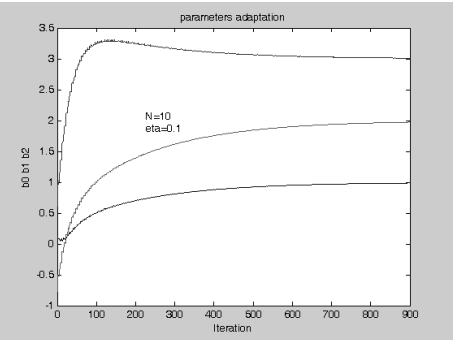


Lecture 4-62

MA Modeling

- **Weight initial : random and** $h = 0.1$
- **N=10; data training set**



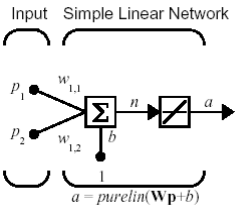


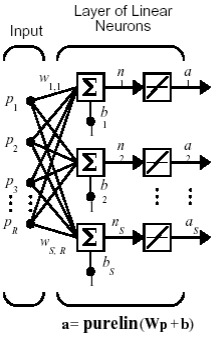
MATLAB TOOLBOX

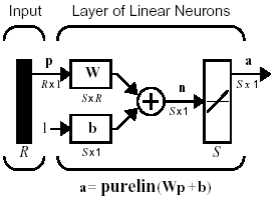
- **net = newlin(PR,S,ID,LR)**
  - **Description of function**
    - **Linear layers are often used as adaptive filters for signal processing and prediction.**
  - **NEWLIN(PR,S,ID,LR) takes these arguments,**
    - **PR - Rx2 matrix of min and max values for R input elements.**
    - **S - Number of elements in the output vector.**
    - **ID - Input delay vector, default = [0].**
    - **LR - Learning rate, default = 0.01;**

    and returns a new linear layer.

- **The linear network is shown as**

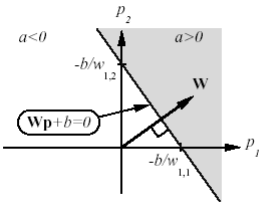






Where... R = numberof elements in input vector

S = numberof neurons in layer



Lecture 4-65