Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control. James C. Spall

ISBN: 0-471-33052-3

CHAPTER 5

STOCHASTIC GRADIENT FORM OF STOCHASTIC APPROXIMATION

Chapter 4 introduced the root-finding (Robbins-Monro) stochastic approximation (SA) algorithm as a general method for nonlinear problems. The aim is to find one or more zeros of the function $\boldsymbol{g}(\boldsymbol{\theta})$ (i.e., roots of $\boldsymbol{g}(\boldsymbol{\theta}) = \mathbf{0}$) when only noisy measurements of the function $\boldsymbol{g}(\boldsymbol{\theta})$ are available. Now, let us return to the essential optimization problem of minimizing a loss function $L = L(\boldsymbol{\theta})$. The root-finding SA algorithm discussed in Chapter 4 plays a central role in optimization through the classical problem of finding a $\boldsymbol{\theta}$ such that $\boldsymbol{g}(\boldsymbol{\theta}) = \partial L/\partial \boldsymbol{\theta} = \mathbf{0}$. That is, noisy measurements of the gradient $\boldsymbol{g}(\boldsymbol{\theta})$ are used in the SA algorithm. For this reason, the SA algorithm in this application is often referred to as a *stochastic gradient* algorithm. Many practical problems have been solved via the stochastic gradient formulation, a few of which are reviewed in this chapter.

Section 5.1 discusses the fundamental principles in the stochastic gradient interpretation of SA. The subsequent three sections are brief discussions of applications of the stochastic gradient-based SA algorithm. Section 5.2 considers neural network training and the interpretation of the well-known backpropagation algorithm as an SA algorithm. Section 5.3 treats discrete-event dynamic systems (e.g., queuing networks), focusing on the "infinitesimal perturbation analysis" method of gradient estimation. Section 5.4 considers image restoration. The discussions of Sections 5.2–5.4 merely touch on the subjects, the aim being to provide a sense of how the stochastic gradient method is used in applications. Section 5.5 offers some concluding remarks.

5.1 ROOT-FINDING STOCHASTIC APPROXIMATION AS A STOCHASTIC GRADIENT METHOD

This relatively long section is in four subsections. These subsections introduce the stochastic gradient algorithm for both recursive and batch processing, and give a brief discussion of the relationship of the least-mean-squares (LMS) algorithm of Chapter 3 to the stochastic gradient method.

5.1.1 Basic Principles

With a differentiable loss function $L(\boldsymbol{\theta})$, recall the familiar set of p equations and p unknowns for use in finding a minimum $\boldsymbol{\theta}^*$:

$$g(\boldsymbol{\theta}) = \frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0}\,. \tag{5.1}$$

Consider the setting where only noisy information is available about the loss function and gradient. We may then express the loss function as

$$L(\boldsymbol{\theta}) = E[Q(\boldsymbol{\theta}, V)]\,, \tag{5.2}$$

where V represents the random effects in the process generating the system output and $Q(\boldsymbol{\theta}, V)$ represents some "observed" cost as a function of the chosen $\boldsymbol{\theta}$ and random effects V (V for *variability* is perhaps a useful mnemonic). The variable V may represent the amalgamation of many individual random effects. The expectation in (5.2) is with respect to all randomness embodied in V. So $L(\boldsymbol{\theta})$ represents an average cost over all possible values of V at the specified $\boldsymbol{\theta}$.

For the discussion of this subsection, Q will represent either an instantaneous (recursive) input or a batch input. In the instantaneous case, as discussed in Subsections 5.1.2 and 5.1.3, Q and V will generally be indexed by k, as they reflect an iteration (k)-varying cost function and associated random input. So, $Q(\boldsymbol{\theta}, V)$ is replaced by $Q_k(\boldsymbol{\theta}, V_k)$. In the batch case, we write $\bar{Q}(\boldsymbol{\theta}, V)$ to denote an average observed cost function, averaged over the number of measurements as in the least-squares criterion of Chapter 3. To avoid having to distinguish between the instantaneous and batch cases in the discussion of this subsection, we simply let $Q(\boldsymbol{\theta}, V)$ represent $Q_k(\boldsymbol{\theta}, V_k)$ or $\bar{Q}(\boldsymbol{\theta}, V)$ as appropriate. It should be understood, for example, that for recursive applications below requiring an unbiased gradient measurement, expression (5.2) is interpreted as $L(\boldsymbol{\theta}) = E[Q_k(\boldsymbol{\theta}, V_k)]$.

Because of the nonlinearity (and possible lack of knowledge about the probability distribution of V), it is almost never the case that L can be computed. However, $Q(\boldsymbol{\theta}, V)$ is typically available since that just represents the outcome for a particular experiment (no expectation involved). Further, it is assumed that $Q(\boldsymbol{\theta}, V)$ is a differentiable function (in $\boldsymbol{\theta}$) for *almost all* V (i.e., for all values of V except possibly a set of values having probability zero; see Appendix C) and that we have knowledge of $\partial Q/\partial \boldsymbol{\theta}$ for any reasonable $\boldsymbol{\theta}$. The expression $\partial Q/\partial \boldsymbol{\theta}$ is called a *stochastic gradient* because it depends on the random term V.

The ability to compute the derivative $\partial Q/\partial \boldsymbol{\theta}$ is, of course, fundamental to the stochastic gradient approach. In many practical problems, it is not possible to compute $\partial Q/\partial \boldsymbol{\theta}$, motivating the gradient-free SA approaches of Chapters 6

and 7. Nonetheless, there is a large set of problems for which it *is* possible to compute the gradient. This set is the focus of this chapter.

Unlike the linear case in Chapter 3 (where the loss function is quadratic in $\boldsymbol{\theta}$), a solution to (5.1) is not necessarily a global minimum to (5.2). Nevertheless, this root-finding approach is adopted in the stochastic gradient SA framework since there is usually a guarantee of at least a local minimum. Obviously, this root-finding approach differs from direct search techniques for optimization, such as those described in Chapter 2. In those, one solves the optimization problem directly rather than through the proxy problem of root finding. Significant advantages of the SA-based root-finding approach over the earlier direct methods are the greater efficiency and the strong theory for guaranteeing performance with noisy measurements of the root-finding function.

While this chapter focuses on basic forms of the stochastic gradient algorithm, Section 8.4 discusses a modification to the basic algorithm to provide for *global* convergence. This modification involves the addition of a Monte Carlo-generated noise term to the right-hand side of the fundamental SA recursions (4.6) or (4.7) when implemented in a stochastic gradient context. This additional noise gives the algorithm enough "jumpiness" to ensure that it will not get prematurely trapped in local minima. The user controls the degree of Monte Carlo noise using principles of annealing (analogous to simulated annealing).

If we assume for the moment that $\boldsymbol{V}$ has a probability density function, then the most general representation of the expectation in (5.2) is

$$E[Q(\boldsymbol{\theta},\boldsymbol{V})] = \int_{\Lambda} Q(\boldsymbol{\theta},\boldsymbol{\upsilon}) p_{\boldsymbol{V}}(\boldsymbol{\upsilon}\,|\,\boldsymbol{\theta})\, d\boldsymbol{\upsilon}, \tag{5.3}$$

where $p_{\boldsymbol{V}}(\boldsymbol{\upsilon}|\boldsymbol{\theta})$ is the density function for $\boldsymbol{V}$ given $\boldsymbol{\theta}$ and the integral is over the domain Λ for $\boldsymbol{V}$ ($\boldsymbol{\upsilon}$ is the dummy variable of integration). In the stochastic gradient formulation, however, the usual form is a special case of that shown in (5.3). In particular, $\boldsymbol{V}$ is usually viewed as some "fundamental" underlying random process that is generated *independently* of $\boldsymbol{\theta}$. Then, (5.3) is replaced by

$$E[Q(\boldsymbol{\theta},\boldsymbol{V})] = \int_{\Lambda} Q(\boldsymbol{\theta},\boldsymbol{\upsilon}) p_{\boldsymbol{V}}(\boldsymbol{\upsilon})\, d\boldsymbol{\upsilon}, \tag{5.4}$$

where $p_{\boldsymbol{V}}(\boldsymbol{\upsilon})$ is the density function for $\boldsymbol{V}$. Although the formulations in (5.3) and (5.4) and the discussion to follow are given in terms of density functions for $\boldsymbol{V}$, precisely the same line of reasoning applies to distributions for which density functions do not exist. In particular, if $\boldsymbol{V}$ involves discrete (or hybrid discrete/continuous) random variables, the arguments apply with probability mass functions replacing probability density functions and sums replacing integrals.

The distinction between (5.3) and (5.4)—with their density functions dependent and independent of $\boldsymbol{\theta}$—is fundamental in the stochastic gradient implementation. It is shown below that (5.4) is the preferred form for the

stochastic gradient algorithm. Let us present an example showing how a simple transformation can *sometimes* convert the general setting of (5.3) to the simpler (5.4).

Example 5.1—Conversion to preferred stochastic gradient form. Suppose that $Q(\theta, V) = f(\theta, Z) + W$, where the scalar $W = W(\theta) \sim N(0, \theta^2\sigma^2)$ (with θ and σ both strictly positive), Z is some scalar random variable independent of θ, and $f(\cdot)$ is some function. Then V represents the combined random effects $\{Z, W(\theta)\}$, which we assume to have a joint probability density function. Clearly, with this definition, the density function for V must be represented by $p_V(\upsilon|\theta)$, not $p_V(\upsilon)$, since the distribution of W depends on θ. Suppose, on the other hand, that $Q(\theta, V)$ is expressed in the equivalent form $Q(\theta, V) = f(\theta, Z) + \theta W'$, where $W' \sim N(0, \sigma^2)$. Then, V represents the combined random effects $\{Z, W'\}$, where V has a density *independent* of θ. Hence, the problem is now in the realm of (5.4), as desired. ❑

Our interest is in unbiased measurements of the gradient $\boldsymbol{g}(\boldsymbol{\theta})$ at any $\boldsymbol{\theta}$. That is, $E[\boldsymbol{Y}(\boldsymbol{\theta})] = \boldsymbol{g}(\boldsymbol{\theta})$. Hence, the mean-zero noise conditions for convergence of root-finding SA discussed in Section 4.3 hold (condition A.3 or the special case of condition B.5 where the bias $\boldsymbol{b}_k = \boldsymbol{0}$). Suppose for the moment that it is valid to interchange the derivative $\partial(\cdot)/\partial\boldsymbol{\theta}$ and integrals in (5.3) or (5.4). We present formal conditions for the interchange in Theorem 5.1 below. Then, in case (5.3) where the density function depends on $\boldsymbol{\theta}$,

$$\begin{aligned}\boldsymbol{g}(\boldsymbol{\theta}) &= \frac{\partial}{\partial\boldsymbol{\theta}}\int_\Lambda Q(\boldsymbol{\theta},\upsilon)p_V(\upsilon\,|\,\boldsymbol{\theta})\,d\upsilon \\ &= \int_\Lambda\left[Q(\boldsymbol{\theta},\upsilon)\frac{\partial p_V(\upsilon\,|\,\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} + \frac{\partial Q(\boldsymbol{\theta},\upsilon)}{\partial\boldsymbol{\theta}}p_V(\upsilon|\boldsymbol{\theta})\right]d\upsilon. \qquad (5.5)\end{aligned}$$

When (5.4) applies, the following simpler form results:

$$\boldsymbol{g}(\boldsymbol{\theta}) = \frac{\partial}{\partial\boldsymbol{\theta}}\int_\Lambda Q(\boldsymbol{\theta},\upsilon)p_V(\upsilon)\,d\upsilon = \int_\Lambda \frac{\partial Q(\boldsymbol{\theta},\upsilon)}{\partial\boldsymbol{\theta}}p_V(\upsilon)\,d\upsilon. \qquad (5.6)$$

The expression appearing on the right-hand side of (5.6) forms the basis for the stochastic gradient algorithm. Let us now use Theorem A.3 in Appendix A to formally state conditions under which the derivative–integral interchange operation in (5.6) is valid.

Theorem 5.1 (Validity of interchange of derivative and integral in (5.6)). Let Λ be the domain for V. Suppose that Θ is an open set. Let $Q(\boldsymbol{\theta}, \upsilon)p_V(\upsilon)$ and $p_V(\upsilon)\partial Q(\boldsymbol{\theta},\upsilon)/\partial\boldsymbol{\theta}$ be continuous on $\Theta \times \Lambda$. Suppose that there exist nonnegative functions $q_0(\upsilon)$ and $q_1(\upsilon)$ such that

$$\left|Q(\boldsymbol{\theta},\boldsymbol{\upsilon})p_V(\boldsymbol{\upsilon})\right| \le q_0(\boldsymbol{\upsilon}),\ \left\|p_V(\boldsymbol{\upsilon})\frac{\partial Q(\boldsymbol{\theta},\boldsymbol{\upsilon})}{\partial\boldsymbol{\theta}}\right\| \le q_1(\boldsymbol{\upsilon}) \text{ for all } (\boldsymbol{\theta},\boldsymbol{\upsilon}) \in \Theta\times\Lambda,$$

where $\int_\Lambda q_0(\boldsymbol{\upsilon})\,d\boldsymbol{\upsilon} < \infty$ and $\int_\Lambda q_1(\boldsymbol{\upsilon})\,d\boldsymbol{\upsilon} < \infty$. Then

$$\frac{\partial}{\partial\boldsymbol{\theta}}\int_\Lambda Q(\boldsymbol{\theta},\boldsymbol{\upsilon})p_V(\boldsymbol{\upsilon})\,d\boldsymbol{\upsilon} = \int_\Lambda \frac{\partial Q(\boldsymbol{\theta},\boldsymbol{\upsilon})}{\partial\boldsymbol{\theta}}p_V(\boldsymbol{\upsilon})\,d\boldsymbol{\upsilon}.$$

That is, (5.6) holds.

If (5.6) and the above interchange hold, a realization $\partial Q/\partial\boldsymbol{\theta}$ yields an unbiased estimate of the true gradient, à la

$$E\left[\frac{\partial Q(\boldsymbol{\theta},\boldsymbol{V})}{\partial\boldsymbol{\theta}}\right] = \boldsymbol{g}(\boldsymbol{\theta}). \tag{5.7}$$

Expression (5.5), on the other hand, does not yield such an "automatic" means of getting an unbiased gradient approximation. Unlike the right-hand side of (5.6), the integrand in (5.5) (specifically, the part before the "+" in the second line) is not in the form (*function*) × (*density*). Hence, there is no natural way of generating a random outcome that is an unbiased estimator of the gradient.

Chapter 15 discusses a "trick" that gets around the problem of an integrand not being in the form (*function*) × (*density*). In particular, one can rewrite the integrand by multiplying through by unity expressed as a ratio of identical density functions $p_V(\boldsymbol{\upsilon}|\boldsymbol{\theta}')/p_V(\boldsymbol{\upsilon}|\boldsymbol{\theta}')$, where $\boldsymbol{\theta}'$ is a fixed value of $\boldsymbol{\theta}$. The new (equivalent) integrand can be expressed in the form (*function*) × (*density*), where the density is $p_V(\boldsymbol{\upsilon}|\boldsymbol{\theta}')$ (see Chapter 15 on the likelihood ratio and sample path methods for gradient estimation in simulation-based optimization). However, in contrast to simulation-based optimization, this trick does not work in *general* SA applications since it requires that "nature" generate $\boldsymbol{V}$ from a density with the known value $\boldsymbol{\theta}'$. In general SA applications, the analyst has no control over how the underlying randomness in the problem is produced. (The "trick" *does* work in the simulation-based optimization setting of Chapter 15 because the analyst controls "nature" by controlling the Monte Carlo randomness produced in the simulation.) A further difficulty with (5.5) relative to general stochastic gradient problems is that the analyst needs complete knowledge of $p_V(\boldsymbol{\upsilon}|\boldsymbol{\theta})$ and $\partial p_V(\boldsymbol{\upsilon}|\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ for varying $\boldsymbol{\upsilon}$ and $\boldsymbol{\theta}$.

In contrast to the problems in using (5.5) for gradient estimation, (5.4) as it manifests itself in (5.6) yields the fundamental gradient estimate:

$$\boldsymbol{Y}(\boldsymbol{\theta}) = \frac{\partial Q(\boldsymbol{\theta},\boldsymbol{V})}{\partial\boldsymbol{\theta}}. \tag{5.8}$$

SA with input (5.8) is the *stochastic gradient algorithm* of interest here (Chapter 15 considers the more general form using (5.5)). From (5.6), $\boldsymbol{Y}(\boldsymbol{\theta})$ is an unbiased estimate of $\boldsymbol{g}(\boldsymbol{\theta})$ (i.e., A.3 and B.5 in Section 4.3 hold). This input is available without knowing the distribution of $\boldsymbol{V}$ because $\boldsymbol{Y}$ is an *observed* outcome.

Algebraically, one can always write the manifestation of $\boldsymbol{V}$ as an error in the L measurement, where the error generally depends on $\boldsymbol{\theta}$. So, $Q(\boldsymbol{\theta}, \boldsymbol{V}) = L(\boldsymbol{\theta}) + \varepsilon(\boldsymbol{\theta})$, although the analytical form of $\varepsilon(\boldsymbol{\theta})$ is usually unknown. Note that $y(\boldsymbol{\theta}) = Q(\boldsymbol{\theta}, \boldsymbol{V})$ in the notation introduced in Chapter 1. In some practical problems, $\boldsymbol{V}$ manifests itself as an additive *independent* random noise term. Then, *assuming that the gradient in (5.8) is available* is equivalent to assuming that the deterministic gradient $\boldsymbol{g}(\boldsymbol{\theta})$ is available. The examples below illustrate the distinction between an error dependent on $\boldsymbol{\theta}$ and not dependent on $\boldsymbol{\theta}$.

Example 5.2—Case where the stochastic gradient method is appropriate. Suppose that $Q(\boldsymbol{\theta}, \boldsymbol{V}) = f(\boldsymbol{\theta}) + \boldsymbol{\theta}^T\boldsymbol{V}$, where $f(\boldsymbol{\theta})$ is a differentiable function and $\boldsymbol{V} \in \mathbb{R}^p$ has an unknown distribution independent of $\boldsymbol{\theta}$. The minimum of $L(\boldsymbol{\theta}) = E[Q(\boldsymbol{\theta}, \boldsymbol{V})] = f(\boldsymbol{\theta}) + \boldsymbol{\theta}^T E(\boldsymbol{V})$ is clearly dependent on the (unknown) mean of $\boldsymbol{V}$ (and $\varepsilon(\boldsymbol{\theta}) = Q(\boldsymbol{\theta}, \boldsymbol{V}) - L(\boldsymbol{\theta}) = \boldsymbol{\theta}^T[\boldsymbol{V} - E(\boldsymbol{V})]$). Given that one knows the form for $Q(\boldsymbol{\theta}, \boldsymbol{V})$, the stochastic gradient method would be appropriate in this problem. The random input according to (5.8) is $\boldsymbol{Y}(\boldsymbol{\theta}) = \partial f(\boldsymbol{\theta})/\partial\boldsymbol{\theta} + \boldsymbol{V}$. ❑

Example 5.3—Stochastic problem solvable by deterministic methods. Now, suppose that the form for Q changes from that in Example 5.2 to $Q(\boldsymbol{\theta}, \boldsymbol{V}) = f(\boldsymbol{\theta}) + \boldsymbol{V}$, where the distribution of the scalar $\boldsymbol{V}$ does not depend on $\boldsymbol{\theta}$. If the analyst has enough information to invoke (5.8), then $\boldsymbol{Y}(\boldsymbol{\theta}) = \partial f(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$. But this is clearly independent of the random effect $\boldsymbol{V}$. Hence, standard deterministic methods can be used on $f(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) - E(\boldsymbol{V})$ (i.e., minimizing $f(\boldsymbol{\theta})$ is the same as minimizing $L(\boldsymbol{\theta})$ since they only differ by the constant corresponding to the mean of $\boldsymbol{V}$). The key qualifier here is the above statement: "...assuming that the gradient in (5.8) is available...." If this qualifier is not true, then one is generally unable to use deterministic methods to solve the problem because of the *random* form of $Q(\boldsymbol{\theta}, \boldsymbol{V})$. In that case, the SA methods for optimization with noisy loss measurements in Chapters 6 and 7 are more appropriate. ❑

5.1.2 Stochastic Gradient Algorithm

Let us now combine the basic SA algorithm in (4.6) or (4.7) with the definition of $\boldsymbol{Y}(\boldsymbol{\theta})$ in (5.8) to obtain a stochastic gradient algorithm. To reflect the common recursive situation where a Q may result from random effects together with a deterministic input at each iteration (as in the instantaneous gradient of Section 3.2 or the external input setting of Section 4.1), we index Q and $\boldsymbol{V}$ by the iteration counter k. Hence, $Q_k(\boldsymbol{\theta}, \boldsymbol{V}_k)$ may represent the instantaneous observation of an overall loss $L(\boldsymbol{\theta})$ or an observation of an instantaneous loss $L_k(\boldsymbol{\theta})$, the latter

case usually associated with time-varying tracking problems. A related implementation is associated with *batch processing*, where the recursive input $Q_k(\boldsymbol{\theta}, V_k)$ is replaced by the overall (sample mean) response $\bar{Q}(\boldsymbol{\theta},V)$ at any $\boldsymbol{\theta}$. In batch processing, the algorithm repeatedly passes through a given fixed set of data (so V represents the randomness in the given set of data). A special case of $\bar{Q}(\boldsymbol{\theta},V)$ is the least-squares criterion for linear systems, $\hat{L}(\boldsymbol{\theta})$, used in Chapter 3.

For the general recursive unconstrained case (SA algorithm (4.6)), we have

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{k+1} &= \hat{\boldsymbol{\theta}}_k - a_k \left.\frac{\partial Q_k(\boldsymbol{\theta},V_k)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k} \\ &= \hat{\boldsymbol{\theta}}_k - a_k \boldsymbol{Y}_k(\hat{\boldsymbol{\theta}}_k), \end{aligned} \tag{5.9}$$

where V_k represents the random input at the kth iteration. Eqn. (5.9) is reflective of recursive processing given that Q_k and V_k may change with k.

If the SA recursion is being used in the above-mentioned batch processing mode, V is the overall set of random inputs, leading to an iterative process of the form

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{k+1} &= \hat{\boldsymbol{\theta}}_k - a_k \left.\frac{\partial \bar{Q}(\boldsymbol{\theta},V)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k} \\ &= \hat{\boldsymbol{\theta}}_k - a_k \boldsymbol{Y}(\hat{\boldsymbol{\theta}}_k), \end{aligned} \tag{5.10}$$

where $\boldsymbol{Y}$ is not indexed by k in the second line to reflect the reprocessing of the same gradient $\partial\bar{Q}/\partial\boldsymbol{\theta}$ with only $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}_k$ changing.

Because the stochastic gradient quantity $\partial Q_k/\partial\boldsymbol{\theta}$ in (5.9) does not, in general, equal the deterministic quantity $\partial L/\partial\boldsymbol{\theta}$, the SA algorithm is fundamentally different from the deterministic steepest descent algorithm in Section 1.4. There is, however, an intuitive connection between (5.9) and steepest descent because $E(\partial Q_k/\partial\boldsymbol{\theta}) = \partial L/\partial\boldsymbol{\theta}$ (or $\partial L_k/\partial\boldsymbol{\theta}$ in the time-varying loss case) under the regularity conditions mentioned above justifying the interchange of a derivative and an (expectation) integral. (In fact, there is some theory that allows for biased $E(\partial Q_k/\partial\boldsymbol{\theta}) \neq \partial L/\partial\boldsymbol{\theta}$ cases. See, for example, condition B.5 of Section 4.3, where the bias $\boldsymbol{b}_k$ can be nonzero but where $\boldsymbol{b}_k \to 0$ as $k \to \infty$. This theory is most useful in the setting of Chapters 6 and 7, where gradients are approximated from loss function measurements.) Figure 5.1 depicts a simple scalar case, suggesting why the recursion in (5.9) yields convergence to $\boldsymbol{\theta}^*$. Note that the a_k sequence is critical to controlling the magnitude of the steps to ensure an algorithm that is neither too timid nor too aggressive.

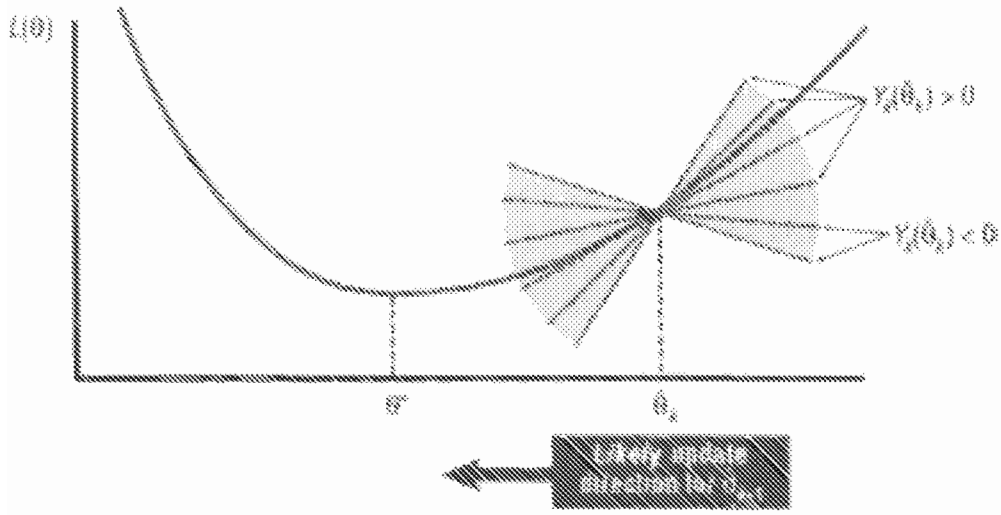

Figure 5.1. Intuitive basis for stochastic gradient search when $p = 1$. For $\hat{\theta}_k > \theta^*$, $Y_k(\hat{\theta}_k) = \partial Q_k / \partial \theta$ at $\theta = \hat{\theta}_k$ *tends* to be positive, leading to updated $\hat{\theta}_{k+1} < \hat{\theta}_k$, as desired. The bow-tie-shaped shaded area is the region likely to contain the line with measured slope $Y_k(\hat{\theta}_k)$. Four of the six measured derivatives shown have the correct (positive) sign. When $\hat{\theta}_k < \theta^*$, the complementary relationship holds, leading to $\hat{\theta}_{k+1} > \hat{\theta}_k$.

There exists formal convergence theory for $\hat{\boldsymbol{\theta}}_k$ in both the recursive and batch settings. The theory for the recursive setting is a special case of the conditions for general root-finding SA in Section 4.3 (a.s. convergence) and Section 4.4 (asymptotic normality). In particular, convergence relies on properly chosen decaying gains a_k. The theory for the batch case is tied to deterministic optimization (see Section 1.4). Batch convergence may *sometimes* occur with fixed gains $a_k = a$ for all k (e.g., in linear models; see Wang et al., 2000). As discussed below, convergence in the batch case is not generally to $\boldsymbol{\theta}^*$, but to a minimum tied to the specific set of data.

Note the connection of the general recursive algorithms in Chapter 3, particularly LMS (Subsection 3.2.2) and recursive least squares (RLS) (Subsection 3.2.4), to the SA form in (5.9). The instantaneous gradient in Section 3.2 corresponds to $\partial Q_k / \partial \boldsymbol{\theta}$ in (5.9). This gradient, of course, is tied to the special linear structure of the model in Chapter 3. We discuss the connection of LMS to the stochastic gradient recursion in more detail in Subsection 5.1.4.

RLS and the stochastic gradient algorithm of (5.9) also have the same generic form except that the specially chosen matrix $\boldsymbol{P}_{k+1}$ replaces the gain coefficient a_k in (5.9). General SA theory allows for choosing a matrix gain for a_k, but in most nonlinear problems (unlike RLS) there is usually not enough information to know how to choose such a matrix. Hence, analysts *typically* use the simpler scalar gain or use an adaptive scheme that seeks to estimate the matrix gain as the algorithm iterates (see Subsection 4.5.2).

While the recursive and batch modes represent, in some sense, the two extremes in implementation of the stochastic gradient algorithm, another hybrid form is popular. We refer to this hybrid form as a *multiple-pass implementation.* This form operates on the data using the instantaneous gradient, as in the recursive form, yet (like batch processing) makes multiple passes through the full set of data. In particular, after the algorithm has passed through the data one time, it returns to the first data point with the initial $\boldsymbol{\theta}$ equal to the terminal estimate from the first pass (the first pass may be carried out concurrent with the arrival of the data). The user may choose to reset the gain value at a_0 as in starting the first pass or may continue with the current gain sequence, processing the first data point in the second pass with gain a_n (producing $\hat{\boldsymbol{\theta}}_{n+1}$) if the terminal estimate for $\boldsymbol{\theta}$ from the first pass was $\hat{\boldsymbol{\theta}}_n$. This process is repeated for as many passes through the data as desired.

The multiple-pass method is particularly popular in neural network estimation, where each pass through the data is referred to as an *epoch.* Exercise 5.8 considers the multiple-pass method.

5.1.3 Implementation in General Nonlinear Regression Problems

An important example of the nonlinear setting of this chapter is nonlinear regression with scalar output. Suppose that the kth output of some process is modeled by

$$z_k = h_k(\boldsymbol{\theta}, \boldsymbol{x}_k) + v_k, \tag{5.11}$$

where $h_k(\cdot)$ is some nonlinear regression function dependent on the parameters being estimated and some input variables $\boldsymbol{x}_k$, and v_k is a mean-zero noise term. In a neural network problem, for example, $h_k(\boldsymbol{\theta}, \boldsymbol{x}_k)$ represents the network output for the given weight values $\boldsymbol{\theta}$ and particular input $\boldsymbol{x}_k$ (see Section 5.2).

Within the context of the nonlinear regression problem associated with (5.11), the algorithm can be implemented in a recursive mode as one collects outputs z_k for each $\boldsymbol{x}_k$ input or in a batch mode where all data have been collected and we work with an overall noisy loss measurement.

In the recursive form based on instantaneous input, (5.9) is implemented with input

$$\begin{aligned} \boldsymbol{Y}_k(\hat{\boldsymbol{\theta}}_k) &= \left.\frac{\partial Q_k(\boldsymbol{\theta}, V_k)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k} = \frac{1}{2}\left.\frac{\partial (z_{k+1} - \hat{z}_{k+1})^2}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k} \\ &= \left[(h_{k+1}(\boldsymbol{\theta}, \boldsymbol{x}_{k+1}) - z_{k+1}) \frac{\partial h_{k+1}}{\partial \boldsymbol{\theta}} \right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k}, \end{aligned} \tag{5.12}$$

where $\hat{z}_k = h_k(\boldsymbol{\theta}, \boldsymbol{x}_k)$ is the prediction at each iteration for the given $\boldsymbol{\theta}$ and $\boldsymbol{x}_k$. The above implies that $Q_k(\boldsymbol{\theta}, V_k) = \frac{1}{2}(z_{k+1} - \hat{z}_{k+1})^2$. This form clearly exhibits

the role of the gradient $\partial h_k/\partial\boldsymbol{\theta}$ in the gradient-based algorithms of interest here. Note that (5.12) is a noisy observation of the corresponding instantaneous gradient $\frac{1}{2}\partial E[(z_{k+1}-\hat{z}_{k+1})^2]/\partial\boldsymbol{\theta}$ at $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}_k$. In a neural network context, for example, a backpropagation type of algorithm would typically be used to calculate $\partial h_{k+1}/\partial\boldsymbol{\theta}$ at the particular $\boldsymbol{x}_{k+1}$ and $\hat{\boldsymbol{\theta}}_k$, as appears in (5.12) (see Section 5.2). If the problem were such that a gradient of this type was unavailable, then we could not use the SA algorithm in (5.9) (but see Chapters 6 and 7).

The randomness V_k is reflected in the real data z_{k+1}, although it does not show explicitly in the bottom line of (5.12). For example, suppose that the process generating the data z_k has the same form as the assumed model (i.e., the actual measurements come from $z_k = h_k(\boldsymbol{\theta}^*, \boldsymbol{x}_k) + v_k$). If the $\boldsymbol{x}_k$ are chosen randomly—as in the LMS discussion of Subsection 3.2.2 and the discussion of Subsection 4.5.1, where the inputs are treated as Markov state variables—then $V_k \equiv \{\boldsymbol{x}_{k+1}, v_{k+1}\}$. Alternatively, if the inputs are chosen deterministically, $V_k = v_{k+1}$. In more general cases, where the true process and assumed model have different forms, V_k represents all randomness in the true process.

In the batch mode, we iterate on $\boldsymbol{\theta}$ for a *fixed* set of data. The batch mode recursion uses (5.10) with input based on n data points:

$$\boldsymbol{Y}(\hat{\boldsymbol{\theta}}_k) = \frac{\partial \bar{Q}}{\partial\boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k} = \frac{1}{2n}\sum_{i=1}^{n}\frac{\partial(z_i-\hat{z}_i)^2}{\partial\boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k}$$

$$= \frac{1}{n}\sum_{i=1}^{n}[h_i(\hat{\boldsymbol{\theta}}_k, \boldsymbol{x}_i) - z_i]\frac{\partial h_i(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial\boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k}. \tag{5.13}$$

Because n is fixed, $\hat{\boldsymbol{\theta}}_k$ will not generally converge to $\boldsymbol{\theta}^*$, but will converge to some minimum that is tied to the fixed n (and associated data z_1, $z_2,\ldots, z_n$). In the linear case, this minimum corresponds to the batch solution $\hat{\boldsymbol{\theta}}^{(n)}$ (see Subsection 3.1.2). More generally, for linear and nonlinear problems, this minimum based on n data is generally close to the true minimum $\boldsymbol{\theta}^*$ if n is large. To implement (5.13), it is necessary to have the full set of data z_1, $z_2,\ldots, z_n$ available at the start of the search process.

5.1.4 Connection of LMS to Stochastic Gradient SA

This subsection shows how the recursive framework associated with input (5.12) applies in the special case of estimating $\boldsymbol{\theta}$ in linear models with LMS as in Chapter 3. Let us begin by considering the following example.

Example 5.4—LMS as an example of the stochastic gradient method. Suppose that $z_k = \boldsymbol{h}_k^T\boldsymbol{\theta} + v_k$, as introduced in Section 3.1, where, in general, $\boldsymbol{h}_k$

depends on some inputs $\boldsymbol{x}_k$ (i.e., $\boldsymbol{h}_k = \boldsymbol{h}_k(\boldsymbol{x}_k)$). In the context of (5.11), $h_k(\boldsymbol{\theta}, \boldsymbol{x}_k)$ is equivalent to $\boldsymbol{h}_k^T\boldsymbol{\theta} = \boldsymbol{h}_k(\boldsymbol{x}_k)^T\boldsymbol{\theta}$. This represents a model linear in $\boldsymbol{\theta}$, which is a special case of the nonlinear models of interest here. Subsection 3.2.2 introduced the basic LMS recursion:

$$\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k - a_k\boldsymbol{h}_{k+1}(\boldsymbol{h}_{k+1}^T\hat{\boldsymbol{\theta}}_k - z_{k+1}), \tag{5.14}$$

with $a_k = a > 0$ for all k. Because $\partial\boldsymbol{h}_{k+1}^T\boldsymbol{\theta}/\partial\boldsymbol{\theta} = \boldsymbol{h}_{k+1}$ (representing $\partial h_{k+1}(\boldsymbol{\theta}, \boldsymbol{x}_{k+1})/\partial\boldsymbol{\theta}$ in (5.12)), the LMS algorithm above corresponds to the recursive algorithm (5.9) with

$$\boldsymbol{Y}_k(\hat{\boldsymbol{\theta}}_k) = \boldsymbol{h}_{k+1}(\boldsymbol{h}_{k+1}^T\hat{\boldsymbol{\theta}}_k - z_{k+1}).$$

Similar arguments can be used for RLS in Subsection 3.2.4 provided that one allows for a matrix gain in place of the scalar a_k. ❑

Let us now continue with LMS as an example of the application of the general SA convergence conditions in Section 4.3. Recall that the two sets of conditions (the "statistics" conditions "A" and the "engineering conditions" "B") are sufficient to guarantee strong (a.s.) convergence of the root-finding SA algorithm to a solution $\boldsymbol{\theta}^*$. The engineering conditions, in particular, are especially powerful for analyzing the LMS algorithm (e.g., Ljung, 1984; Gerencsér, 1995). Proposition 5.1 below gives one set of convergence conditions for LMS, derived from the engineering conditions of Section 4.3. While this illustrates the connection between SA and LMS, the conditions are not the most general possible (see Ljung, 1984, Gerencsér, 1995, or Sections 3.2 and 3.3 for alternative references). The proposition statement uses the Kronecker product ($\otimes$) discussed in Appendix A.

Proposition 5.1 (Convergence of LMS). Consider the LMS recursion in (5.14). Suppose that the real system produces data according to $z_k = \boldsymbol{h}_k^T\boldsymbol{\theta}^* + v_k$, where the v_k are independent, identically distributed (i.i.d.) with mean zero and finite variance and the $\boldsymbol{h}_k$ are random and i.i.d. (independent of the v_k) such that all components of $E(\boldsymbol{h}_k \otimes \boldsymbol{h}_k \otimes \boldsymbol{h}_k \otimes \boldsymbol{h}_k)$ are finite in magnitude (i.e., all possible mixed and unmixed fourth moments of $\boldsymbol{h}_k$ are finite). Suppose that conditions B.1–B.3 (Section 4.3) hold, $\sum_{k=0}^{\infty} a_k^2 < \infty$, and $E\left(\left\|\hat{\boldsymbol{\theta}}_k\right\|^2\right)$ is uniformly bounded above for all k. Then, $\hat{\boldsymbol{\theta}}_k \to \boldsymbol{\theta}^*$ a.s. as $k \to \infty$.

Proof. First, note that $\boldsymbol{\theta}^*$ is the minimum of $L(\boldsymbol{\theta}) \equiv \frac{1}{2}E[(z_k - \boldsymbol{h}_k^T\boldsymbol{\theta})^2]$, where $L(\boldsymbol{\theta})$ does not depend on k because of the assumptions about v_k and $\boldsymbol{h}_k$. From Section 4.3, it is sufficient to have conditions B.1 – B.5 hold. Because B.1–B.3

are assumed to hold, we need only to show that B.4 and B.5 are implied by the other assumptions or the problem structure.

Let us first show that B.5 holds. From Example 5.4, $\boldsymbol{Y}_k(\boldsymbol{\theta}) = \boldsymbol{h}_{k+1}(\boldsymbol{h}_{k+1}^T\boldsymbol{\theta} - z_{k+1})$. This can be expressed as $\boldsymbol{Y}_k(\boldsymbol{\theta}) = \boldsymbol{g}(\boldsymbol{\theta}) + \boldsymbol{e}_k(\boldsymbol{\theta})$, where

$$\boldsymbol{g}(\boldsymbol{\theta}) = [E(\boldsymbol{h}_{k+1}\boldsymbol{h}_{k+1}^T)](\boldsymbol{\theta} - \boldsymbol{\theta}^*) ,$$

$$\boldsymbol{e}_k(\boldsymbol{\theta}) = [\boldsymbol{h}_{k+1}\boldsymbol{h}_{k+1}^T - E(\boldsymbol{h}_{k+1}\boldsymbol{h}_{k+1}^T)](\boldsymbol{\theta} - \boldsymbol{\theta}^*) - \boldsymbol{h}_{k+1}v_{k+1},$$

and z_{k+1} is generated by the true process $\boldsymbol{h}_{k+1}^T\boldsymbol{\theta}^* + v_{k+1}$. Recall that $\Im_k = \{\hat{\boldsymbol{\theta}}_0, \hat{\boldsymbol{\theta}}_1, \ldots, \hat{\boldsymbol{\theta}}_k\}$, as defined in condition B.4. The form for $\boldsymbol{e}_k$ above implies that

$$\boldsymbol{b}_k = E[\boldsymbol{e}_k(\hat{\boldsymbol{\theta}}_k) | \Im_k] = E[\boldsymbol{e}_k(\hat{\boldsymbol{\theta}}_k) | \hat{\boldsymbol{\theta}}_k] = \boldsymbol{0}$$

by the mutual independence of $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots; v_1, v_2, \ldots\}$ and the fact that $E(v_k) = 0$ (the substitution of $\hat{\boldsymbol{\theta}}_k$ for $\Im_k$ in the conditioning is allowed because the independence assumptions on $\{\boldsymbol{h}_k, v_k\}$ imply that conditioning on $\hat{\boldsymbol{\theta}}_k$ alone is equivalent to conditioning on $\hat{\boldsymbol{\theta}}_k$ plus the earlier terms in $\Im_k$).[1] Hence, condition B.5 holds.

Let us now show B.4. Let $\boldsymbol{e}_k = \boldsymbol{e}_k(\hat{\boldsymbol{\theta}}_k)$. Because $\boldsymbol{b}_k = \boldsymbol{0}$, B.4 may be expressed as

$$E\left[\left\|\sum_{k=0}^{\infty} a_k(\boldsymbol{e}_k - \boldsymbol{b}_k)\right\|^2\right] = E\left[\left\|\sum_{k=0}^{\infty} a_k\boldsymbol{e}_k\right\|^2\right] < \infty. \tag{5.15}$$

Note that for all $i < j$, $E(\boldsymbol{e}_i^T\boldsymbol{e}_j) = E[E(\boldsymbol{e}_i^T\boldsymbol{e}_j|\Im_j)] = E[\boldsymbol{e}_i^T E(\boldsymbol{e}_j|\Im_j)] = 0$ (Exercise 5.6). Because $E(\boldsymbol{e}_j^T\boldsymbol{e}_i) = E(\boldsymbol{e}_i^T\boldsymbol{e}_j)$, we have $E(\boldsymbol{e}_i^T\boldsymbol{e}_j) = 0$ for all $i \neq j$. Hence, the cross-product terms in (5.15) disappear, indicating that (5.15) is equivalent to

$$\sum_{k=0}^{\infty} a_k^2 E\left(\|\boldsymbol{e}_k\|^2\right) < \infty. \tag{5.16}$$

It was assumed that $E\left(\|\hat{\boldsymbol{\theta}}_k\|^2\right)$ is uniformly bounded for all k; further, the v_k are i.i.d. with finite variance and the $\boldsymbol{h}_k$ are i.i.d. with finite fourth moments. These facts imply that $E\left(\|\boldsymbol{e}_k\|^2\right)$ is uniformly bounded for all k. Hence, (5.16) holds

[1] Strictly, all conditional expectations are true a.s.; we suppress that clause in the interest of a streamlined discussion.

when $\sum_{k=0}^{\infty} a_k^2 < \infty$. Because this corresponds to one of the assumptions, we have shown B.4, completing the proof. ❑

The sections to follow are *brief* discussions of the application of the stochastic gradient method to the areas of neural networks, discrete-event dynamic systems, and image restoration. These vignettes are not intended to be thorough discussions of the indicated subjects. The reader should consult the indicated references—or other source material—for more complete background on these subjects.

5.2 NEURAL NETWORK TRAINING

Neural networks (NNs) are mathematical models that have become popular over the last two decades as representations of complex nonlinear systems. Among many other areas, applications include forecasting, signal processing, and control. Consider the problem of training a feedforward NN based on input–output data using the well-known backpropagation algorithm (e.g., Jang et al., 1997, pp. 205–210). This has long been recognized as an application of the stochastic gradient version of the SA algorithm, with two of the earlier publications discussing this connection being White (1989) and Kosko (1992, pp. 190–199). One of the main virtues of feedforward (and certain other) NNs is that they are *universal approximators*. That is, they have the ability to approximate broad classes of functions to within any level of accuracy (Haykin, 1999, Sect. 4.13). This property is one of the reasons that NNs have attracted so much attention.

For our purposes here, suppose that the system output z_k can be represented by a NN according to the standard model $z_k = h(\boldsymbol{\theta}, \boldsymbol{x}_k) + v_k$, as introduced in Subsection 5.1.3. (More generally, the output can be a vector $\boldsymbol{z}_k$, but we restrict ourselves to a scalar in this brief discussion.) Here, $h(\boldsymbol{\theta}, \boldsymbol{x}_k)$ represents the NN output, $\boldsymbol{\theta}$ represents the NN weights to be estimated, and, as usual, $\boldsymbol{x}_k$ represents inputs (deterministic or random). The noise v_k captures the difference between a NN output and an actual measurement z_k. Unlike the more general context of Subsection 5.1.3, note that the output $h(\cdot)$ is without a subscript k since we are assuming a fixed NN structure across input–output pairs. Figure 5.2 shows a simple feedforward NN with two hidden layers. This diagram shows two inputs (i.e., $\dim(\boldsymbol{x}_k) = 2$). The output corresponds to a prediction of the scalar z_k.

The NN training problem can be posed as a nonlinear regression problem with $\boldsymbol{\theta}$ representing the NN weights to be estimated. In the nonlinear regression context above, the prediction at the kth input is $\hat{z}_k = h(\boldsymbol{\theta}, \boldsymbol{x}_k)$ for the given input $\boldsymbol{x}_k$ and weights $\boldsymbol{\theta}$. Here the loss function is the normalized sum of mean-squared errors:

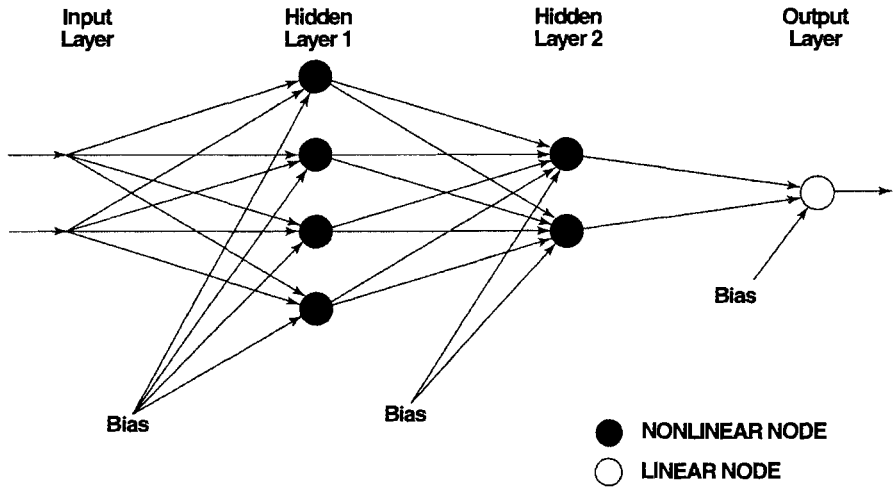

Figure 5.2. Simple feedforward neural network with two inputs in $\boldsymbol{x}_k$. Each of the p connections is weighted by an element of the vector $\boldsymbol{\theta}$ ($p = 25$ here). The inputs to each node are usually summed prior to application of the relevant linear or nonlinear function. The nonlinear nodes are frequently bounded (*sigmoidal*) functions that monotonically increase and have single inflection points. The bias terms represent constant inputs. The rightmost arrow represents the NN output $h(\boldsymbol{\theta}, \boldsymbol{x}_k)$.

$$L(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{k=1}^{n} E\left[(z_k - h(\boldsymbol{\theta}, \boldsymbol{x}_k))^2\right], \tag{5.17}$$

where z_k represents the true system observation. In light of the practical situation where data arrive sequentially (possibly at a high rate), White (1989) emphasizes the backpropagation algorithm based on recursively processing one input–output pair $\boldsymbol{x}_k$, z_k at a time. Using (5.9), the stochastic gradient algorithm then becomes

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{k+1} &= \hat{\boldsymbol{\theta}}_k - a_k \left.\frac{\partial Q_k(\boldsymbol{\theta}, V_k)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k} \\ &= \hat{\boldsymbol{\theta}}_k - a_k \left[\left(h(\boldsymbol{\theta}, \boldsymbol{x}_{k+1}) - z_{k+1}\right)\frac{\partial h}{\partial \boldsymbol{\theta}}\right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k},\end{aligned} \tag{5.18}$$

where the term inside the square brackets represents the noisy observation of the instantaneous gradient of the $(k+1)$st summand in (5.17). In practice, one can take multiple passes through the input–output pairs using (5.18) (e.g., White, 1989, does this in his examples). This only costs computer time (as opposed to the cost of real data) and is generally effective at improving the estimation of $\boldsymbol{\theta}$. It is also possible to run an algorithm with the batch gradient $\partial \bar{Q}/\partial \boldsymbol{\theta}$ as in (5.13) instead of the instantaneous gradient appearing in (5.18). Here, each iteration

requires a set of backpropagation gradients, one for each input–output pair in the data set.

The essential innovation in NN backpropagation is the systematic implementation of the chain rule for computing the $\partial h/\partial \boldsymbol{\theta}$ term that appears in (5.18). Some historical perspective on backpropagation is found, for example, in Kosko (1992, pp. 196–199).

White (1989) also suggests using the backpropagation solution as the initial value for one or more iterations of a Newton–Raphson type of algorithm. This requires the Hessian matrix of the NN (see, e.g., Bishop, 1992). As discussed in Section 1.4, Newton–Raphson is fast, but may be unstable if the initial value is not close to the optimal value. The role of backpropagation in this case is to help avoid the instability.

The asymptotic distribution theory for SA discussed in Section 4.4 can be used to test hypotheses about the effect of particular inputs on the weight identification process. In particular, White (1989) shows how a test statistic can be formed to determine whether particular weights contribute toward the ability of the NN to approximate a function.

White (1989) conducted a numerical study on having a NN emulate a Henon map, which is a nonlinear time series of the form

$$U_t = 1 - 1.4U_{t-1}^2 + 0.3U_{t-2}$$

for scalar U_t. He generated 4000 data points and then sampled randomly (with replacement) from these to get his sequential input–output data for training the NN via (5.18). With output U_t, the inputs used by White were one of the three combinations: U_{t-1} and U_{t-2}, U_{t-1} only, or U_{t-2} only. He used a single hidden-layer NN with nonlinear nodes, each of which had a sigmoidal function. Both a decaying gain, $a_k = a/(k+1)$, and constant gain were used. As might be expected, when both of the inputs were used, the NN provided the best fit. In these studies, decaying gains a_k produced results slightly better than those where constant gains were used. White (1989) also demonstrates the idea of using Newton–Raphson steps on the full data set after achieving effective convergence with the recursion (5.18). This yields further improvement.

There are two other issues strongly related to the stochastic gradient form of SA that arise in the training of NNs. One of these is the momentum form of the gradient input and the other is overfitting, discussed briefly below. *Momentum* is used to smooth out the variations in the individual gradient inputs, especially when using the recursive (one measurement at a time) form of SA as in (5.18). With momentum, the individual gradient inputs are replaced by a weighted combination of the current instantaneous gradient $\partial Q_k/\partial \boldsymbol{\theta}$ and past instantaneous gradients. The modified version of (5.18) then becomes

$$\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k - a_k \tilde{\boldsymbol{Y}}_k ,$$

where

$$\tilde{\boldsymbol{Y}}_0 = \left.\frac{\partial Q_0(\boldsymbol{\theta}, \boldsymbol{V}_0)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_0},$$

$$\tilde{\boldsymbol{Y}}_k = \gamma_k \tilde{\boldsymbol{Y}}_{k-1} + (1-\gamma_k)\left.\frac{\partial Q_k(\boldsymbol{\theta}, \boldsymbol{V}_k)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k}, \quad k \geq 1,$$

and where $0 \leq \gamma_k < 1$. Momentum is sometimes useful in accelerating the algorithm, but the usefulness is at the expense of an additional algorithm sequence that must be specified by the analyst (the weighting coefficient γ_k). Typically, the weighting is greater on the current gradient and less on the previous gradients (Jang et al., 1997, pp. 158–159 and 236–237). Thus, the current input to the recursive algorithm has an exponentially decaying contribution from all past individual gradients when $\gamma_k = \gamma$ for all k, $0 < \gamma < 1$. By letting $\gamma_k \to 0$, the current gradient estimate is given full weight as $k \to \infty$; this is used to ensure that the stochastic gradient algorithm converges to $\boldsymbol{\theta}^*$ as $k \to \infty$ (Spall and Cristion, 1994).

Overtraining (or *overfitting*) is a well-known issue in NNs, regression, and time series. It pertains to tuning the parameter estimates too much to the noise in the data. In fact, n data points can often be used to estimate (uniquely) n parameters, but such a model is likely to be very poor at prediction for a new data set since the original estimate will have adjusted the n parameters to all of the nuances in the particular data set, including the noise. Hence, the general principle is that a parsimonious ("lean") model is preferable to a model with many free parameters. This lets the model parameter estimation uncover overall trends in the data without fitting too closely to the specific noise characteristics of the training data set. This is sometimes referred to as the principle of *Occam's Razor*. However, one must be careful to avoid making the model too parsimonious since there is a risk of providing too little flexibility in the NN to capture the important nonlinearities. Techniques such as cross-validation, the Vapnik–Chervonenkis dimension, and the Akaike information criterion are methods for expressing this tradeoff and choosing the appropriate dimension of the model. Sections 13.1 and 13.2 consider the issues of overtraining and model selection in much greater detail.

Overtraining is a greater threat in batch processing than recursive processing. In batch processing, it is relatively easy to fit the parameters to the noise in the data because there is a *fixed* amount of data and a potentially large number of parameters. For a given NN structure, one practical method for coping with overtraining is to restrict the number of batch iterations to avoid hewing too closely to the specific data set (e.g., Haykin, 1999, Sects. 4.12 and 4.13). There is less of a threat of overtraining in *recursive* processing with a fixed NN structure because there is no hard limit to the amount of data that is to be used in the estimation of the fixed number of weights.

5.3 DISCRETE-EVENT DYNAMIC SYSTEMS

Discrete-event dynamic systems (DEDS) are continuous time systems that evolve by changing states at random points in time. The changes in state correspond to a discrete event. The occurrence of a discrete event, which is one outcome from a discrete set of possible outcomes, triggers a change from one system state to another. Once in a given state, the system will stay in that state until the next discrete event occurs, triggering a jump to a new state. A comprehensive introduction to DEDS is Cassandras and Lafortune (1999).

A large number of practical systems fit within the framework of DEDS, including communications networks, manufacturing systems, computer networks, and transportation systems. Thus, DEDS are models of continuous-time systems whose possible states lie in a discrete set of possible outcomes. Equivalently, the trajectories of the process over time are piecewise constant. For example, if the system state represents the number of users on a computer network, the state remains constant until a user logs off or on, corresponding to a (random) discrete event. From an engineering perspective, the jumps associated with DEDS require fundamentally different tools than traditional models based on ordinary or partial differential equations. Often, DEDS are thought of in the context of queuing networks, where the discrete events might denote a customer arrival or a service completion. In this context, $\boldsymbol{\theta}$ represents parameters that the system designer has some control over, such as mean service rates or other parameters associated with service times.

A typical goal in DEDS design is to come up with the optimal choice of $\boldsymbol{\theta}$ in light of the inherent stochastic aspects of the problem. In DEDS, $\boldsymbol{\theta}$ might represent a collection of direct physical parameters for the system and/or parameters entering the probability distributions of various random processes in the system. Most of the discussion below is drawn from Suri (1989), Glasserman (1991a), L'Ecuyer and Glynn (1994), and Fu and Hu (1997). It is common to optimize DEDS using a Monte Carlo simulation of the process. Chapters 14 and 15 consider simulation-based optimization in detail, including methods relevant to DEDS.

A brute-force approach to optimizing DEDS might be to change various elements of $\boldsymbol{\theta}$, run the simulation several times to get average performance for that chosen value of $\boldsymbol{\theta}$, and then repeat this process for a new value of $\boldsymbol{\theta}$. Obviously, this may be a seriously flawed approach relative to obtaining the true optimal value of $\boldsymbol{\theta}$. If $\boldsymbol{\theta}$ is of even moderate dimension, the number of changes required to ensure reasonable confidence in obtaining a good solution is huge, and, furthermore, each simulation may be very costly to run.

A more systematic approach has been adapted under the rubric of *perturbation analysis* (PA). In this approach, one looks for ways to get a gradient estimate at any $\boldsymbol{\theta}$ value based on only one or a small number of simulation runs. Given the stochastic nature of the simulation, this gradient estimate is only a stochastic estimate of the true gradient $\boldsymbol{g}(\boldsymbol{\theta}) = \partial L/\partial \boldsymbol{\theta}$. A specific form of PA is

the infinitesimal perturbation analysis (IPA) approach to generating a gradient estimate. IPA requires that the probability distribution generating V be independent of $\boldsymbol{\theta}$, consistent with the standard formulation for stochastic gradient methods discussed in Section 5.1. Thus, the IPA gradient estimate has the generic form $\partial Q_k(\boldsymbol{\theta},V_k)/\partial\boldsymbol{\theta}$ or $\partial\bar{Q}(\boldsymbol{\theta},V)/\partial\boldsymbol{\theta}$, as appropriate (recursive or batch, respectively).

The IPA method has a significant potential advantage in efficiency over traditional methods based on approximating the gradient using finite-difference methods (see Chapter 6). The finite-difference methods typically require between $p+1$ and $2p$ simulation runs to form a gradient approximation, in contrast to the one run for IPA. Most discussion of IPA heavily exploits structure that is unique to DEDS. Aside from the above-mentioned assumption regarding the distribution of V being independent of $\boldsymbol{\theta}$ (as in other stochastic gradient methods), IPA requires two problem characteristics to be true:

(i) Small perturbations in $\boldsymbol{\theta}$ do not cause a dramatic change in the output or a change in the order of occurrence of events in the simulation.

(ii) Enough information about the inner workings of the simulation is available to calculate $\partial\bar{Q}(\boldsymbol{\theta},V)/\partial\boldsymbol{\theta}$ or $\partial Q_k(\boldsymbol{\theta},V_k)/\partial\boldsymbol{\theta}$, as appropriate, using one run of the simulation.

Neither of (i) or (ii) is automatically true in most practical DEDS, but if (i) and (ii) do hold, then IPA can be used to provide the required input for the gradient-based SA recursion (5.9). Condition (i) is closely tied to the general requirements for interchange of derivative and integral (Theorem 5.1 in Subsection 5.1.1), so that $E[\partial Q_k(\boldsymbol{\theta},V_k)/\partial\boldsymbol{\theta}] = \boldsymbol{g}(\boldsymbol{\theta})$ at all $\boldsymbol{\theta}$ of interest. A large fraction of Glasserman (1991a), for example, is devoted to establishing formal conditions in DEDS under which (i) holds. (Suri, 1989, presents a realistic example where the interchange of derivative and integral is *invalid.*) Nevertheless, even when (i) holds, the failure of condition (ii) will make it impossible to implement IPA. Monte Carlo simulations are often used for optimization of DEDS. In many complex simulations, it is difficult or impossible to calculate $\partial Q_k(\boldsymbol{\theta},V_k)/\partial\boldsymbol{\theta}$, as that requires detailed analytical knowledge of the effect of each element of $\boldsymbol{\theta}$ on the simulation output.

Because DEDS are frequently associated with queuing systems, we present a simple *conceptual* example of a loss function and associated Q for a queuing-based DEDS and IPA implementation below.

Example 5.5—Simple queuing system. Suppose that a simulation output $Q(\boldsymbol{\theta}, V)$ represents the total waiting time in a system for a group of customers, where $\boldsymbol{\theta}$ is a vector of design parameters that can be adjusted to govern "typical" waiting times and V represents random effects whose distribution does not depend on $\boldsymbol{\theta}$. For example, V might represent random arrival times for customers

into the system, which are unaffected by $\boldsymbol{\theta}$. The loss function is $L(\boldsymbol{\theta}) = E[Q(\boldsymbol{\theta},\boldsymbol{V})]$, representing the typical waiting time given by the simulation, averaged over all possible values of $\boldsymbol{V}$.

Suppose that a small change in $\boldsymbol{\theta}$ causes only small changes in simulation output Q. For example, we do *not* allow a case where, say, a slight change in the position of a piece of equipment—corresponding to one or more components of $\boldsymbol{\theta}$—causes the system to be blocked and forces the times in the queue to become indefinitely long (violating condition (i)). Further, suppose that enough is known about the simulation so that $\partial Q/\partial\boldsymbol{\theta}$ can be calculated (satisfying condition (ii) above). Then the stochastic gradient algorithm (5.9) can be used to estimate $\boldsymbol{\theta}$ based on the principles of IPA. Adding the subscript k to Q, the measurement $\boldsymbol{Y}_k(\boldsymbol{\theta}) = \partial Q_k(\boldsymbol{\theta},\boldsymbol{V}_k)/\partial\boldsymbol{\theta}$ represents the gradient associated with the kth simulation run. For example, each simulation output $Q = Q_k$ may represent the total system wait time for one day of operation. If the conditions for convergence apply (as in Section 4.3), the IPA-based SA algorithm converges to $\boldsymbol{\theta}^*$. ❑

5.4 IMAGE RESTORATION

In image restoration, the task is to recover a true image of a scene from a recorded image of the scene. The recorded image is typically corrupted by noise and/or otherwise degraded from the original image. The image restoration problem may be viewed as one of taking data $\boldsymbol{Z}$ and recovering an image $\boldsymbol{s}$ ("scene") that is observed through some nonlinear and noisy mapping. That is:

Estimate $\boldsymbol{s}$ such that $\boldsymbol{Z}$ is modeled according to $\boldsymbol{Z} = \boldsymbol{F}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{V})$,

where $\boldsymbol{Z}$ is a vector of the observed pixel images, $\boldsymbol{F}$ is the relevant nonlinear mapping, $\boldsymbol{x}$ is some known input vector, and $\boldsymbol{V}$ is the random term, often taking the form of noise. For the problem as given above, the vector of parameters $\boldsymbol{\theta}$ corresponds to $\boldsymbol{s}$. We also consider a case below where $\boldsymbol{\theta}$ represents the parameters (weights) of a NN associated with an image processing problem.

Following Gonzalez and Woods (1992, Chap. 5) (hereafter G&W), suppose that a digitized form of an image is available. The digitized form is often a pixel-by-pixel collection of gray levels of some typically degraded image, as might have been obtained, say, through the quantification of an image obtained through cloud cover. G&W consider the case where the degradation is approximated by linear, position-invariant processes. However, they note that nonlinear mappings often make the restoration more accurate at the expense of greater complexity.

In the linear case, G&W construct a simple least-squares problem:

$$\min_{\boldsymbol{s}} \|\boldsymbol{Z} - \boldsymbol{H} \bullet \boldsymbol{s}\|^2, \tag{5.19}$$

where $\boldsymbol{H} \bullet \boldsymbol{s}$ represents a convolution of the measurement process ($\boldsymbol{H}$) and the true pixel-by-pixel image ($\boldsymbol{s}$) (the convolution represents a smoothing process, but the details are not important for the discussion here—see G&W). This can be solved by either batch linear regression methods or the LMS/RLS methods discussed in Chapter 3. As mentioned in Subsection 5.1.4, the LMS/RLS methods are special cases of the root-finding SA algorithm. G&W also show how constraints can be included through the well-known Lagrange multiplier method.

If we adopt the nonlinear approach mentioned in G&W, then the criterion analogous to that in (5.19) does not lend itself to the customary linear regression-type solution. In this case, the more general SA framework can be applied if the appropriate gradient is available. Suppose that $\boldsymbol{F}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{V}) = \boldsymbol{h}(\boldsymbol{s}, \boldsymbol{x}) + \boldsymbol{V}$, where $\boldsymbol{h}(\cdot)$ is some nonlinear function and $\boldsymbol{V}$ is an independent, mean-zero noise. Then, (5.19) may be changed to the following optimization problem based on a stochastic criterion:

$$\min_{\boldsymbol{s}} E\left(\left\|\boldsymbol{Z} - \boldsymbol{h}(\boldsymbol{s}, \boldsymbol{x})\right\|^2\right). \tag{5.20}$$

Expression (5.20) is then appropriate for the root-finding SA algorithm. The implementation may be in a complete-data (batch) mode, where the algorithm repeatedly passes through all the pixel data at once, leading to each iteration relying on $\partial\left(\left\|\boldsymbol{Z} - \boldsymbol{h}(\boldsymbol{s}, \boldsymbol{x})\right\|^2\right)/\partial \boldsymbol{s}$. Alternatively, the implementation may be in a pixel-by-pixel mode, where the algorithm passes through the data using $\partial\left(\left\|z_k - h_k(\boldsymbol{s}, \boldsymbol{x}_k)\right\|^2\right)/\partial \boldsymbol{s}$ at each iteration, where z_k represents the gray level for the kth pixel, $h_k(\cdot)$ represents the relevant nonlinear mapping for the kth pixel, and $\boldsymbol{x}_k$ represents the relevant user input for that pixel.

To reflect correlation in the pixels and the relative quality of the image data, it is, in general, preferable to include some type of weighting for the various contributions in (5.19) or (5.20). The form shown above does not include such weighting since it is built from a simple Euclidean norm (i.e., sum of squared errors). Obviously, the generalization to weighting of the squared errors does not affect the overall optimization approach.

Specific implementations of the ideas discussed above are given, for example, in Abreu et al. (1996) and Cha and Kassam (1996). Abreu et al. (1996) is concerned with removing impulse noise that may be due, for example, to noisy sensor or transmission errors in collecting the image data. They note that nonlinear methods have often proven superior for this problem relative to linear methods. Their approach is based on a classification algorithm where the current pixel is replaced by a smoothed combination of neighboring pixels if the classifier detects that the current pixel is corrupted by noise. Although this involves a linear combining process, the *overall* process is nonlinear because of the ranking and classification aspects of the filter.

The essential stochastic gradient form of the SA recursion arises in this approach through an adaptive calculation of the weighting coefficients in the

linear combining mentioned above; these weights express the emphasis to put on the current pixel value versus the rank-ordered mean. The number of pairs of weighting coefficients is equal to the number of states that the current set of neighboring pixels can occupy; each state has a separate SA recursion.

Cha and Kassam (1996) describe an approach to image restoration based on a type of NN called the *radial basis function network* (RBFN) (e.g., Jang et al 1997, Sect. 9.5). An N-node RBFN with scalar output has the generic form

$$\text{output}(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi(\|\boldsymbol{x} - \boldsymbol{c}_i\|), \tag{5.21}$$

where $\boldsymbol{x}$ is some network input vector, w_i is the ith weight parameter of the RBFN, $\phi(\cdot)$ is some scalar-valued nonlinear basis function, and $\boldsymbol{c}_i$ is a centering vector. RBFNs have a single-layer structure as shown in (5.21), in contrast to the general multilayer structure of the feedforward NNs discussed in Section 5.2. The most common form for $\phi(\cdot)$ is the Gaussian function associated with the normal probability density function. RBFNs have the advantage of being generally faster to train than feedforward NNs (note the lack of weights to be estimated between the input and single hidden layer) and relative ease of interpretation due to their single-layer structure (Cha and Kassam, 1996; Jang et al, 1997, Sect. 9.5). As with feedforward NNs, RBFNs share the property of being universal approximators of continuous functions.

Cha and Kassam (1996) take the recorded gray scale images of a window of pixels surrounding the pixel of interest as the $\boldsymbol{x}$ vector in (5.21) (so $\boldsymbol{x}$ corresponds to part of the $\boldsymbol{Z}$ vector above). The output of the RBFN is the estimate of the pixel of interest. Through the use of the RBFN, it is not necessary to build up an explicit model for the measurement process (the RBFN acts essentially as an inverse model for recovering $\boldsymbol{s}$).

The stochastic gradient SA approach arises in the training of the weights w_i and centering vectors $\boldsymbol{c}_i$ shown in (5.21) (they also train an additional spread parameter analogous to the standard deviation in the Gaussian density function). The algorithm is aimed at minimizing a mean-squared error loss function measuring the deviation between the actual and estimated gray-scale level. As in Sections 5.1 and 5.2, Cha and Kassam (1996) use the instantaneous (pixel-by-pixel) error in the implementation of the algorithm. Although the estimation for the weights, centering parameters, and scale factors is conducted simultaneously, each set of parameters is assigned its own gain sequence (corresponding to the a_k in (5.9)).

Much of the attention in image restoration is on simulated annealing algorithms, which may be thought of as generalizations of stochastic gradient SA in (5.9) to the global optimization context (see Section 8.6). This is a large subject in image processing (see, e.g., Olsson, 1993). Simulated annealing is considered in Chapter 8.

5.5 CONCLUDING REMARKS

This chapter has focused on what is perhaps the most popular use of the root-finding SA class of algorithms—stochastic gradient algorithms for optimization. These algorithms are associated with the fundamental optimization equation $\boldsymbol{g}(\boldsymbol{\theta}) = \partial L/\partial \boldsymbol{\theta} = \mathbf{0}$. Relative to standard optimization, the main distinction is that only noisy measurements of the gradient $\boldsymbol{g}(\boldsymbol{\theta})$ are used. There are several general implementations in the stochastic gradient setting. These include recursive processing, batch processing, and hybrid forms involving multiple passes through a given set of data (epochs).

Many well-known algorithms are special cases of the stochastic gradient setting. These include the LMS and RLS algorithms of Chapter 3, backpropagation for neural networks, infinitesimal perturbation analysis for discrete-event systems, and many algorithms for signal processing and adaptive control (e.g., in noise cancellation, image restoration, target tracking).

While the stochastic gradient formulation enjoys wide use, its very name also points to its major restriction. The requirement for a gradient—even a noisy gradient—means that one must have available the details of the underlying model. In problems involving complex processes, it is frequently difficult or impossible to get such information. Associated with the fundamental issue of gradient availability, the stochastic gradient method requires that the probability distribution generating the randomness (V) be independent of $\boldsymbol{\theta}$ and that the derivative and integral (expectation) be interchangeable in $\partial E[Q(\boldsymbol{\theta}, V)]/\partial \boldsymbol{\theta}$. There are numerous realistic applications where one or both of these requirements break down.

To address some of the difficulties in implementing stochastic gradient methods, Chapters 6 and 7 consider some gradient-free SA methods. These chapters show how the rich theory of SA can be used in optimization problems where only noisy measurements of the loss function are available.

EXERCISES

5.1 Consider the setting of Example 5.1, except that the scalar $W = W(\theta)$ now has a distribution governed by the density function $\exp(-w/\theta)/\theta$ (w is the dummy variable). Create an equivalent random process and associated replacement for W, say W', such that the joint distribution for $\{Z, W'\}$ (representing the random effects V) does not depend on θ.

5.2 Suppose that V is generated according to a density $p_V(\upsilon|\boldsymbol{\theta})$. Write down an unbiased estimate of $\boldsymbol{g}(\boldsymbol{\theta})$ (the estimate does not necessarily have to be implementable in all applications).

5.3 Paraphrase the arguments in Example A.3 (Appendix A) regarding the invalidity of interchange between the derivative and integral. Plot the

function and comment on how this illustrates the violation of at least one condition for the interchange.

5.4 Suppose that $Q(\boldsymbol{\theta}, \boldsymbol{V}) = L(\boldsymbol{\theta}) + \boldsymbol{\theta}^T \boldsymbol{W} \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{w}$, where $L(\boldsymbol{\theta})$ is a differentiable function, $\boldsymbol{W}$ is a symmetric $p \times p$ random matrix, $\boldsymbol{w}$ is $p \times 1$ random vector, and $\boldsymbol{V} = [\boldsymbol{W}, \boldsymbol{w}]$ has an unknown distribution independent of $\boldsymbol{\theta}$ (it is known that the elements of $E(\boldsymbol{V})$ formally exist in the sense discussed in Appendix C). Determine $\boldsymbol{e}(\boldsymbol{\theta})$ and the conditions on $\boldsymbol{V}$ to ensure that condition A.3 in Section 4.3 holds when $\Theta = \mathbb{R}^p$ (i.e., $E[\boldsymbol{e}(\boldsymbol{\theta})] = \boldsymbol{0}$ for all $\boldsymbol{\theta}$).

5.5 Consider the function $Q(\boldsymbol{\theta}, \boldsymbol{V}) = \sum_{i=1}^{p/2} t_i^4 + \boldsymbol{\theta}^T \boldsymbol{B} \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{V}$, where p is an even number, $\boldsymbol{\theta} = [t_1, t_2, \ldots, t_p]^T$, $\boldsymbol{B}$ is a symmetric matrix, and $\boldsymbol{V}$ is a vector of random errors. Derive $\boldsymbol{Y}(\boldsymbol{\theta}) = \partial Q(\boldsymbol{\theta}, \boldsymbol{V})/\partial \boldsymbol{\theta}$ when $\boldsymbol{B}$ and $\boldsymbol{V}$ are independent of $\boldsymbol{\theta}$ (this function is considered in several numerical exercises throughout this book).

5.6 In the proof of Proposition 5.1 on convergence for LMS, show that for $i \neq j$, $E(\boldsymbol{e}_i^T \boldsymbol{e}_j) = 0$, as required in going from (5.15) to (5.16).

5.7 Consider a model of the form $z = \beta_0 + \boldsymbol{\beta}^T \boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{B} \boldsymbol{x} + v$ (suppressing the subscript k), where β_0 is an additive constant, $\boldsymbol{\beta}$ is a vector, and $\boldsymbol{B}$ is a symmetric matrix. Let $\boldsymbol{\theta}$ represent the unique elements in $\{\beta_0, \boldsymbol{\beta}, \boldsymbol{B}\}$. If $\dim(\boldsymbol{x}) = r$, determine $\dim(\boldsymbol{\theta}) = p$ in terms of r.

5.8 Using the production function from Example 4.4, conduct an analysis of the improvement in the accuracy of estimates obtained by using the stochastic gradient implementation in batch and multiple-pass modes. Generate data on the grid defined by the integer pairs (C, W), where $C \in [1, 2, \ldots, 10]$ and $W \in [11, 12, \ldots, 110]$. Perform 1000 iterations by randomly sampling (uniformly) the 1000 grid points without replacement until all 1000 points have been used. Let $\boldsymbol{\theta}^* = [2.5, 0.7]^T$ and use the production function in Example 4.4 to compute the output z_k, with random error $v_k \sim N(0, 5^2)$. Compare your results with those obtained in Example 4.4. Pick an initial condition of $\hat{\boldsymbol{\theta}}_0 = [1.0, 0.5]^T$ for all runs. Produce a table showing the distance of the final estimate to $\boldsymbol{\theta}^*$ for each of the three implementations according to the following three settings: (i) The basic recursive form (5.9) with $a_k = 0.00125/(k+100)^{0.501}$ as the step size. (ii) The batch mode based on 500 iterations of (5.10) as applied to full data set of 1000 points. Let $a_k = 0.0075/(k+100)^{0.501}$. (iii) The multiple-pass implementation (Subsection 5.1.2) by cycling 10 times through the data (using the final parameter estimate of one cycle as the starting value for the next cycle). Use $a_k = 0.00015/(k+100)^{0.501}$ and do not reset to a_0 at each cycle (so a_k runs from a_0 to a_{9999} over the 10 passes through the data).

5.9 With the 160 input–output measurements in **`reeddata-fit`** (discussed in Section 3.4) (available at the book's Web site), perform a *batch* implementation of SA as in (5.10). Use a simplified form of the quadratic model in Exercise 5.7, where $\boldsymbol{B}$ is a diagonal matrix. (Here $\boldsymbol{x} \in \mathbb{R}^6$ represents the input variables defined in Section 3.4—T, A, E, V, S, and F). Let $\boldsymbol{\theta}$

represent β_0, the elements of $\boldsymbol{\beta}$, and the diagonal elements of $\boldsymbol{B}$. Use the standard sum of squared-error loss function. Calculate three estimates of $\boldsymbol{\theta}$ using a gain sequence of the form $a/(k+10)^{0.501}$: One estimate with a = 0.005, one estimate with a = 0.02, and one estimate with another value of a. Use the initial conditions $\beta_0 = 0$, $\boldsymbol{\beta} = \mathbf{0}$, and $\boldsymbol{B} = \boldsymbol{I}_6$. With the 80 independent measurements in **reeddata-test** (available at the book's Web site), report the mean absolute prediction errors (analogous to Table 3.3) for the three estimates of $\boldsymbol{\theta}$.

5.10 For the model form and problem setting of Exercise 5.9, estimate $\boldsymbol{\theta}$ using a *recursive* implementation of the SA algorithm ((5.9) and (5.12)) and *one* pass through the data **reeddata-fit**. Carry out this process three times with the gain $a/(k+10)^{0.501}$: Use a = 0.005, 0.02, and one other value of your choosing. Then, using the 80 independent measurements in **reeddata-test**, report the mean absolute prediction errors (i.e., the mean absolute deviation [MAD], analogous to Table 3.3) for the three different values of a. If you also did Exercise 5.9, how do the final recursive and iterative batch estimates compare for $\boldsymbol{\theta}$ (based on the analysis with **reeddata-test**)?

5.11 With the data in **reeddata-fit**, estimate a single hidden-layer feedforward NN using backpropagation in a batch implementation (pick eight to 10 nodes in the hidden layer, each node being a common form of sigmoidal function or hyperbolic tangent function). Use a gain sequence of the form $a/(k+10)^{0.501}$ and initial weights generated uniformly over (–0.5, 0.5). (It is not necessary to code backpropagation from scratch; use, e.g., the version in the MATLAB NNs toolbox or any reliable version from the Web where the gain sequence can be user-specified.) After completing the above, use the 80 measurements in **reeddata-test** to determine the mean absolute prediction error (a.k.a. the MAD) from the trained NN. Compare with the numbers in Table 3.3.

5.12 Perform a recursive implementation with a single hidden-layer feedforward NN of the form in Exercise 5.11. This recursive implementation should be run for one pass (160 iterations) through the data based on initial weights as in Exercise 5.11. Compare the MAD using data in **reeddata-test** for several values of a in a gain sequence of the form $a/(k+10)^{0.501}$ and with the values in Table 3.3. If you also did Exercise 5.11, how do the batch and recursive results compare?