

Least-Mean-Square Algorithm

5.1 Introduction

In this chapter we study a primitive class of neural networks consisting of a single neuron and operating under the assumption of linearity. This class of networks is important for three reasons. First, the theory of *linear adaptive filters* employing a single linear neuron model is well developed, and it has been applied successfully in such diverse fields as communications, control, radar, sonar, and biomedical engineering (Haykin, 1991; Widrow and Stearns, 1985). Second, it is a by-product of the pioneering work done on neural networks during the 1960s. Last, but no means least, a study of linear adaptive filters (alongside that of single-layer perceptrons, covered in Chapter 4) paves the way for the theoretical development of the more general case of multilayer perceptrons that includes the use of nonlinear elements, which is undertaken in Chapter 6.

We begin our study by reviewing briefly the linear optimum filtering problem. We then formulate the highly popular *least-mean-square (LMS) algorithm*, also known as the *delta rule* or the *Widrow–Hoff rule* (Widrow and Hoff, 1960). The LMS algorithm operates with a single linear neuron model. The design of the LMS algorithm is very simple, yet a detailed analysis of its convergence behavior is a challenging mathematical task.

The LMS algorithm was originally formulated by Widrow and Hoff for use in *adaptive switching circuits*. The machine used to perform the LMS algorithm was called an *Adaline*, the development of which was inspired by Rosenblatt's perceptron. Subsequently, the application of the LMS algorithm was extended to *adaptive equalization* of telephone channels for high-speed data transmission (Lucky, 1965, 1966), *adaptive antennas* for the suppression of interfering signals originating from unknown directions (Widrow et al., 1967), *adaptive echo cancellation* for the suppression of echoes experienced on long-distance communication circuits (Sondhi and Berkley, 1980; Sondhi, 1967), *adaptive differential pulse code modulation* for efficient encoding of speech signals in digital communications (Jayant and Noll, 1984), *adaptive signal detection* in a nonstationary environment (Zeidler, 1990), and *adaptive cross-polar cancellation* applied to radar polarimetry for precise navigation along a confined waterway (Ukrainec and Haykin, 1989). Indeed, the LMS algorithm has established itself as an important functional block in the ever-expanding field of *adaptive signal processing*.

Organization of the Chapter

The main body of the chapter is organized as follows. The development of the Wiener–Hopf equations for linear optimum filtering is presented in Section 5.2. This is followed by a description of the method of steepest descent in Section 5.3, which is a deterministic

algorithm for the recursive computation of the optimum filter solution. The method of steepest descent provides the heuristics for deriving the LMS algorithm; this is done in Section 5.4. The essential factors affecting the convergence behavior of the LMS algorithm are considered briefly in Section 5.5. In Section 5.6 we consider the learning curve of the LMS algorithm, which is then followed by a discussion on the use of learning-rate annealing schedules in Section 5.7. In Section 5.8 we describe the Adaline based on the LMS algorithm. The chapter concludes with some final remarks in Section 5.9.

5.2 Wiener–Hopf Equations

Consider a set of p sensors located at different points in space, as depicted in Fig. 5.1. Let x_1, x_2, \dots, x_p be the individual signals produced by these sensors. These signals are applied to a corresponding set of weights w_1, w_2, \dots, w_p . The weighted signals are then summed to produce the output signal y . The requirement is to determine the optimum setting of the weights w_1, w_2, \dots, w_p so as to minimize the difference between the system output y and some desired response d in a mean-square sense. The solution to this fundamental problem lies in the Wiener–Hopf equations.

The system described in Fig. 5.1 may be viewed as a *spatial filter*. The input–output relation of the filter is described by the summation

$$y = \sum_{k=1}^p w_k x_k \quad (5.1)$$

Let d denote the *desired response* or *target output* for the filter. We may then define the *error signal*,

$$e = d - y \quad (5.2)$$

As performance measure or cost function, we introduce the *mean-squared error* defined as follows:

$$J = \frac{1}{2} E[e^2] \quad (5.3)$$

where E is the statistical expectation operator. The factor $\frac{1}{2}$ is included for convenience of presentation, which will become apparent later. We may now state the *linear optimum filtering problem* as follows:

Determine the optimum set of weights $w_{o1}, w_{o2}, \dots, w_{op}$ for which the mean-squared error J is minimum.

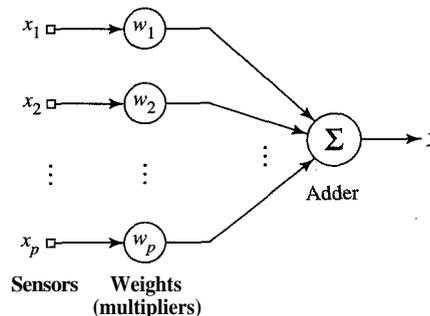


FIGURE 5.1 Spatial filter.

The solution of this filtering problem is referred to in the signal processing literature as the *Wiener filter*, in recognition of the pioneering work done by Wiener (Haykin, 1991; Widrow and Stearns, 1985).

Substituting Eqs. (5.1) and (5.2) in (5.3), then expanding terms, we get

$$J = \frac{1}{2} E[d^2] - E \left[\sum_{k=1}^p w_k x_k d \right] + \frac{1}{2} E \left[\sum_{j=1}^p \sum_{k=1}^p w_j w_k x_j x_k \right] \quad (5.4)$$

where the double summation is used to represent the square of a summation. Since mathematical expectation is a linear operation, we may interchange the order of expectation and summation in Eq. (5.4). Accordingly, we may rewrite Eq. (5.4) in the equivalent form

$$J = \frac{1}{2} E[d^2] - \sum_{k=1}^p w_k E[x_k d] + \frac{1}{2} \sum_{j=1}^p \sum_{k=1}^p w_j w_k E[x_j x_k] \quad (5.5)$$

where the weights are treated as constants and therefore taken outside the expectations.

At this point, we find it convenient to introduce some definitions that account for the three different expectations appearing in Eq. (5.5), as described here:

- The expectation $E[d^2]$ is the *mean-square value* of the desired response d ; let

$$r_d = E[d^2] \quad (5.6)$$

- The expectation $E[dx_k]$ is the *cross-correlation function* between the desired response d and the sensory signal x_k ; let

$$r_{dx}(k) = E[dx_k], \quad k = 1, 2, \dots, p \quad (5.7)$$

- The expectation $E[x_j x_k]$ is the *autocorrelation function* of the set of sensory signals themselves; let

$$r_x(j, k) = E[x_j x_k], \quad j, k = 1, 2, \dots, p \quad (5.8)$$

In light of these definitions, we may now simplify the format of Eq. (5.5) as follows:

$$J = \frac{1}{2} r_d - \sum_{k=1}^p w_k r_{dx}(k) + \frac{1}{2} \sum_{j=1}^p \sum_{k=1}^p w_j w_k r_x(j, k) \quad (5.9)$$

A multidimensional plot of the cost function J versus the weights (free parameters) w_1, w_2, \dots, w_p constitutes the *error-performance surface*, or simply the *error surface* of the filter. The error surface is bowl-shaped, with a well-defined bottom or global minimum point. It is precisely at this point where the spatial filter of Fig. 5.1 is optimum in the sense that the mean-squared error attains its minimum value J_{\min} .

To determine this optimum condition, we differentiate the cost function J with respect to the weight w_k and then set the result equal to zero for all k . The derivative of J with respect to w_k is called the *gradient* of the error surface with respect to that particular weight. Let $\nabla_{w_k} J$ denote this gradient, as shown by

$$\nabla_{w_k} J = \frac{\partial J}{\partial w_k}, \quad k = 1, 2, \dots, p \quad (5.10)$$

Differentiating Eq. (5.9) with respect to w_k , we readily find that

$$\nabla_{w_k} J = -r_{dx}(k) + \sum_{j=1}^p w_j r_x(j, k) \quad (5.11)$$

The optimum condition of the filter is thus defined by

$$\nabla_{w_k} J = 0, \quad k = 1, 2, \dots, p \tag{5.12}$$

Let w_{ok} denote the optimum setting of weight w_k . Then, from Eq. (5.11), we find that the optimum weights of the spatial filter in Fig. 5.1 are determined by the following set of simultaneous equations:

$$\sum_{j=1}^p w_{oj} r_x(j, k) = r_{xd}(k), \quad k = 1, 2, \dots, p \tag{5.13}$$

This system of equations is known as the *Wiener-Hopf equations*, and the filter whose weights satisfy the Wiener-Hopf equations is called a *Wienerfilter*.

5.3 Method of Steepest Descent

To solve the Wiener-Hopf equations (5.13) for tap weights of the optimum spatial filter, we basically need to compute the inverse of a p-by-p matrix made up of the different values of the autocorrelation function $r_x(j, k)$ for $j, k = 1, 2, \dots, p$. We may avoid the need for this matrix inversion by using the *method of steepest descent*. According to this method, the weights of the filter assume a time-varying form, and their values are adjusted in an *iterative* fashion along the error surface with the aim of moving them progressively toward the optimum solution. The method of steepest descent has the task of continually seeking the bottom point of the error surface of the filter. Now it is intuitively reasonable that successive adjustments applied to the tap weights of the filter be in the direction of steepest descent of the error surface, that is, in a *direction opposite to the gradient vector* whose elements are defined by $\nabla_{w_k} J$ for $k = 1, 2, \dots, p$. Such an adjustment is illustrated in Fig. 5.2 for the case of a single weight.

Let $w_k(n)$ denote the value of weight w_k of the spatial filter calculated at iteration or discrete time n by the method of steepest descent. In a corresponding way, the gradient of the error surface of the filter with respect to this weight takes on a time-varying form of its own, as shown by [in light of Eq. (5.11)]

$$\nabla_{w_k} J(n) = -r_{dx}(k) + \sum_{j=1}^p w_j(n) r_x(j, k) \tag{5.14}$$

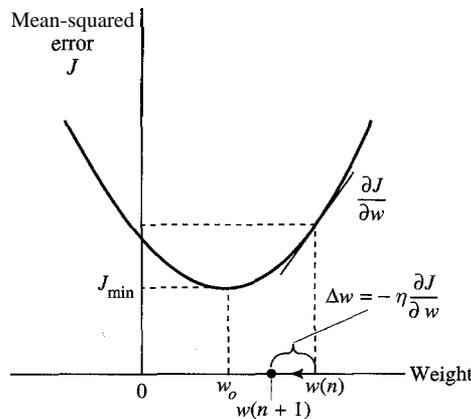


FIGURE 5.2 Illustrating the mean-squared error criterion for a single weight.

Note that the indices j and k refer to locations of different sensors in space, whereas the index n refer to iteration number. According to the method of steepest descent, the adjustment applied to the weight $w_k(n)$ at iteration n is defined by

$$\Delta w_k(n) = -\eta \nabla_{w_k} J(n), \quad k = 1, 2, \dots, p \quad (5.15)$$

where η is a positive constant called the *learning-rate parameter*. Given the *old* value of the k th weight $w_k(n)$ at iteration n , the *updated* value of this weight at the next iteration $n + 1$ is computed as

$$\begin{aligned} w_k(n + 1) &= w_k(n) + \Delta w_k(n) \\ &= w_k(n) - \eta \nabla_{w_k} J(n), \quad k = 1, 2, \dots, p \end{aligned} \quad (5.16)$$

We may thus state the method of steepest descent as follows:

The updated value of the k th weight of a Wiener filter (designed to operate in a minimum mean-square error sense) equals the old value of the tap weight plus a correction that is proportional to the negative of the gradient of the error surface with respect to that particular weight.

Substituting Eq. (5.14) in (5.16), we may finally formulate the method of steepest descent in terms of the correlation functions $r_x(j, k)$ and $r_{dx}(k)$ as follows:

$$w_k(n + 1) = w_k(n) + \eta \left[r_{dx}(k) - \sum_{j=1}^M w_j(n) r_x(j, k) \right], \quad k = 1, 2, \dots, p \quad (5.17)$$

The method of steepest descent is *exact* in the sense that there are no approximations made in its derivation. The derivation presented here is based on minimizing the mean-squared error, defined as

$$J(n) = \frac{1}{2} E[e^2(n)] \quad (5.18)$$

This cost function is an *ensemble average*, taken at a particular instant of time n and over an *ensemble* of spatial filters of an identical design but with different inputs drawn from the same population. The method of steepest descent may also be derived by minimizing the *sum of error squares*:

$$\begin{aligned} \mathcal{E}_{\text{total}}(n) &= \sum_{i=1}^n \mathcal{E}(i) \\ &= \frac{1}{2} \sum_{i=1}^n e^2(i) \end{aligned} \quad (5.19)$$

where the integration is now taken over all iterations of the algorithm, but for a particular realization of the spatial filter. This second approach yields a result identical to that described in Eq. (5.17), but with a new interpretation of the correlation functions. Specifically, the autocorrelation function r_x and cross-correlation function r_{dx} are now defined as *time averages* rather than as ensemble averages. If the physical processes responsible for the generation of the input signals and desired response are *jointly ergodic*, then we are justified in substituting time averages for ensemble averages (Gray and Davisson, 1986; Papoulis, 1984).

In any event, care has to be exercised in the selection of the learning-rate parameter η for the method of steepest descent to work. Also, a practical limitation of the method

of steepest descent is that it requires knowledge of the spatial correlation functions $r_{dx}(k)$ and $r_x(j, k)$. Now, when the filter operates in an *unknown* environment, these correlation functions are not available, in which case we are forced to use *estimates* in their place. The least-mean-square algorithm, described in the next section, results from a simple and yet effective method of providing for these estimates.

5.4 Least-Mean-Square Algorithm

The *least-mean-square (LMS) algorithm* is based on the use of *instantaneous estimates* of the autocorrelation function $r_x(j, k)$ and the cross-correlation function $r_{xd}(k)$. These estimates are deduced directly from the defining equations (5.8) and (5.7) as follows:

$$\hat{r}_x(j, k; n) = x_j(n)x_k(n) \quad (5.20)$$

and

$$\hat{r}_{dx}(k; n) = x_k(n)d(n) \quad (5.21)$$

The use of a hat in \hat{r}_x and \hat{r}_{dx} is intended to signify that these quantities are “estimates.” The definitions introduced in Eqs. (5.20) and (5.21) have been generalized to include a *nonstationary environment*, in which case all the sensory signals and the desired response assume time-varying forms too. Thus, substituting $\hat{r}_x(j, k; n)$ and $\hat{r}_{dx}(k; n)$ in place of $r_x(j, k)$ and $r_{dx}(k)$ in Eq. (5.17), we get

$$\begin{aligned} \hat{w}_k(n+1) &= \hat{w}_k(n) + \eta \left[x_k(n)d(n) - \sum_{j=1}^p \hat{w}_j(n)x_j(n)x_k(n) \right] \\ &= \hat{w}_k(n) + \eta \left[d(n) - \sum_{j=1}^p \hat{w}_j(n)x_j(n) \right] x_k(n) \\ &= \hat{w}_k(n) + \eta [d(n) - y(n)]x_k(n), \quad k = 1, 2, \dots, p \end{aligned} \quad (5.22)$$

where $y(n)$ is the output of the spatial filter computed at iteration n in accordance with the LMS algorithm; that is,

$$y(n) = \sum_{j=1}^p \hat{w}_j(n)x_j(n) \quad (5.23)$$

Note that in Eq. (5.22) we have used $\hat{w}_k(n)$ in place of $w_k(n)$ to emphasize the fact that Eq. (5.22) involves “estimates” of the weights of the spatial filter.

Figure 5.3 illustrates the operational environment of the LMS algorithm, which is completely described by Eqs. (5.22) and (5.23). A summary of the LMS algorithm is presented in Table 5.1, which clearly illustrates the simplicity of the algorithm. As indicated in this table, for the *initialization* of the algorithm, it is customary to set all the initial values of the weights of the filter equal to zero.

In the method of steepest descent applied to a “known” environment, the weight vector $\mathbf{w}(n)$, made up of the weights $w_1(n), w_2(n), \dots, w_p(n)$, starts at some initial value $\mathbf{w}(0)$, and then follows a precisely defined trajectory (along the error surface) that eventually terminates on the optimum solution \mathbf{w}_o , provided that the learning-rate parameter η is chosen properly. In contrast, in the LMS algorithm applied to an “unknown” environment, the weight vector $\hat{\mathbf{w}}(n)$, representing an “estimate” of $\mathbf{w}(n)$, follows 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 optimum solution \mathbf{w}_o ; see Appendix D.

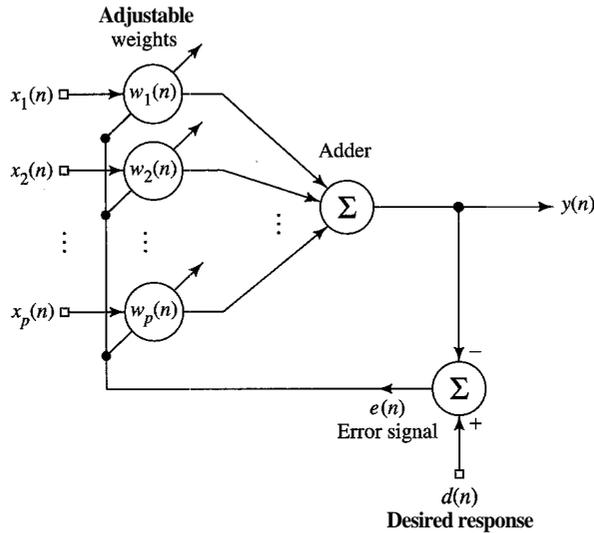


FIGURE 5.3 Adaptive spatial filter.

Another way of stating the basic difference between the method of steepest descent and the **LMS** algorithm is in terms of the error calculations involved. At any iteration n , the method of steepest descent minimizes the mean-squared error $J(n)$. This cost function involves ensemble averaging, the effect of which is to give the method of steepest descent an “exact” gradient vector that improves in pointing accuracy with increasing n . The **LMS** algorithm, on the other hand, minimizes an instantaneous estimate of the cost function $J(n)$. Consequently, the gradient vector in the **LMS** algorithm is “random,” and its pointing accuracy improves “on the average” with increasing n .

The basic difference between the method of steepest descent and the **LMS** algorithm may also be stated in terms of time-domain ideas, emphasizing other aspects of the adaptive filtering problem. The method of steepest descent minimizes the sum of error squares $\mathcal{E}_{\text{total}}(n)$, integrated over all previous iterations of the algorithm up to and including iteration n . Consequently, it requires the storage of information needed to provide running estimates of the autocorrelation function r_x and cross-correlation function r_{dx} . In contrast, the **LMS** algorithm simply minimizes the instantaneous error squared $\mathcal{E}(n)$, defined as $(\frac{1}{2})e^2(n)$, thereby reducing the storage requirement to the minimum possible. In particular, it does not require storing any more information than is present in the weights of the filter.

TABLE 5.1 Summary of the LMS Algorithm

1. Initialization. Set

$$\hat{w}_k(1) = 0 \quad \text{for } k = 1, 2, \dots, p$$

2. Filtering. For time $n = 1, 2, \dots$, compute

$$y(n) = \sum_{j=1}^p \hat{w}_j(n)x_j(n)$$

$$e(n) = d(n) - y(n)$$

$$\hat{w}_k(n+1) = w_k(n) + \eta e(n)x_k(n) \quad \text{for } k = 1, 2, \dots, p$$

It is also important to recognize that the **LMS** algorithm can operate in a stationary or nonstationary environment. By a “nonstationary” environment we mean one in which the statistics vary with time. In such a situation the optimum solution assumes a time-varying form, and the **LMS** algorithm therefore has the task of not only *seeking* the minimum point of the error surface but also *tracking* it. In this context, the smaller we make the learning-rate parameter η , the better will be the tracking behavior of the algorithm. However, this improvement in performance is attained at the cost of a slow adaptation rate (Haykin, 1991; Widrow and Stearns, 1985).

Signal-Flow Graph Representation of the LMS Algorithm

Equation (5.22) provides a complete description of the time evolution of the weights in the **LMS** algorithm. Rewriting the second line of this equation in matrix form, we may express it as follows:

$$\mathbf{w}(n+1) = \hat{\mathbf{w}}(n) + \eta[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)]\mathbf{x}(n) \quad (5.24)$$

where

$$\hat{\mathbf{w}}(n) = [\hat{w}_1(n), \hat{w}_2(n), \dots, \hat{w}_p(n)]^T \quad (5.25)$$

and

$$\mathbf{x}(n) = [x_1(n), x_2(n), \dots, x_p(n)]^T \quad (5.26)$$

Rearranging terms in Eq. (5.24), we have

$$\hat{\mathbf{w}}(n+1) = [\mathbf{I} - \eta\mathbf{x}(n)\mathbf{x}^T(n)]\hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)d(n) \quad (5.27)$$

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

$$\hat{\mathbf{w}}(n) = z^{-1}[\hat{\mathbf{w}}(n+1)] \quad (5.28)$$

where z^{-1} is the *unit-delay operator* implying storage. Using Eqs. (5.27) and (5.28), we may thus represent the **LMS** algorithm by the signal-flow graph depicted in Fig. 5.4.

The signal-flow graph of Fig. 5.4 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, as discussed next.

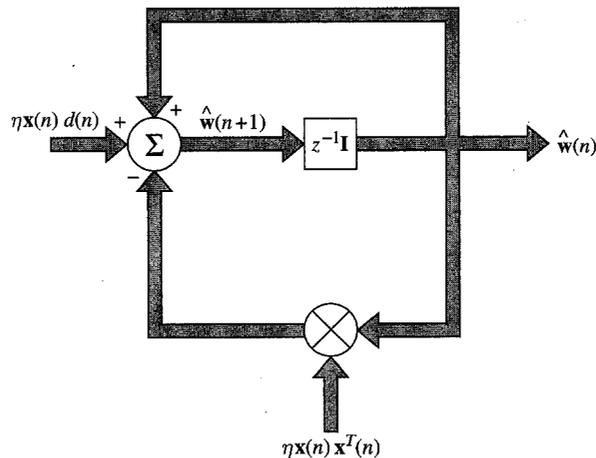


FIGURE 5.4 Signal-flow graph representation of the LMS algorithm.

5.5 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. From Fig. 5.4 we see that it is the lower feedback loop that adds variability to the behavior of the LMS algorithm. In particular, there are two distinct quantities, namely, the learning-rate parameter η and the input vector $\mathbf{x}(n)$, that determine the transmittance of this feedback loop. We therefore deduce that the convergence behavior (i.e., stability) of the LMS algorithm is influenced by the statistical characteristics of the input vector $\mathbf{x}(n)$ and the value assigned to the learning-rate parameter η . Casting this observation in a different way, we may state that for a specified environment that supplies the input vector $\mathbf{x}(n)$, we have to exercise care in the selection of the learning-rate parameter η for the LMS algorithm to be convergent.

There are two distinct aspects of the convergence problem in the LMS algorithm that require special attention (Haykin, 1991):

1. The LMS algorithm is said to be *convergent in the mean* if the mean value of the weight vector $\hat{\mathbf{w}}(n)$ approaches the optimum solution \mathbf{w}_o as the number of iterations n approaches infinity; that is,

$$E[\hat{\mathbf{w}}(n)] \rightarrow \mathbf{w}_o \quad \text{as } n \rightarrow \infty \quad (5.29)$$

2. The LMS algorithm is said to be *convergent in the mean square* if the mean-square value of the error signal $e(n)$ approaches a constant value as the number of iterations n approaches infinity; that is,

$$E[e^2(n)] \rightarrow \text{constant} \quad \text{as } n \rightarrow \infty \quad (5.30)$$

It turns out that the condition for convergence in the mean square is tighter than the condition for convergence in the mean. In other words, if the LMS algorithm is convergent in the mean square, then it is convergent in the mean; however, the converse of this statement is not necessarily true. Both of these issues are considered in the remainder of this section.

Convergence in the Mean

We begin this convergence analysis of the LMS algorithm by making the following assumption: The weight vector \mathbf{w} computed by the LMS algorithm is uncorrelated with the input vector $\mathbf{x}(n)$, as shown by

$$E[\hat{\mathbf{w}}(n)\mathbf{x}(n)] = \mathbf{0} \quad (5.31)$$

Then, taking the expectation of both sides of Eq. (5.27) and then applying this assumption, we get

$$E[\hat{\mathbf{w}}(n+1)] = (\mathbf{I} - \eta\mathbf{R}_x)E[\hat{\mathbf{w}}(n)] + \eta\mathbf{r}_{dx} \quad (5.32)$$

where \mathbf{R}_x and \mathbf{r}_{dx} are defined, respectively, by

$$\mathbf{R}_x = E[\mathbf{x}(n)\mathbf{x}^T(n)] \quad (5.33)$$

and

$$\mathbf{r}_{dx} = E[\mathbf{x}(n)d(n)] \quad (5.34)$$

We refer to \mathbf{R}_x as the *correlation matrix* of the input vector $\mathbf{x}(n)$, and to \mathbf{r}_{dx} as the *cross-correlation vector* between the input vector $\mathbf{x}(n)$ and the desired response $d(n)$.

To find the condition for the convergence of the LMS algorithm in the mean, we perform an *orthogonal similarity transformation* on the correlation matrix \mathbf{R}_x , as shown by (Strang, 1980)

$$\mathbf{Q}^T \mathbf{R}_x \mathbf{Q} = \mathbf{A} \quad (5.35)$$

or, equivalently,

$$\mathbf{R}_x = \mathbf{Q} \mathbf{A} \mathbf{Q}^T \quad (5.36)$$

where \mathbf{A} is a *diagonal matrix* made up of the *eigenvalues* of the correlation matrix \mathbf{R}_x , and \mathbf{Q} is an *orthogonal matrix* whose columns are the associated *eigenvectors* of \mathbf{R}_x . An important property of an orthogonal matrix is that its inverse and transpose have the same value; that is,

$$\mathbf{Q}^{-1} = \mathbf{Q}^T \quad (5.37)$$

or, equivalently,

$$\mathbf{Q} \mathbf{Q}^T = \mathbf{I} \quad (5.38)$$

Rewriting the Wiener–Hopf equations (5.13) in matrix form, we have

$$\mathbf{R}_x \mathbf{w}_o = \mathbf{r}_{dx} \quad (5.39)$$

where \mathbf{R}_x and \mathbf{r}_{dx} are as defined previously, and \mathbf{w}_o is the weight vector of the optimum (Wiener) filter. Using Eq. (5.39) for the cross-correlation vector \mathbf{r}_{dx} in Eq. (5.32), then substituting the orthogonal similarity transformation of Eq. (5.36) for the correlation matrix \mathbf{R}_x , and then using the relation of Eq. (5.37), we get

$$\mathbf{Q}^T E[\hat{\mathbf{w}}(n+1)] = (\mathbf{I} - \eta \mathbf{A}) \mathbf{Q}^T E[\hat{\mathbf{w}}(n)] + \eta \mathbf{A} \mathbf{Q}^T \mathbf{w}_o \quad (5.40)$$

Let a new vector $\mathbf{v}(n)$ be defined as a *transformed version of the deviation between the expectation $E[\hat{\mathbf{w}}(n)]$ and the Wiener solution \mathbf{w}_o* , as shown by

$$\mathbf{v}(n) = \mathbf{Q}^T (E[\hat{\mathbf{w}}(n)] - \mathbf{w}_o) \quad (5.41)$$

Equivalently, we have the affine transformation

$$E[\mathbf{w}(n)] = \mathbf{Q} \mathbf{v}(n) + \mathbf{w}_o \quad (5.42)$$

We may then rewrite Eq. (5.40) in the simplified form

$$\mathbf{v}(n+1) = (\mathbf{I} - \eta \mathbf{A}) \mathbf{v}(n) \quad (5.43)$$

The recursive vector equation (5.43) represents a system of *uncoupled homogeneous first-order difference equations*, as shown by

$$v_k(n+1) = (1 - \eta \lambda_k) v_k(n), \quad k = 1, 2, \dots, p \quad (5.44)$$

where the λ_k are the eigenvalues of the correlation matrix \mathbf{R}_x and $v_k(n)$ is the k th element of the vector $\mathbf{v}(n)$. Let $v_k(0)$ denote the initial value of $v_k(n)$. We may then solve Eq. (5.44) for $v_k(n)$, obtaining

$$v_k(n) = (1 - \eta \lambda_k)^n v_k(0), \quad k = 1, 2, \dots, p \quad (5.45)$$

For the LMS algorithm to be convergent in the mean, we require that for an arbitrary choice of $v_k(0)$ the following condition be satisfied:

$$|1 - \eta \lambda_k| < 1 \quad \text{for } k = 1, 2, \dots, p \quad (5.46)$$

Under this condition, we find that

$$v_k(n) \rightarrow 0 \quad \text{as } n \rightarrow \infty \quad (5.47)$$

which, in the light of Eq. (5.41), is equivalent to the condition described in Eq. (5.29).

The condition of Eq. (5.44), in **turn**, requires that we choose the learning-rate parameter η as follows (Haykin, 1991; Widrow and Stearns, 1985):

$$0 < \eta < \frac{2}{\lambda_{\max}} \quad (5.48)$$

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

In summary, *the LMS algorithm is convergent in the mean* provided that the learning-rate parameter η is a positive constant with an upper bound defined by $2/\lambda_{\max}$. Note that η is measured in units equivalent to the inverse of variance (power).

Convergence in the Mean Square

A detailed analysis of convergence of the **LMS** algorithm in the mean square is much more complicated than convergence analysis of the algorithm in the mean. This analysis is also much more demanding in the assumptions made concerning the behavior of the weight vector $\mathbf{w}(n)$ computed by the **LMS** algorithm (Haykin, 1991). In this subsection we present a simplified result of the analysis.

The **LMS** algorithm is *convergent in the mean square* if the learning-rate parameter η satisfies the following condition (Haykin, 1991; Widrow and Stearns, 1985):

$$0 < \eta < \frac{2}{\text{tr}[\mathbf{R}_x]} \quad (5.49)$$

where $\text{tr}[\mathbf{R}_x]$ is the *trace* of the correlation matrix \mathbf{R}_x . From matrix algebra, we know that (Strang, 1980)

$$\text{tr}[\mathbf{R}_x] = \sum \lambda_k \geq \lambda_{\max} \quad (5.50)$$

It follows therefore that if the learning-rate parameter η satisfies the condition of Eq. (5.49), then it also satisfies the condition of Eq. (5.48). In other words, if the **LMS** algorithm is convergent in the mean square, then it is also convergent in the mean.

From matrix algebra we also know that the trace of a square matrix equals the sum of its diagonal elements. The k th diagonal element of the correlation matrix \mathbf{R}_x equals the mean-square value of the input signal $x(n)$. Hence, the trace of the correlation matrix \mathbf{R}_x equals the *total input power* measured over all the p sensory inputs of the spatial filter in Fig. 5.3. Accordingly, we may reformulate the condition for convergence of the **LMS** algorithm in the mean square given in Eq. (5.49) as follows:

$$0 < \eta < \frac{2}{\text{total input power}} \quad (5.51)$$

For the instrumentation of this condition, we simply have to ensure that the learning-rate parameter η is a positive constant whose value is less than twice the reciprocal of the total input power.

5.6 Learning Curve

A curve obtained by plotting the mean squared error $J(n)$ versus the number of iterations n is called the *ensemble-averaged learning curve*. Such a curve can be highly informative, as it vividly displays important characteristics of the learning process under study.

The learning curve of the **LMS** algorithm consists of the sum of exponentials only. The precise form of this sum depends on the correlation structure of the input vector $\mathbf{x}(n)$.

In any event, to obtain the learning curve, we average the squared error over an ensemble of identical LMS filters with their inputs represented by different realizations of the vector $\mathbf{x}(n)$, which are drawn individually from the same population. For a single realization of the LMS algorithm, the learning curve consists of noisy exponentials; the ensemble averaging has the effect of smoothing out the noise.

For an LMS algorithm convergent in the mean square, the final value $J(\infty)$ of the mean-squared error $J(n)$ is a positive constant, which represents the *steady-state condition* of the learning curve. In fact, $J(\infty)$ is always in excess of the minimum mean-squared error J_{\min} realized by the corresponding Wiener filter for a stationary environment. The difference between $J(\infty)$ and J_{\min} is called the *excess mean-squared error*:

$$J_{\text{ex}} = J(\infty) - J_{\min} \quad (5.52)$$

The ratio of J_{ex} to J_{\min} is called the *misadjustment*:

$$M = \frac{J_{\text{ex}}}{J_{\min}} \quad (5.53)$$

It is customary to express the misadjustment M as a percentage. Thus, for example, a misadjustment of 10 percent means that the LMS algorithm produces a mean-squared error (after completion of the learning process) that is 10 percent greater than the minimum mean squared error J_{\min} . Such a performance is ordinarily considered to be satisfactory.

Another important characteristic of the LMS algorithm is the *settling time*. However, there is no unique definition for the settling time. We may, for example, *approximate* the learning curve by a single exponential with *average time constant* τ_{av} , and so use τ_{av} as a rough measure of the settling time. The smaller the value of τ_{av} is, the faster will be the settling time.

To a good degree of approximation, the misadjustment M of the LMS algorithm is directly proportional to the learning-rate parameter η , whereas the average time constant τ_{av} is inversely proportional to the learning-rate parameter η (Widrow and Stearns, 1985; Haykin, 1991). We therefore have conflicting results in the sense that if the learning-rate parameter is reduced so as to reduce the misadjustment, then the settling time of the LMS algorithm is increased. Conversely, if the learning-rate parameter is increased so as to accelerate the learning process, then the misadjustment is increased. Careful attention has therefore to be given to the choice of the learning parameter η in the design of the LMS algorithm in order to produce a satisfactory overall performance.

5.7 Learning-Rate Annealing Schedules

The difficulties encountered with the LMS algorithm may be attributed to the fact that the learning-rate parameter is maintained constant throughout the computation, as shown by

$$\eta(n) = \eta_0 \quad \text{for all } n \quad (5.54)$$

This is the simplest possible form the learning-rate parameter can assume. In contrast, in *stochastic* approximation, which goes back to the classic paper by Robbins and Monro (1951), the learning-rate parameter is time-varying. The particular time-varying form most commonly used in the stochastic approximation literature is described by

$$\eta(n) = \frac{c}{n} \quad (5.55)$$

where c is a constant. Such a choice is indeed sufficient to guarantee convergence of the stochastic approximation algorithm (Kushner and Clark, 1978; Ljung, 1977). However, when the constant c is large, there is a danger of parameter blowup for small n .

Darken and Moody (1992) have proposed the use of a so-called *search-then-converge schedule*, defined by

$$\eta(n) = \frac{\eta_0}{1 + (n/\tau)} \quad (5.56)$$

where η_0 and τ are constants. In the early stages of adaptation involving a number of iterations n small compared to the *search time constant* τ , the learning-rate parameter $\eta(n)$ is approximately equal to η_0 and the algorithm operates essentially as the “standard” LMS algorithm, as indicated in Fig. 5.5. Hence, by choosing a high value for η_0 within the permissible range, it is hoped that the adjustable weights of the filter will find and hover about a “good” set of values. Then, for a number of iterations n large compared to the search time constant τ , the learning-rate parameter $\eta(n)$ approximates as c/n , where $c = \tau\eta_0$, as shown in Fig. 5.5. The algorithm now operates as a traditional stochastic approximation algorithm, and the weights converge to their optimum values. It thus appears that the search-then-converge scheme combines the desirable features of the standard LMS and traditional stochastic approximation algorithms. The specific learning-rate parameter $\eta(n)$ given in Eq. (5.56) is the simplest member of a class of search-and-converge schedules described in Darken and Moody (1992).

Sutton (1992b) has proposed even more elaborate schemes for adjusting the learning-rate parameter $\eta(n)$, which are based in part on connectionist learning methods, and which result in a significant improvement in performance. The new algorithms described by Sutton are of the same order of complexity as the standard LMS algorithm, $O(p)$, where p is the number of adjustable parameters.

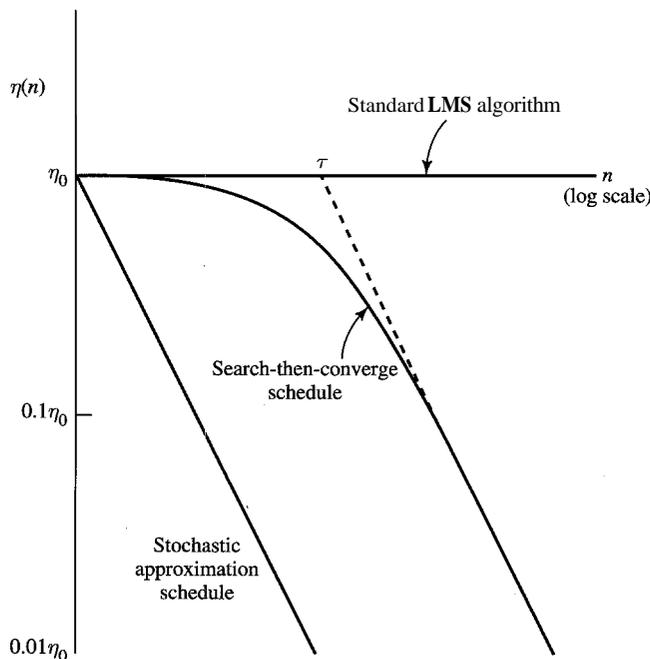


FIGURE 5.5 Learning-rate annealing schedules.

5.8 Adaline

The *Adaline* (adaptive linear element), originally conceived by Widrow and Hoff, is an adaptive pattern-classification machine that uses the LMS algorithm for its operation (Widrow and Lehr, 1990; Widrow and Hoff, 1960).

A block diagram of the Adaline is shown in Fig. 5.6. It consists of a linear combiner, a hard limiter, and a mechanism for adjusting the weights. The inputs x_1, x_2, \dots, x_p applied to the linear combiner are assigned the value -1 or $+1$. A variable threshold θ (whose value lies somewhere between 0 and $+1$) is applied to the hard limiter. The Adaline is also supplied with a desired output d whose value is $+1$ or -1 . The weights w_1, w_2, \dots, w_p and threshold θ are all adjusted in accordance with the LMS algorithm. In particular, the linear combiner output u , produced in response to the inputs x_1, x_2, \dots, x_p , is subtracted from the desired output d to produce the error signal e . The error signal e is, in turn, used to actuate the LMS algorithm in the manner described in Section 5.4.

The Adaline output y is obtained by passing the linear combiner output u through the hard limiter. We thus have

$$y = \begin{cases} +1 & \text{if } u \geq \theta \\ -1 & \text{if } u < \theta \end{cases} \quad (5.57)$$

Note, however, that the control action of the LMS algorithm depends on the error signal e measured as the difference between the desired output d and the linear combiner output u *before* quantization rather than the actual error e , between the desired output d and the actual Adaline output y .

The objective of the adaptive process in the Adaline may be stated as follows:

Given a set of input patterns and the associated desired outputs, find the optimum set of synaptic weights w_1, w_2, \dots, w_p , and threshold θ , to minimize the mean-square value of the actual error e .

Since the values assumed by the actual output of the Adaline and the desired output are $+1$ or -1 , it follows that the actual error e , may have only the values $+2, 0$, or -2 . Minimization of the mean-square value of e , is therefore equivalent to minimizing the average number of actual errors.

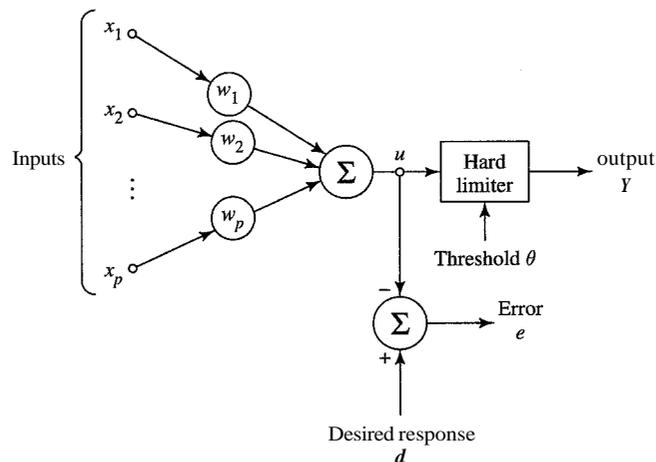


FIGURE 5.6 Block diagram of Adaline.

During the training phase of the Adaline, crude geometric patterns are fed to the machine. The machine then learns something from each pattern, and thereby experiences a change in its design. The total experience gained by the machine is stored in the values assumed by the weights w_1, w_2, \dots, w_p and the threshold θ . The Adaline may be trained on undistorted, noise-free patterns by repeating them over and over until the iterative search process for the optimum setting of the weights converges, or it can be trained on a sequence of noisy patterns on a one-pass basis such that the process converges in a statistical manner (Widrow and Hoff, 1960). Combinations of these methods may also be accommodated simultaneously. After training of the Adaline is completed, it may be used to classify the original patterns and noisy or distorted versions of them.

5.9 Discussion

The LMS algorithm has established itself as an important functional block of *adaptive signal processing*. It offers some highly desirable features:

- Simplicity of implementation, be that in software or hardware form.
- Ability to operate satisfactorily in an unknown environment.
- Ability to track time variations of input statistics.

Indeed, the simplicity of the LMS algorithm has made it the *standard* against which other linear adaptive filtering algorithms are benchmarked.

The formulation of the LMS algorithm presented in this chapter has been from the viewpoint of *spatial filtering*. It may equally well be applied to solve *temporal* filtering problems. In the latter case, the filter takes on the form of a *tapped-delay-line filter*, as depicted in the signal-flow graph of Fig. 5.7. The input vector $\mathbf{x}(n)$ is now defined by the tap inputs of the filter, as shown by

$$\mathbf{x}(n) = [x(n), x(n-1), \dots, x(n-p+1)]^T \quad (5.58)$$

where p is the number of taps. Thus, the tap-input $x(n-k)$ in the tapped-delay-line filter of Fig. 5.7 plays a role similar to that of the sensor input $x_k(n)$ in the spatial filter of Fig. 5.1. The function of the LMS algorithm is to adjust the tap weights of the filter in Fig. 5.7 so as to make the filter output $y(n)$ approximate a desired response $d(n)$ in a least-mean-square sense.

Regardless of the form of the LMS algorithm, its operation consists of two basic processes forming a feedback loop:

- **Adaptive process**, which involves the adjustments of the free parameters of a filter.
- **Filtering process**, which culminates in the computation of an error signal; the error signal is in turn used to actuate the adaptive process, thereby closing the feedback loop.

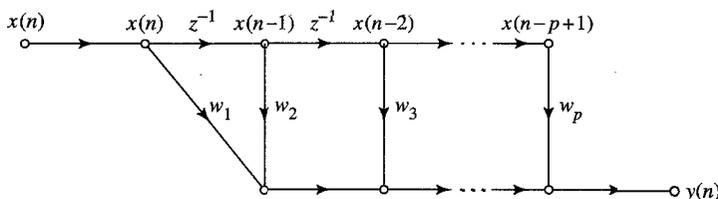


FIGURE 5.7 Signal-flow graph of tapped-delay-line filter.

For a detailed study of the **LMS** algorithm and other linear adaptive filtering algorithms, and their various applications, the reader is referred to Haykin (1991), and Widrow and Stearns (1985).

PROBLEMS

5.1 Consider the method of steepest descent involving a single weight $w(n)$. Do the following:

- Determine the mean-squared error $J(n)$ as a function of $w(n)$.
- Find the optimum solution $w_o(n)$ and the minimum mean-squared error J_{\min} .

5.2 The correlation matrix \mathbf{R}_x of the input vector $\mathbf{x}(n)$ in the LMS algorithm is defined by

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

Define the range of values for the learning-rate parameter η of the **LMS** algorithm for (a) the algorithm to be convergent in the mean, and (b) for it to be convergent in the mean square.

5.3 The *normalized LMS algorithm* is described by the following recursion for the weight vector:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \frac{\eta}{\|\mathbf{x}(n)\|^2} e(n)\mathbf{x}(n)$$

where η is a positive constant and $\|\mathbf{x}(n)\|$ is the Euclidean norm of the input vector $\mathbf{x}(n)$. The error signal $e(n)$ is defined by

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

where $d(n)$ is the desired response. For the normalized **LMS** algorithm to be convergent in the mean square, show that

$$0 < \eta < 2$$

5.4 The **LMS** algorithm is used to implement the dual-input, single-weight adaptive noise canceller shown in Fig. P5.4. Set up the equations that define the operation of this algorithm.

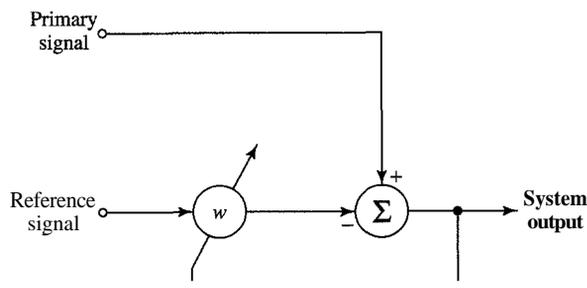


FIGURE P5.4

- 5.5 The signal-flow graph of Fig. P5.5 shows a linear predictor with its input vector made up of the samples $x(n-1), x(n-2), \dots, x(n-p)$, where p is the prediction order. The requirement is to use the LMS algorithm to make a prediction $\hat{x}(n)$ of the input sample $x(n)$. Set up the recursions that may be used to compute the tap weight w_1, w_2, \dots, w_p of the predictor.

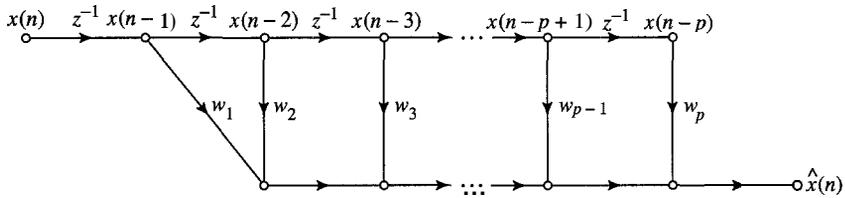


FIGURE P5.5

- 5.6 Compare the distinguishing features of the LMS algorithm and those of a single-layer perceptron involving the use of a single neuron.