Various Algorithms for Optimization and Learning in Adaptive Systems

Thesis by

Brooke P. Anderson

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1993

(Defended Sept. 29, 1992)

To Richard D. and Dorothy L. Anderson, my parents.

Acknowledgements

I would like to thank Professor John Hopfield for his support and for allowing me freedom in choosing my research. Thank you, John, for being an excellent advisor and for making my graduate work at Caltech so gratifying.

I would also like to thank Douglas Kerns for always giving me expert advice on hardware and for helpful discussions on many other matters; David MacKay for turning me into a Bayesian; Ron Benson, Carlos Brody-Pellicer, Andreas Herz, and Marcus Mitchell for insightful discussions and for providing an intellectually stimulating environment; Mike Hasselmo for teaching me about experimental methods in neurobiology; and Joe Bryngleson, Dawei Dong, and Zhaoping Li for help in years past.

Thanks also go to the organizations that gave me financial support: the Office of Naval Research, the Ralph M. Parsons Foundation, and the Department of Health and Human Services.

Abstract

This thesis discusses various methods for learning and optimization in adaptive systems. Overall, it emphasizes the relationship between optimization, learning, and adaptive systems; and it illustrates the influence of underlying hardware upon the construction of efficient algorithms for learning and optimization. Chapter 1 provides a summary and an overview.

Chapter 2 discusses a method for using feed-forward neural networks to filter the noise out of noise-corrupted signals. The networks use back-propagation learning, but they use it in a way that qualifies as unsupervised learning. The networks adapt based only on the raw input data—there are no external teachers providing information on correct operation during training. The chapter contains an analysis of the learning and develops a simple expression that, based only on the geometry of the network, predicts performance.

Chapter 3 explains a simple model of the piriform cortex, an area in the brain involved in the processing of olfactory information. The model was used to explore the possible effect of acetylcholine on learning and on odor classification. According to the model, the piriform cortex can classify odors better when acetylcholine is present during learning but not present during recall. This is interesting since it suggests that learning and recall might be separate neurochemical modes (corresponding to whether or not acetylcholine is present). When acetylcholine is turned

off at all times, even during learning, the model exhibits behavior somewhat similar to Alzheimer's disease, a disease associated with the degeneration of cells that distribute acetylcholine.

Chapters 4, 5, and 6 discuss algorithms appropriate for adaptive systems implemented entirely in analog hardware. The algorithms inject noise into the systems and correlate the noise with the outputs of the systems. This allows them to estimate gradients and to implement noisy versions of gradient descent, without having to calculate gradients explicitly. The methods require only noise generators, adders, multipliers, integrators, and differentiators; and the number of devices needed scales linearly with the number of adjustable parameters in the adaptive systems. With the exception of one global signal, the algorithms require only local information exchange.

Contents

List of Figures

List of Tables

Chapter 1

Summary

This thesis is based on the results of several diverse research projects. It is therefore not monolithic. Nevertheless, there are two overriding themes: the relationship between optimization, learning, and adaptive systems; and the influence of underlying hardware upon the structure of algorithms for learning and optimization. This chapter introduces these two themes, relates them to the other chapters in the thesis, and then summarizes the other chapters in the thesis.

1.1 Optimization, Learning, and Adaptive Systems

Adaptive systems (both manmade and biological) adjust internal variables or structures in response to feedback from the environment. Presumably, this adjustment results in better performance for the systems—hence the link between optimization and adaptive systems. In this thesis, such adjustment is called "learning" or "adaption." The following paragraph explains more precisely how it is related to optimization.

Optimization is clearly a part of many manmade adaptive systems. Backpropagation networks and systems that use LMS learning are two examples that

are built around optimization [57, 44, 58]. In biological systems, however, the role of optimization is less clear. For example, not enough is known about most biological nervous systems to say precisely what, if anything, is being optimized on the neuronal level during learning. Nevertheless, on a macroscopic or system level, there usually exist measures by which one judges the quality or effectiveness of learning. When a rat learns to navigate a maze, the length of time it takes to navigate can serve as a measure of learning. When a person learns calculus, his ability to solve calculus problems can serve as a measure. Thus, adaption or learning in manmade systems is often linked to optimization right from the design phase while learning in biological systems is, at our current level of knowledge, often linked to optimization less directly through the definition of measures by which the learning is judged.

This thesis illustrates these ideas. Chapter 2 explains a method that uses an optimization algorithm to adjust the weights of a feed-forward network. Chapter 6 discusses an optimization algorithm for implementation in analog hardware. In both cases, optimization is intrinsic to the construction of the adaptive systems. Chapter 3 explains a simplified model of the piriform cortex, which involves optimization indirectly. In order to judge the effectiveness of learning, a performance measure was created that increases during learning. This measure is almost certainly not the same function the piriform cortex maximizes during learning (if there even is such a function), but the result—that learning increases *some* measure of performance—is an example of optimization.

1.2 The Influence of Underlying Hardware

The effectiveness of an algorithm depends on the type of hardware on which it is

implemented. For example, digital computers are excellent at performing sequential operations and at executing long lists of logical operations, but they are not very fast at simulating large dynamical systems (like air flow around entire airplanes) compared to the native analog implementation (i.e., the real thing operating in real time). Analog computers are excellent at solving certain differential equations quickly but are not very good at processing lists of logical operations. Biological nervous systems are excellent at producing adaptive behavior but are not very good at multiplication and division.

Thus, hardware influences the construction of algorithms from the start. It is clear, for example, that many of the techniques from traditional artificial intelligence are for digital computers. Such techniques are dominated by operations that search lists or other structures or that test to see whether conditions are true or false, and the corresponding solutions to problems are broken down into series of logical statements. (For examples, see standard texts on artificial intelligence, such as [49], [7], and [60].) Conversely, algorithms meant for implementation in analog hardware can often be written compactly as differential equations. They usually lack the equivalent of "if-then" statements and the accuracy and flexibility of digital programs. (See [39], [54], or [51] for discussions of analog computation.) Biological hardware is different still. Biological nervous systems (even those of insects) cannot currently be functionally duplicated in either digital or analog hardware—not enough is known about how they function. It is unclear if all the computations being performed in such systems will translate well to another medium.

This thesis discusses topics that involve all three of the types of hardware mentioned above. Chapter 2 explains a method that uses conjugate-gradient descent—

an optimization algorithm designed for efficiency on digital computers. Chapter 3 discusses a simplified simulation of an algorithm running on biological hardware, an algorithm that certainly does not run as efficiently on a digital computer. Chapters 4, 5, and 6 all present algorithms meant specifically for analog hardware, and none of these algorithms would be the best choice for implementation on digital computers. Thus, for efficiency, hardware considerations cannot be totally separated from algorithmic considerations—the two are interdependent.

1.3 Summary of Chapter 2

Chapter 2 discusses a method that uses feed-forward neural networks for filtering the noise out of noise-contaminated signals. Specifically, the method uses networks in which the number of input neurons and the number of output neurons are the same but that have fewer middle-layer neurons (i.e., the networks pinch down in the middle). The chapter also presents an analysis of the performance of such networks. This analysis leads to a simple equation that, based only on the geometry of the network and under certain conditions, correctly predicts performance.

It is interesting that the networks discussed in this chapter are trained only with the data available from the environment—there is no external teacher telling the networks what they should produce. Thus, even though the networks adapt by back-propagation learning the adaption in this case is an example of unsupervised learning.

1.4 Summary of Chapter 3

The piriform cortex is the cortical region of the brain that processes olfactory information coming from the olfactory bulb via the lateral olfactory tract. It is

composed of several layers. One layer, layer Ia, contains connections from the lateral olfactory tract to the neurons in the piriform cortex. Another, layer Ib, contains connections among the neurons in the piriform cortex. There is a clear anatomical separation of these connections. Recently, Hasselmo and Bower found that acetylcholine affects the synapses in layers Ia and Ib completely differently: it shuts down the Ib connections and hardly affects the Ia connections [25].

Chapter 3 discusses a simple model of the piriform cortex used to investigate the possible effect of acetylcholine on learning and to explore the computational utility of a differential effect on layers Ia and Ib. In order to judge effect, one needs a measure of performance. This chapter presents a performance measure based on signal-to-noise ratios (which are widely used in the field of signal processing).

By this measure, the model performs better when acetylcholine is present while learning new odors and is absent while recalling odors. This result is interesting since it suggests that learning and recall might be separate modes (corresponding to whether or not acetylcholine is present). It is also interesting since a lack of acetylcholine results in a decreased ability to learn new patterns but not to recall old patterns. This is somewhat similar to the effects of Alzheimer's disease, a disease associated with the degeneration of cells that distribute acetylcholine to the cortex.

1.5 Summary of Chapters 4, 5, and 6

There are many excellent algorithms for optimization, learning, and adaptive systems for use on digital computers. If one must use analog hardware, however, the options are considerably restricted. With analog hardware, the designer no longer has a general-purpose computing machine at his disposal and must be concerned

with how well the necessary computations match the capabilities of the hardware. Storage is not as straightforward and convenient; iterative loops and logical rules are not as easy to implement; division is difficult except in cases where very low precision is tolerable; etc. (See [39], [54], [51], and references therein for general discussions of analog computation.) As a result, algorithms that are straightforward to implement on a digital computer can be prohibitively difficult to implement in analog hardware.

It might seem that one should simply use microprocessors and sidestep this problem, but there are areas in which analog devices, specifically analog VLSI devices, have advantages. For example, analog VLSI devices typically require very little power, are very compact, and can easily implement certain highly parallel operations [39, 53]. These properties are useful in devices such as pacemakers (where low power consumption results in longer time between replacement), hearing aids (where compactness is important), and neural networks (where the fundamental computations are highly parallel). Also, some analog VLSI devices are naturally damage resistant because of parallel or other damage-resistant algorithms. (See [39], [55], and [56] for examples.) This property is useful for systems such as space-probe control systems, which need to be resistant to radiation damage.

Chapters 4, 5, and 6 discuss several adaptive algorithms specifically designed for ease of implementation in analog hardware. Chapter 4 first discusses the use of low-pass filters as expectation operators for use on signals contaminated by multiplicative noise. This is necessary for material that follows. Chapter 5 discusses two methods for estimating gradients. These can be useful for adaptive systems that require gradient information to tune internal parameters (as is often the case). Chapter 6 discusses an algorithm based on some of the ideas in Chapter

5 and specifically designed to implement optimization, which is the key operation of many adaptive systems.

Chapter 2

A Method for Noise Filtering with Neural Networks

One of the appealing aspects of neural networks is that they are not programmed like traditional computers. With back-propagation learning, for example, networks are given input/output pairs during learning [57, 44]. The network is, in effect, shown what it should output for each given input, and it adjusts itself to do so as well as it is able. Thus, the role of program and programmer is replaced by that of adaptive system and trainer. This is an easier task because the trainer need not explain or even know how to solve a given task; he must merely know what the correct responses are. For this reason, as long as there are enough data available for training, neural networks are often useful in situations where methods of solution are unknown.

Such is the case with Klimasauskas' method for noise filtering using neural networks [34]. Here, neural networks are trained to filter out the noise in noise-contaminated signals. Moreover, the neural networks in the Klimasauskas method are trained with input/output pairs taken directly from the environment, so the technique does not even require a trainer. This is an example of unsupervised

[0]Most of the information in this chapter was originally reported in [2].

learning, and it requires even less knowledge on the part of the person implementing the network than is required in the case of supervised learning described above.

This chapter gives an analysis of the Klimasauskas method, compares the method's performance to that of low-pass and optimal filtering, and discusses some advantages and disadvantages of the method. One interesting result of the analysis is that the filtering performance can, under certain conditions, be predicted purely by the geometry of the network. This and the performance comparisons are the central results of the chapter.

Note: For simplicity, this chapter discusses signals corrupted only by additive, Gaussian white noise. Also, there is a technical distinction between noise filtering, which refers to estimation of the signal for the current time using past data, and data smoothing, which refers to estimation of the signal for some earlier time using past and present data. This technical distinction will often be relaxed, and the chapter refers to both processes simply as "noise filtering" or "filtering."

2.1 Description of the Method

The Klimasauskas method uses layered, feed-forward neural networks. The networks are restricted to three layers, and each layer sends outputs only to the adjacent layer (the next higher layer in the network). The neurons all have sigmoidal transfer functions. The networks have m input neurons, k middle-layer neurons, and m output neurons, with $k < m$. Thus, the networks pinch down in the middle. Weights and thresholds adapt by the back-propagation learning rule [57, 44]. During training, the input and target vectors are identical and consist of sequential samples from the corrupted signal. If $c(t)$ denotes the corrupted signal

and if the network has m input neurons, the input vectors to the network are

$$\{x(t_j) \mid t_j = t_0 + (j-1)\Delta t,\ j = 1, 2, \ldots, N_p\}$$

where

$$x(t_j) = \Big(c(t_j), c(t_j + \Delta t), \ldots, c(t_j + (m-1)\Delta t)\Big),$$

t_0 is an initial time, $\Delta t > 0$ is a sampling interval, and N_p is the number of training patterns. One uses the trained network for data smoothing by running sequential samples from the corrupted signal into the network and taking sequential values from the central neuron of the last layer as the smoothed output (see Figure 2.1).

This chapter deals with a method identical to the Klimasauskas method except for three differences whose rationales are as follows. First, the transfer functions of the output-layer neurons are linear rather than sigmoidal so that the output is not restricted to (0,1). This no longer requires the rescaling of all noise-contaminated signals to the range (0,1). Second, the networks can have more than three layers. This allows more complicated and perhaps more effective network structures. Third, the networks adapt using conjugate-gradient descent [42] rather than gradient-descent-like back propagation. This is important for speed of adaption. Our implementation of conjugate-gradient descent was from two to ten times faster than our implementation of standard back propagation when training large networks (networks with tens or hundreds of weights). Conjugate-gradient descent does have more difficulty with local minimae, but this problem can be circumvented. [1]

While Klimasauskas did use his networks for data smoothing, he did not test

[1] When the algorithm stopped, indicating it had reached a minimum, we added noise to the weights, restarted the algorithm, and allowed it to converge to another stopping point. We repeated this process until, after several tries, the stopping points failed to reduce the error (the mean-square difference between the output and the target vectors).

their abilities to perform noise filtering. We also studied the noise filtering task, which was accomplished by taking sequential values from the last neuron in the output layer. Data smoothing was accomplished as described above.

2.2 Analysis and Problem Conditions

A key assumption of this chapter, one central to the following analysis, is that the signals are deterministic. This is not as restrictive as it might seem both because it admits the possibility of chaotic signals (even those with large dimensionality that are similar in many respects to stochastic signals) and because deterministic signals are common. As mentioned above, the noise is additive, Gaussian white noise, which is not deterministic.

Information that is fed into a network starts out being represented as an m-dimensional vector (at the network's input layer) and eventually gets represented as a k-dimensional vector (at the network's middle layer). Since the networks pinch down in the middle (i.e., since $k < m$), this situation suggests that projection of manifolds onto manifolds of reduced dimensionality plays a role in the process. As the analysis shows, this is the case. Under certain conditions, training causes a network to approximate a projection operator that preserves the signal and attenuates the noise.

2.2.1 Analysis

Before analyzing the operation of the Klimasauskas method, we discuss attractors and the noise-filtering properties of projections. The term "attractor" is used here as in the field of chaos. For example, if $s(t)$ is an N-dimensional state vector evolving in time, the attractor is the structure formed from the set of points $\{s(t) \mid t \in [t_0, \infty)\}$, where t_0 is large enough for all of the start-up transients to

have died out. The space R^N is called the "embedding space." The important aspects of the attractor can be reproduced by the time series

$$\{(s_i(t_j), s_i(t_j + \Delta t_L), \ldots, s_i(t_j + (m-1)\Delta t_L)) \mid t_j = t_0 + j\Delta t_I, j \in \{1, 2, 3, \ldots\}\}$$

where s_i is any component of s, $m > 2D_a + 1$, D_a = dimensionality of the original attractor, $\Delta t_I > 0$ is any sampling interval, and $\Delta t_L > 0$ is any time lag. Also, the attractor can *almost* always be reproduced with $D_a < m \leq 2D_a + 1$. Thus, the input vectors described in Section 2.1, where $\Delta t_I = \Delta t_L = \Delta t$, approximate an attractor if m is large enough. (See [48], [15], and [50] for proofs of the validity of and explanations of such reproductions of attractors.)

In [35], Kohonen describes the noise-filtering properties of linear orthogonal projections. Consider an m-dimensional vector s that evolves on a k-dimensional linear subspace L of R^m $(k < m)$, that is corrupted with additive noise n, and that is orthogonally projected back onto L. Let the corrupted vector $x = s + n$, and define P as the orthogonal projection operator. Kohonen shows that, for a spherically symmetric noise distribution,

$$\frac{\sqrt{\langle \|Px - s\|^2 \rangle}}{\sqrt{\langle \|x - s\|^2 \rangle}} = \sqrt{\frac{k}{m}} \tag{2.1}$$

where $\langle \cdot \rangle = \int (\cdot) p(n_1, n_2, \ldots, n_m) dn_1 dn_2 \cdots dn_m$ is an ensemble average, an average over the noise distribution. Thus, P is a filtering operator because Px is closer to s than x is.

Kohonen also shows that any vector x in R^m can be uniquely decomposed into $x = \hat{x} + \tilde{x}$ where $\hat{x} \in L$ and $\tilde{x}$ is orthogonal to L. One can use this as follows to show that linear orthogonal projections filter out more noise than any other linear projection. Again, let P be the orthogonal projection onto L, and let $x = s + n$ where $s \in L$. Then, $Px = Ps + P(\hat{n} + \tilde{n}) = s + \hat{n}$, and $\langle \|Px - s\|^2 \rangle = \langle \|\hat{n}\|^2 \rangle$.

For a nonorthogonal projection P' that projects onto L, $P'x = P's + P'(\hat{n} + \tilde{n}) = s + \hat{n} + P'\tilde{n}$, and $\langle \|P'x - s\|^2 \rangle = \langle \|\hat{n} + P'\tilde{n}\|^2 \rangle$. Assume that the noise distribution is spherically symmetric and that $\hat{n}$ and $\tilde{n}$ are independent. This is the case for Gaussian white noise, where the components of the noise vector are independent in any orthonormal basis. Then $\langle \|\hat{n} + P'\tilde{n}\|^2 \rangle > \langle \|\hat{n}\|^2 \rangle$, $\langle \|Px - s\|^2 \rangle < \langle \|P'x - s\|^2 \rangle$, and the orthogonal projection filters out the most noise.

This same principle can work for orthogonally projecting noise-corrupted vectors onto nonlinear attractors. If the attractor is smooth enough so that small regions of it appear linear and if the amplitude of the added noise is not larger than this scale of linearity, the above noise-reduction ratio will still approximately hold. This is because each small region of the attractor will appear as a hyperplane onto which the corrupted vector is projected.

With the above introduction, we can now analyze the Klimasauskas method. Since the network takes m-dimensional vectors as inputs, represents them as k-dimensional vectors at the middle layer (where $k < m$), and finally gives m-dimensional vectors as outputs, it is taking m-dimensional vectors and mapping them onto a k-dimensional surface embedded in R^m. Thus, the network is doing a projection onto a surface. If that surface is similar to the pure signal's attractor, if the projection is approximately orthogonal, and if the conditions in the previous paragraph hold, (2.1) will approximately hold where Px is the output vector of the network. This turns out to be the case.

The reason that the surface of projection is similar to the pure signal's attractor and that the projection is approximately orthogonal can be illustrated by considering the optimization done during training. The error function to be minimized

is

$$E = \frac{1}{N_p} \sum_{p=1}^{N_p} \|y(p) - s(p) - n(p)\|^2 = \frac{1}{N_p} \sum_{p} [\|y - s\|^2 + \|n\|^2 - 2(y - s) \cdot n]$$

where s is the pure part of the target vector, n is the noisy part of the target vector, y is the output vector, p is the training-pattern number, and N_p is the number of training patterns. Assuming that N_p is large,

$$E \approx \langle \|y - s\|^2 + \|n\|^2 - 2(y - s) \cdot n \rangle = \langle \|y - s\|^2 \rangle + \langle \|n\|^2 \rangle + 2\langle s \cdot n \rangle - 2\langle y \cdot n \rangle.$$

Since s and n are by definition independent, $\langle s \cdot n \rangle = 0$. Since the network projects m-dimensional vectors onto a k-dimensional surface where $k < m$, according to the projection arguments given above, any noise present in the input is not reproduced exactly at the output. (White noise has an attractor of infinite dimensionality, and some information must be lost in the projection operation.) This suggests that n and the output, y, are not completely correlated. If they are not very strongly correlated, $\langle y \cdot n \rangle \approx 0$, which simplifies the analysis. In fact, Figure 2.2 shows that, for various trained networks, the mean of $y \cdot n$ is near zero, and its error bars (out to one standard deviation) straddle zero. Thus, $\langle y \cdot n \rangle \approx 0$ is a good approximation, and

$$E \approx \langle \|y - s\|^2 \rangle + \langle \|n\|^2 \rangle.$$

This function is minimized when $y = s$.

We already know that the network projects input vectors onto a surface. Since the training process works to minimize E, i.e., to produce $y \approx s$ as closely as the network is able, the training process must work to mold that surface onto the pure signal's attractor. If the network used a significantly different surface, the output would be significantly different from s. s projected onto its own attractor is still s,

but s projected onto a different surface is no longer s, and the projection would introduce error. Likewise, for linear patches of the surface, an orthogonal projection reduces more noise than any other projection, so the training process must work to produce an orthogonal projection in order to minimize $\langle \|y - s\|^2 \rangle$. Thus, assuming that the network has a rich enough structure so that it can represent the pure signal's attractor (i.e., that $k \geq D_a$ and that the network has enough layers and large enough layers), the training process produces a projection for which $y \approx s$, and the network is projecting approximately onto the pure signal's attractor. Then, assuming that the conditions in paragraph 4 of this section hold, the projection is approximately orthogonal, and (2.1) is approximately valid.

Since time averages were more convenient than ensemble averages in the computer simulations, they are used to express the left-hand side of (2.1). Call the resulting expression "R_v." R_v is a measure of the reduction in the noise present in the input vectors. Since white noise is ergodic, time averages are equal to ensemble averages [18]. Then, altogether for the network,

$$R_v \triangleq \frac{\sqrt{\langle \|y - s\|^2 \rangle_t}}{\sqrt{\langle \|x - s\|^2 \rangle_t}} \approx \sqrt{\frac{k}{m}}. \tag{2.2}$$

Because the network reduces the vector noise, it is reasonable to expect that it reduces the noise in each component of the output vector as well. Thus, take $y_i(t)$ as a filtered version of $x_i(t) = c(t + (i-1)\Delta t)$. Figure 2.3 graphically depicts this operation. The left side of the figure shows traditional filtering: a signal is contaminated with noise and then filtered. The filtered version is closer to the pure signal than the noise-contaminated version. The right side of the figure shows how an attractor can be filtered: an attractor is contaminated with noise and then projected. The projected version is closer to the pure attractor than the noise-contaminated version. The arrows linking the two sides show how to use projection

to filter a scalar signal: a noise-corrupted attractor is formed from the noise-corrupted signal, the noise-corrupted attractor is projected, and the appropriate component of the projected version is taken as the output. This component is then the filtered version of the noise-corrupted scalar signal.

2.2.2 Problem Conditions

As discussed above, k must be $\geq D_a$, and m must be of sufficient size for the neural network to perform effectively as a noise filter. Although necessary, these conditions are not sufficient to ensure that the network can do the correct mapping. Obviously, one finite layer of neurons and weights cannot perform all possible mappings. Thus, additional layers (between the input layer and the middle layer and between the middle layer and the output layer) or layers that are larger than the minimum size based on the criteria above might be required. (See [17] and [27] for discussions of the network structures that are necessary for approximating arbitrary mappings.)

Inadequate simulation of the Klimasauskas method on a digital computer can cause problems. Roundoff error and digitization of continuous values can produce small wrinkles and flat spots in the error surface in weight space. Let w be a vector representing the network's weights. Let w^* denote the weights that minimize E. Even if the network is capable of perfect filtering with $w = w^*$ (i.e., $y(x(t); w^*) = s(t)$), the wrinkles can cause learning to stop when $w = w^f \neq w^*$. When this is the case and letting $\Delta y = y(x(t); w^f) - y(x(t); w^*)$, (2.2) becomes

$$R_v = \frac{\sqrt{\langle \|\Delta y\|^2 \rangle_t}}{\sqrt{\langle \|n\|^2 \rangle_t}} = \frac{\sqrt{\langle \|\Delta y\|^2 \rangle_t}}{\sigma\sqrt{m}}$$

where σ is the standard deviation of the noise. As $\sigma \to 0$, $R_v \to \infty$, and the filtering performance is poor. However, at large noise levels, the network should operate as

expected because, as σ increases, $y(x(t); w^*) = y(s(t) + n(t); w^*)$ will on average deviate more and more from $s(t)$ and perfect filtering. Thus, $\|y(x; w^f) - y(x; w^*)\|$ will at some point become negligible compared to $\|y(x; w^*) - s\|$, and one can neglect any difference between the true minimum of E and the point where learning actually stops.

Also, the analysis in this chapter relies on the noise's having an infinite dimensionality. This is true for white noise, but it is not true for practical noise sources (because of finite bandwidths or because the noise is actually from a dynamical system with a large but finite attractor dimensionality, for example). If the noise does not have an infinite dimensionality, it is possible that it could be exactly reproduced at the outputs of a network even though $k < m$. Consider the case in which each neuron in a network can represent B bits of data and in which a single sample $c(t)$ of the noise-corrupted signal contains C bits of data. If $Cm \leq Bk$, it is possible that the network would learn to reproduce everything—including the noise—at its output, and the analysis in Section 2.2.1 would be invalid. However, this seems to be a pathological case: a network would probably require many large layers between the input and middle layers and between the output and middle layers in order to do the complicated encoding and decoding required. The computer simulations described below used a pseudorandom-number generator (with an attractor of finite dimensionality) apparently without disrupting the predictive power of the analysis.

2.3 Computer-Simulation Results

The measure for comparison of the various filtering techniques is based on a ratio of root-mean-square Euclidean distances. Define it as

$$R \triangleq \frac{\sqrt{\sum_{t=1}^{T}(y_i(t) - b(i-1+t))^2}}{\sqrt{\sum_t (c(i-1+t) - b(i-1+t))^2}} \approx \frac{\sqrt{\langle (y_i - b)^2 \rangle_t}}{\sqrt{\langle (c-b)^2 \rangle_t}} \tag{2.3}$$

where $b(t)$ is the scalar signal (as distinguished from the vector s), $c(t)$ is the noise-corrupted scalar signal (as distinguished from x), $y(t)$ is the vector output of the filter, t is a discrete-time variable, and T is large ($\sim 10^4$). In the case of filters other than the network type, the filter generally has only one output, and the subscript i on $y_i(t)$ is meaningless.

2.3.1 Signals and Noise

Four signals were used for testing: (1) a sum of two random-frequency sine waves, (2) a sum of four random-frequency sine waves, (3) the output of the Mackey-Glass equation, and (4) the output of the logistic mapping. The Mackey-Glass signal was generated from a discrete-time version of the Mackey-Glass equation with $a = .2$, $b = .1$, and $\tau = 30$ [38]:

$$s(t+1) - s(t) = \frac{.2s(t-\tau)}{1 + s^{10}(t-\tau)} - .1s(t).$$

The continuous-time version of this equation is chaotic and has an attractor of dimension 3.5. The logistic signal was generated from the function $b(t+1) = 4b(t)(1-b(t))$. $b(t)$ is chaotic and has an attractor of dimension .54 [19]. Figures 2.6, 2.7, and 2.8 display plots of some of these signals, demonstrating the noise-filtering effects of various networks. In terms of notation, all of the computer simulations had $t \in \{1, 2, 3, \ldots\}$ and Δt, referred to in Section 2.1, set to 1.

The pure signal was corrupted with computer-generated, zero-mean Gaussian noise added to the signal at each time step. The amount of noise is reported in decibels (db), where db $= 10 \log$ SNR and SNR is the signal-to-noise ratio defined

as (signal power)/(noise power). Signal power is $P_s = \frac{1}{T}\sum_{t=1}^{T} b^2(t) - \left(\frac{1}{T}\sum_t b(t)\right)^2$. One calculates noise power the same way.

All of the signals were tested at a sampling ratio of 1:1. The two-sine-wave and Mackey-Glass signals were also tested at sampling ratios of 1:19 and 1:3, respectively, where A:B means A points were used out of every B points generated. With these ratios, the sampling frequency is closer to the maximum frequency component in the signal (as shown by an FFT power-spectrum analysis), and filtering is more challenging.

2.3.2 Other Methods Used for Comparison

The following sections compare the Klimasauskas method to low-pass filtering, optimal filtering, and the predictions of (2.2). For low-pass filtering, optimal RC time constants (time constants that gave the best filtering) were used. Optimal filtering was accomplished in the frequency domain using FFT's [42].

2.3.3 Results

Test of the Analysis

To examine the predictive power of (2.2), various networks were trained on the first 5% ($\sim$ 500 data points) of a noise-corrupted signal. Then the entire series was run through the network, and R_v was calculated. Figures 2.4 and 2.5 display the results graphically, showing R_v vs. $\sqrt{k/m}$ for two different signals.

As Figure 2.4 shows, a three-layer network with two middle-layer neurons seems capable of mapping vectors onto the attractor of the two-sine-wave signal. This is reasonable as the attractor is the surface of a 2-torus and as sine waves can be reproduced by single-layer neural networks. With a single middle-layer neuron,

however, $k < D_a$ and the network cannot map vectors onto a 2-dimensional surface. This suggests that the network will have problems accomplishing the needed projection and that it will not behave according to (2.2). Indeed, Figure 2.4 shows that, in this case, R_v rapidly deviates from $\sqrt{k/m}$ as k/m becomes small. It is interesting that, even though one of the necessary conditions for the analysis is violated, R_v is still $\approx \sqrt{k/m}$ as long as k/m is not too small.

For the Mackey-Glass signal, Figure 2.5 shows that a three-layer network with one or three middle-layer neurons seems unable to accomplish the needed mapping. This is expected since $k < D_a$. Again it is interesting that, even though $k < D_a$, $R_v \approx \sqrt{k/m}$ as long as k/m is not too small. The networks with five middle-layer neurons produce $R_v \approx \sqrt{k/m}$ to as low a value of k/m as we were able to handle. On the computers we had access to at the time, the training took too long for smaller k/m (i.e., larger m).

Note that the networks occasionally achieve an R_v better than $\sqrt{k/m}$. This is interesting since it is better than a linear orthogonal projection can do. The nonlinear mapping that the network does is clearly of benefit here.

Comparison with Low-pass and Optimal Filtering

Here, various networks were trained and given data as in the previous section. Then R (see (2.3)) was calculated for the networks and the other methods. Figures 2.6, 2.7, and 2.8 display plots of signals corrupted with 6 db noise, pure signals, and sample network outputs (in smoothing mode) for some of the cases tested. Table 2.1 summarizes the comparative results.

There are several features in Table 2.1 that merit comment. First, the data-smoothing mode of the Klimasauskas method routinely achieves better R values

Table 2.1: Filtering comparison. signal = signal name, db = SNR in decibels, LP = R value for low-pass filtering, opt = R for optimal filtering, net-s = R for Klimasauskas method in smoothing mode, net-f = R for Klimasauskas method in filtering mode, SW = sine waves, MG = Mackey-Glass, LM = logistic map, and config = network configuration listed as number of neurons in the consecutive layers.

signal	db	LP	opt	net-s	net-f	config
2 SW	0	.32	.057	.14	.26	100-2-100
	6	.37	.082	.16	.25	100-2-100
	20	.61	.15	.22	.37	100-2-100
	40	.93	.20	4.8	5.7	100-2-100
2 SW at 1:19	6	.82	.15	.49	.48	100-2-100
4 SW	6	.38	.11	.41	.51	21-4-21
MG	0	.48	.25	.34	.56	35-5-35
	6	.60	.30	.38	.60	35-5-35
	20	.86	.42	.42	.69	35-5-35
	40	1.0	.54	1.9	2.8	35-5-35
MG at 1:3	6	.77	.46	.58	.87	21-5-21
LM	6	.94	.81	.79	.95	3-6-2-6-3

than does the noise-filtering mode. Thus, not all output neurons have equal effect. Second, the Klimasauskas method is not effective for low noise levels (e.g., SNR $\geq$ 40 db) as predicted in Section 2.2.2. Third, the smoothing mode of the Klimasauskas method compares well to optimal filtering on the chaotic signals and generally outperforms low-pass filtering.

2.4 Conclusions

The Klimasauskas method performs well on chaotic signals. For mixtures of sine waves, optimal filtering is much better. The Klimasauskas method does a much better job for data smoothing than for noise filtering. However, even if the performance were better than all other methods, the Klimasauskas method is more cumbersome to implement. It cannot currently be implemented in readily avail-

able special-purpose hardware. Also, the digital-computer implementation of the method is slower than low-pass filtering. Other methods discussed in signal-processing literature, such as Kalman filtering and the adaptive line enhancer, would generally be faster as well since they require fewer computations than the Klimasauskas method does.

The attribute of the Klimasauskas method most likely to lead to its use in spite of the disadvantages is that it requires very little knowledge about the signal and noise. Of course, knowledge about the dimensionality and complexity of the signal's attractor is useful in deciding which network configuration to employ for best effect. However, the Klimasauskas method will generally do some useful filtering as long as k/m is not too small. Optimal filtering, Kalman filtering, and many others require the estimation of noise spectra, of parameters for the process emitting the signal, etc. Thus, the Klimasauskas method is potentially most useful in cases where very little information about the signal or noise is available and where one wants a "smoother" version (in a qualitative sense) of a suspected noise-corrupted signal.

The analysis shows that the Klimasauskas method operates by building up a representation of the signal's attractor and then by projecting inputs onto that representation. Assuming that the network employed has a rich enough structure to represent the attractor, and assuming that the conditions in paragraph 4, Section 2.2.1 are met, the noise-filtering properties of the network are predicted solely by $\sqrt{k/m}$ (see (2.2)).

The Klimasauskas method should be able to handle signals with slowly changing characteristics as long as: first, the characteristics are approximately constant over the length of time required to gather one training set, and second, the signal's

attractor does not eventually violate the necessary conditions (conditions such as $D_a \leq k$). A rough rule of thumb is that a network with N adjustable parameters (weights and thresholds) needs N independent training vectors in order to determine the correct values of the parameters and more than that if the vectors are not independent (as will likely be the case). Thus, the Klimasauskas method should be able to handle signals whose characteristic time for change is an order of magnitude or two greater than $N\Delta t$ where Δt is the sampling interval discussed in Section 2.1.

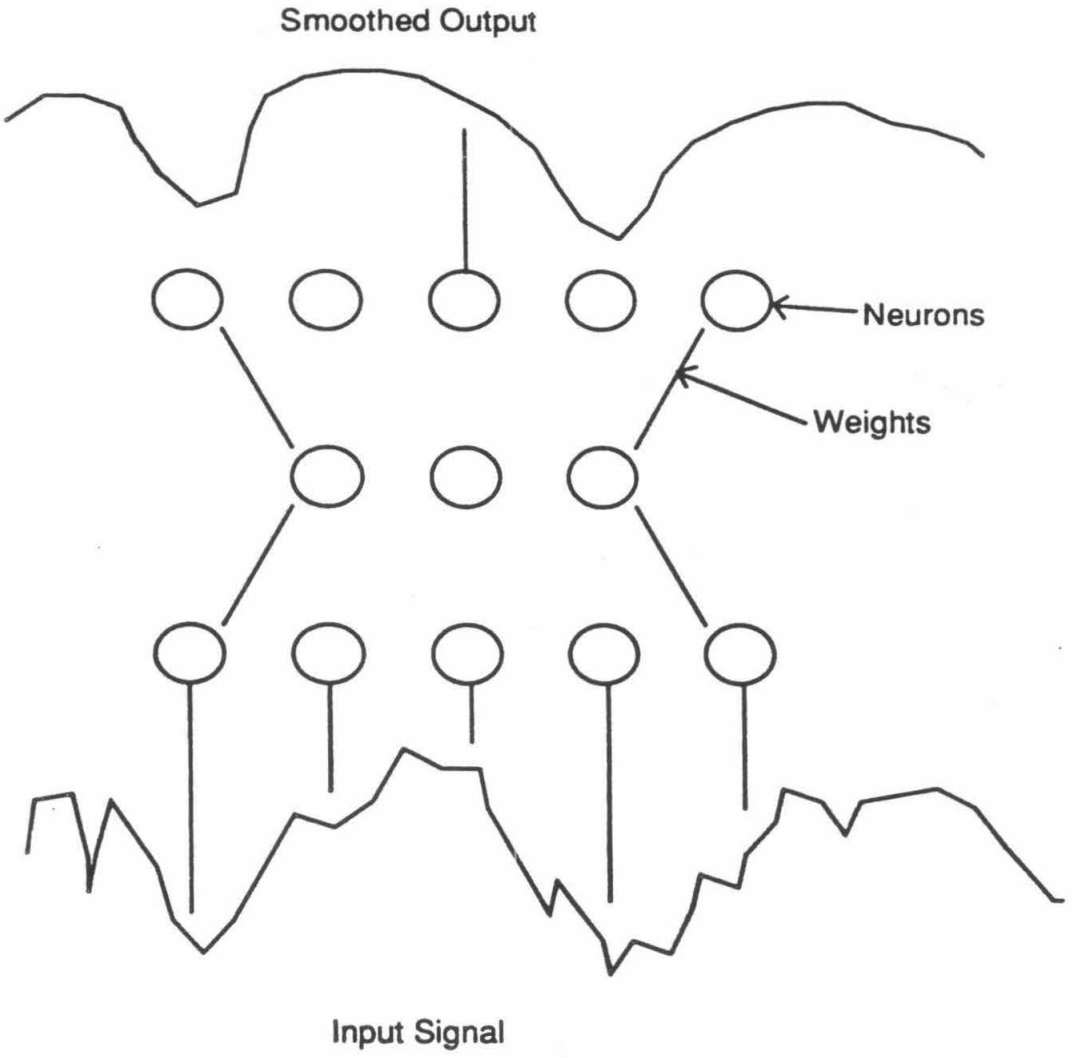

Figure 2.1: Use of a network for data smoothing.

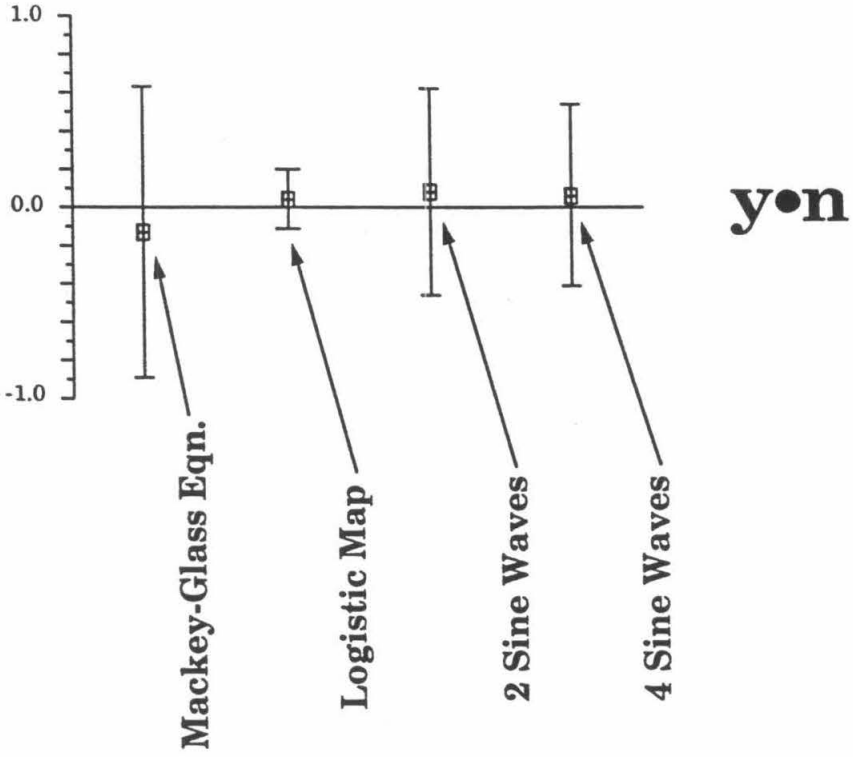

Figure 2.2: Mean and standard deviation of $y \cdot n$.

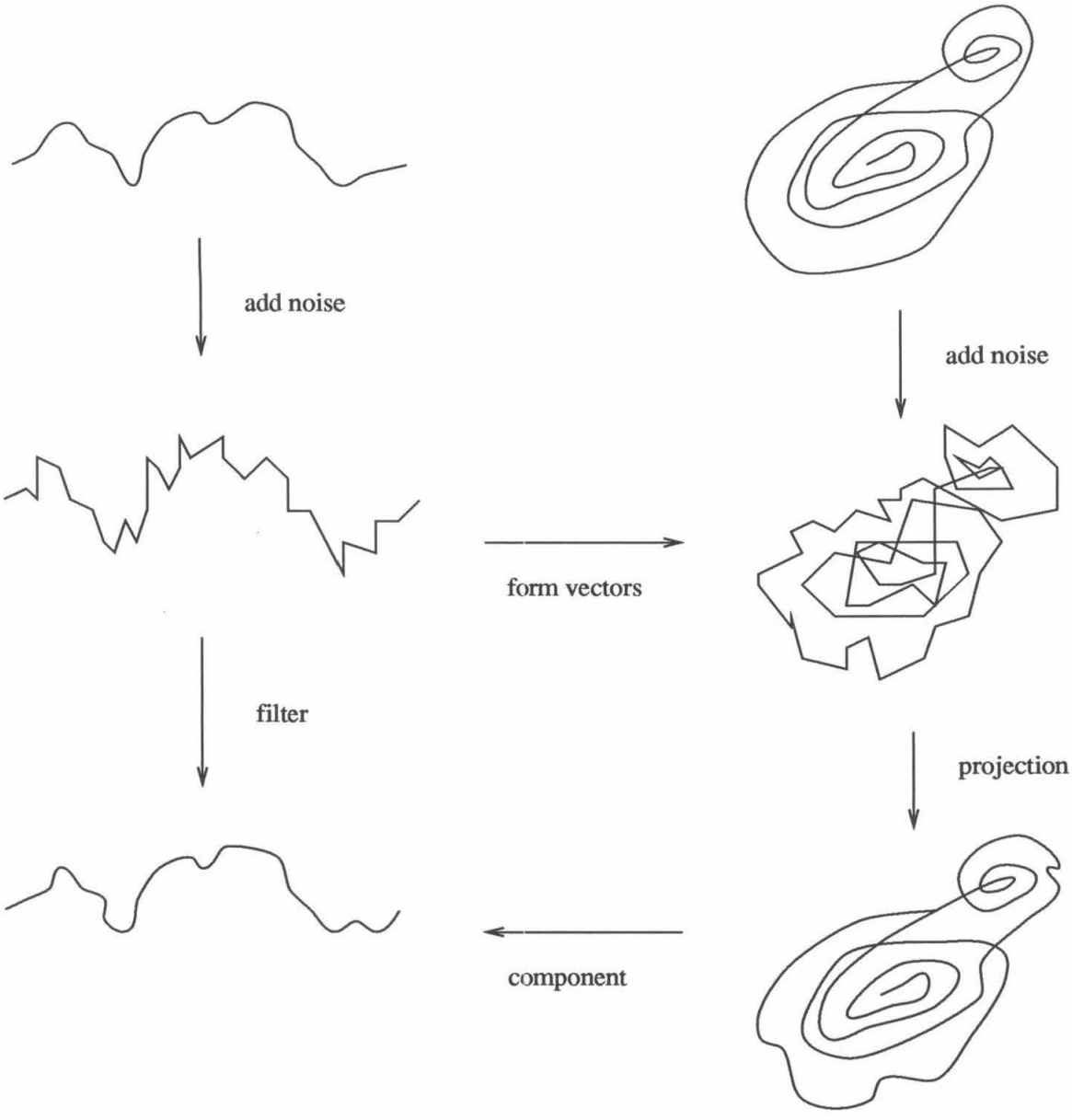

Figure 2.3: Graphical depiction of filtering by projection.

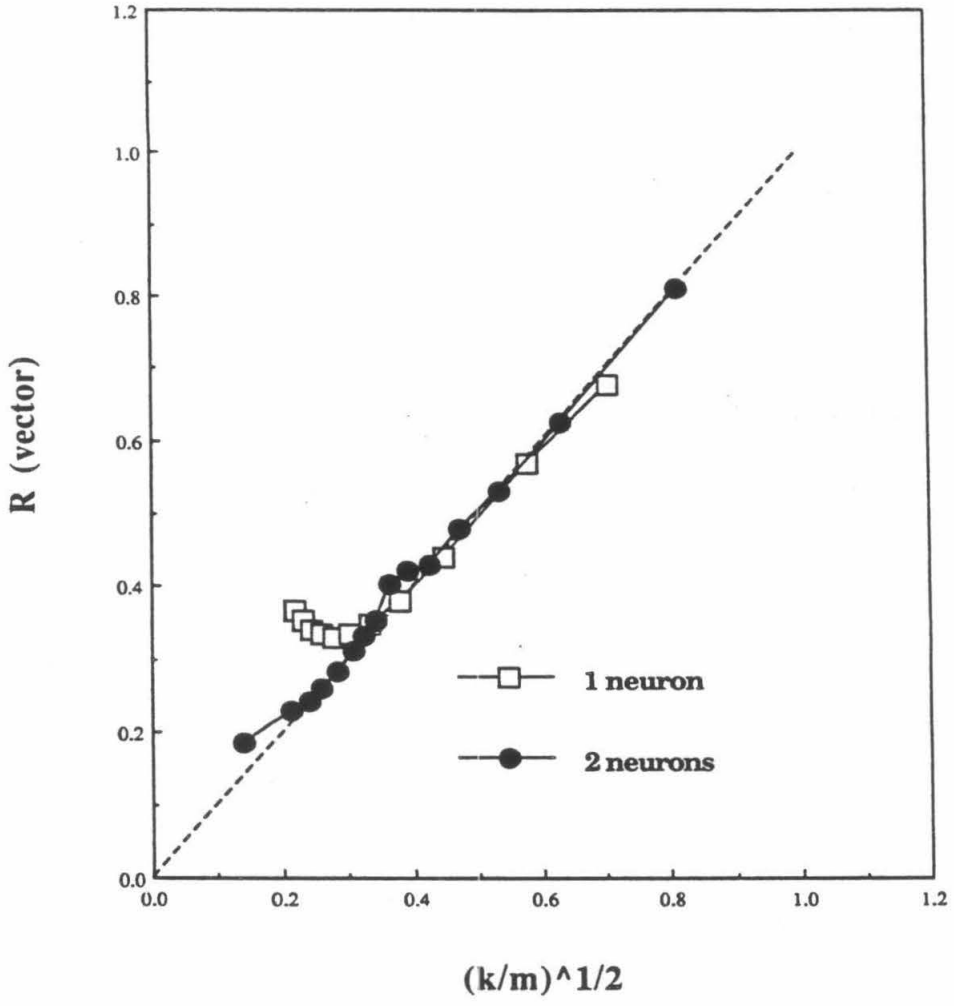

Figure 2.4: R_v for the two-sine-wave case (one vs. two middle-layer neurons).

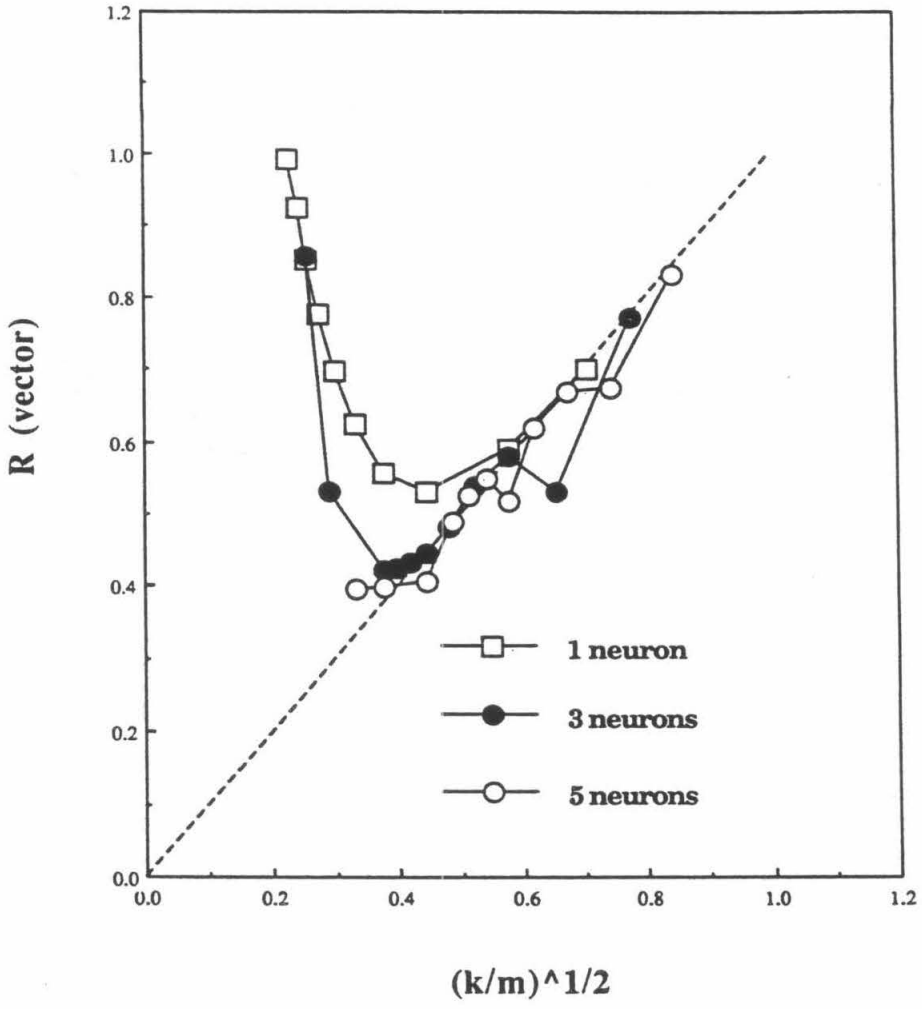

Figure 2.5: R_v for the Mackey-Glass case (one vs. three vs. five middle-layer neurons.)

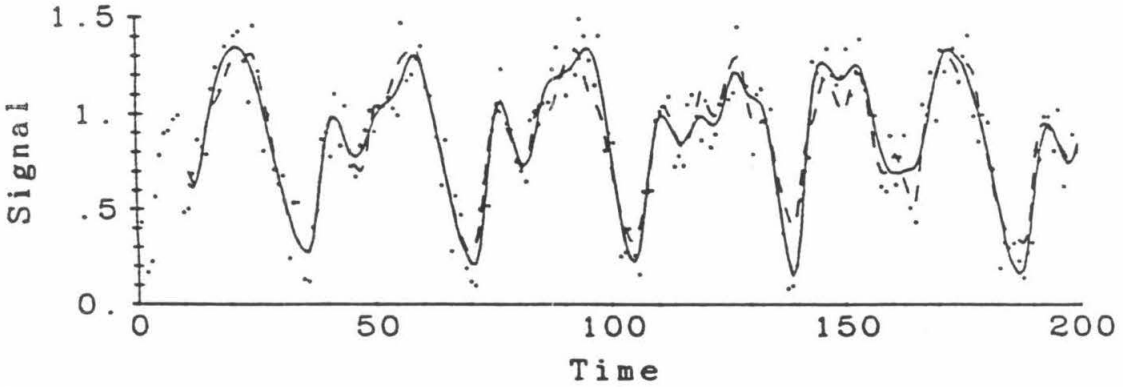

Figure 2.6: Mackey-Glass equation at 1:3. Dots = 6 db noise-corrupted signal, dashed lines = pure signal, and solid lines = smoothed signal.

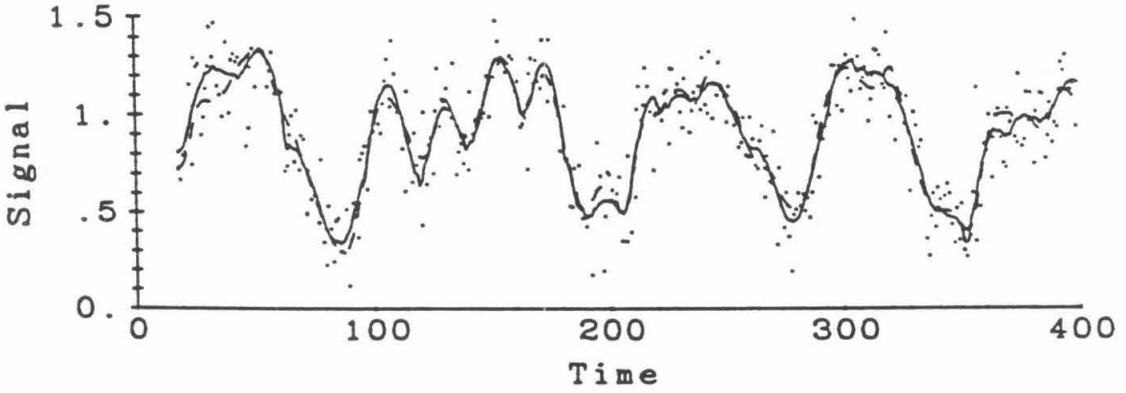

Figure 2.7: Mackey-Glass equation. Dots = 6 db noise-corrupted signal, dashed lines = pure signal, and solid lines = smoothed signal.

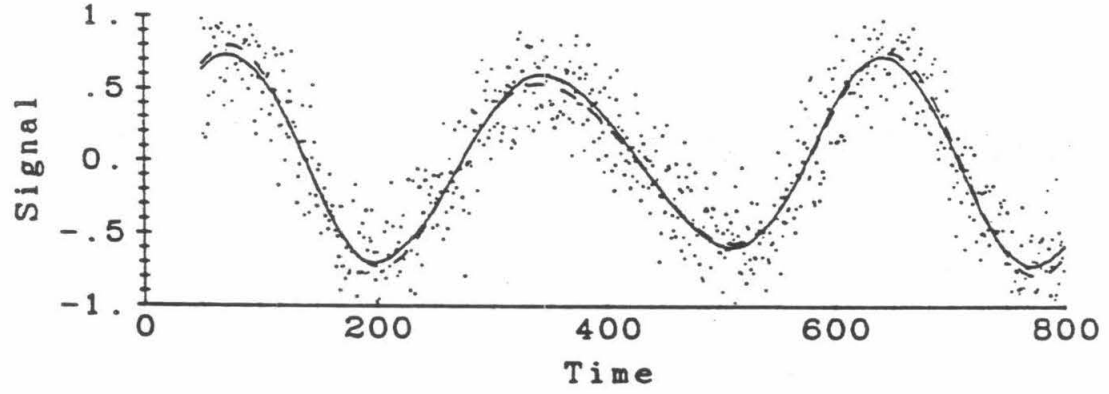

Figure 2.8: Mixture of two sine waves. Dots = 6 db noise-corrupted signal, dashed lines = pure signal, and solid lines = smoothed signal.

Chapter 3

A Model for Learning in the Piriform Cortex

A wide range of behavioral studies support a role for acetylcholine in memory function [37, 21]. However, researchers have not linked specific neuropharmacological effects of acetylcholine to its influence on memory function. With this goal in mind, Hasselmo and Bower recently studied the effect of acetylcholine on the piriform cortex [25].

The piriform cortex is the cortical region of the brain that processes olfactory information coming from the olfactory bulb via the lateral olfactory tract. It has several layers. One layer, layer Ia, contains the afferent fibers, the connections from the lateral olfactory tract to the neurons in the piriform cortex. Another, layer Ib, contains the intrinsic fibers, the excitatory connections among the neurons in the piriform cortex. Figure 3.1 shows this structure schematically. The piriform cortex has widely distributed input and intrinsic connections. Assuming that it also has modifiable synapses (as suggested by experiments that show it exibits long-term potentiation [30, 29]), it has the main elements of abstract associative-memory models [20]. (For descriptions of such models, see, for example, [3] and [36].) Thus,

[0]Most of the information in this chapter was originally reported in [23].

researchers have explored cerebral-cortical associative-memory function using the piriform cortex as a model system, and they have developed computational models of the piriform cortex that show such associative-memory function [59, 6, 24].

This chapter presents a theoretical exploration of the consequences of Hasselmo and Bower's results in the context of associative-memory function. Section 3.1 explains Hasselmo and Bower's experimental results, and Section 3.2 explains the computational model used in the research. The model predicts that perfusion of acetylcholine into the piriform cortex during learning (but not during recall) enhances memory function.

3.1 Experiments

The afferent-fiber and intrinsic-fiber systems of the piriform cortex are physically separated in the piriform cortex. To study differences in the effect of acetylcholine[1] on these two fiber systems, Hasselmo and Bower applied the pharmacological agent carbachol (a chemical analogue of acetylcholine) to a brain-slice preparation of piriform cortex while monitoring changes in the strength of synaptic transmission associated with each fiber system [25]. In these experiments, both extracellular and intracellular recordings demonstrated clear differences in the effects of carbachol on synaptic transmission. The results in Figure 3.2 show that synaptic potentials evoked by activating intrinsic fibers in layer Ib were strongly suppressed in the presence of 100 μM carbachol, while, at the same concentration, synaptic potentials evoked by stimulation of afferent fibers in layer Ia showed almost no change. Hasselmo and Bower found that 100 μM acetylcholine (mixed with 1 μM of the

[1] Acetylcholine is the neurotransmitter in some areas of the nervous system, but research suggests that either glutamate or aspartate is instead the neurotransmitter in the piriform cortex. [9, 10]

acetylcholinesterase blocker neostigmine) produced similar results.

These experiments demonstrate that there is a substantial difference in the neurochemical modulation of synapses associated with the afferent-fiber and intrinsic-fiber systems within piriform cortex. Acetylcholine selectively suppresses intrinsic-fiber synaptic transmission without affecting afferent-fiber synaptic transmission. While interesting in purely pharmacological terms, these differential effects are even more intriguing when considered in the context of computational models of memory function in this region.

3.2 Modeling

There are two aspects to modeling the effects of cholinergic suppression in the piriform cortex. First, obviously, a specific model of the piriform cortex must be developed. Second, a performance measure must be developed so that one has a basis for judging the effects of cholinergic suppression on computational function. These tasks are complicated conceptually by the limited understanding of the exact function of the piriform cortex and, in simulations, also made difficult by limited computer resources. Estimates must be made concerning which aspects will be important and which will only add complication. Otherwise, one is left with the task of modeling the system from the molecular dynamics on up. Even if this were possible, the limited understanding of the piriform cortex would then result in an unworkably large number of undetermined, adjustable parameters in the model.

This chapter presents a simple model of the piriform cortex, one biased towards the assumption that the piriform cortex operates similarly to an associative memory. This model is based on the schematic diagram of the piriform cortex (Figure 3.1), which serves as the schematic diagram of the model. The neurons

in the model have nonlinear transfer functions, but they are nonspiking (i.e., the classic mean-firing-rate representation is used). The synapses have linear transfer functions. The set of odors that the model is to learn is represented by a randomly generated set of input vectors. The learning is modelled as a Hebbian process. More precise details are given below.

A performance measure must also be defined. Associative memories learn mappings between sets of memories, and they tend to reject noise in the inputs. For example, if an associative memory is trained on the set of vector pairs $\{(x^m, y^m) \mid m = 1, 2, \ldots, M\}$ and if a noise-corrupted version of x^k is given as the input, the model should still give an output that is close to y^k. One can quantify this with signal-to-noise ratios, and these ratios form the basis for the performance measure used for testing. The details are given in Section 3.2.3.

3.2.1 Specifics of the Model

Figure 3.1 shows a schematic representation of the model. At each time step, a neuron was picked at random, and its activation was updated as

$$a_i(t+1) = A_i(t) + \sum_{j=1}^{N} [(1-c)B_{ij} - H_{ij}]g(a_j(t))$$

where N = the number of neurons; t = time $\in \{0, 1, 2, \ldots\}$; c = a parameter representing the amount of acetylcholine present $\in [0, 1]$; a_i = the activation or membrane potential of neuron i; $g(a_i)$ = the output or firing frequency of neuron i given a_i; A_i = the input to neuron i, representing the afferent inputs from the olfactory bulb; B_{ij} = the weight matrix or the synaptic strength from neuron j to neuron i; and H_{ij} = the inhibition matrix or the amount that neuron j inhibits neuron i. To account for the local nature of inhibition in the piriform cortex, $H_{ij} = 0$ for $|i-j| > r$, and $H_{ij} = h$ for $|i-j| \leq r$, where r is the inhibition radius.

The function $g(a_i)$ was set to 0 if $a_i < \theta_a$, where θ_a = a firing threshold; otherwise, it was set to $\gamma_a \tanh(a_i - \theta_a)$ where γ_a = a firing gain.

The weights were updated every N time steps according to the following Hebbian learning rule.

$$B_{ij} = f(W_{ij})$$

$$\Delta W_{ij} = W_{ij}(t+N) - W_{ij}(t) = (1-c)\gamma_\ell (a_i - \theta_\ell) g(a_j)$$

The function $f(\cdot)$ is a saturating function, similar to $g(\cdot)$, used so that the weights could not become negative or grow arbitrarily large (representing a restriction on how strong synapses could become). γ_ℓ is a parameter that adjusts learning speed, and θ_ℓ is a learning threshold. The weights were updated every N time steps to account for the different time scales between synapse modification and neuron settling. The same thing can be accomplished by updating the weights every time step and using a smaller γ_ℓ. However, by updating the weights only every N time steps, one reduces the amount of processor time required.

3.2.2 Training of the Model

During learning, the model was presented with various vectors (taken to represent odors) at the input, $A_i(t)$. The network was then allowed to run and the weights to adapt. The input vectors were created randomly, constrained to have no components with values less than zero, and then normalized. Without normalization, some input vectors are much more excitatory than others. For such vectors, a_i and $g(a_j)$ in the learning rule are on average larger than for other vectors, and learning proceeds more rapidly. Normalization makes the learning rates more even.

Specifically, the procedure for creating the set of training vectors $\{A^m \mid m = 1, 2, \ldots, M\}$ was: first, set $A_i^m = \max\{0, G(\mu, \sigma)\}$ where G = Gaussian with aver-

age μ and standard deviation σ, and second, normalize the whole vector so that $\|A^m\|^2 = N(\sigma^2 + \mu^2)$. M = number of memories or odors presented to network during training, and A_i^m = the input to neuron i while odor m is present.

During learning, in the asynchronous update equation, $A_i(t) = A_i^1$ for τ time steps, then $A_i(t) = A_i^2$ for the next τ time steps, and so on; i.e., the various odors were presented cyclically.

3.2.3 Performance Measure for the Model

The piriform cortex receives inputs from the olfactory bulb and sends outputs to other areas of the brain. Assuming that during recall the network receives noisy versions of the learned input patterns (or odors), we presume the function of the piriform cortex is to reduce the noise present in its outputs, creating a more stable representation as a basis for classification or recognition of the odors. Thus, the term "recall" here does not mean that the piriform cortex retrieves an exact copy of a learned input pattern. Recall is simply the process during which the piriform cortex receives a noisy input and produces an output vector which is more suitable for further computation.

One can use signal-to-noise ratios as a quantitative measure of this, but care must be taken in defining the "signal" part of the ratio. For example, consider the vector x corrupted by noise with standard deviation σ. σ is a measure of the amount of noise, but what measures the amount of signal? One might use $\|x\|$, but this is not a good measure if x's being near zero conveys a lot of information. One might assume that all points in space convey equal information, in which case one could define the signal-to-noise ratio as SNR $\triangleq$ const/σ. However, once one is dealing with more than a single point, there is a more useful relationship. Consider two vectors, x^1 and x^2, which are corrupted by noise with standard deviation σ.

What really matters in terms of resolution of the two points is how σ compares to the distance between the two points. Define $d \triangleq \|x^1 - x^2\|$. If d/σ is large, the points seem well resolved—it is easy to tell one from the other. If d/σ is small, the noise will blur the two points together. Thus, d/σ is a reasonable measure of the SNR for a pair of points. Likewise, one reasonable measure for more than two points is the average of d/σ over all pairs of points:

$$\frac{2}{M(M-1)} \sum_{i<j} \frac{\|x^i - x^j\|}{\sigma}$$

where M is the number of points.[2]

With these plausibility arguments in mind, define the performance measure (call it "ρ") as the average d/σ over pairs of outputs (SNR_{out}) divided by the average d/σ over pairs of inputs (SNR_{in}). d is the distance between the means of two vectors, and σ is the standard deviation of the vectors. Thus,

$$\text{SNR}_{in} = \frac{2}{M(M-1)} \sum_{i<j} \frac{\|A^i - A^j\|}{\sigma_{in}}$$

where A^i is training vector i (i.e., the vector that represents odor i) and where σ_{in} is the standard deviation of the noise added to the odor vectors to simulate environmental noise. SNR_{out} is more complicated to compute since the standard deviations of the output vectors are not constrained to be all the same. Thus, use the average standard deviation for each pair:

$$\text{SNR}_{out} = \frac{2}{M(M-1)} \sum_{i<j} \frac{\|V^i - V^j\|}{\frac{1}{2}(\sigma_{out,i} + \sigma_{out,j})}$$

where V^i is the mean of output vector i and where $\sigma_{out,i}$ is the standard deviation

[2]The above ideas about resolving points are similar to those used in the analysis of error-correcting codes. There, one is interested in resolving one code from another in the presense of transmission noise, and the codes can be represented as points in code space. See texts on information theory, such as [1] and [4], for details.

measured for output vector i. Finally,

$$\rho = \frac{\text{SNR}_{out}}{\text{SNR}_{in}}.$$

$\rho > 1$ means that the average SNR for the outputs of the model is greater than that for the raw inputs, and the model is performing a useful function: it is creating a more easily resolved representation of the odors. Figure 3.3 illustrates a situation in which $\rho > 1$.

3.2.4 Testing the Model

The purpose of the model is to show the influence of acetylcholine on learning performance. To that end, the model was allowed to learn for a time with various levels of acetylcholine present; then acetylcholine was turned off ($c = 0$), and the model was tested. Acetylcholine was not left on during testing because it shuts down the intrinsic connections, which results in poor performance during recall.

For testing, weight adaption was turned off, acetylcholine influence was turned off ($c = 0$), and ρ was calculated as follows. Gaussian noise was added to A^1, and the resulting vector was given as an input to the trained network. The network was allowed to settle into a stable state (which it did routinely even though its connection matrix was not constrained to be symmetric), and the corresponding output was noted. This process was repeated for A^1 so that a mean and standard deviation for the output could be estimated. The same was done for A^2, A^3, and so on. Then SNR_{in}, SNR_{out}, and ρ were calculated. Finally, the state of the network could either be reset (for a new learning run) or be set to what it was before the test (so that learning could continue as if uninterrupted).

3.2.5 Results of Testing

A typical example of a test run is shown in Figure 3.4. The figure shows $\rho = \rho(c,t)$ as a function of time, t, and acetylcholine concentration, c. The z-axis values for each c and t are represented by the sizes of the black rectangles—white space represents $\rho \leq 1$, and the largest rectangles represent $\rho \approx 2$. c varied from 0 (no acetylcholine) to .9 (a large concentration). The various other parameters were: $N = 10$, $M = 10$, $r = 2$, $h = .3$, $\gamma_a = 1$, $\theta_a = 1$, $\gamma_\ell = 10^{-3}$, $\theta_\ell = 1$, and $\tau = 10$. μ and σ for the memory vectors (see Section 3.2.2) were 1 and .5, respectively. For testing, $\sigma_{in} = .1$.

Notice that, for a fixed amount of acetylcholine, the model's performance rises and then falls over time. Ideally, the performance should rise and then flatten out, as further learning should not degrade performance. This is a problem with pure Hebbian learning where $\Delta W_{ij} \propto g(a_i)g(a_j)$ and where there is nothing in the learning rule that prevents the weights from growing arbitrarily large. For the model used in this chapter, the function $f(\cdot)$ could possibly be used to control growth, perhaps in conjunction with a weight-decay term, so that weight change would stop near the peak in ρ; or other features could be added to the model to achieve this. However, since the peak performance is what indicates whether or not acetylcholine has a useful effect, little effort was invested in fiddling with the model to get it to stop weight change at a convenient point. Note also that the more acetylcholine was present, the longer the learning took. This is reasonable since $\Delta W = O(1-c)$, which means that learning time scales at least like $1/(1-c)$ near $c = 1$.

Since the tests (the measurements of $\rho(c,t)$) were stochastic in nature, Figure 3.4 is not used as the final measure of the effectiveness of acetylcholine in the model.

Averages, $\mu_\rho(c,t)$, and standard deviations, $\sigma_\rho(c,t)$, of $\rho(c,t)$ were estimated by repeating the entire testing procedure many times, keeping the odor vectors and other parameters fixed, and gathering statistics. Then, a maximum average performance measure was calculated as $\rho_M(c) = \max_t \mu_\rho(c,t)$, and a standard deviation for this measure was calculated as $\sigma_M(c) = \sigma_\rho(c, t_M(c))/\sqrt{N_T}$ where $t_M(c)$ is the time value at which $\mu_\rho(c,t)$ achieves its maximum and where N_T is the number of times the testing procedure was repeated. Figure 3.5 shows the maximum average performance and its standard deviation for various values of acetylcholine. Obviously, the larger values of acetylcholine enhance performance. [3]

3.3 Conclusions

The results from the model show that suppression of connections among cells within the piriform cortex during learning enhances the performance during recall. (See the beginning of Section 3.2.3 for a definition of "recall.") Thus, acetylcholine released in the cortex during learning might enhance associative-memory function. These results might explain some of the behavioral evidence for the role of acetylcholine in memory function, and they predict that acetylcholine might be released in cortical structures preferentially during learning. Further biological experiments are necessary to test this prediction. If acetylcholine is released preferentially during learning, learning and recall can be considered as two separate modes, with acetylcholine being the switch that toggles between them.

Since reduction in acetylcholine does not affect the recall of previously stored memories but does affect the learning of new memories, the model can exhibit

[3]Note: in signal processing, the common units for measuring SNR are decibels (db). For the SNR measure used in this chapter, db $= 20 \log \text{SNR}$, and $20 \log \rho = 20 \log \text{SNR}_{out} - 20 \log \text{SNR}_{in}$. Thus, $\rho = 1.7$ (the largest value of ρ_M) corresponds to an improvement in SNR of 4.6 db, which is a significant improvement.

symptoms somewhat similar to Alzheimer's disease. Alzheimer's disease is associated with the degeneration of cells that distribute acetylcholine to the cortex. While the causal relationship between the disease and the degeneration is not known, the disease does affect memory. Both the learning of new memories and the recall of old memories are impaired. However, of the two, the learning of new memories is more strongly affected. See [43], [40], [11], and [28] for more information on Alzheimer's disease.

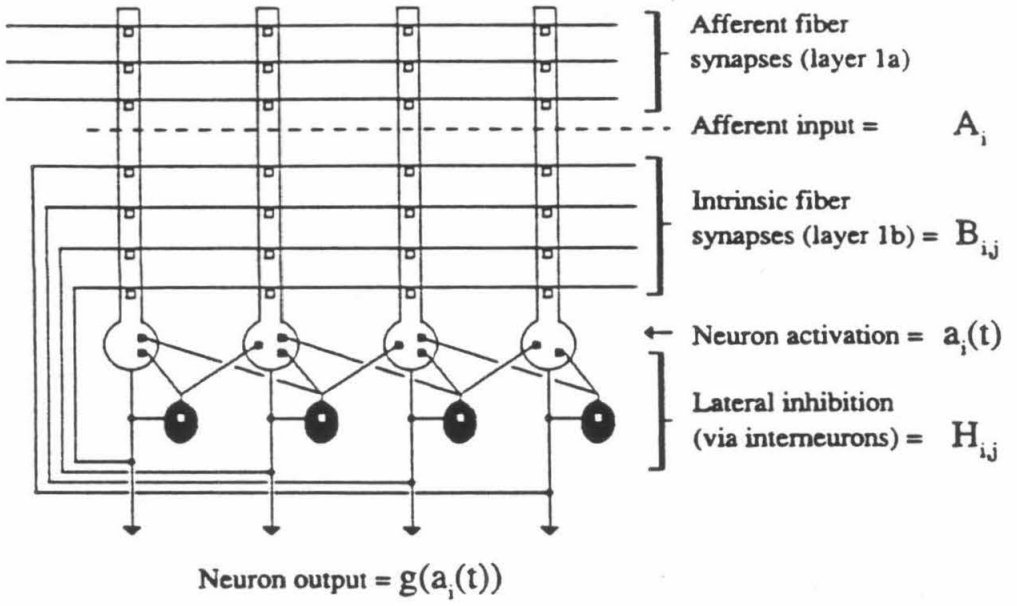

Figure 3.1: Schematic representation of piriform cortex, showing afferent input A_i (layer Ia) and intrinsic connections B_{ij} (layer Ib)

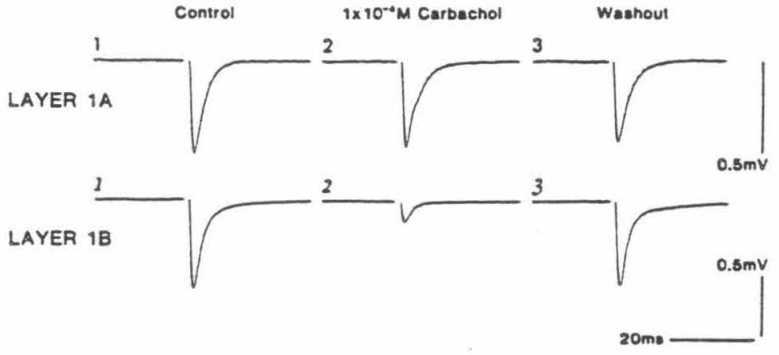

Figure 3.2: Synaptic potentials recorded in layer Ia and layer Ib before, during, and after perfusion with the acetylcholine analogue carbachol. Carbachol selectively suppresses layer Ib (intrinsic fiber) synaptic transmission.

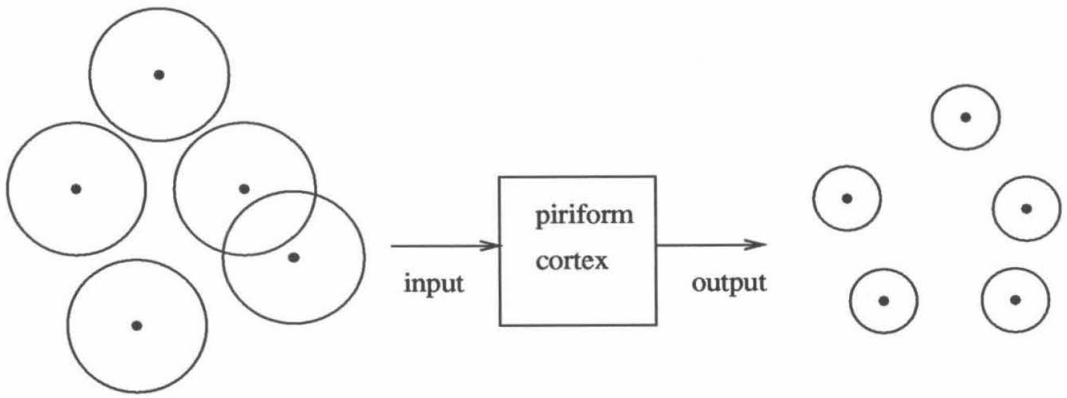

Figure 3.3: An illustration of a situation in which $\rho > 1$. The circles on the left represent noise-corrupted odor vectors. The circles on the right represent the corresponding output vectors. The centers of the circles represent the pure vectors, and the boundaries of the circles represent noise out to one standard deviation. The output vectors are more easily resolved than the input vectors.

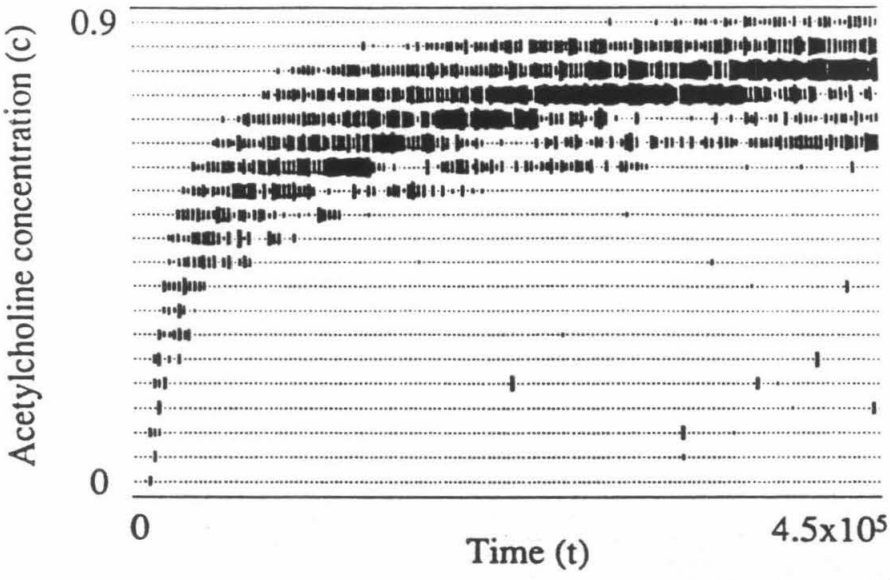

Figure 3.4: Sample test run, with time on the horizontal axis (x axis), acetylcholine level on the vertical axis (y axis), and ρ on the z axis. The z-axis value is represented by the size of a black rectangle. White space indicates $\rho \leq 1$, and the largest rectangles represent $\rho \approx 2$.

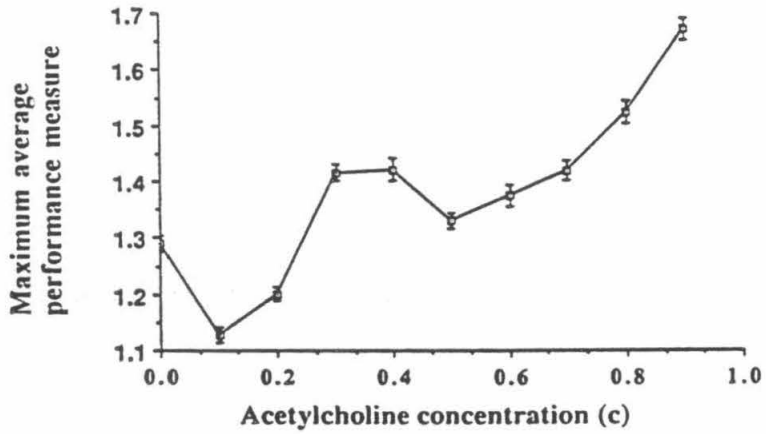

Figure 3.5: Maximum average performance vs. acetylcholine level. Acetylcholine increases the performance level attained.

Chapter 4

Low-pass Filters and Multiplicative Noise

Analog hardware (especially analog VLSI) typically has several advantages over digital hardware, such as compactness and low power consumption [39, 53]. However, analog hardware also has disadvantages, the most prominent being that it cannot easily embody as wide a range of algorithms as can digital hardware. Thus, algorithms developed for digital hardware are often prohibitively difficult to implement in analog hardware. This chapter and the two that follow discuss algorithms for signal processing and optimization that are specifically designed for easy and compact implementation in analog hardware.

This chapter deals with the filtering of signals contaminated by multiplicative noise. The study of such filtering is a fairly well-established subfield of signal processing since it has applications in a variety of areas such as control systems, image reconstruction, and adaptive systems [16, 46, 5]. Many algorithms for such filtering already exist; but current methods, while well suited to implementation on digital computers, involve complicated recursive algorithms that are ill suited to implementation in analog hardware [5, 8, 41].

Out of all the possible problems to consider (each application adds its own com-

plications), this chapter concentrates on the approximation of the expected value or expectation of signals contaminated by multiplicative noise. There are two reasons for such a choice. First, it is a very basic and common form of filtering—such approximations are used as smoothed versions of contaminated signals. Second, some adaptive systems (such as those in [16] and, more importantly for this thesis, those built using the techniques discussed in Chapter 5) require the expectations. Through a simple analysis, this chapter shows that the expectations can be approximated using only low-pass filters (which are extremely simple to implement in analog hardware), and it gives a bound on the accuracy of the approximation. Since the bound involves only parameters that are easily estimated or easily measured, it can be useful for design work.

The analysis below assumes that noise $n(t)$ contaminates a signal $s(t)$ to give $n(t)s(t)$. All are scalars evolving in time. Let the operator E denote the expectation (the average over the probability distribution of the noise). Let the operator A denote the low-pass filter. The goal, then, is to find out how closely $A[ns]$ approximates $E[ns]$ and to quantify the disparity.

4.1 Analysis

Let $x(t) = n(t)s(t)$ where $s(t)$ is the signal, $n(t)$ is the noise, and both are scalars evolving in time. Let $y(t) = A[x(t)]$ be a low-pass filtering of $x(t)$. The goal in analog hardware is to have $y(t) \approx E[n(t)]s(t)$ where E is the expectation operator for the noise. This section examines this approximation.

The following are some conditions on n, A, and s. Let $n(t)$ have a stationary (i.e., time-independent) probability density $P(n)$, and formally define the expectation as $E[\cdot] \triangleq \int(\cdot)P(n)dn$. Define the average as $\mu \triangleq E[n(t)]$, which is a constant

because $P(n)$ is stationary. n must be independent of s. Let A be a linear, time-invariant low-pass filter defined by $A[x(t)] \triangleq \int_{-\infty}^{\infty} a(\tau)x(t-\tau)d\tau$ where $a(\tau)$ is real for all τ. Let $A[\text{const}] = \text{const}$ (i.e., $\int_{-\infty}^{\infty} a(\tau)d\tau = 1$). The only restriction on s is that it is bandlimited to within the bandwidth of the filter so that $|A[s(t)] - s(t)|^2 \ll 1$.

One quantitative measure of the accuracy of the approximation is the mean-square error:

$$\text{MSE} = E\left[\left\{y(t) - E[n(t)s(t)]\right\}^2\right] = E[y^2] - 2E[y]\mu s + (\mu s)^2.$$

From the conditions in the previous paragraph, $E[y] = E[A[ns]] = A[E[n]s] = \mu A[s(t)]$, and the MSE can be rewritten as

$$\text{MSE} = V[y(t)] + \mu^2\left\{A[s(t)] - s(t)\right\}^2 \tag{4.1}$$

where $V[y] = E[y^2] - (E[y])^2$ is the variance of the output of the filter. The two important terms are the variance $V[y(t)]$ and the ability of the low-pass filter to pass the signal (as quantified by $|A[s] - s|^2$). Since the latter is small, the low-pass filter is a good approximation to the expectation operator as long as the variance of the output is also small.

The variance is

$$\begin{aligned} V[y(t)] &= E\left[\left\{y(t) - E[y(t)]\right\}^2\right] = E\left[\left\{A[n(t)s(t) - \mu s(t)]\right\}^2\right] \\ &= E\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} a(\tau)a(\tau')s(t-\tau)s(t-\tau')[n(t-\tau) - \mu] \times \right. \\ &\qquad \left. [n(t-\tau') - \mu]d\tau d\tau'\right] \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} a(\tau)a(\tau')s(t-\tau)s(t-\tau')\phi_{nn}(\tau - \tau')d\tau d\tau' \end{aligned}$$

where $\phi_{nn}(\Delta t) = E[(n(t)-\mu)(n(t+\Delta t)-\mu)]$ is the autocovariance of the noise at lag

Δt. Since n is stationary, ϕ_{nn} is independent of time, and $E[(n(t_1)-\mu)(n(t_2)-\mu)]$ depends only on $|t_1 - t_2|$.

Since most filters are more easily specified in the frequency domain, we will use the Fourier transforms of all the variables.

$$\begin{aligned}
V[y(t)] &= \frac{1}{(2\pi)^5}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} e^{i\tau(\omega_1-\omega_3+\omega_5)}e^{i\tau'(\omega_2-\omega_4-\omega_5)} \times \\
&\qquad e^{it(\omega_3+\omega_4)}A(\omega_1)A(\omega_2)S(\omega_3)S(\omega_4) \times \\
&\qquad \Phi_{nn}(\omega_5)d\tau d\tau' d\omega_1 d\omega_2\cdots d\omega_5 \\
&= \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\delta(\omega_1-\omega_3+\omega_5)\delta(\omega_2-\omega_4-\omega_5) \times \\
&\qquad e^{it(\omega_3+\omega_4)}A(\omega_1)A(\omega_2)S(\omega_3)S(\omega_4) \times \\
&\qquad \Phi_{nn}(\omega_5)d\omega_1 d\omega_2\cdots d\omega_5 \\
&= \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{it(\omega_1+\omega_2)}A(\omega_1)A(\omega_2) \times \\
&\qquad S(\omega_3)S(\omega_1+\omega_2-\omega_3)\Phi_{nn}(\omega_3-\omega_1)d\omega_1 d\omega_2 d\omega_3 \\
&= \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{it(\omega'+\omega'')}A(\omega)A(\omega'+\omega''-\omega) \times \\
&\qquad S(\omega')S(\omega'')\Phi_{nn}(\omega'-\omega)d\omega d\omega' d\omega'' \qquad (4.2)
\end{aligned}$$

Now we can develop a bound.

$$\begin{aligned}
|V[y(t)]| &\le \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}|A(\omega)A(\omega'+\omega''-\omega)| \times \\
&\qquad |S(\omega')||S(\omega'')||\Phi_{nn}(\omega'-\omega)|d\omega d\omega' d\omega'' \\
&\le \frac{\Phi_m}{(2\pi)^3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty}|A(\omega)A(\omega'+\omega''-\omega)|d\omega\right] \times \\
&\qquad |S(\omega')||S(\omega'')|d\omega' d\omega''
\end{aligned}$$

where $\Phi_m \triangleq \max_w |\Phi_{nn}(\omega)|$. The Cauchy-Schwarz inequality for integrals is

$$\left|\int a(t)b(t)dt\right|^2 \le \left[\int |a(t)|^2 dt\right]\left[\int |b(t)|^2 dt\right].$$

Thus,

$$\int_{-\infty}^{\infty} |A(\omega)A(\omega' + \omega'' - \omega)|d\omega \leq \int_{-\infty}^{\infty} |A(\omega)|^2 d\omega.$$

Define $A_m \triangleq \max_w |A(\omega)|$, and $2A_m^2 W_a \triangleq \int_{-\infty}^{\infty} |A(\omega)|^2 d\omega$. For most filters, A_m (the maximum amplitude of the transfer function) is equal to or very close to one. The definition involving W_a is merely an alternative definition of the filter's bandwidth. Now,

$$|V[y(t)]| \leq \frac{2}{(2\pi)^3} A_m^2 W_a \Phi_m \left[\int_{-\infty}^{\infty} |S(\omega)|d\omega\right]^2.$$

Define $P_s \triangleq \left[\frac{1}{2\pi}\int_{-\infty}^{\infty} |S(\omega)|d\omega\right]^2$. This is just an alternative definition of the power in the signal. Since $V[y(t)] \geq 0$, we can drop the absolute value and get

$$V[y(t)] \leq \frac{1}{\pi} A_m^2 W_a \Phi_m P_s. \tag{4.3}$$

As long as the characteristics of the noise are best known in the frequency domain, this is the final expression.

However, if the time-domain characteristics of the noise are easiest to determine (as is often the case if one has to determine the properties experimentally), a different expression is more convenient. Bound Φ_m as follows. $\Phi_m = \max_w |\Phi_{nn}(\omega)| \leq \int_{-\infty}^{\infty} |\phi_{nn}(\tau)|d\tau$. Define $2\sigma^2 T_{ac} \triangleq \int_{-\infty}^{\infty} |\phi_{nn}(\tau)|d\tau$. Since in hardware $\phi_{nn}(0)$ is finite, let $\sigma^2 = \phi_{nn}(0) =$ the variance of the noise. Then, T_{ac} is simply a measure of how quickly the autocovariance of the noise decays with increasing lag. In any case, the final expression for the bound is

$$V[y(t)] \leq \frac{2}{\pi} A_m^2 W_a \sigma^2 T_{ac} P_s. \tag{4.4}$$

Note that many of the factors in (4.4) can be easily estimated if one needs only a rough estimate of the variance. For most low-pass filters, $A_m \approx 1$. W_a can be estimated as any usual measure of the filter's bandwidth, such as the cutoff

frequency. Many processes driven by white noise have an autocovariance that dies out exponentially with increasing lag (see [45] for examples). In such cases, $\phi_{nn}(0)$ is finite and T_{ac} can be estimated as the lag at which the autocovariance of the noise has decayed by a factor of $1/e$. If $\phi_{nn}(\Delta t)$ has an oscillatory component, use the decay of an exponential bound (i.e., choose T_{ac} such that $|\phi_{nn}(T_{ac} + \Delta t)| \leq \phi_{nn}(0)/e \quad \forall \quad \Delta t \geq 0$). P_s can be estimated as follows.

$$\max_t |s(t)| = \max_t \left| \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} S(\omega) d\omega \right| \leq \frac{1}{2\pi} \int_{-\infty}^{\infty} |S(\omega)| d\omega = \sqrt{P_s}$$

Since $[\max_t |s(t)|]^2 = \max_t s^2(t)$, P_s is at least as large as $[\max_t s^2(t)]$, and this can serve as a quick estimate.

(4.3) and (4.4) become tighter bounds when the bandwidth of the signal is very small and the bandwidth of the noise is large. In this case, $\Phi_{nn}(\omega) \approx \Phi_m$ over the bandwidth of the filter and $|S(\omega)|$ is sharply peaked. Thus, the S terms in (4.2) are appreciable only when $|\omega'|$ and $|\omega''|$ are small, $A(w' + w'' - w) \approx A(-w) = A^*(w)$, and (4.2) becomes

$$\begin{aligned} V[y(t)] &\approx \frac{\Phi_m}{(2\pi)^3} \left[\int_{-\infty}^{\infty} |A(\omega)|^2 d\omega \right] \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{it(\omega' + \omega'')} S(\omega') S(\omega'') d\omega' d\omega'' \\ &= \frac{1}{\pi} A_m^2 W_a \Phi_m s^2(t). \end{aligned}$$

Because $|S(\omega)|$ is sharply peaked, $P_s \approx \max_t s^2(t)$. Because of the broad bandwidth of Φ_{nn}, $\phi_{nn}(\tau)$ is a sharply peaked, positive function, and $\Phi_m \approx \int_{-\infty}^{\infty} |\phi_{nn}(\tau)| d\tau$. Thus, under these conditions, the maximum over time of (4.2) is approximately the same as the right-hand sides of (4.3) and (4.4), and the bounds are tight.

Overall, since $P_s \geq \max_t s^2(t)$ and $A_m \approx 1$ for most low-pass filters, W_a, σ^2, and T_{ac} are the parameters to reduce in order to reduce the variance and the MSE of the approximation. However, one must be careful not to reduce W_a so far that

the filter no longer passes the signal well (i.e., that $|A[s] - s|^2$ ceases to be small, increasing the MSE as shown in (4.1)).

4.2 Conclusions

In the case of multiplicative noise, a linear, time-invariant low-pass filter is a good approximation to an expectation operator as long as the filter passes the signal well and the variance of the output is small. A bound for this variance is given by (4.4) (or by (4.3) if the frequency-domain properties of the noise are more easily discovered than the time-domain properties).

Chapter 5

Gradient Estimation in Analog Hardware

Many adaptive systems require the computation of gradients that are difficult to calculate directly in analog VLSI. As an example, consider back-propagation learning in neural networks. The back-propagation algorithm uses the gradient-descent method of optimization (see Appendix A) to minimize an objective function (the squared error between actual and desired outputs) [57, 44]. While the algorithm is straightforward to implement on a digital computer, exact computation of the objective function's gradient is very difficult in analog VLSI. Any algorithm that requires the computation of gradients—this includes many adaptive systems and optimizers—is susceptible to the same problem.

However, there are techniques to *estimate* the gradients of objective functions, techniques that are easy to implement in analog VLSI. This chapter discusses two such techniques and the results of implementing them in analog hardware.

[0]A portion of the research done for this chapter was done in collaboration with Douglas A. Kerns.

5.1 Analysis

The following three sections describe the two methods, indicate why the methods should work, and give conditions under which one method is superior to the other.

5.1.1 Method 1

The first method is an extension of work described in [13] and [16]. Dembo and Kailath recently applied this method to learning in neural networks [12].

Assume that we would like to find the gradient of $f(x)$, where x is a scalar and f is a scalar-valued function. (Section 5.3 discusses the the case where x is a vector.) Let $x = s + n$, where $s(t)$ is the point at which we would like to evaluate the gradient (this point may vary in time) and where $n(t)$ is noise (a stochastic scalar evolving in time). If $|n| \ll 1$, and assuming that f and its derviatives exist, f can be expanded in a Taylor series:

$$f(x) = f(s) + f'(s)n + \frac{1}{2}f''(s)n^2 + O(n^3).$$

Now, compute $E[nf(x)]$ where E is the expectation operator for the noise. Assume that n has a stationary (i.e., time-independent) probability density $P(n)$. Then, E can be formally defined as $E[\cdot] \triangleq \int(\cdot)P(n)dn$. Assume that s and n are independent and that $E[n] = E[n^3] = 0$. Let $\sigma^2 = E[n^2]$, which is a constant since $P(n)$ is stationary. Then,

$$nf(x) = nf(s) + n^2 f'(s) + \frac{1}{2}n^3 f''(s) + O(n^4), \tag{5.1}$$

and

$$E[nf(x)] = \sigma^2 f'(s) + O(E[n^4]). \tag{5.2}$$

Thus, $E[nf(x)]$ is approximately proportional to the gradient as long as $|n|$ is small.

However, there is a problem. While most of (5.2) can be computed in analog hardware (a noise generator can generate n, an adder can compute $x = s + n$, and a multiplier can compute $nf(x)$), there is no way to make an expectation operator exactly. That would require either an ensemble average or, if the noise is ergodic, a time average with an infinite time interval [18]. Nevertheless, Chapter 4 shows that it is possible to approximate E by a linear, time-invariant low-pass filter; and low-pass filters are easy to build in analog hardware.

Denote the low-pass operator by A. The object now is to approximate $\sigma^2 f'(s)$ by $A[nf(x)]$, instead of by $E[nf(x)]$ as in (5.2). Define $y_0 \triangleq \sigma^2 f'(s)$ and $y \triangleq A[nf(x)]$. If $|y - y_0| \ll |y_0|$, the approximation is good. A quantitative measure of this is the signal-to-noise ratio: SNR $= y_0^2 / E[(y - y_0)^2]$. The denominator is the mean square error, MSE. It can be expanded as

$$\text{MSE} = E[(y - y_0)^2] = V[y] + (y_0 - E[y])^2$$

where $V[y] = E[y^2] - (E[y])^2$ is the variance of y. The MSE is computed as follows.

First, find $E[y]$. Since A is linear,

$$E[y] = E[A[nf(x)]] = A[E[nf(x)]] = \sigma^2 A[f'(s)] + O(E[n^4]).$$

Assume that the signal s is bandlimited so that $|A[f'(s)] - f'(s)|$ is small; i.e., $A[f'(s)] = f'(s) + \Delta_s$ where $|\Delta_s| \ll 1$. Then,

$$E[y] = y_0 + O(E[n^4]) + \sigma^2 \Delta_s.$$

Now, find $V[y]$. In (5.1), $nf(x)$ is expanded into terms that factor into a noise component and a signal component. Thus, consider the case of filtering a signal contaminated by multiplicative noise: $A[mr]$ where $m(t)$ is noise with a stationary

probability distribution, $r(t)$ is an independent signal, and A is a linear, time-invariant low-pass filter. From (4.4) in Chapter 4,

$$V[A[mr]] \leq \frac{2}{\pi} A_m^2 W_a V[m] T_{ac} P_s \tag{5.3}$$

where A_m is the maximum amplitude of the filter's transfer function, W_a is the bandwidth of the filter, T_{ac} is the autocovariance time for the noise, and P_s is a particular measure of the power in $r(t)$. Assuming that $A_m \approx 1$, which is true for most simple low-pass filters, $V[A[mr]] = O(W_a V[m] T_{ac} P_s)$. From (5.1), it is clear that the term that will contribute the most variance (lowest order in n) is the first. Setting $m(t) = n(t)$ and $r(t) = f(s(t))$, $V[A[nf(s)]] = O(W_a \sigma^2 T_{ac} P_s)$ where T_{ac} is the autocovariance time for n and where P_s is a measure of the power in $f(s)$. Thus, to lowest order in n,

$$V[y] = O(V[A[nf(s)]]) = O(W_a \sigma^2 T_{ac} P_s).$$

Putting all of this together,

$$\text{SNR} = \frac{\sigma^4 [f'(s)]^2}{O(W_a \sigma^2 T_{ac} P_s) + [O(E[n^4]) + \sigma^2 \Delta_s]^2}. \tag{5.4}$$

Before continuing, it will be useful to develop the following notation. Let x be called the "order parameter" (i.e., we are interested in expansions where $|x|$ is small), and let y be another adjustable parameter that can be made arbitrarily small. If $z = O(yx^p)$ and $p > 0$, denote the situation by $z < O(y)$. Conversely, if $p < 0$, denote the situation by $z > O(y)$. Also, it will be useful to express final results in terms of only one adjustable parameter. For example, for $z = O(xy)$, since y is arbitrarily adjustable, let y be a convenient function of x, say, $y = x^q$ where q is a constant. In that case, $z = O(x^{(1+q)})$.

Now back to the problem at hand. Using σ as the order parameter in (5.4), the SNR should be $> O(1)$ so that it can be increased to acceptable levels by

decreasing the amplitude of the noise. To achieve this, the denominator in (5.4) should be $< O(\sigma^4)$ because the numerator is $O(\sigma^4)$. Adjust Δ_s so that it is $< O(1)$. (It is already small, so its role in the SNR should be minimal anyway.) Then, since $E[n^4] = O(\sigma^4)$, the second term in the denominator is $< O(\sigma^4)$. As $P_s \geq \max_t[f(s(t))]^2$ (see Chapter 4), P_s is not in general small. Thus, we must have $W_a T_{ac} < O(\sigma^2)$. If all of these conditions are satisfied and if the noise amplitude is made small enough, $A[nf(x)] \approx \sigma^2 f'(s)$.

5.1.2 Method 2

Method 2 is quite similar to method 1. However, instead of computing $E[nf(x)]$ in order to find the direction of the gradient, one uses $E[\dot{n}\dot{f}(x)]$, where the dots represent differentiation in time. It works as follows. Using the same symbol definitions as in Section 5.1.1 and again assuming that $|n| \ll 1$ (although $|\dot{n}|$ will not in general be small),

$$\dot{f}(x) = (\dot{n} + \dot{s})f'(x) = (\dot{n} + \dot{s})[f'(s) + nf''(s) + O(n^2)],$$

and

$$\dot{n}\dot{f}(x) = \dot{n}^2 f'(s) + \dot{n}\dot{s}f'(s) + \dot{n}^2 n f''(s) + \dot{n}n\dot{s}f''(s) + O(n^2). \tag{5.5}$$

Since n is independent of s, since $E[\dot{n}] = \frac{d}{dt}E[n] = 0$, and since $E[\dot{n}n] = \frac{1}{2}\frac{d}{dt}E[n^2] = 0$,

$$E[\dot{n}\dot{f}(x)] = \nu^2 f'(s) + O(E[n\dot{n}^2])$$

where $\nu^2 = E[\dot{n}^2]$ = a constant (because the noise has a stationary probability distribution). Under these conditions, $E[\dot{n}\dot{f}(x)]$ is approximately proportional to the gradient as long as $|n|$ is small.

Again, let's use a low-pass filter A to approximate E. Define $y \triangleq A[\dot{n}\dot{f}(x)]$ and

$y_0 \triangleq \nu^2 f'(s)$.

$$E[y] = E[A[\dot{n}\dot{f}(s)]] = A[E[\dot{n}\dot{f}(x)]] = \nu^2 f'(s) + O(E[n\dot{n}^2]) + \nu^2 \Delta_s$$

where $\Delta_s = A[f'(s)] - f'(s)$. Assume that s is slowly varying so that $|\dot{s}|$ is small and so that $|\Delta_s| \ll 1$. From (5.5), again it is clear that the first term contributes the most variance. (All other terms are higher order in n except the second, and the second is smaller because $|\dot{s}|$ is small.) From (5.3), $V[A[\dot{n}^2 f'(s)]] = O(W_a \eta^2 T_{ac} P_s)$ where W_a is the bandwidth of the filter, T_{ac} is the autocovariance time of $\dot{n}$, P_s is a measure of the power in $f'(s)$, and $\eta^2 = E[\dot{n}^4]$. Thus, to lowest order in n,

$$V[y] = O(W_a \eta^2 T_{ac} P_s).$$

Altogether,

$$\text{SNR} = \frac{\nu^4 [f'(s)]^2}{O(W_a \eta^2 T_{ac} P_s) + [O(E[n\dot{n}^2]) + \nu^2 \Delta_s]^2}. \tag{5.6}$$

The numerator is $O(1)$—remember that ν is not small. To get SNR $> O(1)$, the denominator has to be $< O(1)$. (See the end of Section 5.1.1 for an explanation of inequalities involving order parameters.) Adjust Δ_s so that it is $< O(1)$. (It is already small, so its role in the SNR should be minimal anyway.) Then, since $E[n\dot{n}^2] = O(\sigma)$, the second term in the denominator is $< O(1)$. As $P_s \geq \max_t [f'(s(t))]^2$ (see Chapter 4), P_s is not in general small, so we must have $W_a T_{ac} < O(1)$. If all of these conditions are satisfied and if the noise amplitude is made small enough, $A[\dot{n}\dot{f}(x)] \approx \nu^2 f'(s)$.

5.1.3 Comparing the Two Methods

From the previous two sections, note that one needs $W_a T_{ac} < O(1)$ for method 2 and $W_a T_{ac} < O(\sigma^2)$ for method 1. Obviously, as $\sigma \to 0$, $W_a T_{ac}$ must be much smaller for method 1 than for method 2. Using the following approximations, one

can examine what happens to the SNR's for the two methods when $W_a T_{ac}$ is small enough to satisfy the conditions for both methods yet is the same for both. We have already assumed that s varies slowly enough so that $|\Delta_s|$ is small. Assuming further that the bandwidth of s is very small and that the bandwidth of n is very large, (5.3) becomes a tight bound (as explained in Chapter 4). Thus, for method 1, to lowest order in σ,

$$\mathrm{SNR}_1 \approx \frac{\sigma^4 [f'(s)]^2}{\frac{2}{\pi} A_m^2 W_a \sigma^2 T_{ac} P_{s1}} = \frac{\pi \sigma^2 [f'(s)]^2}{2 A_m^2 W_a T_{ac} P_{s1}}.$$

Similarly, for method 2,

$$\mathrm{SNR}_2 \approx \frac{\pi \nu^4 [f'(s)]^2}{2 A_m^2 \eta^2 W_a T_{ac} P_{s2}}.$$

Since the signal bandwidth is assumed to be very small, $P_{s1} \approx [f(s)]^2$, and $P_{s2} \approx [f'(s)]^2$. The relative performance at the same value of s, W_a, and T_{ac} is

$$\frac{\mathrm{SNR}_2}{\mathrm{SNR}_1} \approx \left(\frac{\nu^2}{\eta\sigma}\right)^2 \frac{P_{s1}}{P_{s2}} \approx \left(\frac{\nu^2}{\eta\sigma} \frac{f(s)}{f'(s)}\right)^2. \tag{5.7}$$

Thus, method 1 should be superior near $f/f' = 0$, and method 2 should be superior when $1/\sigma$ is large enough to overwhelm the other terms (i.e., when f is far enough away from zero). As σ is reduced, method 1 will outperform method 2 in a smaller and smaller region around $f/f' = 0$.

5.2 Experiments

The following sections describe the results of implementing both methods in hardware.

5.2.1 The Definition of "Small"

In hardware implementations of the methods, the values of f, n, etc. are not dimensionless. This brings up the problem of deciding which values of $|n|$ (and

hence σ) are small: $\sigma = 0.1$ V is a number smaller than 1 when measured in volts but much larger than 1 when measured in millivolts. The key to the correct interpretation comes from the Taylor expansions—of $f(s+n)$ for method 1 and of $f'(s+n)$ for method 2. Regardless of units, $|n|$ is small if the first-order and higher-order terms in the expansions are small compared to the zeroth-order terms (at least away from the regions where the zeroth-order terms are zero):

$$\left|\frac{f(s+n)-f(s)}{f(s)}\right| \ll 1$$

for method 1 and

$$\left|\frac{f'(s+n)-f'(s)}{f'(s)}\right| \ll 1$$

for method 2.

5.2.2 Method 1

Figure 5.1 shows a block diagram of the implementation of $A[Bnf(x)]$ where B is an adjustable gain parameter described below. This system was built on a breadboard using standard discrete components. Specifically, LM324 op-amps were used for summing, gain, and low-pass operations, and an MC1495L multiplier was used for multiplication. The operator A was a single-pole low-pass filter. The function f was built of several CA3080 transconductance amplifiers, as shown in Figure 5.2. The noise $n(t)$ was generated by a custom analog-VLSI chip built to amplify the intrinsic noise in MOS devices [32].

To test the circuit, a slow triangular waveform was applied to the $s(t)$ input. Thus, s swept over the range of interest. The output of the system was examined as a function of s rather than as a function of t since this allowed better visualization of the gradients involved.

Figure 5.3 shows the best result we were able to obtain by varying the noise amplitude and the gain parameter B. The gray plot shows a single sweep of the output of the system vs. s. Note that it is still noisy and not very symmetrical. The black plot is an average, performed by the digital oscilloscope, of many sweeps. One might think that the black plot could be obtained merely by decreasing the bandwidth, W_a, of the low-pass filter. Unfortunately, one can reduce W_a only so far until it starts to interfere with the transmission of $f'(s)$. In other words, for a given $s(t)$, which varies in time, W_a can be made only so small before Δ_s becomes large enough to spoil the SNR (see (5.4)).

It was also difficult to keep the multiplier from clipping. The multiplier clips if $|Bn|$ or $|f(x)|$ or both are too large. On the other hand, $|Bn|$ and $|f(x)|$ cannot be too small, or the multiplier gives no appreciable output. B was used to balance these two effects.

5.2.3 Method 2 and Comparison

Figure 5.4 shows a block diagram of the implementation of $A[B\dot{n}\dot{f}(x)]$ where B is an adjustable gain parameter as discussed in Section 5.2.2. The construction of this circuit was identical to that used for method 1 except for the insertion of differentiators. The method of testing was likewise identical.

Figure 5.5 shows the results. Again, the gray plot is a single sweep, while the black plot is an average of many sweeps. The results are clearly much better than those obtained from method 1 (see Figure 5.3).

Also, compared to method 1, the parameters had a much broader range of acceptable settings. The insertion of the time-derivative operators in the paths leading to the multiplier greatly reduces the scaling and clipping problems encountered during the implementation of method 1. This is because both inputs

to the multiplier are approximately centered around zero. Also, the differentiation reduces any low-frequency components present in the noise, making the low-pass filter, A, all the more effective without having to reduce its bandwidth. Thus, one can more easily avoid the problem of making W_a so small that $|\Delta_s|$ ceases to be small, as discussed in Section 5.2.2.

There can still be problems with method 2, however. If $|\dot{s}|$ is large, P_s and $|\Delta_s|$ can become large, spoiling the SNR (see (5.6)). Also, one might still encounter clipping under two conditions. First, if $|\dot{s}|$ is too large, $|\dot{f}|$ can become large, and the multiplier might clip. Second, if the noise spectrum extends to very high frequencies, $|B\dot{n}|$ can be large, and again the multiplier might clip. However, for practical noise-generation circuits—especially in low-power analog VLSI—limitation of the noise bandwidth is essentially free, as the dominant noise source is usually flicker noise with a 1/(frequency) spectrum.

f, W_a, T_{ac}, and σ were approximately the same for the implementations of methods 1 and 2. Also, in each case, s was slowly varying and had a bandwidth of about 1 Hz, while n had a bandwidth of about 1 kHz. Thus, (5.7) should approximately hold. ((5.7) remains unchanged for values of $B \neq 1$, as B cancels in the SNR expressions. Using B is equivalent to scaling f, and so it appears to the same power in P_s, the denominator, in f^2, the numerator for method 1, and in $(f')^2$, the numerator for method 2.) From the bottom traces in Figures 5.3 and 5.5, one can find values for $f(s)$ and $f'(s)$ and can estimate $\mathrm{SNR}_1(s)$ and $\mathrm{SNR}_2(s)$ as the mean square of the outputs divided by the variance of the outputs at specific values of s. As can be seen from the figures, the only place where method 1 matches or outperforms method 2 is near $f(s) = 0$. At most other places (i.e., away from $f(s) = 0$), method 2 significantly outperforms method 1. This agrees with the

discussion in Section 5.1.3.

5.3 Generalization to Higher Dimensionality

Both methods can be generalized to the case where x is a vector. There is one added complication, however. Both methods rely on sending perturbation information through a single channel (namely, the value of $f(x)$, which is broadcast down a single wire). In terms of information theory, if there are N adjustable parameters (i.e., if x is an N-dimensional vector), if each perturbation of a parameter transmits M bits of information, and if the channel is bandlimited so that it can transmit at most L bits of information per second, one expects to wait NM/L seconds in order to extract all the information for estimating the gradient. All hardware will have such bandwidth limitations. Thus, according to information theory, the settling times for the methods (the time it takes to get gradient estimates) should be at best proportional to N. The following two sections describe the generalization to higher dimensionality.

Note: the following two sections involve many approximations and rely on scaling relationships. The symbol " $\tilde{\propto}$ " means "approximately proportional to."

5.3.1 Method 1

Assume that x, n, and s are now N-dimensional vectors. Assume that n_i is independent of n_j for all $i \neq j$ and that all the noise sources n_i have the same statistics, each one satisfying the conditions given for the 1-dimensional case. Now, for method 1,

$$f(x) = f(s) + \sum_i n_i \frac{\partial f}{\partial x_i}(s) + \frac{1}{2} \sum_{i,j} n_i n_j \frac{\partial^2 f}{\partial x_i \partial x_j}(s) + O(\|n\|^3),$$

$$n_k f(x) = n_k f(x) + \sum_i n_k n_i \frac{\partial f}{\partial x_i} + \frac{1}{2}\sum_{i,j} n_k n_i n_j \frac{\partial^2 f}{\partial x_i \partial x_j} + O(\|n\|^4),$$

$$E[n_k f(x)] = 0 + \sum_i \sigma^2 \delta_{i,k} \frac{\partial f}{\partial x_i} + 0 + O(E[\|n\|^4]) = \sigma^2 \frac{\partial f}{\partial x_k}(s) + O(E[\|n\|^4]),$$

and thus

$$E[nf(x)] = \sigma^2 \nabla f(s) + O(E[\|n\|^4]).$$

Similarly to the 1-dimensional case, define $y^0 \triangleq \sigma^2 \nabla f(s)$ and $y \triangleq A[nf(x)]$. Now, MSE $= E[\|y - y^0\|^2]$, and SNR $= \|y^0\|^2/\text{MSE}$. One can rewrite the MSE as

$$\begin{aligned} \text{MSE} &= E[\|y\|^2 + \|y^0\|^2 - 2y^0 \cdot y] = \|y^0 - E[y]\|^2 - \|E[y]\|^2 + E[\|y\|^2] \\ &= \sum_i \{E[y_i^2] - (E[y_i])^2\} + \|y^0 - E[y]\|^2 = \sum_i \{V[y_i] + (y_i^0 - E[y_i])^2\}. \end{aligned}$$

Thus, the MSE scales like N times the MSE of the 1-dimensional case, and

$$\text{SNR} = \frac{\sigma^4 \|\nabla f(s)\|^2}{O(N W_a T_{ac} \sigma^2 P_s) + \sum_i \{O(E[n_i^4]) + \sigma^2 \Delta_{s,i}\}^2} \tag{5.8}$$

where $\Delta_s = A[\nabla f(s)] - \nabla f(s)$.

(5.8) contains N, but it does not yet explicitly show all of its dependence on N. We now have to take into account the finite bandwidth of f. Call this bandwidth W_f and define it by the condition $\dot{f}(x) \leq W_f$. This is just a convenient way to define the bandwidth for what follows. Since $\dot{f} = \nabla f \cdot (\dot{n} + \dot{s})$, this condition implies that $\dot{n}_i$ must typically be bounded. This is true in hardware. Nevertheless, if $\dot{n}_i$ were unbounded but had a finite variance, one could still define a bandwidth condition in terms of variances.

Since W_a is the limiting bandwidth of the gradient estimation process, one should look at how W_a changes with N while all other aspects are held constant. $1/W_a$ is the characteristic or settling time of the estimation process.

Assume that f and $\|\nabla f\|^2$ remain constant as N changes. This is reasonable in that f and $\|\nabla f\|^2$ will remain within certain limits when implemented in hardware

and will not grow to be arbitrarily large. Also, assume that the bandwidth of s is very small and that $|\Delta_s|$ is small so that various simplifying approximations hold as in Section 5.1.3.

Then, to lowest order in σ,

$$\text{SNR} \approx \frac{\sigma^4 \|\nabla f(s)\|^2}{\frac{2}{\pi} A_m^2 N W_a T_{ac} \sigma^2 P_s} \approx \frac{\pi \sigma^2 \|\nabla f(s)\|^2}{2 A_m^2 W_a T_{ac} f^2(s) N}.$$

Now, take into account the bandlimit on f.

$$\dot{f}(x) = \dot{x} \cdot \nabla f(x) \approx \dot{n} \cdot \nabla f(s)$$

Since σ scales the size of n_i and thus also of $\dot{n}_i$,

$$\|\dot{n}\|^2 = \sum_{i=1}^{N} \dot{n}_i^2 \tilde{\propto} N\sigma^2,$$

and $\dot{f}(x) \tilde{\propto} \sigma\sqrt{N}$. At the bandlimit, $\dot{f}(x) = W_f \tilde{\propto} \sigma\sqrt{N}$. Thus, to stay within the bandwidth, one must have $\sigma \tilde{\propto} 1/\sqrt{N}$. Altogether,

$$\text{SNR} \tilde{\propto} \frac{\sigma^2}{N W_a} \tilde{\propto} \frac{1}{N^2 W_a},$$

and to keep the SNR constant, one must have $W_a \tilde{\propto} 1/N^2$. The settling time, $1/W_a$, is thus $\tilde{\propto} N^2$, which is not optimal.

5.3.2 Method 2

Following steps similar to those in the previous section, one finds for method 2 that

$$\text{SNR} = \frac{\nu^4 \|\nabla f(s)\|^2}{\sum_{i=1}^{N} \left[O(W_a T_{ac} \eta^2 P_{s,i}) + \{E[\dot{n}_i^2 n_i] + \nu^2 \Delta_{s,i}\}^2 \right]} \tag{5.9}$$

where, again, the noise components all have the same statistics. $\Delta_s = A[\nabla f(s)] - \nabla f(s)$, and $P_{s,i}$ is the power in $\frac{\partial f}{\partial x_i}(s)$.

When the bandwidth of s is very small, $P_{s,i} \approx (\partial f/\partial x_i)^2$, and assuming that $|\Delta_s|$ is small,

$$\text{SNR} \approx \frac{\pi\nu^4\|\nabla f(s)\|^2}{2A_m^2 W_a T_{ac}\eta^2\|\nabla f(s)\|^2} = \frac{\pi\nu^4}{2A_m^2\eta^2 W_a T_{ac}}.$$

Here, the bandwidth condition on $\dot{f}$ gives

$$W_f = \dot{f} \approx \dot{n}\cdot\nabla f(s) \tilde{\propto} \nu\sqrt{N},$$

and one must have $\nu \tilde{\propto} 1/\sqrt{N}$. To find an estimate of how $\eta^2 = E[\dot{n}_i^4]$ scales, consider the following.

$$\|\dot{n}\|^4 = (\sum_i \dot{n}_i^2)^2 = \sum_i \dot{n}_i^4 + \sum_{i\neq j} \dot{n}_i^2\dot{n}_j^2$$

$$\sum_i \dot{n}_i^4 = (\sum_i \dot{n}_i^2)^2 - \sum_{i\neq j} \dot{n}_i^2\dot{n}_j^2 \tag{5.10}$$

The left-hand side of (5.10) is $\tilde{\propto} N\eta^2$, the first term on the right-hand side is $\tilde{\propto} N^2\nu^4$, and the second term on the right-hand side is $\tilde{\propto} N^2\nu^4$. Since $\nu \tilde{\propto} 1/\sqrt{N}$, the right-hand side is $\tilde{\propto} 1$, and thus so must the left-hand side be; i.e., $\eta \tilde{\propto} 1/\sqrt{N}$. Altogether,

$$\text{SNR} \tilde{\propto} \frac{1}{NW_a},$$

and for a constant SNR, the settling time scales approximately linearly with N, which is the best possible scaling relationship.

5.4 Conclusions

The experimental results show that method 2 is more robust and easier to implement than method 1. Both of the methods use only a noise generator, an adder, a multiplier, and a low-pass filter. Method 2 also requires two differentiators. Method 1 approximates $\sigma^2 f'(s)$ by computing $A[nf(x)]$ where $\sigma^2 = E[n^2]$, n is the

noise, $x = s + n$, s is the point where the gradient is to be evaluated, and A is the low-pass filter. Method 2 approximates $\nu^2 f'(s)$ by computing $A[\dot{n}\dot{f}(x)]$ where $\nu^2 = E[\dot{n}^2]$. The final expressions for the accuracy of the two approximations are given by (5.4) and (5.6). The analysis requires the following assumptions: f and its derivatives exist; the noise and the point of interest s are independent; the noise has a stationary, zero-mean probability distribution; σ^2 is small; the low-pass filter used to approximate the expectation operator is linear and time-invariant; and s is slowly varying. Both methods can be generalized to the case where f has more than one adjustable parameter. (5.8) and (5.9) are the multidimensional analogues of (5.4) and (5.6). In this case, assuming that all other aspects are constant, the settling or characteristic time for method 1 is approximately proportional to N^2, while for method 2 it is approximately proportional to N.

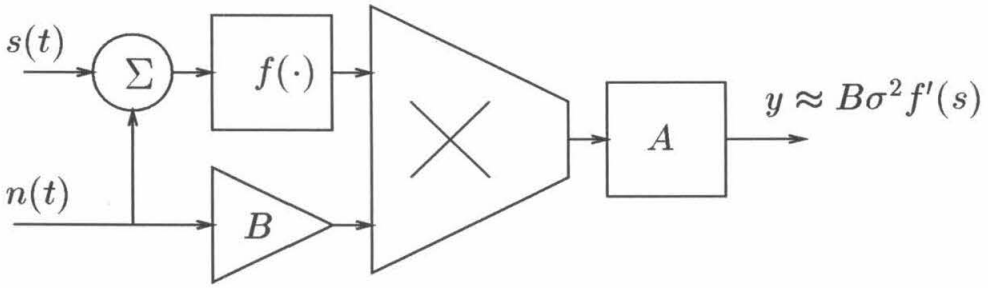

Figure 5.1: A gradient estimator based on method 1. The system computes $A[Bnf(s+n)]$.

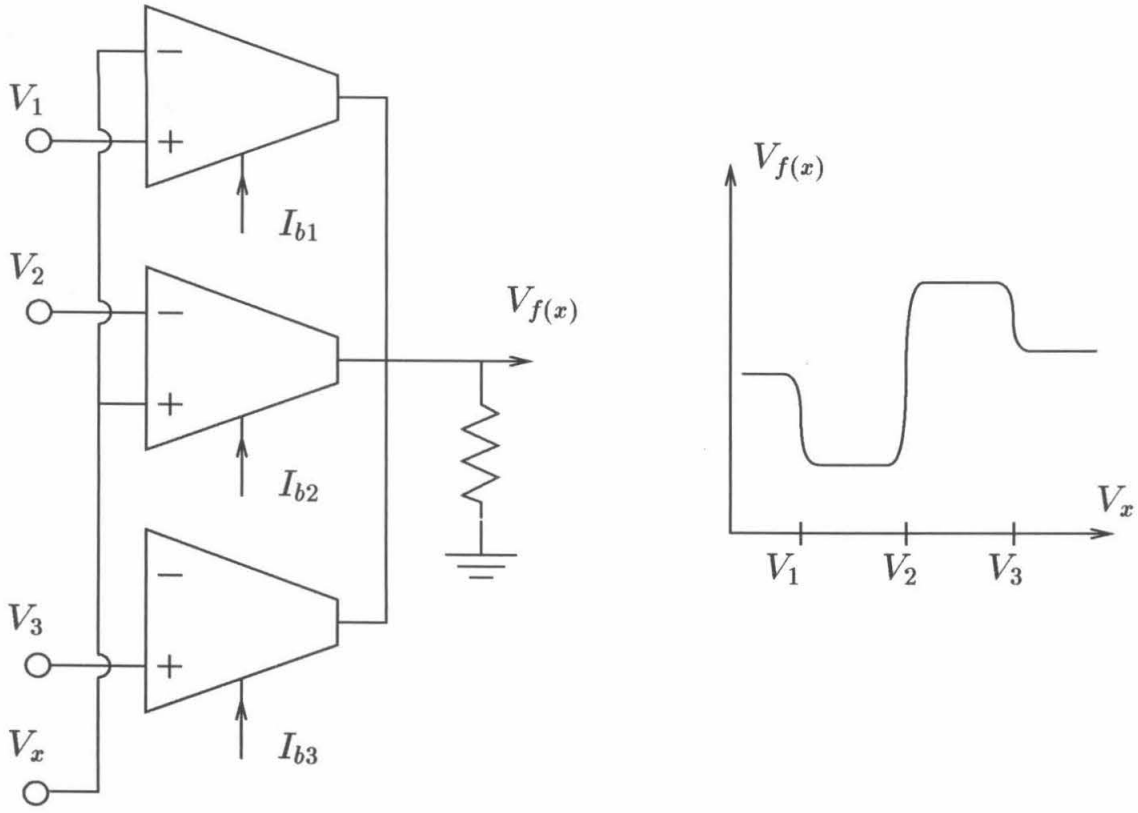

Figure 5.2: Breadboard construction of the function f.

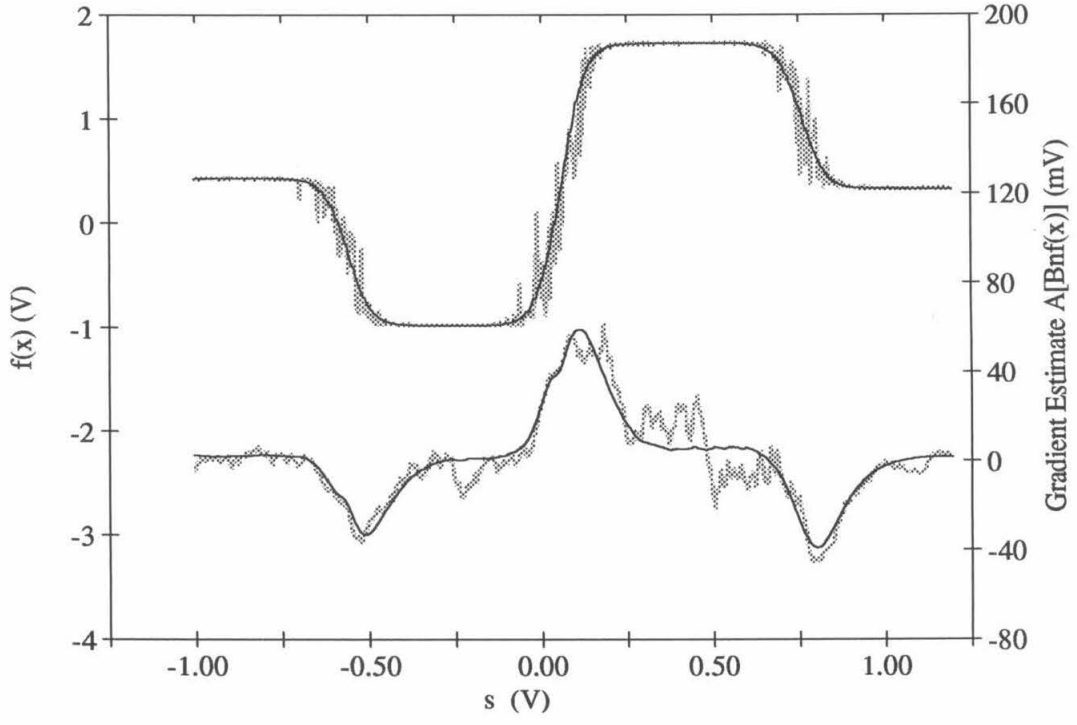

Figure 5.3: Experimental results from method 1. Digital-oscilloscope traces show $f(x)$ (top) and the gradient estimate $A[Bnf(x)]$ (the output of the system, shown at bottom) as functions of the applied signal $s(t)$. The gray line is a single sweep, and the solid line is an average of many sweeps.

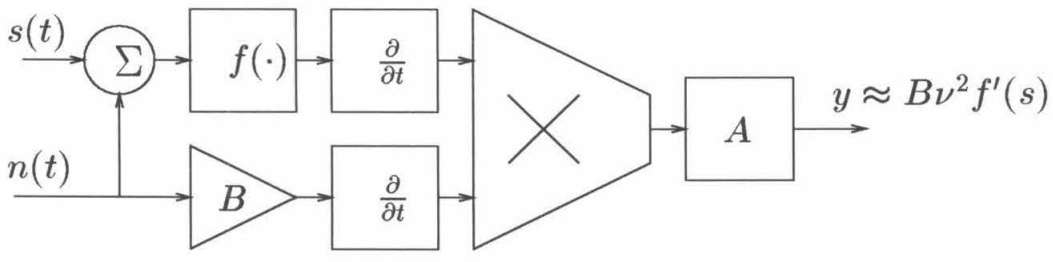

Figure 5.4: A gradient estimator based on method 2. The system computes $A[B\dot{n}\dot{f}(s+n)]$.

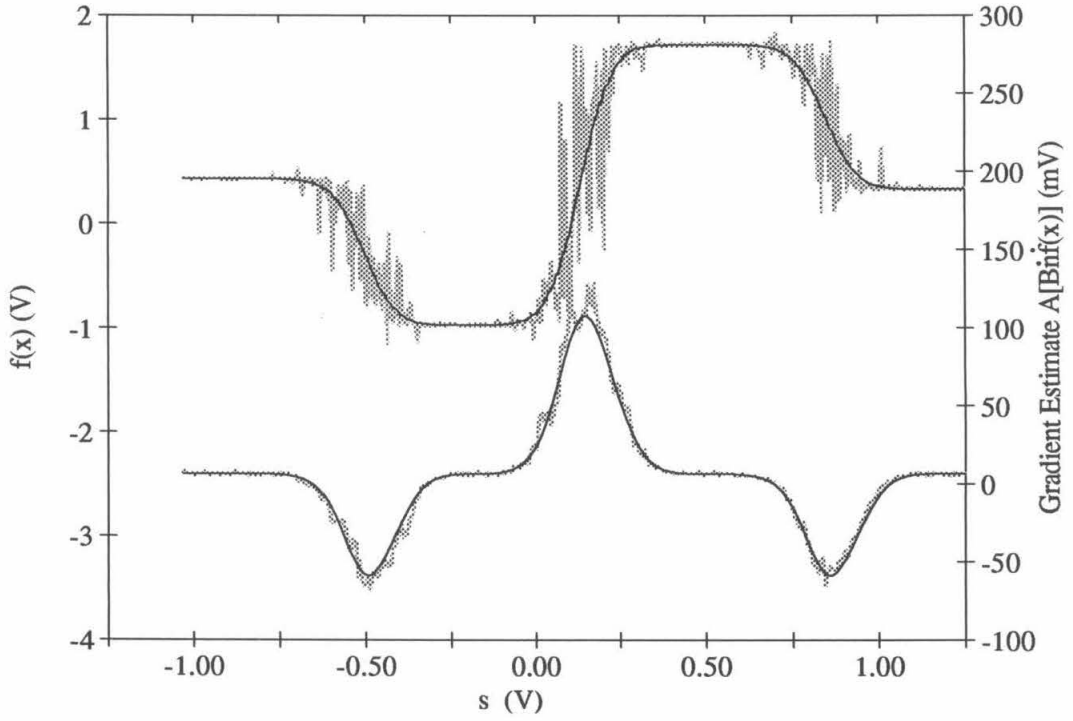

Figure 5.5: Experimental results from method 2. Digital-oscilloscope traces show $f(x)$ (top) and the gradient estimate $A[B\dot{n}\dot{f}(x)]$ (the output of the system, shown at bottom) as functions of the applied signal $s(t)$. The gray line is single sweep, and the solid line is an average of many sweeps.

Chapter 6

Optimization in Analog Hardware

This chapter discusses an algorithm for optimization that is specifically designed for ease of implementation in analog hardware. The basic idea behind the algorithm is as follows. Assume that you are given a black box that has one output and one adjustable knob, and you are told to minimize the output of the box. Assume further that you are not given gradient information (since exact computation of the gradient might be very difficult in analog hardware). One intuitive solution is simply to adjust the knob and to note the change in the output. If the output goes up, turn the knob in the opposite direction; if the output goes down, keep turning the knob in the same direction. The quantitative details are more involved, but the basic idea is to perturb the parameters, to note the correlations between the changes in the parameters and the changes in the output, and to use that correlation to indicate the direction the parameters should be moved.

The use of perturbation and correlation in optimization has an interesting history. In the 1950's, Draper and Li studied the optimization of adaptive control systems with one adjustable parameter [13]. They used sine waves and triangle waves for perturbation. Eykhoff and Smith extended the method to systems with

two parameters, adapted the method to other problems in control theory, and showed how noise can be used as the perturbation [16]. In [47], Stromer gives a bibliography of other work done in this area until 1959. Although it was concerned mainly with control systems, this work pointed the way to general-purpose optimization algorithms for analog hardware. However, by the mid-1960's, much of the research had stopped. According to Dembo and Kailath, this was due to concerns that perturbation could cause a loss of stability in control systems [12]. Perhaps another reason is that digital computers became more prevalent and allowed the implementation of more efficient algorithms. In the 1970's, Harth, Tzanakou, and Michalak independently developed a general-purpose, discrete-time optimization algorithm, called "ALOPEX,"[1] which uses perturbation and correlation [22, 52]. Working to discover the strongest stimulus for a neuron in a frog's visual cortex, they put a frog in front of a video monitor, put a probe into a neuron in the frog's brain, and used the ALOPEX algorithm to adjust the pixels on the video monitor so that the output of the neuron was maximized. ALOPEX has many desirable features for analog implementation; for example, it requires only multiplication and addition operations, and it does not require the explicit computation of gradients. Also, it uses noise as its source of perturbation, which might be helpful for avoiding phase locking of the sources (as might happen more readily with periodic sources in analog VLSI). However, ALOPEX is a discrete-time algorithm, and [22] and [52] do not discuss hardware time delays or give conditions under which the algorithm converges.

This chapter explains an algorithm that resembles a continuous-time version of ALOPEX—thus the algorithm is called "CALOPEX." This chapter gives an analysis of CALOPEX, taking into account time delays in the hardware, and gives

[1]This is an acronym for "ALgorithm Of Pattern EXtraction."

conditions under which the expected output of the system decreases. Section 6.1 gives a mathematical description of CALOPEX. Section 6.2 shows that CALOPEX approximately minimizes the expected output of the system. Section 6.3 explains how the convergence time scales with the number of adjustable parameters. Section 6.4 summarizes the conditions for convergence and discusses the suitability of CALOPEX for implementation in analog VLSI.

6.1 The Method

Let $f(x)$ be a scalar-valued function to be minimized. x is a vector whose components represent adjustable parameters. Let $x(t) = s(t) + \alpha n(t)$, where n is a stochastic vector, α is a small constant, and s represents the part of x adjusted by CALOPEX. Under certain conditions, which will be described below, the following dynamical system approximately minimizes the expectation of f.

$$\dot{s}(t + \tau_1) = -\alpha\eta\dot{f}(t - \tau_2)\dot{n}(t). \tag{6.1}$$

τ_1 and τ_2 represent time delays in the system (due perhaps to limitations in hardware implementation), η is a positive constant (which adjusts adaption speed), and $f(t)$ is a short notation for $f(x(t))$. See Figure 6.1 for a schematic representation of (6.1).

From the discussion of method 2 in Chapter 5, one can see that the right-hand side of (6.1) develops a noisy estimate of ∇f, at least as long as τ_2 is negligible. Thus, one might expect CALOPEX to operate on average qualitatively like gradient descent. This is indeed the dominant behavior as can be seen from (6.5). (See Section 6.2 for more details on the similarity.)

6.2 Minimization

The goal of this section is to show that, under certain conditions, the expectation of f decreases over time and that the expectation of f is approximately minimized. This section will develop an expression for the change in the expectation of f and then will show how various approximations simplify the expression.

To begin, let $\dot{n}$ be a stochastic vector with a stationary (i.e., time-independent) probability density $P(\dot{n})$. Assume that $\dot{n}$ has a mean of zero, has a finite variance, and can be integrated (i.e., $n = \int \dot{n} dt$ exists). Define the expectation as $E[\cdot] \triangleq \int(\cdot)P(\dot{n})d\dot{n}$. Since $\frac{d}{dt}E[f] = E[\dot{f}]$, first find an expression for $\dot{f}$. Using (6.1) and assuming that ∇f exists,

$$\begin{aligned}
\dot{f}(t) &= \dot{f}(s(t) + \alpha n(t)) = [\alpha\dot{n}(t) + \dot{s}(t)]^T \nabla f(t) \qquad (6.2)\\
&= \alpha\dot{n}^T(t)\nabla f(t) - \alpha\eta\dot{f}(t-\tau_T)\dot{n}^T(t-\tau_1)\nabla f(t)\\
&= \alpha\dot{n}^T(t)\nabla f(t) - \alpha\eta[\alpha\dot{n}^T(t-\tau_T) + \dot{s}^T(t-\tau_T)]\nabla f(t-\tau_T)\dot{n}^T(t-\tau_1)\nabla f(t)\\
&= \alpha\dot{n}^T(t)\nabla f(t) - \alpha^2\eta\nabla f^T(t-\tau_T)\dot{n}(t-\tau_T)\dot{n}^T(t-\tau_1)\nabla f(t)\\
&\qquad -\alpha\eta\nabla f^T(t-\tau_T)\dot{s}(t-\tau_T)\dot{n}^T(t-\tau_1)\nabla f(t)
\end{aligned}$$

where the "T" superscript represents the transpose of a vector, and $\tau_T = \tau_1 + \tau_2$.

(6.2) is complicated. Nevertheless, there are approximations that will cause terms in $E[\dot{f}]$ to factor conveniently, revealing the underlying behavior of the method. Assume that α can be made small and that correlations in $\dot{n}(t)$ die out exponentially in time, as is the case with Ornstein-Uhlenbeck processes. (Ornstein-Uhlenbeck processes are often used as realistic noise models. See [18] for rigorous mathematical details.) Specifically, assume that

$$\left|\mathrm{Cov}[g_1(\dot{n}(t_1)), g_2(\dot{n}(t_2))]\right| \leq \left|\mathrm{Cov}[g_1(\dot{n}(t_1)), g_2(\dot{n}(t_1))]\right| e^{-|t_1-t_2|/T_{ac}}$$

where g_1 and g_2 are arbitrary scalar-valued functions, $\mathrm{Cov}[a,b] = E[(a - E[a])(b - E[b])]$, and T_{ac} is the characteristic time for the decay, called the "autocorrelation time." Then, assuming that αT_{ac} is small, terms which involve products of functions of $\dot{n}$ and functions of αn will approximately factor under the expectation operator.

$$\begin{aligned}
&E[g_1(\dot{n}(t_1))g_2(\alpha n(t_2))] \\
&\quad = E\left[g_1(\dot{n}(t_1))\left\{g_2(0) + \nabla g_2^T(0)\alpha\int_{-\infty}^{t_2}\dot{n}^T(t')dt' + O(\alpha^2)\right\}\right] \\
&\quad = E[g_1(\dot{n})]g_2(0) + \alpha\nabla g_2^T(0)\int E[g_1(\dot{n}(t_1))\dot{n}^T(t')]dt' + O(\alpha^2) \\
&\quad = E[g_1(\dot{n})]g_2(0) + O\left(\alpha\int E[g_1(\dot{n})\|\dot{n}\|]e^{-|t_1-t'|/T_{ac}}dt'\right) + O(\alpha^2) \\
&\quad = E[g_1(\dot{n})]g_2(0) + O(\alpha T_{ac}) + O(\alpha^2)
\end{aligned}$$

where g_1 and g_2 are arbitrary differentiable functions. Since

$$E[g_2(\alpha n)] = E[g_2(0) + \nabla g_2^T(0)\alpha n + O(\alpha^2)] = g_2(0) + O(\alpha^2),$$

we have $g_2(0) = E[g_2(\alpha n)] + O(\alpha^2)$ and

$$E[g_1(\dot{n}(t_1))g_2(\alpha n(t_2))] = E[g_1(\dot{n})]E[g_2(\alpha n)] + O(\alpha T_{ac}) + O(\alpha^2). \tag{6.3}$$

Now, after E operates upon (6.2) and after using (6.3), various terms will factor and others will be negligible. The first term in (6.2) becomes

$$E[\alpha\dot{n}^T(t)\nabla f(t)] = \alpha E[\dot{n}^T]E[\nabla f] + O(\alpha^2 T_{ac}) + O(\alpha^3) = O(\alpha^2 T_{ac}) + O(\alpha^3).$$

The second term becomes

$$\begin{aligned}
&E[\alpha^2\eta\nabla f^T(t-\tau_T)\dot{n}(t-\tau_T)\dot{n}^T(t-\tau_1)\nabla f(t)] \\
&\quad = \alpha^2\eta E\left[\nabla f^T(t-\tau_T)E[\dot{n}(t-\tau_T)\dot{n}^T(t-\tau_1)]\nabla f(t)\right] + O(\alpha^3\eta T_{ac}) + O(\alpha^4\eta) \\
&\quad = \alpha^2\eta E[\nabla f^T(t-\tau_T)C(\tau_2)\nabla f(t)] + O(\alpha^3\eta T_{ac}) + O(\alpha^4\eta)
\end{aligned}$$

where $C(\tau_2) = E[\dot{n}(t-\tau_T)\dot{n}^T(t-\tau_1)]$ is the autocorrelation matrix for $\dot{n}$. (It depends only on $\tau_2 = \tau_T - \tau_1$ because $P(\dot{n})$ is independent of time.) Assuming that τ_T is small and that ∇f is differentiable in time, $\nabla f(t-\tau_T) = \nabla f(t) + O(\tau_T)$, and the second term becomes

$$\alpha^2\eta E[\nabla f^T(t)C(\tau_2)\nabla f(t)] + O(\alpha^2\eta\tau_T) + O(\alpha^3\eta T_{ac}) + O(\alpha^4\eta).$$

Since

$$\dot{s}(t+\tau_1) = -\alpha\eta\dot{f}(t-\tau_2)\dot{n}(t) = -\alpha\eta[\alpha\dot{n}(t-\tau_2)+\dot{s}(t-\tau_2)]\nabla f^T(t-\tau_2)\dot{n}(t) = O(\alpha^2\eta),$$

the third-and-final term becomes

$$E[\alpha\eta\nabla f^T(t-\tau_T)\dot{s}(t-\tau_T)\dot{n}^T(t-\tau_1)\nabla f(t)] = O(\alpha^3\eta^2). \tag{6.4}$$

η, T_{ac}, τ_1, and τ_2 remain in the order expressions because they are also adjustable during construction of the hardware. Altogether,

$$\begin{aligned}\frac{d}{dt}E[f] = E[\dot{f}] \;=\; & -\alpha^2\eta E[\nabla f^T(t)C(\tau_2)\nabla f(t)] \\ & +O(\alpha^3\eta^2) + O(\alpha^3) + O(\alpha^2\eta\tau_T) \\ & +O(\alpha^2 T_{ac}) + O(\alpha^3\eta T_{ac}) + O(\alpha^4\eta).\end{aligned} \tag{6.5}$$

The first term in (6.5) will be the dominant term if: $\tau_2 \leq T_{ac}$, so that $C(\tau_2) = O(e^{-\tau_2/T_{ac}})$ is not small; $\eta = O(\alpha^p)$ where $-1 < p < 1$; $\tau_T = O(\alpha^q)$ where $q > 0$; and $T_{ac} = O(\alpha^r)$ where $r > p$. Assume that $\alpha \neq 0$ (so that f is being perturbed) and that f is not exactly flat (so that the perturbation insures that ∇f is not always 0—otherwise x is already at a local minimum). Assume that $\dot{n}_i$ and $\dot{n}_j$ are independent for all $i \neq j$ so that $C(0)$ will be diagonal and positive definite. Assume that τ_2 is small enough so that $C(\tau_2)$ is positive definite. Then, $E[\nabla f^T C \nabla f] > 0$, and $\frac{d}{dt}E[f] < 0$ until the formerly small terms overwhelm the

first term. Thus, f decreases on average towards a local minimum (where $\nabla f = 0$) until $\|\nabla f\|^2$ becomes too small. This is not an exact optimization of $E[f]$ since $x = s+n$ will wander around the local minimum point (call it "s^*") as long as noise is added to the system, causing $E[f]$ to be $> f(s^*)$. Nevertheless, since the first term in (6.5) is of lower order in α than the other terms, (6.5) becomes closer to pure gradient descent and thus to exact optimization as $\alpha \to 0$. $\lim_{t\to\infty} |E[f(t)]-f(s^*)|^2$ quantifies the imperfection of the minimization—call it the "mismatch." Since $\alpha \to 0$ leads to a mismatch of zero, α can serve as a qualitative measure of mismatch.

The leading-order behavior of (6.5) is similar to gradient descent. If C is a scaled version of the identity matrix (as would be the case if all the noise generators were independent and had the same statistics), (6.5) becomes to leading order $E[\dot{f}] \propto -E[\|\nabla f\|^2]$. This is similar to (A.1) in Appendix A—i.e., it is similar to gradient descent. If C is not a scaled version of the identity matrix, the direction of descent is skewed away from the gradient, but it is still downhill as long as C is positive definite.

6.3 Convergence Time vs. Number of Parameters

In terms of information theory, CALOPEX relies on sending information (the perturbations) down a single channel (the wire leading from f). If each perturbation carries M bits of information, if there are N adjustable parameters (i.e., if s is an N-dimensional vector), and if the channel can transmit L bits of information per second, one expects to wait MN/L seconds in order to gather all the information necessary for one complete update of s. Thus, bandwidth limitations force one

to wait longer as N increases; and at best, the convergence time for CALOPEX should scale linearly with N.

The scaling relationship for CALOPEX is complicated by the presence of η. In Figure 6.1, one can see a channel leading from f to $-\eta(d/dt)$ and another leading from $-\eta(d/dt)$ to the multiplier. These channels cannot be lumped together and be considered as one because, according to the previous section, η can be extremely small or extremely large as α becomes small, depending on whether $p > 0$ or $p < 0$. Since f can be small while $\eta\dot{f}$ is large (or *vice versa*), one has to keep track of both.

Define the bandwidth limitations for the two channels as $\dot{f}(x) \leq W_1$ and $\eta\dot{f} \leq W_2$. These are just convenient ways to define bandwidths for what follows. Since $\dot{f} = \nabla f \cdot (\dot{s} + \alpha\dot{n})$, these conditions imply that $\dot{n}_i$ must typically be bounded. This will be true in hardware. Nevertheless, if $\dot{n}_i$ were unbounded but had a finite variance, one could still develop bandwidth limitations in terms of variances.

Assume that $\|\nabla f\|^2$ remains constant as N changes. This is reasonable in that f and $\|\nabla f\|^2$ will remain within certain limits when implemented in hardware and will not grow to be arbitrarily large as N increases.

Then, at the bandlimit, the first condition gives

$$W_1 = \dot{f}(x) = \dot{x} \cdot \nabla f(x) \approx \alpha\dot{n} \cdot \nabla f$$

to first order in α. Since

$$\|\dot{n}\|^2 = \sum_{i=1}^{N} \dot{n}_i^2 \approx Nc^2$$

where c^2 is a typical maximum size for $\dot{n}_i^2$, $W_1 \tilde{\propto} \alpha\sqrt{N}$ where " $\tilde{\propto}$ " means "approximately proportional to." Thus, $\alpha \tilde{\propto} 1/\sqrt{N}$. Similarly, for the condition involving W_2, $\eta\alpha \tilde{\propto} 1/\sqrt{N}$.

Note that, from (6.4) and (6.5), both $\dot{s}_i$ and $E[\dot{f}]$ exhibit their dependence on N by being $\tilde{\propto} \alpha^2\eta$. Thus, $1/(\alpha^2\eta)$ is the characteristic time for CALOPEX—define

this as τ_c. Since $\eta \propto \alpha^p$, the two bandwidth conditions give

$$\tau_c \tilde{\propto} N^{(1+p/2)} \tag{6.6}$$

and

$$\tau_c \tilde{\propto} N^{\left(\frac{1+p/2}{1+p}\right)}. \tag{6.7}$$

One must satisfy whichever is the most restrictive condition in order to stay within both bandwidth conditions. For $-1 < p < 0$, (6.7) is the most restrictive; for $0 < p < 1$, (6.6) is the most restrictive. The best scaling relationship occurs at $p = 0$, where the two conditions are identical and where $\tau_c \tilde{\propto} N$. Thus, at best, the convergence time for CALOPEX scales approximately linearly with N, which is the optimal scaling relationship.

6.4 Implementation Issues

The following is a summary of conditions that ensure the operation of the method: (1) f must be bounded from below; (2) f and its derivatives must exist; (3) $P(\dot{n})$ must be stationary; (4) $\dot{n}$ must be integrable, have a finite variance, and have a mean of zero; (5) correlations in $\dot{n}$ must die out exponentially; (6) it must be possible to make α and τ_T small; (7) τ_2 must be $\leq T_{ac}$; and (8) $C(\tau_2)$ must be positive definite.

Conditions 1 and 2 are not very restrictive. Conditions 3, 4, and 5 will be satisfied for many noise generators. If the noise n has a stationary probability distribution, so will that of $\dot{n}$. If n is generated by hardware that does not have an infinite bandwidth, $\dot{n}$ will exist, and its variance will be finite. If $E[n] = 0$, $E[\dot{n}] = \frac{d}{dt}E[n] = 0$. If correlations in n die out exponentially, the same will be true of correlations in $\dot{n}$. It should not be difficult to build noise generators with

these properties. Then, one needs only to run n through a differentiator to obtain a suitable $\dot{n}$. Conditions 6 and 7 will be achievable if one has enough latitude in setting α, T_{ac}, τ_1, and τ_2 during hardware construction. Condition 8 will be satisfied if τ_2 is small enough compared to T_{ac} (condition 7 will likely insure this for most well-behaved noise generators) and if $\dot{n}_i$ and $\dot{n}_j$ are independent for all $i \neq j$. Thus, not surprisingly, independent noise generators should be used for the components of the noise vector.

CALOPEX needs only adders, multipliers, differentiators, integrators, and noise generators. Most of these devices are straightforward to implement in analog hardware, but noise generators and suitably long-term integrators might be difficult to build. In [32], Kerns shows how to build an analog-VLSI noise generator that is compact and that requires only a small number of devices. However, it is not yet clear how resistant a collection of such noise generators is to phase locking or to the pickup of 60 Hz oscillations from the power supply, both of which would cause CALOPEX to fail. [31] and [26] discuss storage devices that are appropriate for constructing integrators in analog VLSI that have very long time constants. However, these devices use ultraviolet light, which can interfere with other circuit components if the chip is not correctly laid out. This also might cause CALOPEX to fail.

Nevertheless, there are four features of CALOPEX that are especially helpful for implementing the method in hardware. First, CALOPEX does not require explicit computation of any gradients. This is useful in cases where ∇f is difficult to compute. Second, the inputs to the multiplier in Figure 6.1 are time derivatives. This makes it easier to adjust the hardware so that the multiplier does not clip, since the inputs are approximately centered around zero (because $E[\dot{n}] = 0$ and

$E[\dot{f}] = O(\alpha^2\eta)$, which is small). If instead the inputs had large offsets, the dynamic range of the multiplier would be reduced. Third, the computation of (6.1) involves only local information except for one global signal. In other words, the change in parameter s_i requires only the local values n_i and s_i and the global signal $\dot{f}$ (which is transmitted identically to all sites of adaption). This means that the hardware construction does not require complicated transmission of signals among all the sites of adaption. Fourth, CALOPEX needs only one differentiator for f and one differentiator, one adder, one multiplier, and one noise generator for each component of the vector x. Thus, the complexity of the algorithm (measured in number of devices needed) scales linearly with the number of adjustable parameters.

Also, when implemented in analog hardware, CALOPEX uses parallel computation since it adjusts all the parameters of f at the same time. This increases the convergence speed of the optimization compared to serial execution, but it also provides CALOPEX with a measure of damage resistance. The damage resistance comes from the fact that destruction of one site of adaption does not necessarily destroy the other sites' abilities to adapt. Thus, while damage might cause s_k to become frozen, all the other s_i, $i \neq k$, could keep adapting and would naturally adjust themselves to take into account that s_k is fixed. In other words, CALOPEX would minimize f subject to the constraint $s_k =$ a constant. This degradation because of accumulated damage is closer to that exhibited by biological systems than to that exhibited by digital computers.

Of course, the convergence time for CALOPEX increases as the number of parameters increases, all other aspects being constant. Section 6.3 shows that, at best, the convergence time scales approximately linearly with the number of adjustable parameters (the dimension of s). According to information-theoretical

arguments, this is the best possible scaling relationship.

Lastly, the end of Section 6.2 defines mismatch and shows that α can serve as a qualitative indicator of the amount of mismatch. If α is adjustable during the course of optimization, it is possible to perform a sort of simulated annealing by starting with larger α (for faster adaption) and gradually reducing it (for smaller mismatch). (See [33] for a discussion of simulated annealing.) α is similar to the temperature parameter in simulated annealing, and its adjustment might likewise be useful as a technique for avoiding local minimae.

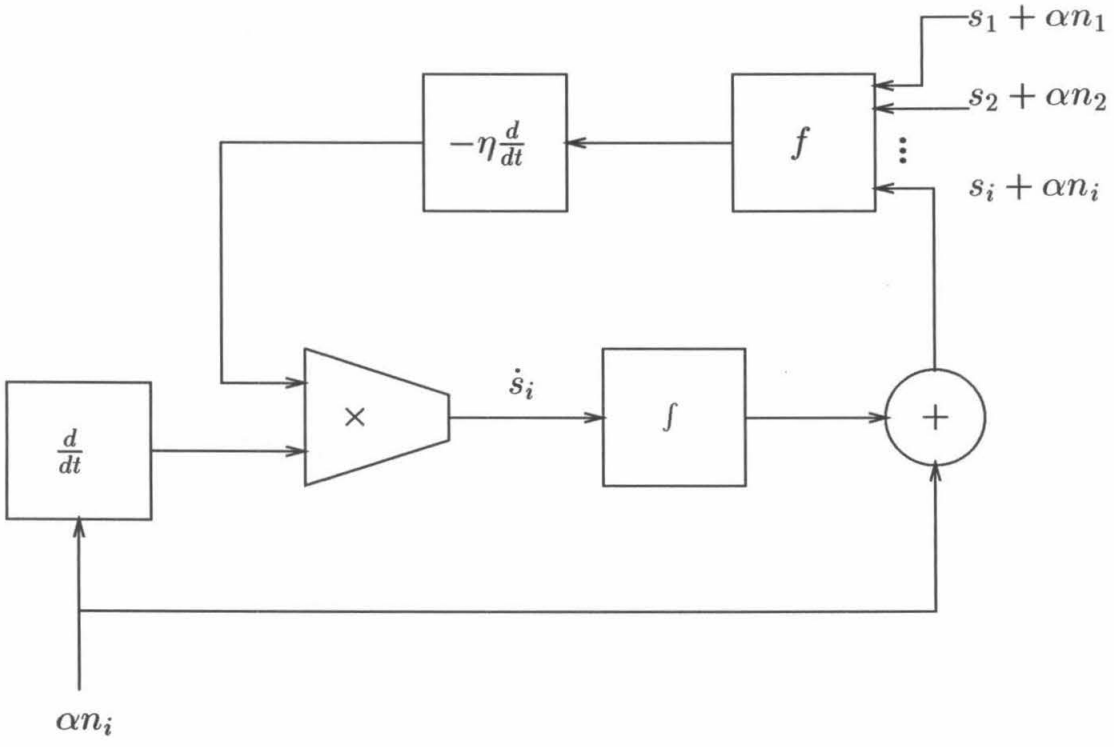

Figure 6.1: A schematic representation of CALOPEX. f is the function to be minimized; n is a stochastic vector; α is a small constant; η is a constant that adjusts convergence speed; and s is a vector that represents the adjustable parameters in f. CALOPEX can tolerate small time delays in the computation of f and in the multiplier/integrator path.

Appendix A

Optimization Through Gradient Descent

Assume that we would like to minimize the function $f(x)$, where f is bounded below and where both f and x are scalars. This might be necessary as part of an adaptive-control system where f is an error between actual and desired outputs or as part of a learning system (such as a neural network). One common approach to optimization is gradient descent, where one sets

$$\frac{dx}{dt} = -\alpha \frac{df}{dx}$$

and where $\alpha > 0$ is a constant. Using these dynamics,

$$\frac{df}{dt} = \frac{df}{dx}\frac{dx}{dt} = -\frac{1}{\alpha}\left(\frac{dx}{dt}\right)^2 \Rightarrow \begin{cases} = 0 & \text{if } \dot{x} = 0 \\ < 0 & \text{otherwise.} \end{cases}$$

Thus, f decreases until x stops changing, at which point $df/dx = 0$. This point is at least a local minimum.

Likewise, for the case where x is a vector and where $\dot{x} = -\alpha \nabla f$, one can find

$$\frac{df}{dt} = \nabla f \cdot \frac{dx}{dt} = -\frac{1}{\alpha}\left\|\frac{dx}{dt}\right\|^2 = -\alpha \|\nabla f\|^2 \Rightarrow \begin{cases} = 0 & \text{if } \nabla f = 0 \text{ and } \dot{x} = 0 \\ < 0 & \text{otherwise.} \end{cases} \quad \text{(A.1)}$$

Bibliography

[1] N. Abramson, *Information Theory and Coding.* New York: McGraw-Hill, 1963.

[2] B. P. Anderson and D. D. Montgomery, "A method for noise filtering with feed-forward neural networks: analysis and comparison with low-pass and optimal filtering," in *Proceedings of the International Joint Conference on Neural Networks, San Diego, '90*, vol. 1. New York: IEEE, 1990, pp. 209–214.

[3] J. A. Anderson, J. W. Silverstein, S. A. Ritz and R. S. Jones, "Distinctive features, categorical perception, and probability learning: some applications of a neural model," *Psychological Review*, vol. 84, pp. 413–451, 1977.

[4] R. Ash, *Information Theory.* New York: Wiley, 1965.

[5] D. S. Bernstein and D. C. Hyland, "Optimal projection equations for reduced-order modeling, estimation, and control of linear-systems with multiplicative white noise," *Journal of Optimization Theory and Applications*, vol. 58, no. 3, pp. 387–409, 1988.

[6] J. M. Bower, "Reverse engineering the nervous system: an anatomical, physiological and computer based approach," in *An Introduction to Neural and Electronic Networks*, S. Zornetzer, J. Davis, and C. Lau, Eds. San Diego: Academic Press, 1990.

[7] E. Charniak, C. K. Riesbeck, and D. V. McDermott, *Artificial Intelligence Programming.* New York: Wiley, 1980.

[8] B. S. Chow and W. P. Birkemeier, "A new recursive filter for systems with multiplicative noise," *IEEE Transactions on Information Theory*, vol. 36, no. 6, pp. 1430–1435, 1990.

[9] G. G. S. Collins, J. Anson, and G. A. Probett, "Patterns of endogenous amino acid release from slices of rat and guinea-pig olfactory cortex," *Brain Research*, vol. 204, no. 1, pp. 103–120, 1981.

[10] A. Constanti, J. D. Connor, M. Galvan, and A. Nistri, "Intracellularly-recorded effects of glutamate and aspartate on neurones in the guinea-pig olfactory cortex slice," *Brain Research*, vol. 195, no. 2, pp. 403–420, 1980.

[11] J. T. Coyle, D. L. Price, and M. R. DeLong, "Alzheimer's disease: a disorder of cortical cholinergic innervation," *Science*, vol. 219, pp. 1184–1190, 1983.

[12] A. Dembo and T. Kailath, "Model-free distributed learning," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 58–70, 1990.

[13] C. S. Draper and Y. T. Li, *Principles of Optimalizing Control Systems and an Application to the Internal Combustion Engine.* New York: ASME, 1951.

[14] R. Duda and P. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[15] J. Eckmann and D. Ruelle, "Ergodic theory of chaos and strange attractors," *Reviews of Modern Physics*, vol. 57, pp. 617–619, 1985.

[16] P. Eykhoff and O. J. M. Smith, "Optimalizing control with process-dynamics identification," *IRE Transactions on Automatic Control*, vol. AC-7, pp. 140–155, 1962.

[17] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989.

[18] C. W. Gardiner, *Handbook of Stochastic Methods for Physics, Chemistry, and the Natural Sciences*. Berlin: Springer-Verlag, 1985.

[19] P. Grassberger and I. Procaccia, "Characterization of strange attractors," *Physical Review Letters*, vol. 50, no. 5 , pp. 346–349, 1983.

[20] L. B. Haberly and J. M. Bower "Olfactory cortex: model circuit for study of associative memory?" *Trends in Neurosciences*, vol. 12, no. 7, pp. 258–264, 1989.

[21] J. J. Hagan and R. G. M. Morris, "The cholinergic hypothesis of memory: a review of animal experiments," in *Handbook of Psychopharmacology*, vol. 20, L. L. Iversen, S. D. Iversen, and S. H. Snyder, Eds. New York: Plenum Press, 1989.

[22] E. Harth and E. Tzanakou, "ALOPEX: a stochastic method for determining visual receptive fields," *Vision Research*, vol. 14, pp. 1475–1482, 1974.

[23] M. E. Hasselmo, B. P. Anderson, and J. M. Bower, "Cholinergic modulation may enhance cortical associative memory function," in *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 46–52.

[24] M. E. Hasselmo, M. A. Wilson, B. P. Anderson and J. M. Bower, "Associative memory function in piriform (olfactory) cortex: computational modeling and neuropharmacology," in *Cold Spring Harbor Symposia on Quantitative Biology*, vol. LV. Cold Spring Harbor, NY: Cold Spring Harbor Laboratory, 1990, pp. 599–609.

[25] M. E. Hasselmo and J. M. Bower, "Cholinergic suppression specific to intrinsic not afferent fiber synapses in rat piriform (olfactory) cortex," *Journal of Neurophysiology*, vol. 67, no. 5, pp. 1222–1229, 1992.

[26] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 'floating gate' synapses," in *Proceedings of the International Joint Conference on Neural Networks, Washington D.C., '89*, vol. II. New York: IEEE, 1989, pp. 191–196.

[27] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[28] B. T. Hyman, A. R. Damasio, G. W. Van Hoesen, and C. L. Barnes, "Cell specific pathology isolates the hippocampal formation in Alzheimer's disease," *Science* vol. 225, pp. 1168–1170, 1984.

[29] M. W. Jung, J. Larson, and G. Lynch, "Long-term potentiation of monosynaptic EPSPs in rat piriform cortex in vitro," *Synapse*, vol. 6, no. 3, pp. 279–283, 1990.

[30] E. D. Kanter and L. B. Haberly, "NMDA-dependent induction of long-term potentiation in afferent and association fiber systems of piriform cortex in vitro," *Brain Research*, vol. 525, no. 1, pp. 175–179, 1990.

[31] D. A. Kerns, J. E. Tanner, M. A. Sivilotti, and J. Luo, "CMOS UV-writable non-volatile analog storage," in *Advanced Research in VLSI: Proceedings of the 1991 University of California/Santa Cruz Conference*, C. H. Sequin, Ed. Cambridge, MA: MIT Press, 1991, pp. 245–261.

[32] D. A. Kerns, "A compact noise source for VLSI applications" (work in progress).

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[34] C. Klimasauskas, "Neural nets and noise filtering," *Dr. Dobb's Journal*, p. 32, Jan. 1989.

[35] T. Kohonen, *Self-Organization and Associative Memory.* Berlin: Springer-Verlag, 1984, pp. 35–41.

[36] T. Kohonen, P. Lehtio, J. Rovamo, J. Hyvarinen, K. Bry, and L. Vainio, "A principle of neural associative memory," *Neuroscience*, vol. 2, pp. 1065–1076, 1977.

[37] M. D. Kopelman, "The cholinergic neurotransmitter system in human memory and dementia: a review," *Quarterly Journal of Experimental Psychology*, vol. 38A, pp. 535–573, 1986.

[38] A. Lapedes and R. Farber, "How neural networks work," in *Neural Information Processing Systems*, D. Z. Anderson, Ed. New York: American Institute of Physics, 1988, pp. 442–456.

[39] C. Mead, *Analog VLSI and Neural Systems.* New York: Addison-Wesley, 1989.

[40] R. G. Morris and M. D. Kopelman, "The memory deficits in Alzheimer-type dementia: a review," *Quarterly Journal of Experimental Psychology*, vol. 38A, pp. 575–602, 1986.

[41] Y. A. Phillis, "A smoothing algorithm for systems with multiplicative noise," *IEEE Transactions on Automatic Control*, vol. 33, no. 4, pp. 401–403, 1988.

[42] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes: the Art of Scientific Computing.* London: Cambridge University Press, 1986.

[43] D. L. Price, "New perspectives on Alzheimer's disease," *Annual Review of Neuroscience*, vol. 9, pp. 489–512, 1986.

[44] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol. 1, D. Rumelhart *et al.*, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[45] T. L. Saaty, *Modern Nonlinear Equations.* New York: Dover, 1981, pp. 393–408.

[46] T. Simchony, R. Chellappa, and Z. Lichtenstein, "Relaxation algorithms for map estimation of gray-level images with multiplicative noise," *IEEE Transactions on Information Theory*, vol. 36, no. 3, pp. 608–613, 1990.

[47] P. R. Stromer, "Adaptive or self-optimizing control systems—a bibliography," *IRE Transactions on Automatic Control*, vol. AC-4, pp. 65–68, 1959.

[48] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick, '80*, D. Rand and L. Young, Eds. Berlin: Springer-Verlag, 1981, pp. 366–381.

[49] S. L. Tanimoto, *The Elements of Artificial Intelligence: An Introduction Using LISP*. Rockville, MD: Computer Science Press, 1987.

[50] J. Theiler, "Estimating fractal dimension," *Journal of the Optical Society of America A—Optics and Image Science*, vol. 7, no. 6, pp. 1055–1073, 1990.

[51] R. Tomovic and W. J. Karplus, *High-speed Analog Computers*. New York: Wiley, 1962.

[52] E. Tzanakou, R. Michalak, and E. Harth, "The alopex process: visual receptive fields by response feedback," *Biological Cybernetics*, vol. 35, pp. 161–174, 1979.

[53] E. Vittoz, "Future of analog in the VLSI environment," in *Proceedings of the IEEE International Symposium on Circuits and Systems, New Orleans, '90*, vol. 2. Piscataway, NJ: IEEE, 1990, pp. 1372–1375.

[54] C. A. A. Wass and K. C. Garner, *Introduction to Electronic Analogue Computers*. London: Pergamon Press, 1965.

[55] L. Watts, R. Lyon , and C. Mead, "A bidirectional analog VLSI cochlear model," in *Advanced Research in VLSI: Proceedings of the 1991 University of California/Santa Cruz Conference*, C. H. Sequin, Ed. Cambridge, MA: MIT Press, 1991, p. 156.

[56] L. Watts, *Cochlear Mechanics: Analysis and Analog VLSI Simulation*, Ph.D. Thesis, Department of Electrical Engineering, California Institute of Technology, 1992.

[57] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Thesis, Department of Applied Mathematics, Harvard, 1974.

[58] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[59] M. A. Wilson and J. M. Bower, "A computer simulation of olfactory cortex with functional implications for storage and retrieval of olfactory information," in *Neural Information Processing Systems*, D. Z. Anderson, Ed. New York: American Institute of Physics, 1988, pp. 114–126

[60] P. H. Winston, *Artificial Intelligence.* Reading, MA: Addison-Wesley, 1977.