EENG582: Artificial Neural Networks

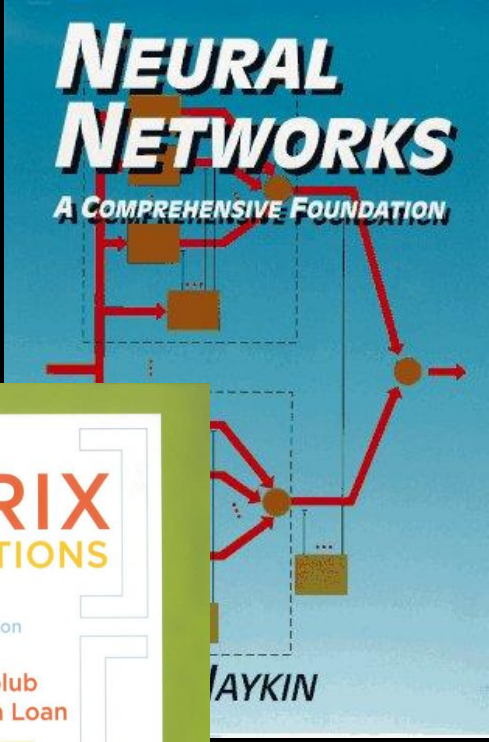
Reference 1: **Neural Networks**
A Comprehensive Foundation
by S. Haykin

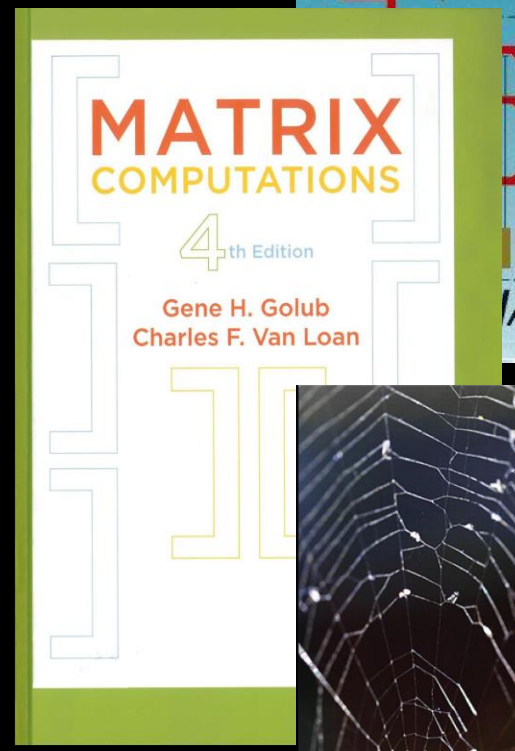
Sections 3.1 – 3.5

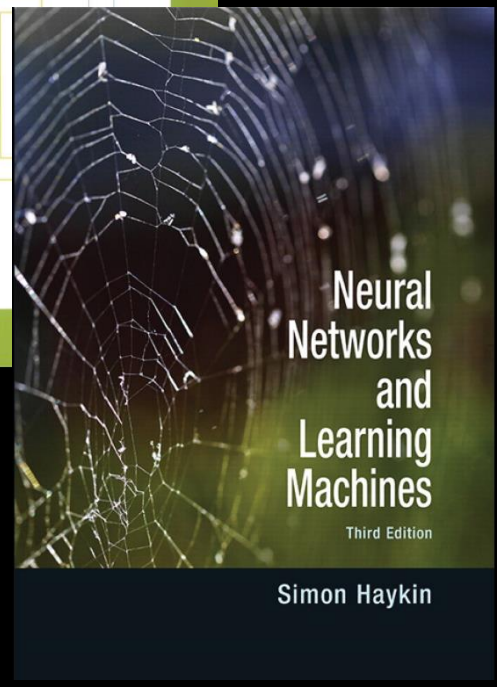
3. Single Layer Perceptrons
+ Optimization

Prof. Dr. Hasan AMCA
Electrical and Electronic Engineering Department
(ee.emu.edu.tr)
Eastern Mediterranean University
(emu.edu.tr)

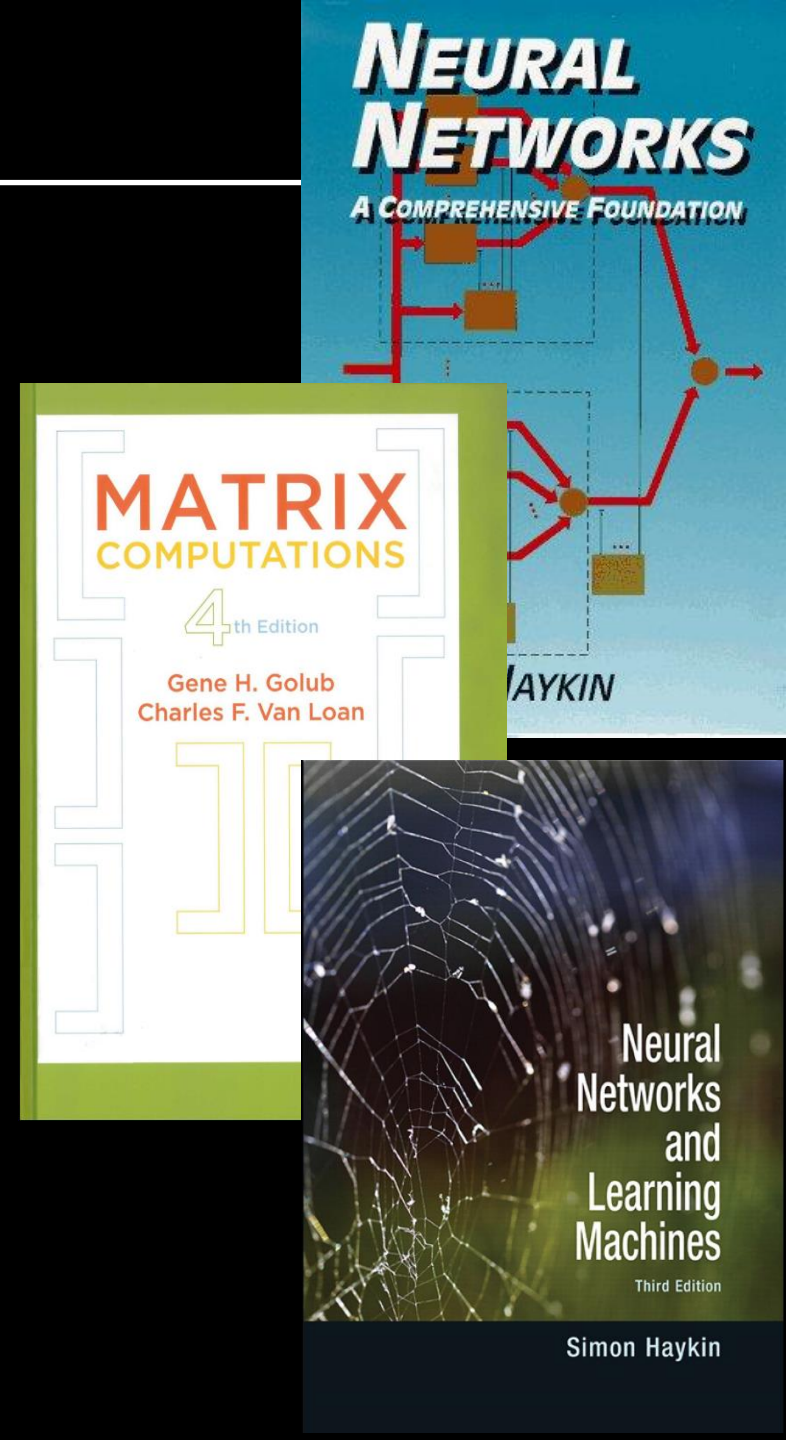
Reference 2: Matrix Computations, 4th Ed. By Golub and Van Loan - Chapter 11, Large Sparse Linear System Problems







3. Single Layer Perceptrons



3 Single Layer Perceptrons

3.1 Introduction

- The perceptron is the simplest form of a neural network used for the classification of patterns said to be linearly separable
  ~ i.e., patterns that lie on opposite sides of a hyperplane.
- Basically, it consists of **a single neuron** with adjustable synaptic weights and bias.
- The algorithm used to adjust the free parameters of this neural network first appeared in a learning procedure developed by Rosenblatt (1958, 1962) for his perceptron brain model.
- The perceptron built around **a single neuron** is limited to performing pattern classification with only two classes (hypotheses).

3 Single Layer Perceptrons

- The single neuron also forms the basis of an adaptive filter,
  - a functional block that is basic to the ever-expanding subject of signal processing.
- The development of **adaptive filtering** owes itself to the least-mean-square (LMS) algorithm, also known as the delta rule.
- Adaptive filters have been successfully applied in such diverse fields as antennas, communication systems, control systems, radar, sonar, seismology, and biomedical engineering.
- The LMS algorithm and the perceptron are naturally related. It is therefore proper for us to study them together in one chapter.

3.2 Adaptive Filtering Problem

- Consider a dynamical unknown system,
- All that we have know is an m-dimensional stimulus $\mathbf{x}(i)$ applied across m input nodes of the system.
- The system responds by producing a scalar output $d(i)$, where $i = 1, 2, \dots, n, \dots$ as depicted in Fig. 3.1a. Thus, the external behavior of the system is described by the data set

$$\mathcal{T}: \{\mathbf{x}(i), d(i); i = 1, 2, \dots, n, \dots\} \tag{3.1}$$

where

$$\mathbf{x}(i) = [x_1(i), x_2(i), \dots, x_m(i)]^T$$

- Samples of $\mathcal{T}$ are identically distributed with an unknown probability law.
- The dimension m pertaining to the input vector $\mathbf{x}(i)$ is referred to as the *dimensionality of the input space*.

3.2 Adaptive Filtering Problem

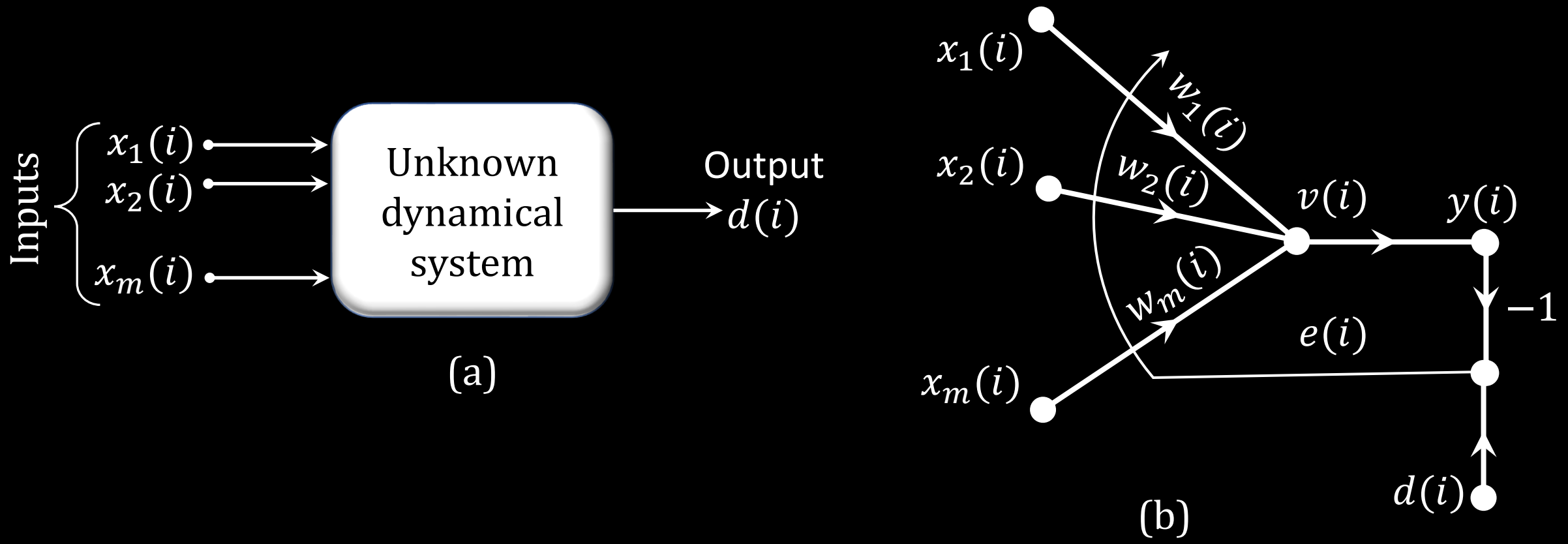


Fig. 3.1 (a) Unknown dynamical system.
(b) Signal flow graph of adaptive model for the system.

3.2 Adaptive Filtering Problem

- The problem we address is: How to design a multiple input-single output model of the unknown dynamical system by building it around a single linear neuron, under the influence of an algorithm that controls necessary adjustments to the synaptic weights of the neuron, with the following points in mind:
  - ~ The algorithm starts from an arbitrary setting of the neuron's synaptic weights,
  - ~ Adjustments to the synaptic weights, in response to statistical variations in the system's behavior, are made on a continuous basis,
  - ~ Computations of adjustments to the synaptic weights are completed inside a time interval that is one sampling period long.
- The neuronal model described is referred to as an **adaptive filter**.

3.2 Adaptive Filtering Problem

- Fig. 3.1b shows a signal-flow graph of the adaptive filter. Its operation consists of two continuous processes:

1. Filtering process, which involves the computation of two signals:
   - An output, denoted by $y(i)$, that is produced in response to the m elements of the stimulus vector $\mathbf{x}(i)$, namely, $x_1(i), x_2(i), \ldots, x_m(i)$.
   - An error signal, denoted by $e(i)$, that is obtained by comparing the output $y(i)$ to the corresponding output $d(i)$ produced by the unknown system. In effect, $d(i)$ acts as a desired response or target signal.
2. Adaptive process, which involves the automatic adjustment of the svnaptic weights of the neuron in accordance with the error signal $e(i)$.

3.2 Adaptive Filtering Problem

- Since the neuron is linear, the output $y(i)$ is exactly the same as the induced local field $v(i)$; that is,

$$y(i) = v(i) = \sum_{k=1}^{m} w_k(i)x_k(i) \tag{3.2}$$

where $w_k(i)$ are the m synaptic weights of the neuron measured at time i.

- In matrix form

$$y(i) = \mathbf{x}^T(i)\mathbf{w}(i) \tag{3.3}$$

where $\mathbf{w}(i) = [w_1(i), w_2(i), \dots, w_m(i)]^T$ for a single neuron.

- The error signal

$$e(i) = d(i) - y(i) \tag{3.4}$$

- The manner in which the error signal $e(i)$ is used to control the adjustments to the neuron's synaptic weights is determined by the cost function used to derive the *adaptive filtering algorithm* of interest.

3.4 Linear Least-Squares Filter

- A *linear least-squares filter* has two distinctive characteristics:
  1. The single neuron around which it is built is linear, as shown in the model of Fig. 3.1b.
  2. The cost function $\mathcal{C}(\boldsymbol{w})$ used to design the filter consists of the sum of error squares, as defined in Eq. (3.17) as

$$\mathcal{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i).$$

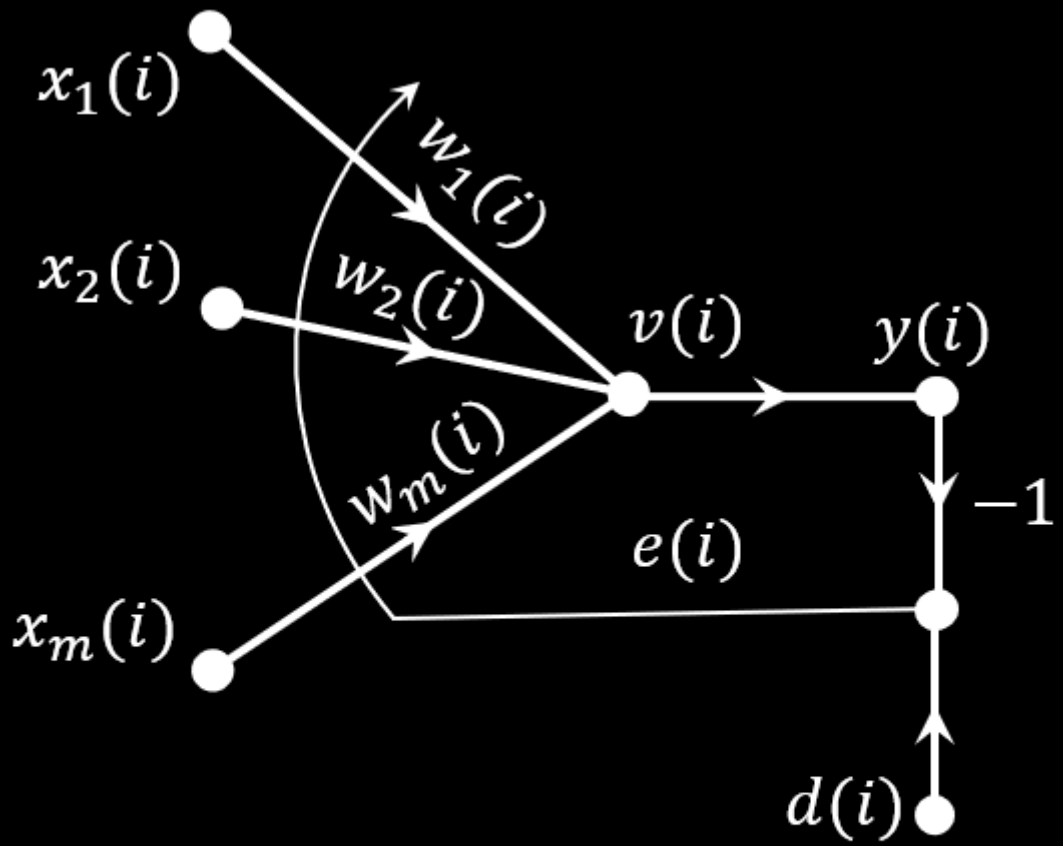


Fig. 3.1b Signal flow graph of adaptive model for the system.

3.4 Linear Least-Squares Filter

- The error vector $\mathbf{e}(n)$ may be expressed as:

$$\mathbf{e}(n) = \mathbf{d}(n) - [\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(n)]^T \mathbf{w}(n) = \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n) \qquad (3.25)$$

  where $\mathbf{d}(n)$ is the n-by-1 desired response vector

$$\mathbf{d}(n) = [d(1), d(2), \dots, d(n)]^T$$

  and $\mathbf{X}(n)$ is the n-by-m data matrix,

$$\mathbf{X}(n) = [\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(n)]^T$$

- Differentiating (3.25) with respect to $\mathbf{w}(n)$ yields the gradient matrix

$$\nabla \mathbf{e}(n) = -\mathbf{X}^T(n)$$

- And the Jacobian of $\mathbf{e}(n)$ is

$$\mathbf{J}(n) = -\mathbf{X}(n) \qquad (3.26)$$

3.4 Linear Least-Squares Filter

- The Gauss-Newton method converges in a single iteration, as follows

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1}\mathbf{X}^T(n)\left(\mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)\right)$$

$$= \left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1}\mathbf{X}^T(n)\mathbf{d}(n) \qquad (3.27)$$

- $\left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1}\mathbf{X}^T(n)$ is the *pseudoinverse* of the data matrix $\mathbf{X}(n)$ as shown by

$$\mathbf{X}^+(n) = \left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1}\mathbf{X}^T(n) \qquad (3.28)$$

- We can rewrite (3.27) as

$$\mathbf{w}(n+1) = \mathbf{X}^+(n)\mathbf{d}(n) \qquad (3.29)$$

- (3.29) means that the weight vector $\mathbf{w}(n+1)$ solves the linear least-squares problem defined over an observation interval of duration n.

Linear Least-Squares Filter Example

```
% Constrained Linear Least-Squares,SolverBased
% www.mathworks.com/help/optim/ug/deblur-solver-based.html
% The Problem: Here is a photo of people sitting in a car
% having an interesting license plate.
load optdeblur
[m,n] = size(P);
mn = m*n;
subplot(411);
imshow(P)
title(sprintf('Original Image, size %d-by-%d, %d
pixels',m,n,mn))

% Add Motion: Simulate the effect of vertical motion
% blurring by averaging each pixel with the 5 pixels above
% and below. Construct a sparse matrix D to blur with a
% single matrix multiply.
blur = 5;  mindex = 1:mn;  nindex = 1:mn;
for i = 1:blur
  mindex=[mindex i+1:mn 1:mn-i];
  nindex=[nindex 1:mn-i i+1:mn];
end
D = sparse(mindex,nindex,1/(2*blur+1));
```

```
% Draw a picture of D.
subplot(412);
cla
axis off ij
xs = 31;
ys = 15;
xlim([0,xs+1]);
ylim([0,ys+1]);
%subplot(412);
[ix,iy] = meshgrid(1:(xs-1),1:(ys-1));
l = abs(ix-iy)<=5;
text(ix(l),iy(l),'x')
text(ix(~l),iy(~l),'0')
text(xs*ones(ys,1),1:ys,'...');
text(1:xs,ys*ones(xs,1),'...');
title('Blurring Operator D (x = 1/11)')

% Blurred Image
G = D*(P(:));
subplot(413);
imshow(reshape(G,m,n));
title('Blurred Image')
```

Linear Least-Squares Filter Example

```
% Deblurred Image
% To deblur, suppose that you know the blurring
operator D. How well can you
% remove the blur and recover the original image
P?
% The simplest approach is to solve a least
squares problem for x:
lb = zeros(mn,1);
ub = 1 + lb;
sol = lsqlin(D,G,[],[],[],[],lb,ub);

% Minimum found that satisfies the constraints.
% Optimization completed because the objective
function is non-decreasing in
% feasible directions, to within the value of
the optimality tolerance,
% and constraints are satisfied to within the
value of the constraint tolerance.
xpic = reshape(sol,m,n);
subplot(414);
imshow(xpic)
title('Deblurred Image')
```

```
% Regularization
% Regularization is a way to smooth the
solution. There are many regularization methods.
% For a simple approach, add a term to the
objective function as follows:
ddI = speye(mn);
sol2 = lsqlin(D+0.02*addI,G,[],[],[],[],lb,ub);

% Minimum found that satisfies the constraints.
% Optimization completed because the objective
function is non-decreasing in
% feasible directions, to within the value of
the optimality tolerance,
% and constraints are satisfied to within the
value of the constraint tolerance.
xpic2 = reshape(sol2,m,n);
figure
imshow(xpic2)
title('Deblurred Regularized Image')
```

Linear Least-Squares Filter Example

Original Image, 149-by-311, 46339 pixels

Blurred Image

Deblurred Image

Deblurred Regularized Image

Linear Least-Squares Filter **Example**

Blurring Operator $D(x = 1/11)$

x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Wiener Filter: limiting form of the Linear Least-Squares Filter for an Ergodic Environment

- A case of particular interest is when the input vector $\mathbf{x}(i)$ and desired response $\mathbf{d}(i)$ are drawn from a *stationary ergodic environment*
- We may then use long-term sample averages or time-averages for expectations or ensemble averages.
- Such an environment is partially described by the second order statistics:
  - ~ Correlation matrix of the input vector $\mathbf{x}(i)$; it is denoted by $\mathbf{R}_x$.

$$\mathbf{R}_x = E[\mathbf{x}^T(i)\mathbf{x}(i)] = \lim_{n\to\infty} \sum_{i=1}^{n} \mathbf{x}(i)\mathbf{x}^T(i) = \lim_{n\to\infty} \mathbf{X}(n)\mathbf{X}^T(n) \quad (3.30)$$

  - ~ Cross-correlation vector between the input vector $\mathbf{x}(i)$ and desired response $\mathbf{d}(i)$; denoted by $\mathbf{r}_{xd}$.

$$\mathbf{r}_{xd} = E[\mathbf{x}(i)\mathbf{d}(i)] = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}(i)\mathbf{d}(i) = \lim_{n\to\infty} \frac{1}{n} \mathbf{X}^T(n)\mathbf{d}(n) \quad (3.31)$$

- where E denotes the statistical expectation operator.

Wiener Filter: limiting form of the Linear Least-Squares Filter for an Ergodic Environment

- Accordingly, we may reformulate the linear least-squares solution of (3.27) as

$$\mathbf{w}_0 = \lim_{n\to\infty} \mathbf{w}(n+1) = \lim_{n\to\infty} \left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1} \mathbf{X}^T(n)\mathbf{d}(n)$$

$$= \lim_{n\to\infty} \left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1} \lim_{n\to\infty} \mathbf{X}^T(n)\mathbf{d}(n) = \mathbf{R}_x^{-1}\mathbf{r}_{xd} \qquad (3.32)$$

- The weight vector $\mathbf{w}_0$ is called the *Wiener solution* to the linear optimum filtering problem. Accordingly, we may make the following statement:

For an ergodic process, the linear least-squares filter asymptotically approaches the Wiener filter as the number of observations approaches ∞

Wiener Filter: limiting form of the Linear Least-Squares Filter for an Ergodic Environment

- The Wiener filter requires knowledge of the 2nd-order statistics, which are often not available in practice.
- Linear adaptive filters which are able to adjust its free parameters in response to statistical variations in environment can obtain 2nd order stats
- A popular algorithm for doing this kind of adjustment on a continuing basis is the least-mean-square algorithm, which is intimately related to the Wiener filter.

3.5 Least-Mean-Square Algorithm

- The *least-mean-square* (LMS) algorithm is based on the use of *instantaneous values for the cost function*, namely,

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}e^2(n) \tag{3.33}$$

where $e(n)$ is the error signal measured at time n. Differentiating $\mathcal{E}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$ yields

$$\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}} = e(n)\frac{\partial e(n)}{\partial \mathbf{w}} \tag{3.34}$$

- the LMS algorithm operates with a linear neuron where the error signal is

$$e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n) \tag{3.35}$$

- Hence, $\frac{\partial e(n)}{\partial \mathbf{w}(n)} = -\boldsymbol{x}(n)$ and $\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)e(n)$. The gradient vector may then be written as

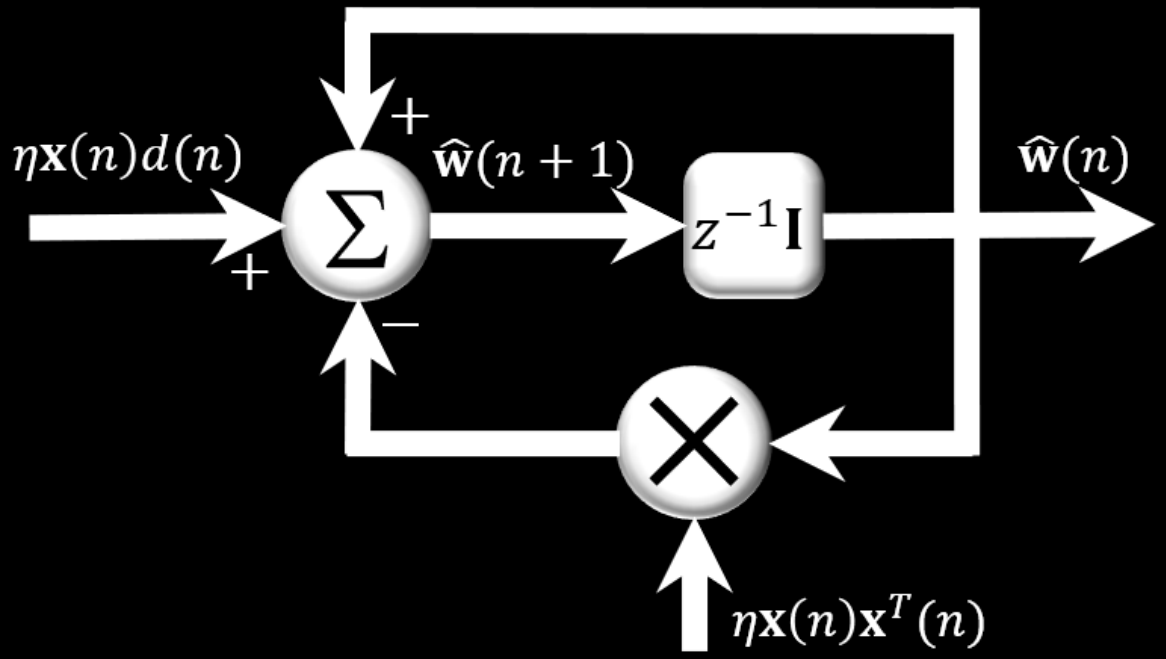
$$\hat{\mathbf{g}}(n) = -\mathbf{x}(n)e(n) \tag{3.36}$$

3.5 Least-Mean-Square Algorithm

- Finally, using (3.36) for the gradient vector in (3.12) for the method of steepest descent, we may formulate the LMS algorithm as follows:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n) e(n) \qquad (3.37)$$

Where η is the learning-rate parameter and $\hat{\mathbf{w}}(n)$ is the estimate of the weight vector which results from the use of steepest descent method by the LMS algorithm.



3.5 Least-Mean-Square Algorithm

Table 3.1 Summary of LMS Algorithm

Training Sample:
Input signal vector= $\mathbf{x}(n)$
Desired response= $d(n)$

User-selected parameter. η

Initialization:
Set weight vector $\hat{\mathbf{w}}$=0

Computation.
For $n = 1, 2, \ldots$ compute

$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$$

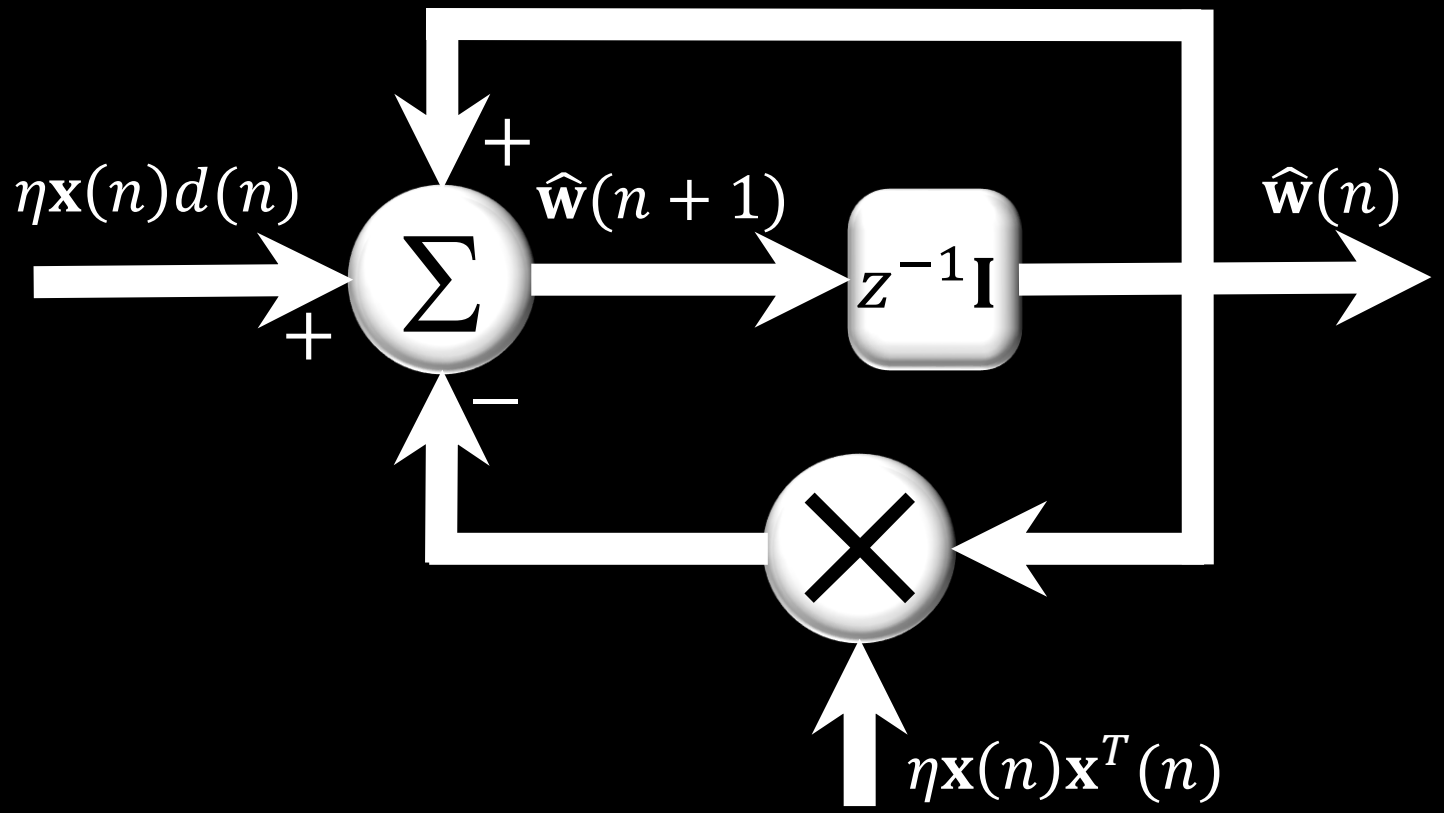


Fig. 3.3 Signal-flow graph representation of the LMS algorithm.

3.5 Least-Mean-Square Algorithm

- In the steepest descent algorithm, the weight vector $\mathbf{w}(n)$ follows a well-defined trajectory in weight space for a prescribed η.
- In contrast, in the LMS algorithm the weight vector $\mathbf{w}(n)$ traces a random trajectory.
- In using the LMS algorithm, we sacrifice a distinctive feature of the steepest descent algorithm.
- However, unlike the method of steepest descent, the LMS algorithm does not require knowledge of the statistics of the environment since the initial values of the weight vector $\hat{\mathbf{w}} = 0$.

Signal-Flow Graph Representation of the LMS Algorithm

- By combining (3.35) and (3.37) we may express the evolution of the weight vector in the LMS algorithm as follows:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)]$$

$$= [1 - \eta\mathbf{x}(n)\mathbf{x}^T(n)]\hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)d(n) \qquad (3.38)$$

- where $\mathbf{I}$ is the identity matrix. In using the LMS algorithm, we recognize that

$$\hat{\mathbf{w}}(n) = z^{-1}[\hat{\mathbf{w}}(n+1)] \qquad (3.39)$$

- where z^{-1} is the unit-delay operators shown in signal-flow graph of the stochastic feedback system in fig Fig. 3.3.
- Feedback has a profound impact on the convergence behavior of the LMS algorithm.

Convergence Considerations of the LMS Algorithm

- From control theory we know that the stability of a feedback system is determined by the parameters that constitute its feedback loop such as the learning-rate parameter η.
- The first criterion for convergence of the LMS algorithm is convergence of the mean, described by

$$E[\hat{\mathbf{w}}(n)] \rightarrow \mathbf{w}_0 \quad \text{as } n \rightarrow \infty \tag{3.40}$$

- where $\mathbf{w}_0$ is the Wiener solution.
- From a practical point of view, the convergence issue that really matters is convergence in the mean square, described by

$$E[e^2(n)] \rightarrow \text{constant } as\ n \rightarrow \infty \tag{3.41}$$

Convergence Considerations of the LMS Algorithm

- LMS algorithm is convergent in the mean square provided that η satisfies

$$0 < \eta < \frac{2}{\lambda_{max}} \tag{3.42}$$

where λ_{max} is the largest eigenvalue of the correlation matrix $\mathbf{R}_x$.

- Since λ_{max} typically not available, the trace of the $\mathbf{R}_x$ matrix may be taken as a conservative estimate for λ_{max} as

$$0 < \eta < \frac{2}{\mathrm{tr}[\mathbf{R}_x]} \tag{3.43}$$

- By definition, the trace of a square matrix is equal to the sum of its diagonal elements.

Convergence Considerations of the LMS Algorithm

- Since each diagonal element of the correlation matrix $\mathbf{R}_x$ equals the mean-square value of the corresponding sensor input

$$0 < \eta < \frac{2}{\text{sum of mean}-\text{square values of the sensor inputs}} \tag{3.44}$$

Virtues and limitations of the LMS Algorithm

- An important virtue of the LMS algorithm is that it is simple and model independent and therefore robust, which means that small model uncertainty and small disturbances can only result in small estimation errors
- The primary limitations of the LMS algorithm are its slow rate of convergence and sensitivity to variations in the eigen structure of the input.
- The LMS algorithm typically requires a number of iterations equal to about 10 times the dimensionality of the input space for it to reach a steady-state condition.
- LMS algorithm is particularly sensitive to variations in the condition number or eigenvalue spread (λ_{max} , λ_{min}) of the correlation matrix $\mathbf{R}_x$ of the input vector $\mathbf{x}$ defined by

$$\chi(\mathbf{R}_x) = \frac{\lambda_{max}}{\lambda_{min}} \tag{3.45}$$

Available Optimizers
Available Types for Optimization-Variables

Optimizer	Error-Function (EF) Formulation	Search Method
Random Optimizer	Least-Squares EF	Random Search
Gradient Optimizer	Least-Squares EF	Gradient Search
Random Minimax Optimizer	Minimax L1 EF	Random Search
Gradient Minimax Optimizer	Minimax L1 EF	Gradient Search
Quasi-Newton Optimizer	Least-Squares EF	Quasi-Newton Search
Least Pth Optimizer	Least Pth EF	Quasi-Newton Search
Minimax Optimizer	Minimax EF	Gauss-Newton/Quasi-Newton (minimax) Search
Random Max Optimizer	Negated Least-Squares EF	Random Search
Hybrid Optimizer	Least-Squares EF	Random Search and Quasi-Newton Search
Discrete Optimizer	Least-Squares EF	Exhaustive Search
Genetic Optimizer	Least-Squares EF	Genetic Algorithm Search
Simulated Annealing Optimizer	Least-Squares EF	Simulated Annealing Algorithm Search
Sensitivity Analysis†		

http://literature.cdn.keysight.com/litweb/pdf/ads2008/optstat/ads2008/Summary_of_Optimizers.html