

3.1 INTRODUCTION

Rosenblatt's perceptron, discussed in Chapter 1, was the first learning algorithm for solving a linearly separable pattern-classification problem. The *least-mean-square (LMS) algorithm*, developed by Widrow and Hoff (1960), was the first linear adaptive-filtering

algorithm for solving problems such as prediction and communication-channel equalization. Development of the LMS algorithm was indeed inspired by the perceptron. Though different in applications, these two algorithms share a common feature: They both involve the use of a *linear combiner*, hence the designation “linear.”

The amazing thing about the LMS algorithm is that it has established itself not only as the workhorse for adaptive-filtering applications, but also as the benchmark against which other adaptive-filtering algorithms are evaluated. The reasons behind this amazing track record are multifold:

- In terms of computational complexity, the LMS algorithm’s complexity is *linear* with respect to adjustable parameters, which makes the algorithm *computationally efficient*, yet the algorithm is effective in performance.
- The algorithm is *simple* to code and therefore easy to build.
- Above all, the algorithm is *robust* with respect to external disturbances.

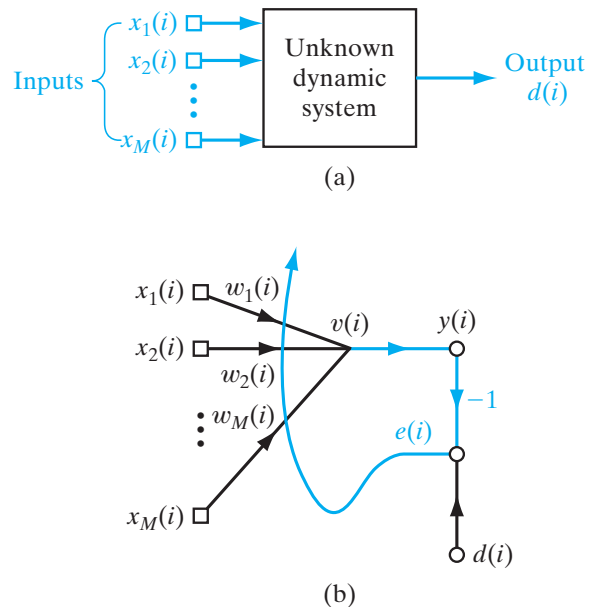
From an engineering perspective, these qualities are all highly desirable. It is therefore not surprising to see that the LMS algorithm has withstood the test of time.

In this chapter, we derive the LMS algorithm in its most basic form and discuss its virtues and limitations. Most importantly, the material presented herein sets the stage for the back-propagation algorithm to be discussed in the next chapter.

3.2 FILTERING STRUCTURE OF THE LMS ALGORITHM

Figure 3.1 shows the block diagram of an unknown dynamic system that is stimulated by an input vector consisting of the elements $x_1(i)$, $x_2(i)$, ..., $x_M(i)$, where i denotes the instant of time at which the stimulus (excitation) is applied to the system. The time index

FIGURE 3.1 (a) Unknown dynamic system. (b) Signal-flow graph of adaptive model for the system; the graph embodies a feedback loop set in color.



$i = 1, 2, \dots, n$. In response to this stimulus, the system produces an output denoted by $y(i)$. Thus, the external behavior of the system is described by the data set

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

where

$$\mathbf{x}(i) = [x_1(i), x_2(i), \dots, x_M(i)]^T \quad (3.2)$$

The sample pairs composing $\mathcal{T}$ are identically distributed according to 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*, or simply as the *input dimensionality*.

The stimulus vector $\mathbf{x}(i)$ can arise in one of two fundamentally different ways, one spatial and the other temporal:

- The M elements of $\mathbf{x}(i)$ originate at different points in space; in this case, we speak of $\mathbf{x}(i)$ as a *snapshot* of data.
- The M elements of $\mathbf{x}(i)$ represent the set of present and $(M - 1)$ past values of some excitation that are *uniformly spaced in time*.

The problem we address is how to design a multiple-input–single-output *model* of the unknown dynamic system by building it around a *single linear neuron*. The neural model operates 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 (i.e time is incorporated into the constitution of the algorithm).
- Computations of adjustments to the synaptic weights are completed inside an interval that is one sampling period long.

The neural model just described is referred to as an *adaptive filter*. Although the description is presented in the context of a task clearly recognized as one of *system identification*, the characterization of the adaptive filter is general enough to have wide application.

Figure 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, $x_1(i), x_2(i), \dots, x_M(i)$;
 - an error signal, denoted by $e(i)$, that is obtained by comparing the output $y(i)$ with 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 synaptic weights of the neuron in accordance with the error signal $e(i)$.

Thus, the combination of these two processes working together constitutes a *feedback loop* acting around the neuron, as shown in Fig. 3.1b.

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

$$y(i) = v(i) = \sum_{k=1}^M w_k(i) x_k(i) \quad (3.3)$$

where $w_1(i), w_2(i), \dots, w_M(i)$ are the M synaptic weights of the neuron, measured at time i . In matrix form, we may express $y(i)$ as an inner product of the vectors $\mathbf{x}(i)$ and $\mathbf{w}(i)$ as

$$y(i) = \mathbf{x}^T(i) \mathbf{w}(i) \quad (3.4)$$

where

$$\mathbf{w}(i) = [w_1(i), w_2(i), \dots, w_M(i)]^T$$

Note that the notation for a synaptic weight has been simplified here by *not* including an additional subscript to identify the neuron, since we have only a single neuron to deal with. This practice is followed throughout the book, whenever a single neuron is involved. The neuron's output $y(i)$ is compared with the corresponding output $d(i)$ received from the unknown system at time i . Typically, $y(i)$ is different from $d(i)$; hence, their comparison results in the error signal

$$e(i) = d(i) - y(i) \quad (3.5)$$

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. This issue is closely related to that of optimization. It is therefore appropriate to present a review of unconstrained-optimization methods. The material is applicable not only to linear adaptive filters, but also to neural networks in general.

3.3 UNCONSTRAINED OPTIMIZATION: A REVIEW

Consider a cost function $\mathcal{E}(\mathbf{w})$ that is a *continuously differentiable* function of some unknown weight (parameter) vector $\mathbf{w}$. The function $\mathcal{E}(\mathbf{w})$ maps the elements of $\mathbf{w}$ into real numbers. It is a measure of how to choose the weight (parameter) vector $\mathbf{w}$ of an adaptive-filtering algorithm so that it behaves in an optimum manner. We want to find an optimal solution $\mathbf{w}^*$ that satisfies the condition

$$\mathcal{E}(\mathbf{w}^*) \leq \mathcal{E}(\mathbf{w}) \quad (3.6)$$

That is, we need to solve an *unconstrained-optimization problem*, stated as follows:

Minimize the cost function $\mathcal{E}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$.

The necessary condition for optimality is

$$\nabla \mathcal{E}(\mathbf{w}^*) = \mathbf{0} \quad (3.7)$$

where ∇ is the *gradient operator*,

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_M} \right]^T \quad (3.8)$$

and $\nabla \mathcal{E}(\mathbf{w})$ is the *gradient vector* of the cost function,

$$\nabla \mathcal{E}(\mathbf{w}) = \left[\frac{\partial \mathcal{E}}{\partial w_1}, \frac{\partial \mathcal{E}}{\partial w_2}, \dots, \frac{\partial \mathcal{E}}{\partial w_M} \right]^T \quad (3.9)$$

(Differentiation with respect to a vector is discussed in Note 1 at the end of this chapter.)

A class of unconstrained-optimization algorithms that is particularly well suited for the design of adaptive filters is based on the idea of *local iterative descent*:

Starting with an initial guess denoted by $\mathbf{w}(0)$, generate a sequence of weight vectors $\mathbf{w}(1)$, $\mathbf{w}(2)$, $\dots$, such that the cost function $\mathcal{E}(\mathbf{w})$ is reduced at each iteration of the algorithm, as shown by

$$\mathcal{E}(\mathbf{w}(n+1)) < \mathcal{E}(\mathbf{w}(n)) \quad (3.10)$$

where $\mathbf{w}(n)$ is the old value of the weight vector and $\mathbf{w}(n+1)$ is its updated value.

We hope that the algorithm will eventually converge onto the optimal solution $\mathbf{w}^*$. We say “hope” because there is a distinct possibility that the algorithm will diverge (i.e., become unstable) unless special precautions are taken.

In this section, we describe three unconstrained-optimization methods that rely on the idea of iterative descent in one form or another (Bertsekas, 1995).

Method of Steepest Descent

In the method of steepest descent, the successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, that is, in a direction opposite to the gradient vector $\nabla \mathcal{E}(\mathbf{w})$. For convenience of presentation, we write

$$\mathbf{g} = \nabla \mathcal{E}(\mathbf{w}) \quad (3.11)$$

Accordingly, the steepest-descent algorithm is formally described by

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \mathbf{g}(n) \quad (3.12)$$

where η is a positive constant called the *stepsize*, or *learning-rate, parameter*, and $\mathbf{g}(n)$ is the gradient vector evaluated at the point $\mathbf{w}(n)$. In going from iteration n to $n+1$, the algorithm applies the *correction*

$$\begin{aligned} \Delta \mathbf{w}(n) &= \mathbf{w}(n+1) - \mathbf{w}(n) \\ &= -\eta \mathbf{g}(n) \end{aligned} \quad (3.13)$$

Equation (3.13) is in fact a formal statement of the error-correction rule described in the introductory chapter.

To show that the formulation of the steepest-descent algorithm satisfies the condition of Eq. (3.10) for iterative descent, we use a first-order Taylor series expansion around $\mathbf{w}(n)$ to approximate $\mathcal{E}(\mathbf{w}(n+1))$ as

$$\mathcal{E}(\mathbf{w}(n+1)) \approx \mathcal{E}(\mathbf{w}(n)) + \mathbf{g}^T(n) \Delta \mathbf{w}(n)$$

the use of which is justified for small η . Substituting Eq. (3.13) into this approximate relation yields

$$\begin{aligned}\mathcal{E}(\mathbf{w}(n+1)) &\approx \mathcal{E}(\mathbf{w}(n)) - \eta \mathbf{g}^T(n) \mathbf{g}(n) \\ &= \mathcal{E}(\mathbf{w}(n)) - \eta \|\mathbf{g}(n)\|^2\end{aligned}$$

which shows that, for a positive learning-rate parameter η , the cost function is decreased as the algorithm progresses from one iteration to the next. The reasoning presented here is approximate in that this end result is true only for small enough learning rates.

The method of steepest descent converges to the optimal solution $\mathbf{w}^*$ slowly. Moreover, the learning-rate parameter η has a profound influence on its convergence behavior:

- When η is small, the transient response of the algorithm is *overdamped*, in that the trajectory traced by $\mathbf{w}(n)$ follows a smooth path in the $\mathcal{W}$ -plane, as illustrated in Fig. 3.2a.
- When η is large, the transient response of the algorithm is *underdamped*, in that the trajectory of $\mathbf{w}(n)$ follows a zigzagging (oscillatory) path, as illustrated in Fig. 3.2b.
- When η exceeds a certain critical value, the algorithm becomes unstable (i.e., it diverges).

Newton's Method

For a more elaborate optimization technique, we may look to *Newton's method*, the basic idea of which is to minimize the quadratic approximation of the cost function $\mathcal{E}(\mathbf{w})$ around the current point $\mathbf{w}(n)$; this minimization is performed at each iteration of the algorithm. Specifically, using a *second-order* Taylor series expansion of the cost function around the point $\mathbf{w}(n)$, we may write

$$\begin{aligned}\Delta \mathcal{E}(\mathbf{w}(n)) &= \mathcal{E}(\mathbf{w}(n+1)) - \mathcal{E}(\mathbf{w}(n)) \\ &\approx \mathbf{g}^T(n) \Delta \mathbf{w}(n) + \frac{1}{2} \Delta \mathbf{w}^T(n) \mathbf{H}(n) \Delta \mathbf{w}(n)\end{aligned}\tag{3.14}$$

As before, $\mathbf{g}(n)$ is the M -by-1 gradient vector of the cost function $\mathcal{E}(\mathbf{w})$ evaluated at the point $\mathbf{w}(n)$. The matrix $\mathbf{H}(n)$ is the m -by- m *Hessian* of $\mathcal{E}(\mathbf{w})$, also evaluated at $\mathbf{w}(n)$. The Hessian of $\mathcal{E}(\mathbf{w})$ is defined by

$$\begin{aligned}\mathbf{H} &= \nabla^2 \mathcal{E}(\mathbf{w}) \\ &= \begin{bmatrix} \frac{\partial^2 \mathcal{E}}{\partial w_1^2} & \frac{\partial^2 \mathcal{E}}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \mathcal{E}}{\partial w_1 \partial w_M} \\ \frac{\partial^2 \mathcal{E}}{\partial w_2 \partial w_1} & \frac{\partial^2 \mathcal{E}}{\partial w_2^2} & \cdots & \frac{\partial^2 \mathcal{E}}{\partial w_2 \partial w_M} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 \mathcal{E}}{\partial w_M \partial w_1} & \frac{\partial^2 \mathcal{E}}{\partial w_M \partial w_2} & \cdots & \frac{\partial^2 \mathcal{E}}{\partial w_M^2} \end{bmatrix}\end{aligned}\tag{3.15}$$

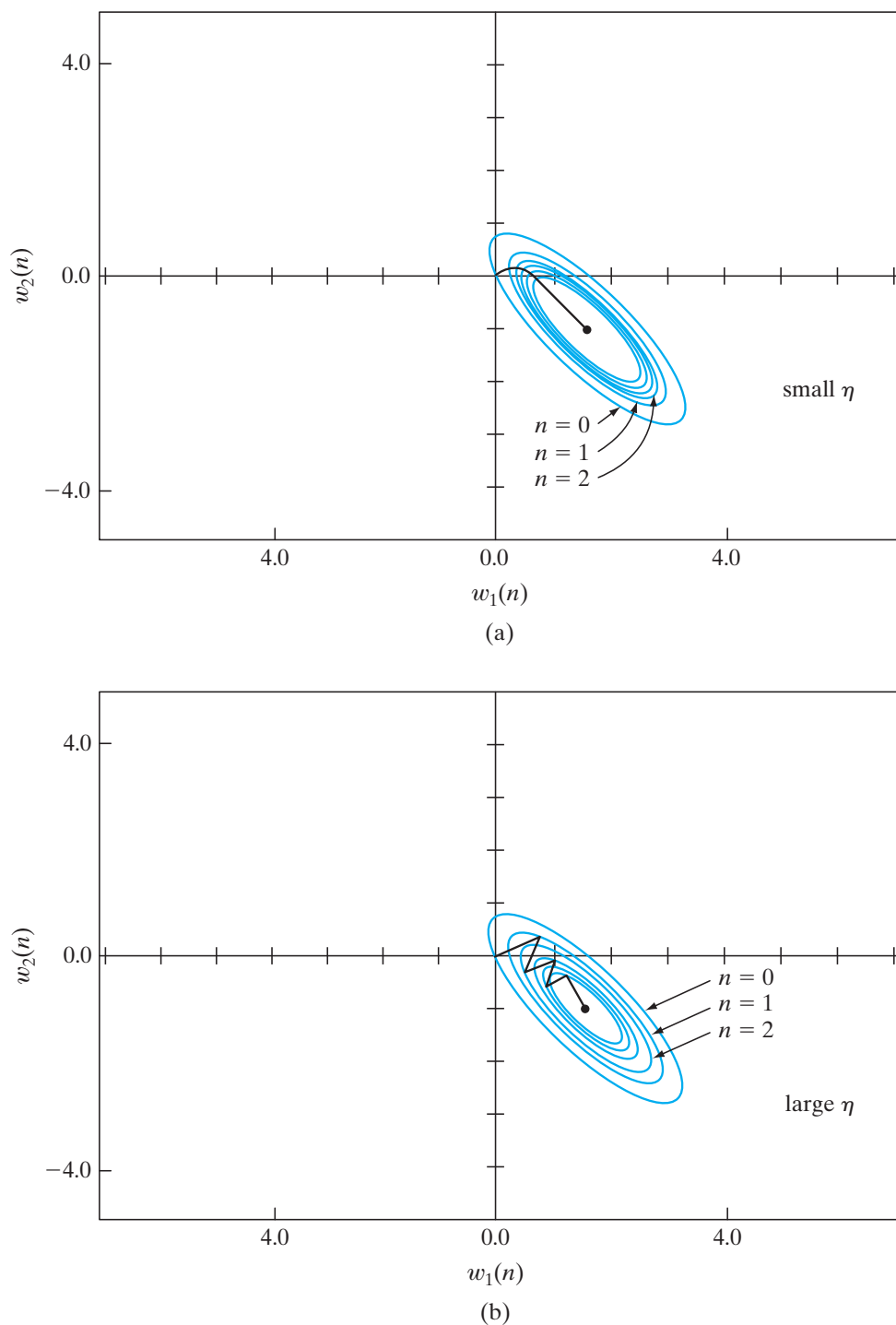


FIGURE 3.2 Trajectory of the method of steepest descent in a two-dimensional space for two different values of learning-rate parameter: (a) small η (b) large η . The coordinates w_1 and w_2 are elements of the weight vector $\mathbf{w}$; they both lie in the $\mathcal{W}$ -plane.

Equation (3.15) requires the cost function $\mathcal{E}(\mathbf{w})$ to be twice continuously differentiable with respect to the elements of $\mathbf{w}$. Differentiating¹ Eq. (3.14) with respect to $\Delta\mathbf{w}$, we minimize the resulting change $\Delta\mathcal{E}(\mathbf{w})$ when

$$\mathbf{g}(n) + \mathbf{H}(n)\Delta\mathbf{w}(n) = \mathbf{0}$$

Solving this equation for $\Delta\mathbf{w}(n)$ yields

$$\Delta\mathbf{w}(n) = -\mathbf{H}^{-1}(n)\mathbf{g}(n)$$

That is,

$$\begin{aligned}\mathbf{w}(n+1) &= \mathbf{w}(n) + \Delta\mathbf{w}(n) \\ &= \mathbf{w}(n) - \mathbf{H}^{-1}(n)\mathbf{g}(n)\end{aligned}\tag{3.16}$$

where $\mathbf{H}^{-1}(n)$ is the inverse of the Hessian of $\mathcal{E}(\mathbf{w})$.

Generally speaking, Newton's method converges quickly asymptotically and does *not* exhibit the zigzagging behavior that sometimes characterizes the method of steepest descent. However, for Newton's method to work, the Hessian $\mathbf{H}(n)$ has to be a *positive definite matrix* for all n . Unfortunately, in general, there is no guarantee that $\mathbf{H}(n)$ is positive definite at every iteration of the algorithm. If the Hessian $\mathbf{H}(n)$ is not positive definite, modification of Newton's method is necessary (Powell, 1987; Bertsekas, 1995). In any event, a major limitation of Newton's method is its computational complexity.

Gauss–Newton Method

To deal with the computational complexity of Newton's method without seriously compromising its convergence behavior, we may use the *Gauss–Newton method*. To apply this method, we adopt a cost function that is expressed as the sum of error squares. Let

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n e^2(i)\tag{3.17}$$

where the scaling factor $\frac{1}{2}$ is included to simplify matters in subsequent analysis. All the error terms in this formula are calculated on the basis of a weight vector $\mathbf{w}$ that is fixed over the entire observation interval $1 \leq i \leq n$.

The error signal $e(i)$ is a function of the adjustable weight vector $\mathbf{w}$. Given an operating point $\mathbf{w}(n)$, we linearize the dependence of $e(i)$ on $\mathbf{w}$ by introducing the new term

$$e'(i, \mathbf{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \mathbf{w}} \right]_{\mathbf{w}=\mathbf{w}(n)}^T \times (\mathbf{w} - \mathbf{w}(n)), \quad i = 1, 2, \dots, n$$

Equivalently, by using matrix notation, we may write

$$\mathbf{e}'(n, \mathbf{w}) = \mathbf{e}(n) + \mathbf{J}(n) (\mathbf{w} - \mathbf{w}(n))\tag{3.18}$$

where $\mathbf{e}(n)$ is the error vector

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

and $\mathbf{J}(n)$ is the n -by- m Jacobian of $\mathbf{e}(n)$:

$$\mathbf{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_M} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_M} \end{bmatrix}_{\mathbf{w}=\mathbf{w}(n)} \quad (3.19)$$

The Jacobian $\mathbf{J}(n)$ is the transpose of the m -by- n gradient matrix $\nabla \mathbf{e}(n)$, where

$$\nabla \mathbf{e}(n) = [\nabla e(1), \nabla e(2), \dots, \nabla e(n)]$$

The updated weight vector $\mathbf{w}(n+1)$ is now defined by

$$\mathbf{w}(n+1) = \arg \min_{\mathbf{w}} \left\{ \frac{1}{2} \|\mathbf{e}'(n, \mathbf{w})\|^2 \right\} \quad (3.20)$$

Using Eq. (3.18) to evaluate the squared Euclidean norm of $\mathbf{e}'(n, \mathbf{w})$, we get

$$\begin{aligned} \frac{1}{2} \|\mathbf{e}'(n, \mathbf{w})\|^2 &= \frac{1}{2} \|\mathbf{e}(n)\|^2 + \mathbf{e}^T(n) \mathbf{J}(n) (\mathbf{w} - \mathbf{w}(n)) \\ &\quad + \frac{1}{2} (\mathbf{w} - \mathbf{w}(n))^T \mathbf{J}^T(n) \mathbf{J}(n) (\mathbf{w} - \mathbf{w}(n)) \end{aligned}$$

Hence, differentiating this expression with respect to $\mathbf{w}$ and setting the result equal to zero, we obtain

$$\mathbf{J}^T(n) \mathbf{e}(n) + \mathbf{J}^T(n) \mathbf{J}(n) (\mathbf{w} - \mathbf{w}(n)) = \mathbf{0}$$

Solving this equation for $\mathbf{w}$, we may thus write, in light of Eq. 3.20,

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n) \mathbf{J}(n))^{-1} \mathbf{J}^T(n) \mathbf{e}(n) \quad (3.21)$$

which describes the *pure* form of the Gauss–Newton method.

Unlike Newton's method, which requires knowledge of the Hessian of the cost function $\mathcal{E}(n)$, the Gauss–Newton method requires only the Jacobian of the error vector $\mathbf{e}(n)$. However, for the Gauss–Newton iteration to be computable, the matrix product $\mathbf{J}^T(n) \mathbf{J}(n)$ must be nonsingular.

With regard to the latter point, we recognize that $\mathbf{J}^T(n) \mathbf{J}(n)$ is always nonnegative definite. To ensure that it is nonsingular, the Jacobian $\mathbf{J}(n)$ must have row *rank* n ; that is, the n rows of $\mathbf{J}(n)$ in Eq. (3.19) must be linearly independent. Unfortunately, there is no guarantee that this condition will always hold. To guard against the possibility that $\mathbf{J}(n)$ is rank deficient, the customary practice is to add the diagonal matrix $\delta \mathbf{I}$ to the matrix $\mathbf{J}^T(n) \mathbf{J}(n)$, where $\mathbf{I}$ is the identity matrix. The parameter δ is a small positive constant chosen to ensure that

$$\mathbf{J}^T(n) \mathbf{J}(n) + \delta \mathbf{I} \text{ is positive definite for all } n$$

On this basis, the Gauss–Newton method is implemented in the slightly modified form

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n) + \delta\mathbf{I})^{-1}\mathbf{J}^T(n)\mathbf{e}(n) \quad (3.22)$$

The effect of the added term $\delta\mathbf{I}$ is progressively reduced as the number of iterations, n , is increased. Note also that the recursive equation (3.22) is the solution of the *modified* cost function

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2} \left\{ \sum_{i=1}^n e^2(i) + \delta \|\mathbf{w} - \mathbf{w}(n)\|^2 \right\} \quad (3.23)$$

where $\mathbf{w}(n)$ is the *current value* of the weight vector $\mathbf{w}(i)$.

In the literature on signal processing, the addition of the term $\delta\mathbf{I}$ in Eq. (3.22) is referred to as *diagonal loading*. The addition of this term is accounted for by expanding the cost function $\mathcal{E}(\mathbf{w})$ in the manner described in Eq. (3.23), where we now have two terms (ignoring the scaling factor $\frac{1}{2}$):

- The first term, $\sum_{i=1}^n e^2(i)$, is the standard sum of squared errors, which depends on the training data.
- The second term contains the squared Euclidean norm, $\|\mathbf{w} - \mathbf{w}(n)\|^2$, which depends on the filter structure. In effect, this term acts as a *stabilizer*.

The scaling factor δ is commonly referred to as a *regularization parameter*, and the resulting modification of the cost function is correspondingly referred to as *structural regularization*. The issue of regularization is discussed in great detail in Chapter 7.

3.4 THE WIENER FILTER

The ordinary least-squares estimator was discussed in Chapter 2, where the traditional approach to minimization was used to find the least-squares solution from an observation model of the environment. To conform to the terminology adopted in this chapter, we will refer to it as the *least-squares filter*. Moreover, we will rederive the formula for this filter by using the Gauss–Newton method.

To proceed then, we use Eqs. (3.3) and (3.4) to define the error vector as

$$\begin{aligned} \mathbf{e}(n) &= \mathbf{d}(n) - [\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(n)]^T \mathbf{w}(n) \\ &= \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n) \end{aligned} \quad (3.24)$$

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 the error vector $\mathbf{e}(n)$ with respect to $\mathbf{w}(n)$ yields the gradient matrix

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

Correspondingly, the Jacobian of $\mathbf{e}(n)$ is

$$\mathbf{J}(n) = -\mathbf{X}(n) \quad (3.25)$$

Since the error equation (3.18) is already linear in the weight vector $\mathbf{w}(n)$, the Gauss–Newton method converges in a single iteration, as shown here. Substituting Eqs. (3.24) and (3.25) into (3.21) yields

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)(\mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)) \\ &= (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n) \end{aligned} \quad (3.26)$$

The term $(\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)$ is called the *pseudoinverse* of the data matrix $\mathbf{X}(n)$; that is,²

$$\mathbf{X}^+(n) = (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n) \quad (3.27)$$

Hence, we may rewrite Eq. (3.26) in the compact form

$$\mathbf{w}(n+1) = \mathbf{X}^+(n)\mathbf{d}(n) \quad (3.28)$$

This formula represents a convenient way of stating the following:

The weight vector $\mathbf{w}(n+1)$ solves the linear least-squares problem, defined over an observation interval of duration n , as the product of two terms: the pseudoinverse $\mathbf{X}^+(n)$ and the desired response vector $\mathbf{d}(n)$.

Wiener Filter: Limiting Form of the Least-Squares Filter for an Ergodic Environment

Let $\mathbf{w}_o$ denote the limiting form of the least-squares filter as the number of observations, n , is allowed to approach infinity. We may then use Eq. (3.26) to write

$$\begin{aligned} \mathbf{w}_o &= \lim_{n \rightarrow \infty} \mathbf{w}(n+1) \\ &= \lim_{n \rightarrow \infty} (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n) \\ &= \lim_{n \rightarrow \infty} \left(\frac{1}{n} \mathbf{X}^T(n)\mathbf{X}(n) \right)^{-1} \times \lim_{n \rightarrow \infty} \frac{1}{n} \mathbf{X}^T(n)\mathbf{d}(n) \end{aligned} \quad (3.29)$$

Suppose now the input vector $\mathbf{x}(i)$ and the corresponding desired response $d(i)$ are drawn from a jointly *ergodic environment* that is also stationary. We may then substitute time averages for ensemble averages. By definition, the ensemble-averaged form of the correlation matrix of the input vector $\mathbf{x}(i)$ is

$$\mathbf{R}_{xx} = \mathbb{E}[\mathbf{x}(i)\mathbf{x}^T(i)] \quad (3.30)$$

and, correspondingly, the ensemble-averaged form of the *cross-correlation vector* between the input vector $\mathbf{x}(i)$ and the desired response vector $d(i)$ is

$$\mathbf{r}_{dx} = \mathbb{E}[\mathbf{x}(i)d(i)] \quad (3.31)$$

where $\mathbb{E}$ is the expectation operator. Therefore, under the ergodicity assumption, we may now write

$$\mathbf{R}_{xx} = \lim_{n \rightarrow \infty} \frac{1}{n} \mathbf{X}(n)\mathbf{X}^T(n)$$

and

$$\mathbf{r}_{dx} = \lim_{n \rightarrow \infty} \mathbf{X}^T(n) \mathbf{d}(n)$$

Accordingly, we may recast Eq. (3.29) in terms of ensemble-averaged correlation parameters as

$$\mathbf{w}_o = \mathbf{R}_{xx}^{-1} \mathbf{r}_{dx} \quad (3.32)$$

where $\mathbf{R}_{xx}^{-1}$ is the inverse of the correlation matrix $\mathbf{R}_{xx}$. The formula of Eq. (3.32) is the ensemble-averaged version of the least-squares solution defined in Eq. (2.32).

The weight vector $\mathbf{w}_o$ is called the *Wiener solution* to the optimum linear filtering problem (Widrow and Stearns, 1985; Haykin, 2002). Accordingly, we may make the statement:

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

Designing the Wiener filter requires knowledge of the second-order statistics: the correlation matrix $\mathbf{R}_{xx}$ of the input vector $\mathbf{x}(n)$, and the cross-correlation vector $\mathbf{r}_{xd}$ between $\mathbf{x}(n)$ and the desired response $d(n)$. However, this information is not available when the environment in which the filter operates is unknown. We may deal with such an environment by using a *linear adaptive filter*, adaptive in the sense that the filter is able to adjust its free parameters in response to statistical variations in the environment. A highly popular algorithm for doing this kind of adjustment on a continuing-time basis is the least-mean-square algorithm, discussed next.

3.5 THE LEAST-MEAN-SQUARE ALGORITHM

The *least-mean-square (LMS) algorithm* is configured to minimize the *instantaneous value* of the cost function,

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

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

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

As with the least-squares filter, the LMS algorithm operates with a linear neuron, so we may express the error signal as

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

Hence,

$$\frac{\partial e(n)}{\partial \hat{\mathbf{w}}(n)} = -\mathbf{x}(n)$$

and

$$\frac{\partial \mathcal{E}(\hat{\mathbf{w}})}{\partial \hat{\mathbf{w}}(n)} = -\mathbf{x}(n)e(n)$$

Using this latter result as the *instantaneous estimate* of the gradient vector, we may write

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

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

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

It is also noteworthy that the inverse of the learning-rate parameter η acts as a measure of the *memory* of the LMS algorithm: The smaller we make η , the longer the memory span over which the LMS algorithm remembers past data will be. Consequently, when η is small, the LMS algorithm performs accurately, but the convergence rate of the algorithm is slow.

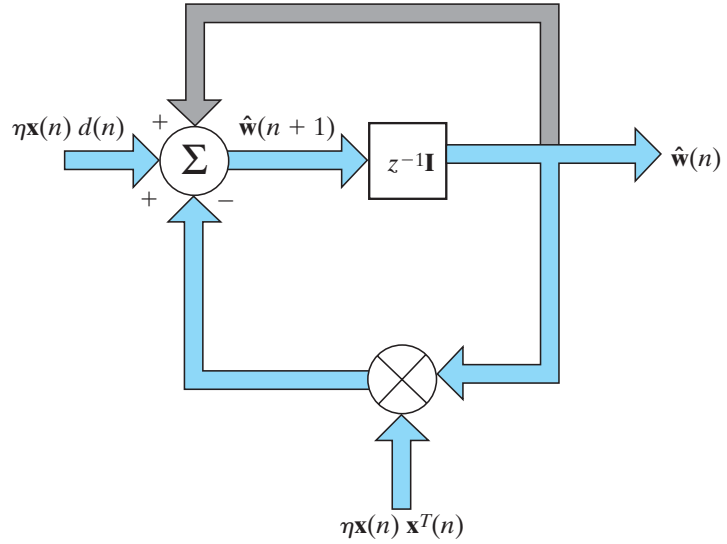
In deriving Eq. (3.37), we have used $\hat{\mathbf{w}}(n)$ in place of $\mathbf{w}(n)$ to emphasize the fact that the LMS algorithm produces an *instantaneous estimate* of the weight vector that would result from the use of the method of steepest-descent. As a consequence, in using the LMS algorithm we sacrifice a distinctive feature of the steepest-descent algorithm. In the steepest-descent algorithm, the weight vector $\mathbf{w}(n)$ follows a well-defined trajectory in the weight space $\mathcal{W}$ for a prescribed η . In contrast, in the LMS algorithm, the weight vector $\hat{\mathbf{w}}(n)$ traces a random trajectory. For this reason, the LMS algorithm is sometimes referred to as a “stochastic gradient algorithm.” As the number of iterations in the LMS algorithm approaches infinity, $\hat{\mathbf{w}}(n)$ performs a random walk (Brownian motion) about the Wiener solution $\mathbf{w}_o$. The important point to note, however, is the fact that, unlike the method of steepest descent, the LMS algorithm does *not* require knowledge of the statistics of the environment. This feature of the LMS algorithm is important from a practical perspective.

A summary of the LMS algorithm, based on Eqs. (3.35) and (3.37), is presented in Table 3.1, which clearly illustrates the simplicity of the algorithm. As indicated in this table, *initialization* of the algorithm is done by simply setting the value of the weight vector $\hat{\mathbf{w}}(0) = \mathbf{0}$.

TABLE 3.1 Summary of the LMS Algorithm

<i>Training Sample:</i>	Input signal vector = $\mathbf{x}(n)$ Desired response = $d(n)$
<i>User-selected parameter:</i>	η
<i>Initialization.</i>	Set $\hat{\mathbf{w}}(0) = \mathbf{0}$.
<i>Computation.</i>	For $n = 1, 2, \dots$, compute $e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$ $\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n)e(n)$

FIGURE 3.3 Signal-flow graph representation of the LMS algorithm. The graph embodies feedback depicted in color.



Signal-Flow Graph Representation of the LMS Algorithm

By combining Eqs. (3.35) and (3.37), we may express the evolution of the weight vector in the LMS algorithm as

$$\begin{aligned}\hat{\mathbf{w}}(n+1) &= \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n)[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)] \\ &= [\mathbf{I} - \eta \mathbf{x}(n)\mathbf{x}^T(n)]\hat{\mathbf{w}}(n) + \eta \mathbf{x}(n)d(n)\end{aligned}\quad (3.38)$$

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

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

where z^{-1} is the *unit-time delay operator*, implying storage. Using Eqs. (3.38) and (3.39), we may thus represent the LMS algorithm by the signal-flow graph depicted in Fig. 3.3. This signal-flow graph reveals that the LMS algorithm is an example of a *stochastic feedback system*. The presence of feedback has a profound impact on the convergence behavior of the LMS algorithm.