Analog VLSI Autonomous Systems for Learning and Optimization

Dissertation by

Gert Cauwenberghs

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1994

(Defended December 8, 1993)

Acknowledgments

The California Institute of Technology has been an especially exciting and stimulating environment to me. During the five years of my Caltech experience I have had the privilege to interact with many fine individuals who directly and indirectly contributed to the success of my dissertation work presented here, and more importantly provided me the opportunity for growth and expanding my knowledge.

It was during the early years at Caltech that I initially developed a strong interest in analog VLSI and neural systems, in particular under the stimulus of Carver Mead. Ever since, the work done by him and his co-workers has strongly inspired mine, and I have been so fortunate to interact with them and exchange ideas on many occasions. Also, with the encouragement and appreciation Carver extended to me, the pursuit of my work has been increasingly rewarding and enjoyable.

I can hardly imagine completing the work without the constant source of support which my adviser, Amnon Yariv, has generously provided me in many ways. His genuine interest in my research and his care for my professional growth have been a pleasant experience to me. Other members in his group have extended me much cordiality and support as well. I especially enjoyed working together with Chuck Neugebauer and Volnei Pedroni on several related projects, gaining me valuable experience which proved very useful in carrying out the experimental work.

While the Caltech of academic excellence has provided me a stimulating environment to pursue my work, it also brought me a treasure of human nature. At Caltech I was fortunate to meet Langche Zeng, whose love and warm care for me has always been a unique source of happiness. She motivates the perspective of my work and has constantly stimulated me in pursuing higher standards in life in general. My deep gratitude also goes to my parents for having given me the indispensable warmth and care of a loving family.

Finally, I wish to express thanks for various ways of feedback on the presented work I received from Josh Alspector, Pierre Baldi, Bhusan Gupta, Marwan Jabri, David Kirk, Maurice van Putten, Jim Spall, Volnei Pedroni, and others whose names I mentioned

above or I forgot to include. I also thank my committee members for their commitment and their interest in my work, and certainly for their flexibility in arranging a time for the Ph.D. exam despite their busy schedules. A Francqui fellowship of the Belgian American Education Foundation provided financial support for the first year of graduate study at Caltech, and the prompt ARPA/NSF MOSIS fabrication service was instrumental in obtaining the VLSI chips for the experiments.

Analog VLSI Autonomous Systems for Learning and Optimization

Gert Cauwenberghs

Abstract

The integration of adaptive functions within analog neural hardware, while certainly promising to enhance system performance, has for long been hindered by technological difficulties due to the complexity and sensitivity of standard adaptive algorithms. We present a general framework for self-contained adaptation in analog VLSI supporting a broad class of supervised learning and optimization tasks, which largely alleviates the implementation problems by virtue of a robust system approach exploiting statistics and redundancy in stochastic processes. Specifically, the framework includes: *i)* a perturbative algorithm based on stochastic approximation to optimize a set of parameters in an arbitrary deterministic system, these parameters being adjusted according to global performance evaluations rather than using explicit knowledge about the internal structure of the system; and *ii)* a scalable and modular CMOS architecture that implements this algorithm, and that additionally provides for embedded long-term dynamic storage of the volatile analog parameter values, quantized locally and refreshed autonomously on capacitors with direct external access in both digital and analog formats. We analyze the convergence and scaling properties of the stochastic algorithm, present on-line versions of the algorithm for supervised learning in dynamical systems, and provide experimental results demonstrating real-time trajectory learning on an analog CMOS chip containing a network of six fully recurrent dynamical neurons. We also include results demonstrating robust long-term retention of locally stored volatile information in analog VLSI using the autonomous refresh technique.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Most on-line adaptive algorithms for training dynamic recurrent neural networks use explicit gradient information on the network dynamics to perform on-line steepest descent of the observed training error in the parameter space. Since the calculation of the error gradient at a given instant needs to account for the accumulated error dynamics over an extended time span, gradient descent methods are rather computationally complex. For real-time implementations, they require extensive provisions for storage and interconnectivity which scale disproportionally with the dimension of the system.

Chapter 2 investigates a stochastic approximation technique for error-descent supervised learning in nonlinear dynamical systems, using global performance evaluations on the physical network under parallel stochastic perturbations of the parameters rather than calculating gradients from an assumed internal structural model. The stochastic method applies to general parameter-driven dynamical systems with arbitrary continuous characteristics, of which neither the functional form nor the internal structure need to be known. Two practical on-line schemes are presented that implement the learning method in two different modes of operation under real-time conditions, for analysis and prediction of time-varying processes, and for identification and control of unknown dynamical

systems. Simulation results validating both schemes are included.

Chapter 3 covers architecture and implementation issues regarding the integration of the learning method in real-time hardware. Scalable and modular VLSI architectures are presented which implement the two on-line schemes of the stochastic method outlined in Chapter 2. Of both schemes, the second one is particularly amenable for integration in faulty analog environments, as it reflects and preserves the model-independent nature of the stochastic learning algorithm. Other than discussing accuracy issues related to the analog VLSI implementation, Chapter 3 also describes a technique for long-term local storage of the analog parameters in capacitive format, which can be directly embedded in the learning architecture using available resources already provided by the learning update units. The technique repetitively quantizes and refreshes the volatile analog parameter values, incorporating redundancy and statistical averaging to avoid catastrophic loss of information triggered by occasional quantization errors. The description of a compact and fault-tolerant circuit implementing a quantization element, which comprises a bi-directional A/D (analog-to-digital) and D/A (digital-to-analog) converter, is given in Appendix A, together with measurements on the characteristics of the device.

Chapter 4 describes the analog VLSI implementation of the integrated learning and storage architecture, and presents experimental results obtained from two chips, one incorporating refresh and quantization portions of the architecture and the other comprising the full functionality of the learning and storage system embedded in an analog VLSI neural network. The network contains six fully recurrent neurons with continuous-time dynamics, providing 42 free parameters which include connection strengths and thresholds, and is trained to generate two neuron output signals following a trajectory of prescribed periodic waveforms. The chip implementing the network includes local provisions supporting both the learning and storage of the parameters, and can be readily expanded for applications of learning recurrent dynamical networks requiring larger dimensionality. The chapter describes and characterizes the functional elements comprising the implemented recurrent network and integrated learning system, and includes experimental results obtained from

training the network to produce two outputs following a circular trajectory, representing a quadrature-phase oscillator. The robust and autonomous operation of the refresh scheme is verified on a separate chip containing a 128-element array of capacitive storage cells with integrated quantization elements, demonstrating long-term analog storage in excess of 8 bit resolution.

Finally, Chapter 5 concludes the findings and marks areas for further investigation.

1.2 Optimization in Dynamical Systems

Dynamic recurrent neural networks have become a powerful tool for the analysis and prediction of time-varying processes [Sompolinsky 86], [Sato 90b], [Lopez 93] and for the modelling and control of unknown dynamical systems [Narendra 90], [Narendra 93]. In effect, the dynamic behavior of any deterministic system with continuous characteristics can be approximated by means of an appropriately sized recurrent neural network, exploiting the vast degree of freedom constituted by the free parameters of the network to capture the underlying dynamics of the unknown system [Funahashi 93]. To assign optimal values for the parameters of the network—connection strengths, thresholds and time constants, and possibly others depending on the network model, a variety of learning algorithms have been derived. Supervised learning algorithms, which steer the response of the network towards an exemplary training signal by adjusting the parameters accordingly, are preferred when training signal and performance evaluation are continuously available. The complexity of current supervised learning algorithms limits the practical use of recurrent neural nets as generalized dynamical systems. The computational requirements tend to increase strongly with the size of the network, especially for real-time supervised learning, where the parameters of the network need to be adjusted on-line according to a continuous stream of data obtained from the instantaneous network response and the training signal [Williams 89]. Additionally, implementation of these learning algorithms requires considerable accuracy in the computation and the modelling, and the learning performance is quite sensitive to errors and model mismatches which frequently arise in

hardware implementations.

In the following chapter, we investigate a simple method which significantly reduces the computation complexity and sensitivity of supervised learning in dynamical systems at little or no cost in performance over alternative, mostly gradient-based approaches. The learning architecture employs a stochastic perturbative algorithm which probes the dependence of the network error on the parameters directly, rather than deriving an estimate of the gradient based on an explicit model of the network dynamics [Cauwenberghs 93a]. The direct approach of observing the error gradient on the physical network relates to techniques of stochastic approximation [Robins 51], [Kushner 78] and finds parallels in other recent algorithmic developments [Styblinski 90], [Spall 92a] and hardware learning architectures as well [Dembo 90], [Kirk 93], [Alspector 93], [Flower 93]. We address practical issues that arise for implementation of the learning method in real-time environments and with special-purpose hardware, especially concerning the scaling of the learning architecture with the dimension of the dynamical system, and the natural causality constraints imposed by real-time hardware on the time-domain computations. The algorithmic framework of the method assumes no predefined structure or functional form for the network, and is applicable to general parameter-driven structures other than recurrent neural networks as well.

1.2.1 Parameter-Driven Dynamical Systems

The dynamics of the general class of network structures under consideration can be cast in the general abstract form

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = F(\mathbf{p}, \mathbf{x}, \mathbf{y}) \ , \tag{1.1}$$

where $\mathbf{x}$ represents the vector of state variables of the network, $\mathbf{p}$ denotes the vector of adjustable parameters, and the vector $\mathbf{y}$ corresponds to the inputs external to the network. The functional form of (1.1) complies with that of dynamical recurrent neural networks,

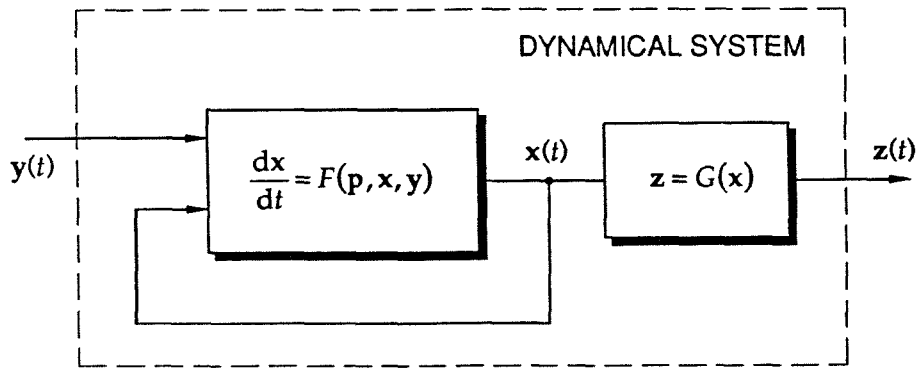


Figure 1.1: General form of the dynamical system under optimization.

typically defined by

$$\tau_i \frac{dx_i}{dt} = -x_i + \sigma(\sum_j W_{ij}x_j + \theta_i + y_i) \tag{1.2}$$

with a parameter vector given by the combined weights W_{ij}, thresholds θ_i and time constants τ_i of the network. The nonlinear activation σ in (1.2) is usually specified as a sigmoidal function, such as $\sigma(x) \equiv \tanh(x)$. The network output vector $\mathbf{z}(t)$ is obtained indirectly from the state variables x_i governing the network dynamics (1.1), through a transformation

$$\mathbf{z}(t) = G(\mathbf{x}(t)) \ . \tag{1.3}$$

The configuration of the functional elements F and G, comprising the dynamical system, is illustrated symbolically in Figure 1.1.

The additional transformation G accounts for a diversity of network configurations, specific to the application of interest. Similar to (1.1), neither assumptions on the functional form nor explicit knowledge of (1.3) are necessary to support the learning method. The instantaneous transformation (1.3) can be further generalized to include dynamic features. As an example, the operator G may include the response of an unknown dynamical system

$$\mathbf{z}(t) = \int_{-\infty}^{t} g(\mathbf{x}(\theta), t, \theta)d\theta \tag{1.3b}$$

driven by the network, such as a nonlinear plant under control by the network. In a different context, more typical in applications for identification of dynamical systems and prediction of time-varying sequences, the transformation G conforms to the instantaneous format (1.3) and accounts for a projection of the internal dynamics onto the output, usually contracting towards lower dimensionality. Such is useful to generate complex dynamical systems of higher internal dimensionality than what can be obtained externally through straight projection of the time derivatives of the visible units on their present states, therefore encompassing a potentially richer spectrum of dynamical characteristics. Additional degrees of freedom in the optimization can be obtained by allowing for extra parameters in the functional from of (1.3) as well. In the case of the dynamical recurrent neural networks considered under (1.2), the outputs z_i may be obtained through an additional layer of weighted sums and sigmoids, or alternatively may simply consist of a select subset of the network states x_i.

For the optimization method under study, no particular assumptions on the functional form and the physical configuration of F and G are needed, other than their combination produces a dynamical response at the outputs $\mathbf{z}(t)$ to a time-varying stimulus $\mathbf{y}(t)$ at the inputs, in a consistent manner as determined by a set of continuous and constant parameters $\{p_i\}$ which can be tuned to optimize performance. The details of the internal structure comprising the functional blocks F and G merely relate to specific configurations supporting different applications. For simplicity, the dynamical system under optimization can be considered as an integrated entity, and a clear distinction between the network and the surrounding environment participating in the dynamical process is not needed.

1.2.2 Supervised Learning

In the context of supervised learning, the system is presented a training signal $\mathbf{z}^T(t)$, encoding the desired outputs in response to the supplied inputs $\mathbf{y}(t)$. The origin of the training signals, supplied externally to the network, depends on the specific application. Two typical configurations of the dynamical system, in relation to the reference supplying

the target signals, are given in Figure 1.2. The first configuration, shown in Figure 1.2 (a), features a reference source which generates both input and output training signals $\mathbf{y}(t)$ and $\mathbf{z}^T(t)$, as given by time-varying signals extracted from an external process with causal dynamical characteristics. Conversely, the alternative configuration of Figure 1.2 (b) employs a reference system which embodies the target dynamical response of the system under optimization, whereby the input training sequence $\mathbf{y}(t)$ is chosen as to represent the complete range of inputs under typical operating conditions. The distinction between both configurations and their corresponding application domains forms the basis of the two distinct real-time realizations of the stochastic method and their implementation architectures presented in Sections 2.3 and 3.1.

An error functional $\mathcal{E}(\mathbf{p})$ is constructed on the system outputs $\mathbf{z}$ to quantify the global performance of the system in tracking the training signal $\mathbf{z}^T(t)$. As with other approaches for supervised learning, the method aims at decreasing the error measure with respect to the parameters, with the optimum state of the system defined as that corresponding to the absolute minimum of the error. The error functional can be constructed freely, provided it satisfies obvious continuity and consistency conditions. Typically, an error measure satisfies the format

$$\mathcal{E}(\mathbf{p}) = \int_{-\infty}^{+\infty} e(\mathbf{z}(t), \mathbf{z}^T(t))\mathrm{d}t \ , \tag{1.4}$$

with a distance metric $e(\mathbf{z}(t), \mathbf{z}^T(t))$ which usually takes the form of a mean-square error (MSE)

$$e(\mathbf{z}(t), \mathbf{z}^T(t)) = \frac{1}{2}\sum_i (z_i(t) - z_i^T(t))^2 \ . \tag{1.5}$$

Besides network outputs $\mathbf{z}(t)$ and training signals $\mathbf{z}^T(t)$, the error functional may include other factors which contribute to the "performance" in other ways, provided the thus constructed error measure can be directly observed on the network.

The infinite horizon of the formulation in (1.4) hinders implementation, since the observed or supplied values of $\mathbf{z}(t)$ and $\mathbf{z}^T(t)$ are available over a limited time range only. Often a truncated error measure is used in practice, defined over a finite observation time

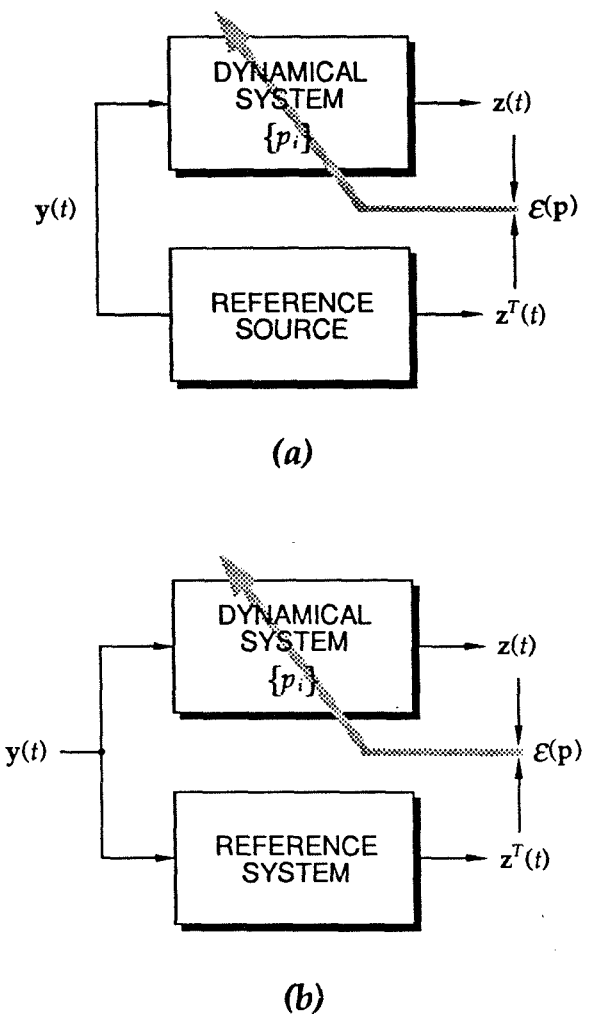


Figure 1.2: Configurations for supervised learning on the dynamical system. *(a)* Configuration with reference source. *(b)* Alternative configuration with reference system and externally supplied training input sequence.

interval $[t_0, t_f]$

$$\mathcal{E}(\mathbf{p}; \mathbf{x}_0; t_0, t_f) = \int_{t_0}^{t_f} e(\mathbf{z}(t), \mathbf{z}^T(t))\mathrm{d}t \ . \tag{1.6}$$

Obviously, minimization of the truncated error measure (1.6), at given values for t_0 and t_f, will generally not yield the same set of parameters as the optimum values obtained by minimizing the theoretical measure (1.4). Furthermore, the value of the error measure (1.6) depends on the initial state of the network $\mathbf{x}_0 = \mathbf{x}(t_0)$, which needs to be specified explicitly for the error measure to be uniquely defined. These issues are the subject of the analysis in Section 2.3. Another more basic issue, related to the integration time intervals, is the timing organization of the computations during learning with regard to the timing of the network dynamics. This issue distinguishes on-line methods from off-line methods for supervised learning. Off-line methods, while generally computationally efficient, require extensive access to the history of network variables for the computation of the parameter updates, implying provisions for large memory storage proportional to the integration time interval of (1.6). Typically, off-line methods consider a fixed time interval $[t_0, t_f]$ and given initial conditions $\mathbf{x}_0$ for the error functional (1.6), forcing a reset of time on the network for every new learning iteration. They effectively perform batch-mode learning, repetitively circulating the given "data-set" constituted by the training signals $(\mathbf{y}(t), \mathbf{z}^T(t))$ over the interval $[t_0, t_f]$, for every "epoch" of the learning session. In contrast, on-line schemes construct values for the parameter updates through a sequence of operations which proceed exclusively in forward-time, synchronously with the network dynamics. As a consequence, on-line methods are able to operate on the network and update its parameters while it runs continuously, without the need to reset time and control initial conditions for the state variables. This makes on-line methods far more suitable than off-line methods for real-time applications, under a continuous stream of incoming learning and network signals $\mathbf{y}(t)$, $\mathbf{z}^T(t)$, and $\mathbf{z}(t)$.

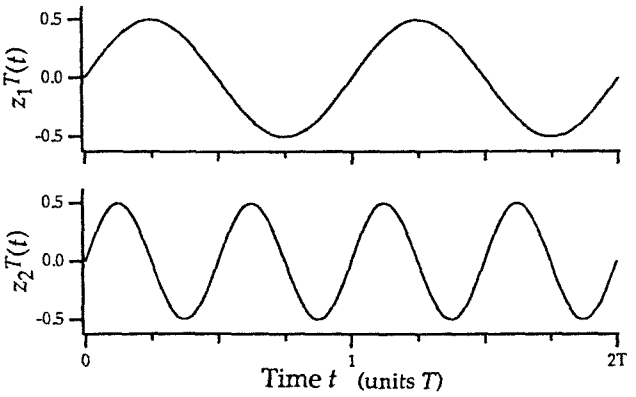
1.2.3 Trajectory Learning and Teacher Forcing

The following chapter also includes simulation results to experimentally confirm the analysis carried out there, and in particular to assess the performance of the stochastic method relative to gradient descent and to demonstrate the real-time embodiments of the method. All simulations are performed on the same learning example, which consists of a Lissajous dynamic trajectory defined by a pair of sinusoidal target output signals

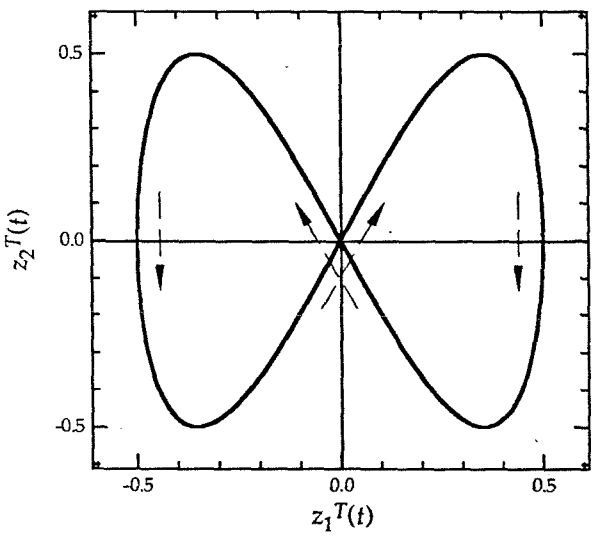
$$\begin{cases} z_1^T(t) &= \sin(t) \\ z_2^T(t) &= \sin(2t) \end{cases} \tag{1.7}$$

with periodicity $T = 2\pi$, in the absence of externally supplied inputs, $\mathbf{y}(t) \equiv 0$ [Pearlmutter 89]. The periodic waveforms of the target training signal and the corresponding phase diagram representation are depicted in Figure 1.3. A similar but less complex periodic trajectory, comprising a quadrature-phase oscillator, is used for the experimental learning sessions on the analog neural network chip as described in Chapter 4.

The motivation for choosing the particular form of training signals in (1.7) follows from the complex nonlinear dynamics needed to generate the trajectory without external stimulus, constituting a non-trivial learning problem. In particular, the trajectory phase diagram of Figure 1.3 (b) contains a singularity in the direction of motion at the origin, indicating that the two state variables comprising the outputs of the system are not sufficient to completely model the underlying dynamics generating the waveforms (1.7). As a consequence, learning methods merely relating the time derivative of the output vector to its state directly, through a nonlinear transformation such as a feedforward neural network, fail to reproduce the desired trajectory. The waveforms (1.7) can be successfully generated by training a six-neuron dynamic recurrent network (1.3) through gradient descent on the error (1.6) over an integration interval including a couple of periods of (1.7) [Toomarian 92]. For the simulations we use the same network structure and learning parameters as in [Toomarian 92]. The network contains six neurons according to (1.2), with fixed time constants τ_i set to unity. The output $\mathbf{z}$ is represented by two selected neuron state variables,



(a)



(b)

Figure 1.3: Benchmark learning example for the simulations: a "Figure 8" dynamical trajectory. *(a)* Training signals. *(b)* Phase diagram representation.

$z_i(t) \equiv x_i(t); \quad i = 1, 2.$

As in [Toomarian 92], a teacher forcing signal is applied during learning in the form of an external input $\mathbf{y}$, according to

$$y_i(t) = \lambda \left(z_i^T(t)\right)^{1-\beta} \left(z_i^T(t) - z_i(t)\right)^{\beta} \tag{1.8}$$

for $i = 1, 2$ and with $\lambda = 1$ and $\beta = 7/9$. This signal serves the purpose of a feedback mechanism, terminally attracting the network outputs towards the training signal $\mathbf{z}^T(t)$ to facilitate the learning [Toomarian 92], [Zak 89]. The amplitude of the teacher forcing signal (1.8) decreases gradually as the network outputs approach the target output sequence, practically vanishing at convergence such that the bias introduced by the forcing signal is eventually almost removed[1]. The feedback mechanism of teacher forcing is quite essential for trajectory learning in general, to provide an external stimulus that synchronizes the network dynamics with the training signal. We note that in the general learning case an external input $\mathbf{y}(t)$ is supplied with the training signal, in addition to the target output $\mathbf{z}^T(t)$. The training input $\mathbf{y}(t)$ then naturally serves as a reference to the target outputs for the network under learning, since obviously both signals are related and synchronized through the inherent dynamics represented by the training data. In most practical situations this reference provides sufficient stimulus to guide the learning process in absence of explicit learning feedback in the network through teacher forcing. Therefore, the example case of trajectory learning considered in the simulations is expected to provide a proper benchmark platform to test the stochastic learning method under more general operating conditions, representative of a much wider spectrum of training signals for a variety of different applications.

[1]For complete removal of the bias due to residual errors at convergence, the teacher forcing constant λ needs to be gradually attenuated along the learning process [Toomarian 92].

1.3 Analog VLSI Implementation

1.3.1 Learning in Neural Hardware

Analog VLSI implementation of neural networks with learning capabilities have received much attention lately, and several working analog chips or systems employing analog chips incorporating learning have been reported: [Furman 88], [Vittoz 89], [Alspector 89] among others. The advantages of using analog VLSI as technology medium for special-purpose neural network implementations include the inherent parallelism of the summing operations [Graf 89] and the compact size and low power consumption of the elements performing the local processing functions [Mead 89], [Andreou 91]. Some disadvantages attributed to analog VLSI, in general, are the limited available dynamic range and the strong requirements on precise matching between components in order to achieve a reasonable degree of accuracy. However, the sensitivity to precision of implementation depends strongly on the system-level specifications, and one of the intended properties of artificial neural networks is exactly "graceful" degradation of performance under errors in the implementation, such as offsets in a typical analog VLSI process, through redundancy in distributed representation [Rumelhart 86a]. For learning neural networks, in particular, the effects of offsets and mismatches in analog hardware implementations can be significantly reduced by "learning" the set of parameters directly on the implemented network, rather than programming the network with the parameter values obtained from learning off-line using a model of the implemented network [Smith 90], [Frye 91]. Parallel architectures for fast and efficient learning, embedded with the implemented network, can be obtained for certain classes of learning algorithms, mostly those based on incremental outer-product rules [Furman 88], [Cauwenberghs 92], [Donald 93], [Benson 93] and other local update algorithms [Vittoz 89], [Alspector 89], [Hochet 91], [Cohen 92], [Linares-Barranco 93]. On the other hand, the learning performance of such integrated learning networks may still be affected by the analog precision of the implemented learning functions themselves, depending on the nature of the algorithm used.

Virtually all of the learning hardware implementations of neural networks which have

been developed so far exclude dynamical effects in the inputs and outputs, basically for applications of pattern recognition and association of input-output pairs. Such applications typically deal with feedforward networks, for which the learning is fairly standardized [Rumelhart 86b] and easy to implement in VLSI. A few exceptions of VLSI learning architectures, capable of training recurrent networks as well, have been reported [Alspector 89], [Benson 93], [Jabri 92] though for learning fixed point attractors discarding transient response. While analog recurrent networks learning time-varying features offer a wide range of attractive applications, *e.g.*, for process control and identification of dynamical systems [Narendra 90], their implementation in special-purpose hardware has currently not been demonstrated. One factor seriously inhibiting implementation is the complexity of the available learning algorithms in a dynamic setting. Several versions of gradient descent algorithms for supervised learning in dynamical recurrent neural networks exist [Williams 89], [Pearlmutter 89], [Toomarian 92], [Schmidhuber 92], [Sun 92], [Baldi 93]. However, none of those support a scalable implementation for on-line (real-time) operation in a two-dimensional arrangement, as required for a typical VLSI process technology. In contrast, the perturbative stochastic algorithm studied in the next chapter provides a scalable and modular on-line implementation, described in further detail in Chapter 3 and demonstrated experimentally in Chapter 4.

1.3.2 Analog Volatile Storage

One of the most challenging problems faced by massively parallel analog information and signal processing in VLSI is the local storage of analog information, preferably in analog format. In particular for analog VLSI implementations of neural networks and other parameter-driven systems for signal processing, an integrated analog storage medium is desirable to locally represent the parameters p_i at the appropriate positions in the network architecture, and to retain their values over an extended period of time after last being written externally or updated by the adaptive scheme.

Unlike well developed methods for digital programmable storage using SRAM and

DRAM technology, the random-access long-term storage of analog information in VLSI has known significantly less progress in the past. Similar to digital memory storage, a voltage level on a capacitor encodes an analog memory value. However, the drift of that voltage due to leakage and noise affects the analog representation much more drastically than for a binary representation. The drift due to leakage is unavoidable if the storage capacitor is to be directly accessible for writing, in ohmic contact with surrounding circuitry. In contrast, a floating-gate storage capacitor, completely insulated by a surrounding oxide, avoids memory degradation due to leakage, but precludes direct write access for programming except by slow electron transport through the oxide. Most applications requiring direct write-access to analog voltages stored on an array of capacitors in VLSI have rather used off-chip digital storage and external D/A conversion to periodically refresh the programmed voltages. For large-scale storage, this method requires high-bandwidth off-chip communication and extra external components.

Local storage mechanisms for large-scale integrated programmable analog memories, that avoid the need of external storage and off-chip communication, have been explored recently [Terman 81], [Hochet 91]. The basic principle here is to effectively quantize the values stored in the memory, restricting their analog range to a finite set of discrete levels, and to periodically refresh the drifting values to the nearest discrete level. Therefore the analog values encode digital information while no explicit digital storage is required. Instead, the inherent digital information is stored in analog format on a memory capacitor and is repeatedly retrieved from the identified nearest discrete level. The quantization provides a certain excursion margin for the memory values between consecutive refresh operations, which retrieve the correct discrete level as long as the refresh rate is fast enough to counteract the effect of the drift. Basically, to ensure the stored value never escapes from one memory level toward another, the expected drift accumulated over one refresh interval should be considerably smaller than the separation between neighboring quantization levels. Therefore the maximum number of quantization levels that can be resolved by the analog memory depends on the time scale of the refresh intervals relative

to that of the nominal voltage drift.

Implementation of the quantization and refresh functions in VLSI basically involves activation of an A/D/A (analog-to-digital and digital-to-analog) conversion loop [Terman 81], which may be shared in sequence by multiple storage devices. Variants on this mode of implementation [Hochet 89], [Hochet 91], [Vittoz 91] provide a periodic signal which sequences through the complete set of discrete memory levels for direct comparison with the stored value in memory, identifying and copying the nearest level in a manner somewhat related to dual-slope A/D conversion. While either configuration is able to store an analog value over an extended period of time, occasional errors in the quantization and refresh, occurring in practice due to noise and analog VLSI imprecisions, have a catastrophic impact on the data retention of the memory. Indeed, a wrong classification of the nearest discrete analog level at quantization causes a complete loss of the stored information at the successive refresh operation. Furthermore, offsets occurring at refresh, due to incorrect replication of the identified level onto the storage device, increase the likelihood of a subsequent error in the quantization. The stringent requirements on precision in the implementation severely limit the analog resolution practically attainable with the dynamic storage method. An alternative realization, which incorporates error correction at the LSB level to mask quantization errors of limited magnitude, has been proposed recently [Lee 91]. However, the modified scheme involves extra resources for local storage and handling of one bit of information.

We have developed an alternative technique for dynamic multi-level storage, which avoids the sensitivity to quantization errors by specifying partial updates for the analog value, with small fixed size increments in the direction of the nearest discrete level, rather than completely substituting the current analog value at refresh [Cauwenberghs 93c]. Hence occasional errors in the quantization merely cause a small increment in the wrong direction, which is likely removed in subsequent cycles unless errors occur rather frequently. The modified refresh scheme therefore allows for reliable long-term storage at increased analog resolution. The details and further characteristics of the technique are described in

Section 3.2, and the description of a compact quantization element supporting fairly high resolution and bandwidth, using a bi-directional A/D and D/A converter, is presented in Appendix A.

1.3.3 A Unified Analog VLSI Framework

While the learning and the refresh of the parameter values p_i are two distinct processes which each address the system from a different perspective, the implementation of their functions needs to be coordinated since both interface with the same parameter storage medium. Essentially, both processes need to alternate, with the refresh activated when the learning is disabled, and the refresh disabled when the learning is activated. The need for switching between learning and retention phases of operation in an adaptive network arises since a constant source of training data for supervised learning cannot always be guaranteed. In particular, whenever on-line target signals are not available or the system is in an idle state, continued adaptation is detrimental to the performance of the network, and the parameter values need to be retained.

Since the learning and refresh processes alternate and both access the same storage device, it is not only possible but furthermore desirable to combine the local parameter update functions for both in one implementation architecture, partly sharing the same hardware. Fortunately, both the stochastic learning algorithm and the partial refresh method mentioned above specify the same type of incremental updates in the parameters, such that an efficient time-sharing of the update hardware is indeed feasible.

In Chapter 3, we present a simple scalable architecture implementing the stochastic perturbative algorithm for on-line supervised learning in dynamical recurrent neural networks, which includes integrated provisions for partial refresh of the parameter storage. The local functions of both learning and refresh are combined in a unified cell structure, of which instances are locally replicated in direct contact with the respective parameter storage devices, one for every parameter value in the network structure.

The scalable and modular architecture of the learning architecture owes largely to the

simple form of the stochastic perturbative algorithm, avoiding the complexity of on-line gradient-descent methods. In addition to the significant reduction in implementation complexity, the direct observation of the gradient avoids the offsets which may arise in a derived gradient estimate, due to mismatches between the assumed network model and its physical implementation. In fact, the direct approach further extends directly towards optimization of arbitrary parameter-driven dynamical systems, of which the dependence of the error index on the parameters and a model of the network does not need to be known or specified [Dembo 90].

As a demonstration of principle, we have implemented a small recurrent neural network with continuous-time dynamics, integrated with the learning circuitry on a CMOS chip, and have experimentally verified its learning performance with a simple trajectory learning task, representative of more general and useful applications for system identification and adaptive control. The structure of the learning network and its experimental performance are described in Chapter 4, which also includes test results obtained on a related chip with integrated quantization functions, demonstrating the robustness of the partial refresh method to attain long-term data retention at high analog resolution. By virtue of the scalable and modular architecture of the integrated learning and storage functions, the combination of the two chips presented here can be extended, with minor modifications to the internal structure of the cells, to accommodate applications of adaptive autonomous dynamical recurrent systems of very large dimensionality.

Chapter 2

Supervised Learning and Optimization

The scope of the present chapter is to explore functionally simple and robust, yet computationally efficient techniques for error-descent supervised learning and optimization in dynamical systems of which the characteristics are not necessarily known *a priori*. The study is motivated by experience gained from practical analog hardware implementations, and particularly addresses the limitations of analog VLSI systems in terms of the implementation accuracy and the strict causality of time-domain operations.

Section 2.1 briefly analyzes the shortcomings of gradient descent methods for real-time supervised learning in terms of computational complexity and sensitivity to model mismatches, motivating gradient-free alternatives. Section 2.2 investigates a stochastic parallel perturbation technique for supervised learning in dynamical systems, probing the dependence of the error functional (1.6) on the parameters on the network directly rather than calculating the components of the gradient from an assumed structural model of the network. The learning performance of the stochastic method is compared with that of standard gradient descent methods, in terms of convergence behavior, learning speed, and scaling properties. Finally, practical on-line schemes for real-time implementation of the stochastic method are presented in Section 2.3, for two distinct classes of applications each

with a different mode of operation. The first scheme aims at applications for analysis and prediction of time-varying sequences, and the second scheme applies to identification and control of unknown continuously running dynamical systems. Simulation results on the trajectory learning example of Section 1.2 are included to illustrate the statements made.

2.1 Gradient-Based Supervised Learning

The standard methods for supervised learning in dynamical neural networks minimize an MSE functional of the form (1.6) with error specification (1.5), by gradient descent of the error in the parameter space [Williams 89], [Pearlmutter 89], [Toomarian 92], [Baldi 93]. Gradient descent is an iterative method of function optimization, by means of incremental updates in the parameter values according to the locally steepest direction of error descent, given by the opposite direction of the gradient vector:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \eta \frac{\mathrm{d}\mathcal{E}^{(k)}}{\mathrm{d}\mathbf{p}} \quad . \tag{2.1}$$

Under reasonable assumptions on the continuity of $\mathcal{E}$ and at sufficiently small learning rate η, the sequence $\{\mathbf{p}^{(k)}\}$ converges to a local minimum of the error functional $\mathcal{E}$. Usually, with appropriate sizing of the network and a thoughtful formulation of the learning task, the existence of multiple minima can be avoided, and the found local minimum coincides with the desired global minimum. The issue of dealing with local minima falls beyond the scope here, rather we include practical guidelines to avoid those in dynamical systems and suggest a simple scheme to circumvent them in case their existence persists nevertheless. Instead, we address the computational complexity of supervised learning and the sensitivity of learning performance under poor conditions common in real-world environments, with regard to gradient-descent methods.

2.1.1 Algorithmic Complexity

Several versions of gradient-descent algorithms based on the error formulation (1.6) and the recurrent neural network structure (1.2) have been developed, each calculating the gradient in (2.1) from a different computational perspective. The different approaches, each arriving at the same value of the gradient, can roughly be divided in three categories, depending on the functionality of the learning procedure in terms of the timing of the computation and the required access to the network and training data. Off-line methods, such as error back-propagation through time [Werbos 90] and related variational methods [Pearlmutter 89], [Sato 90a], have superior computational efficiency. However, they need a backward integration pass to construct the gradient, which assumes a reversal of the time dimension during the learning process. Real-time implementation of these methods requires the storage of the intermediate values of the state variables, cumulative over the complete integration interval $[t_0, t_f]$. On-line methods significantly reduce the storage requirement by avoiding the backward integration step altogether. A matrix sensitivity formalism is used in [Williams 89] that allows all calculations to be performed in forward time with minimal storage; however, the computation complexity, in terms of number of operations required per gradient result, rises sharply with the dimension of the network. On-line alternatives that reduce the amount of computation to obtain the gradient have been proposed recently [Toomarian 92], [Schmidhuber 92], [Sun 92], which all lead to an equivalent improvement in computation efficiency over the latter method, at approximately equal levels of storage required. The performance of the various gradient descent methods in the three categories is summarized in table 2.1. In the scaling formulas, N denotes the number of neural state variables in the recurrent network, and L is defined as the interval length of the error functional (1.6) in terms of the number of integration time steps. An extensive review, comparing and analyzing the different methods, can be found in [Baldi 93].

For the purpose of the investigation here, in the context of real-time integration of learning methods in VLSI hardware and related implementation environments, we note

Table 2.1: Scaling properties of gradient descent methods for supervised learning in dynamical recurrent neural networks.

Method	Computational Complexity *	Memory Requirement **
Off-Line		
Pearlmutter (1989) Sato (1990) Werbos (1990)	$O(LN^2)$	$O(LN^2)$
On-Line		
Williams & Zipser (1989) Narendra & Parthasarathy (1991)	$O(LN^4)$	$O(N^3)$
Toomarian & Barhen (1992) Schmidhuber (1992) Sun, Chen & Lee (1992)	$O(LN^3)$	$O(N^3)$

* Number of operations per update, in batch-mode.
** Number of state variables used in the calculations.
N = Number of fully interconnected dynamical neurons in the network.
L = Integration length of the error functional, in number of time steps.

that the best on-line gradient descent method still implies an $\mathcal{O}(N^3)$ storage requirement, which is hard to realize in a scalable architecture given the intrinsic 2-D nature of a typical VLSI wafer fabrication process[1].

2.1.2 Dependence on Structural Model Specification

Other than the issue of scalable implementation, there are more disadvantages specific to the nature of gradient-descent in general, which motivate the search for a gradient-free alternative. The procedure to obtain the gradient from the network and training data assumes an explicit structural model for the network dynamics and the output filter, represented by $F(\mathbf{p}, \mathbf{x}, \mathbf{y})$ in (1.1) and $G(\mathbf{x})$ in (1.3) or (3b). If the topological structure of the implemented network does not exactly reflect the functional form assumed by the gradient formula, a bias will result in the learning process. The amplitude of this bias is difficult to estimate. Since the long-term dynamics of a recurrent network generally depends strongly on the parameter settings, the effect of the bias may be severe to the learning performance in certain cases, even if the mismatch between the network structure and the assumed model is relatively small. Such mismatch is unavoidable in analog implementations of the network structure and the learning circuitry, due to random offset and gain errors induced by impurities in the fabrication process. Digital implementations are usually robust to errors of this kind. Nevertheless, the functional implementation of the derivatives of F and G and integration of the error dynamics becomes demanding in order to satisfy the accuracy requirements.

Regardless of the issue of the bias due to model mismatches, the fact that the derivatives of F and G need to be known analytically to construct the gradient precludes applications where the inherent dynamical structure of the network F and the output filter G is not known *a priori* to the user. One example already mentioned is the case of direct adaptive control, for which the operator G can be considered to represent the dynamics of an un-

[1]It is unlikely that further developments will bring about an on-line, full-gradient method with a simple, scalable $\mathcal{O}(N^2)$ storage architecture. This is because the calculation of the gradient needs to account for the extended effect of the parameters over a long time range, involving in principle the complete history of the error propagation dynamics.

known plant driven by the state variables of the recurrent network. A more general case concerns the optimization of unknown (though observable) dynamical systems whose behavior is governed by a set of adjustable parameters, which need to be programmed to appropriate constant levels to establish a given desired dynamic response. For these important classes of generic optimization and control problems, gradient-descent methods are inadequate and an alternative, gradient-free technique is needed which minimizes the error through direct evaluations of the error on the physical network. Gradient-free methods also offer an attractive alternative for gradient-based methods in certain classes of optimization problems, where the structure of the dynamical system under optimization is well known and the derivatives can be derived to compute the gradient. The benefits of a gradient-free, model-independent method when explicit model knowledge is available include the robustness of learning performance in the presence of model mismatches in the physical network [Jabri 92], and the increased flexibility to rearrange the network structure, *e.g.*, when training modular systems of reconfigurable neural networks [Satyanarayana 92]. Obviously, the increased fault-tolerance and flexibility of a direct evaluation method necessarily comes at the expense of a loss in learning efficiency, since several evaluations on the network are required to extract the equivalent amount of information contained in the error gradient, which otherwise is constructed from a single network observation and prior knowledge on the network structure. Nevertheless, as will be demonstrated in the section below, a high degree of efficiency can still be maintained with a gradient-free method, when the samples of network performance evaluations are selected according to a random distribution of parallel perturbations in the parameters.

2.2 Stochastic Error-Descent Optimization

The gradient-free method we investigate for supervised learning in dynamical systems complies in format and in scope with stochastic approximation methods, for optimization of a scalar performance index in an unknown process by means of noisy observations of the performance in the parameter space [Robins 51], [Kushner 78]. The method applies to

a broad class of optimization problems, of which the origin and functional dependence of the performance index in the context of the application need not to be specified. In particular, the method applies to the performance index identified by the error functional (1.6), of which the internal dynamics of the network producing the observed error value is irrelevant to the optimization process itself [Cauwenberghs 93a]. Consequently, the analysis of the stochastic method and its convergence and scaling properties in this section can proceed mostly without further reference to the dynamics of the network and the form of the error functional, which we specify conform to (1.6) with given values for t_0, t_f and fixed initial conditions $\mathbf{x}_0$. The restrictive batch-mode format that this error specification implies on the timing of the network dynamics under learning, in accordance with the discussion in Section 1.2, will be relaxed in Section 2.3 where we will present on-line variants of the training method for real-time implementations.

2.2.1 Formulation

Let $\mathcal{E}(\mathbf{p})$ be a scalar error functional to be minimized, defined at any point $\mathbf{p}$ in the parameter space as the result of direct observation of an error measure on the physical system of interest, under application of the corresponding parameter values to that system. The stochastic method specifies incremental updates in the parameter values according to

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \mu\, \widehat{\mathcal{E}}^{(k)}\, \boldsymbol{\pi}^{(k)} \tag{2.2}$$

with the constant μ defining the learning rate, and with the scalar quantity

$$\widehat{\mathcal{E}}^{(k)} = \frac{1}{2}\left(\mathcal{E}(\widehat{\mathbf{p}}^{+(k)}) - \mathcal{E}(\widehat{\mathbf{p}}^{-(k)})\right) \tag{2.3}$$

obtained from two observations of the error measure $\mathcal{E}$ under complementary two-sided activation of a parallel perturbation $\boldsymbol{\pi}^{(k)}$ onto the parameter vector $\mathbf{p}^{(k)}$,

$$\widehat{\mathbf{p}}^{+(k)} \;=\; \mathbf{p}^{(k)} + \boldsymbol{\pi}^{(k)} \tag{2.4}$$

$$\hat{\mathbf{p}}^{-(k)} = \mathbf{p}^{(k)} - \boldsymbol{\pi}^{(k)} .$$

The perturbation values $\pi_i^{(k)}$ are selected at random from a given distribution, synchronously for all components i in parallel, and once for every iteration k. For optimal performance, we assume the components i of the perturbation vector, as well as different instances k of any single component, are mutually uncorrelated. In terms of the expectation value, this translates into the condition

$$E(\pi_i^{(k)} \pi_j^{(l)}) = \sigma^2 \delta_{ij} \delta_{kl} \tag{2.5}$$

with δ the Kronecker delta symbol (1 if both arguments are equal; 0 otherwise), and with σ representing the strength of the perturbation.

The observed differential error (2.3) yields an estimate for the derivative of the error function with respect to the parameters in the neighborhood of $\mathbf{p}^{(k)}$, along a single direction in parameter space given by the perturbation vector $\boldsymbol{\pi}^{(k)}$. Accordingly, the incremental update specified in (2.2) adjusts the parameter vector along that direction. In essence, the method implements a random-direction extension to the finite difference Kiefer-Wolfowitz algorithm for multivariate stochastic approximation [Kushner 78]. The key factor which contributes the relative increase in speed of the method over finite difference stochastic methods is the parallelism in the statistics (2.5) of the perturbations, which by uniformly exciting the system in all dimensions simultaneously allow for an accurate and complete estimate of gradient information through significantly less observations. Parallel stochastic perturbation methods more or less conform with the functional form outlined in Eqns. (2.2)-(2.4), have been investigated in a variety of different contexts. A simultaneous perturbation stochastic approximation (SPSA) method, for optimization in noisy settings for the observations, was analyzed in [Spall 92a] and showed a significant improvement in efficiency and quality of approximation over sequential perturbation finite difference methods. Recently, much attention was devoted to development and application of related model-independent algorithms for supervised learning in neural networks, under paral-

lel perturbation of the network parameters, mostly inspired by arguments of efficiency and fault-tolerance in practical hardware-oriented learning systems [Cauwenberghs 93a], [Alspector 93], [Flower 93]. Directly related to these methods are continuous-time learning variants, which use injection of white noise signals onto the parameters and time-averaged correlations between the performance index and the noise signals to continuously update the parameters [Dembo 90]. The continuous-time version supports fairly elegant analog VLSI architectures [Kirk 93], but is limited to steady-state optimization problems, since the underlying assumption on the correlations for the construction of the updates necessitates an instantaneous response of the network performance index to the time-varying perturbation signals [Dembo 90]. For optimization of dynamical systems, as considered here, the dynamics of the network response and the learning system assigning parameter values need to be decoupled. The discrete-time formulation of the updates (2.2), independent of the internal network dynamics giving rise to the observed values of the error under the specified parameter settings, satisfies this requirement. Issues of continuity in the real-time process of learning, in contrast with the discrete nature of the observations (1.6), will be covered in Section 2.3.

The differences in functional form between the various parallel perturbation methods for optimization and learning are quite subtle, mainly concerning the format in which the perturbations contribute to the values of the parameter updates in (2.2), which in turn varies the regularity conditions on the statistics of the stochastic perturbation vector. Unlike the random-direction format with multiplicative contribution of the perturbation components in (2.2) and in [Cauwenberghs 93a], most of the related methods [Spall 92a], [Alspector 93], [Flower 93] specify a component-wise division by the respective perturbations in the construction of the parameter updates instead, resembling a finite-difference gradient approximation when considering the components individually. The divisive formats places special requirements on the statistical distribution for the perturbations near the origin, to avoid potential singularities arising from contributions dividing by small values for the perturbation components. In particular, the distribution for the pertur-

bation components used in SPSA and related algorithms needs to contain finite inverse moments [Spall 92a]. In a comparative study of perturbative stochastic approximation methods [Chin 93], the divisive version of SPSA is found to be more efficient than the random-direction multiplicative version (2.2), but the difference in performance appears to depend on the form specified for the statistical distribution of the perturbations. For practical purposes, we specify a symmetric binary distribution $\pi_i^{(k)} = \pm\sigma$ with equal probability for both polarities, in which case the distinction between multiplicative and divisive contributions of the perturbation in the updates disappears completely (except for a trivial uniform scalar factor σ^2). The symmetric binary distribution is easy to implement, and seems to offer optimal performance for either divisive and multiplicative variants of (2.2).

The multiplicative version furthermore allows for a direct extension of the method to global optimization tasks, using uniformly distributed random perturbations of larger size to sample distant points in the parameter space. A random-direction method similar to (2.2)-(2.5) was proposed in [Styblinski 90] for global non-convex function optimization, with an annealing cooling scheme controlling the amplitude of the perturbations to obtain a gradually decreasing convolutive smoothing effect on the error functional. This global optimization scheme was found to be significantly more effective than simulated annealing methods [Kirkpatrick 83], [Szu 87] in terms of speed of convergence as well as quality of the asymptotic results, for a broad class of multi-extremal functions. Therefore, the method as it is used here for optimization in dynamical systems holds a potential to deal with difficult learning problems beyond what is achievable with gradient-descent methods, by manipulating the amplitude of the perturbations applied to the parameters along the learning process. Hard optimization error functionals, contaminated by spurious local minima or abrupt discontinuities due to bifurcations, are common in learning tasks involving complex oscillatory trajectories [Baldi 92]. In such cases the effective smoothing of the error functional through large-amplitude perturbations may serve to mask these irregularities in the minimization process. Nevertheless, some of the convergence problems due to the

structure of the error surface can be avoided from the start, through proper choice of the timing in the error specification as will be shown in Section 2.3.

2.2.2 Convergence Properties

We present some important convergence properties of the stochastic method (2.2)-(2.4) below, with reference to gradient descent methods. For brevity, we adopt the formalism of the analysis in [Cauwenberghs 93a], augmented to account for the symmetrical two-sided activation by the perturbations in (2.3). A thorough mathematical analysis of related convergence properties of parallel perturbation, random-direction, and component-wise finite difference methods for stochastic approximation can be found in [Spall 92a] and [Chin 93]. The stochastic approximation methods considered therein specify a decaying sequence for the gain μ in the parameter updates, to gradually remove residual fluctuations in the parameter values due to the observation noise near convergence. In contrast, we consider a constant value for the learning rate μ in (2.2), which is more appropriate for real-time adaptive implementations continuously updating the parameters, where most recent information on the performance error $\mathcal{E}$ is more relevant than distant information in the past. Especially in non-stationary environments, fast adaptation to structural changes in the dynamics of the system is often a requirement. In principle, the amplitude of the learning rate μ can be set accordingly to that purpose, with a greater value of μ corresponding to a faster response to changing conditions. However, an increased level of the learning rate μ increases the amplitude of the parameter fluctuations due to the observation noise in $\mathcal{E}$ causing a residual error persisting at convergence, such that the chosen level for μ needs to compromise both considerations. In the convergence analysis below, we assume a stationary process at sufficiently low levels of observation noise, such that the value for μ can be freely chosen over a wide range according to other considerations. An important additional factor in the choice of μ is the stability of the parameter update process with regard to the structure of the error surface, defining a limit on the convergence speed as will be shown below. Similar stability considerations apply to any other incremental update

method as well, allowing a comparison of relative performance between the stochastic method and other methods, including gradient descent.

Mean Path of Error Descent in Parameter Space

Under small uncorrelated perturbations with uniform variance σ^2 in accordance with (2.5), the stochastic method performs steepest descent of the error in the parameter space on average. Formally, for small perturbations $\pi_i^{(k)}$ we expand the differential observed error in a Taylor series around $\mathbf{p}^{(k)}$:[2]

$$\widehat{\mathcal{E}}^{(k)} = \sum_{j=1}^{P} \frac{\partial \mathcal{E}}{\partial p_j} \pi_j^{(k)} + o(|\boldsymbol{\pi}^{(k)}|^3) + n^{(k)} \quad , \tag{2.6}$$

with P the number of parameters, $\partial \mathcal{E}^{(k)}/\partial p_j$ the components of the error gradient evaluated at $\mathbf{p}^{(k)}$, and $n^{(k)}$ the observation noise corresponding to the error evaluations $\mathcal{E}(\widehat{\mathbf{p}}^{+(k)})$ and $\mathcal{E}(\widehat{\mathbf{p}}^{-(k)})$. Substituting (2.6) into (2.2), the incremental parameter updates can be expressed as:

$$\begin{aligned} \triangle p_i^{(k)} &= p_i^{(k+1)} - p_i^{(k)} \\ &= -\mu \sum_{j=1}^{P} \frac{\partial \mathcal{E}}{\partial p_j} \pi_i^{(k)} \pi_j^{(k)} \\ &\quad - \mu\, o(|\boldsymbol{\pi}^{(k)}|^3)\, \pi_i^{(k)} - \mu\, n^{(k)} \pi_i^{(k)} \quad . \end{aligned} \tag{2.7}$$

On average and under the orthogonality conditions of (2.5) for the perturbation statistics, the increment in the parameter vector (2.7) reduces to

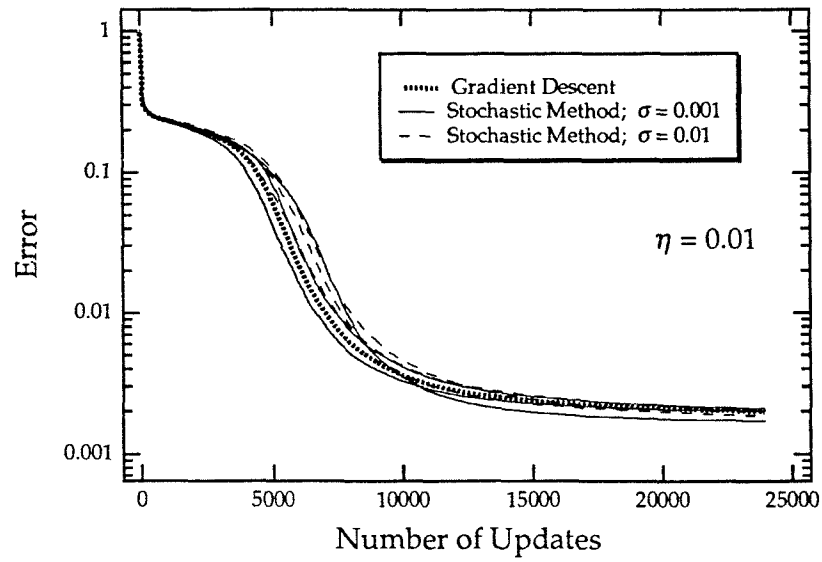
$$\mathrm{E}(\triangle \mathbf{p}^{(k)}) = -\mu\sigma^2 \frac{\mathrm{d}\mathcal{E}}{\mathrm{d}\mathbf{p}} + \mu\, o(\sigma^4) \quad . \tag{2.8}$$

[2]The $o(.)$ terms denote the nonlinear components of the Taylor expansion. In general, the definition of the $o(.)$ symbol in (2.6) and in subsequent formulas is as follows: for every $\chi > 0$ there exists a constant $\alpha > 0$ such that $|o(f(x))| < \alpha |f(x)|$ for $|x| < \chi$.

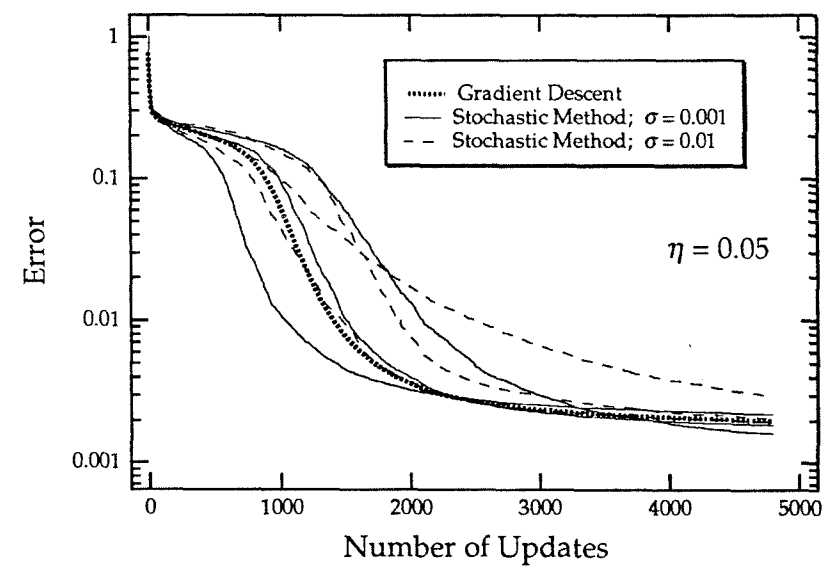
The absence of noise-related terms in (2.8) follows from the statistics of the observation noise $n^{(k)}$ in (2.7) which is assumed uncorrelated with the applied perturbations $\pi_i^{(k)}$, that is $\mathrm{E}(n^{(k)}\pi_i^{(k)}) = 0$. The expression for the incremental update (2.8) asymptotically approaches that for gradient descent as the perturbation strength σ decreases, with an effective gradient descent learning rate of $\eta_{\text{eff}} = \mu\sigma^2$ in accordance with (2.1). Hence, on average the path of the parameter vector under iteration of the stochastic updates follows the steepest path of error descent in parameter space, as with a strict gradient descent method. The fluctuations of the parameter updates (2.7) with respect to their average (2.8), due to the observation noise and the randomness of the perturbations, give rise to diffusion in the error descent process around the mean path of steepest descent.

Figure 2.1 illustrates the similarity in error descent profile on a macroscopic scale between the stochastic method and gradient descent, showing the simulated error profile under training of the network with the stochastic method for different values of μ and σ, in comparison to gradient descent under equivalent conditions ($\eta = \mu\sigma^2$). The simulations here and in the rest of the section use the network structure and the training signal from the Lissajous dynamic trajectory example defined in Section 1.2, for an error functional (1.6) with $t_0 = 0, t_f = T = 2\pi$, and fixed initial conditions x_{0i} in the $[-0.1, 0.1]$ interval. The resemblance between the stochastic and the gradient descent error profiles in Figure 2.1 is particularly clear at low values for σ and η_{eff}. The degradation of convergence at larger values of η_{eff} for the stochastic method relative to gradient descent is caused by diffusion and local instabilities in the update process, as will be evident from the analysis below.

The above conditions imposed on the statistics and amplitude of the perturbations, under which the stochastic method closely follows a gradient descent characteristic, are not essential to warrant convergence. Relaxation of these conditions does not necessarily cause a loss in performance. The stochastic method is still guaranteed to converge towards local minima in the more general case of cross-coupled and non-uniform statistics in the perturbations, violating condition (2.5). Strictly, for convergence it is only required that the components of the perturbation vector be linearly independent. Stated otherwise, none of



(a)



(b)

Figure 2.1: Error descent profiles for trajectory learning sessions with gradient descent and several runs of the stochastic method. *(a)* Effective learning rate $\eta = 0.01$. *(b)* $\eta = 0.05$.

the components are allowed to depend on other components in *deterministic* fashion. Formally, with the matrix $c_{ij} = E(\pi_i^{(k)}\ \pi_j^{(k)})$ representing the expected correlations between simultaneous perturbation components, the expression for the mean incremental updates (2.8) transforms into:

$$\mathrm{E}(\triangle \mathbf{p}^{(k)}) = -\mu\, \mathbf{c}\, \frac{\mathrm{d}\mathcal{E}}{\mathrm{d}\mathbf{p}} + \mu\, o(\sigma^4)\ . \tag{2.9}$$

If the perturbation components are linearly independent, the matrix $\mathbf{c}$ is non-singular and consequently is also strictly positive-definite. Then iteration of (2.9) leads to a local minimum of $\mathcal{E}$, albeit not necessarily along the steepest path of descent. The pre-conditioning of the gradient by the correlation matrix $\mathbf{c}$ slows down convergence in general, though it may actually prove beneficial in certain cases[3].

The restrictive condition on the size of the perturbations can be relaxed as well. A larger perturbation amplitude, as mentioned earlier, effects a virtual smoothing of the error functional [Styblinski 90], offering a convenient means of escaping local minima, discontinuities and other singularities in the error surface. An annealing scheme as in [Styblinski 90], gradually decreasing the perturbation amplitude along the optimization process, is then necessary to remove the residual bias near convergence.

Error Descent Fluctuations

The stochastic nature of the parameter updates under learning causes the parameter values to fluctuate around the mean path of steepest error descent. Of the two main contributive factors inducing these fluctuations, the randomness of the perturbations and the noise in the network observations, only the latter really affects the learning accuracy, provided the perturbation amplitude and learning rate are reasonably small. The random perturbations, on the other hand, bias the statistics of the parameter updates strictly in favor of a decrease in error. In particular, for small perturbations, for a small learning rate, and in absence of observation noise, the error $\mathcal{E}^{(k)}$ always decreases under an update (2.2) regardless of the

[3]For instance, in analogy with second-order Newton methods for gradient descent learning, a correlation matrix $\mathbf{c}$ proportional to the inverse of the local Hessian $\partial^2\mathcal{E}\ /\ \partial p_i \partial p_j$ would lead to a more uniform and superior convergence behavior provided the optimization error functional is convex-shaped.

particular instance of the perturbation vector used in the update. Formally, the incremental error between consecutive parameter updates can be written as

$$\begin{aligned} \triangle\mathcal{E}^{(k)} &= \mathcal{E}(\mathbf{p}^{(k+1)}) - \mathcal{E}(\mathbf{p}^{(k)}) \\ &= \sum_{i=1}^{P} \frac{\partial \mathcal{E}}{\partial p_i} \triangle p_i^{(k)} + o(|\triangle \mathbf{p}^{(k)}|^2) \end{aligned} \tag{2.10}$$

and, with the aid of (2.7), further expanded into

$$\begin{aligned} \triangle\mathcal{E}^{(k)} &= -\mu \sum_{i=1}^{P} \sum_{j=1}^{P} \frac{\partial \mathcal{E}}{\partial p_i} \pi_i^{(k)} \frac{\partial \mathcal{E}}{\partial p_j} \pi_j^{(k)} + o(|\triangle \mathbf{p}^{(k)}|^2) \\ &\quad - \mu \sum_{i=1}^{P} \frac{\partial \mathcal{E}}{\partial p_i} \pi_i^{(k)} \left(o(|\boldsymbol{\pi}^{(k)}|^3) + n^{(k)} \right) . \end{aligned} \tag{2.11}$$

The nonlinear terms in (2.11) depend on the perturbation strength, the learning rate and the structure of the error profile in the parameter space. In the limit of $\sigma \to 0$ and $\mu \to 0$ (hence $\boldsymbol{\pi}^{(k)} \to 0$ and $\triangle \mathbf{p}^{(k)} \to 0$), and in absence of observation noise $n^{(k)} \to 0$, the expression for the incremental error (2.11) reduces to a quadratic form

$$\triangle\mathcal{E}^{(k)} \approx -\mu \, (\sum_{i=1}^{P} \frac{\partial \mathcal{E}}{\partial p_i} \pi_i^{(k)})^2 = -\mu \left| \frac{\mathrm{d}\mathcal{E}}{\mathrm{d}\mathbf{p}} \cdot \boldsymbol{\pi}^{(k)} \right|^2 \leq 0 \tag{2.12}$$

yielding a decrease in error for every learning update[4]. Under finite non-zero values for the perturbation strength σ and the learning rate μ, the residual error terms in (2.11) account for persistent fluctuations in the parameter updates. However, the amplitude of the effective "perturbation noise" remaining at convergence depends strongly on the value chosen for σ, and can be significantly reduced by decreasing the value of σ to the extent allowed by the intensity level of the observation noise $n^{(k)}$. In particular, from expression (2.7) it can be observed that the portion of the parameter fluctuations at convergence due to the finite perturbation amplitude σ scales as $o(\sigma^4)$.

[4]We note that the property of a decrease in error under individual updates does not hold in case a divisive formula is specified for the updates (2.2) as discussed earlier, unless a binary symmetric distribution is used for the perturbations ($\pi_i{}^{(k)} = \pm\sigma$).

The effective decrease of the value for the error functional under any single (sufficiently small) instance of the perturbation vector comes at little surprise, since it is programmed to that purpose in the formula for the updates (2.2). Indeed, the algorithm iteratively updates the parameters in a single random direction of the parameter space, after probing the dependence of the error on the parameters in that particular direction and adjusting the size of the update vector accordingly. Consequently, the effective decrease in error achieved by an update in a given direction depends on the degree of alignment between the chosen direction and the local error gradient, as can be verified from (2.12). In particular, the most efficient updates are obtained along the direction of steepest descent, parallel to the gradient. In contrast, updates along directions orthogonal to the gradient fail to contribute a decrease in error. Nevertheless, since an increase in error cannot occur and not all instances of the perturbation vector can possibly be orthogonal to the gradient[5], the iteration of the learning updates gives rise to a strict decrease of the error on a global scale, and hence the algorithm converges towards a local minimum of the error function just like a gradient descent method.

Figure 2.2 illustrates the statistics of the error fluctuations, obtained from simulations of training sessions on the network with the dynamic trajectory, using the same network settings as before. The graph in Figure 2.2 shows the frequency distribution of the error decrements for different values of the perturbation strength σ and the learning rate μ, in absence of observation noise. The error decrements in the simulated distribution are normalized relative to their corresponding expected values, which from (2.10) and (2.8) are obtained as $-\mu\sigma^2 \, |\mathrm{d}\mathcal{E}/\mathrm{dp}|^2$ and therefore conform to the values of the equivalent decrements under gradient descent[6]. As expected from the above arguments, the error decreases in most of the instances. In addition, the shape of the distributions in Figure 2.2

[5]If statistically all instances of the perturbation vector are orthogonal to one or more directions in the vector space, then the perturbation components are not linearly independent. Formally, the correlation matrix $\mathbf{c}$ becomes singular, and (2.9) will not converge to a minimum of the error. The minimization of the error is then effectively restricted to the subspace excluding the "dead" dimensions in parameter space.

[6]The purpose of the normalization is to isolate the effect of the perturbation statistics in the distribution, avoiding interference from the structure of the error surface. Under normalization, the profile of the frequency distribution is to a large extent independent of the location in parameter space, as verified through simulations and confirmed by theory.

clearly shows a dominant concentration for particularly small values of the error decrement, with a sharp edge at the origin where the distribution vanishes. The asymmetry of the distribution around the expected value of error decrements is a direct consequence of the geometrical distribution of random directions in the parameter space, with reference to the gradient vector. Since most random directions are nearly orthogonal to the gradient, the largest fraction of updates contribute rather little decrease in error. The specification of random directions for the updates by the stochastic method may seem particularly inefficient. Nevertheless, as will be shown later below, the method achieves a fair degree of efficiency through other factors relevant to optimization in dynamical systems, in particular the low computational complexity of constructing the updates without explicit use of gradient information. In fact, the method can be augmented with features to boost the update efficiency as well[7], at the expense of an increase in implementation complexity.

At higher levels of the perturbation strength or the learning rate, the error undergoes occasional upward transitions in the error, as testified in Figure 2.2 by the smearing of the distribution around the origin under an increasing value of μ. The upward transitions are caused by strong nonlinearities in certain areas of the parameter space. Upward transitions in the error, when induced by a large value of the learning rate, may trigger instabilities in the error descent minimization process. The impact of the learning rate on the intensity of the instabilities and the corresponding limits imposed on the learning speed form the subject of the analysis below.

Relative Convergence Rate

A simple derivation of the optimal convergence rate by the stochastic learning method, based on physical arguments, follows below. The derivation allows to formulate a fair estimate of the maximum attainable speed of the stochastic method, relative to that of gradient descent and other incremental update methods, at low levels of observation

[7]Possible extensions include the addition of a momentum term in (2.2) [Styblinsky 83], [Styblinski 90], or an adaptive biasing of the statistics of the perturbation components towards the direction of the local gradient.

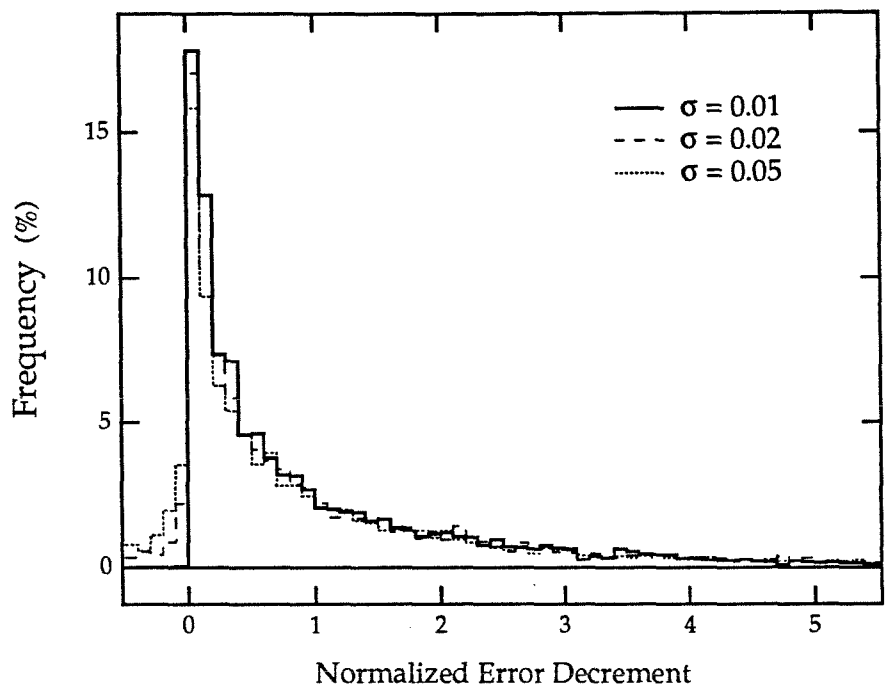


Figure 2.2: Frequency distribution of the error decrements under updates with the stochastic method, relative to gradient descent.

noise[8].

The maximum attainable learning speed is achieved at the highest value for the learning rate μ still supporting stable dynamics of the parameter increments. The error descent process $\triangle\mathcal{E}^{(k)}$ becomes unstable, causing an increase in error and possibly a runaway condition, when the amplitude of the parameter updates $\triangle\mathbf{p}^{(k)}$ in (2.10) exceeds a certain value. The physical limit on $|\triangle\mathbf{p}^{(k)}|$ depends mainly on the nonlinearity of error functional in the neighborhood of $\mathbf{p}^{(k)}$, and not as much on the specifics of the algorithm used to generate the increments. Since the increment amplitude can be controlled directly by the value for the learning rate, which in turn determines the speed of convergence, the limit on the increment amplitude $|\triangle\mathbf{p}^{(k)}|_{\max}$ defines the maximum attainable learning speed supported by the algorithm. The optimal value for the learning rate, through $|\triangle\mathbf{p}^{(k)}|_{\max}$, strongly depends on the nonlinear shape of the error functional $\mathcal{E}(\mathbf{p})$ and the location of the parameter vector $\mathbf{p}^{(k)}$ on the error surface[9]. Nevertheless, since the maximum value for the increment amplitude $|\triangle\mathbf{p}^{(k)}|_{\max}$ applies as well to other minimization methods specifying incremental updates in the parameters, the performance of the stochastic method can be compared, in relative terms, to that of other incremental methods applied to minimizing the same error functional $\mathcal{E}(\mathbf{p})$.

For gradient descent (GD), the intensity of the incremental updates is obtained from (2.1) as

$$|\triangle\mathbf{p}^{(k)}|^2_{\mathrm{GD}} = \eta^2 \left|\frac{\mathrm{d}\mathcal{E}}{\mathrm{d}\mathbf{p}}\right|^2 . \tag{2.13}$$

The corresponding intensity for the stochastic error descent (SD) method is derived from (2.2), yielding

$$|\triangle\mathbf{p}^{(k)}|^2_{\mathrm{SD}} = \mu^2\, |\boldsymbol{\pi}^{(k)}|^2\, (\widehat{\mathcal{E}}^{(k)})^2 . \tag{2.14}$$

Under mutually uncorrelated and uniform perturbation statistics (2.5), the intensity of the

[8] An elaborate treatment of the performance of similar parallel methods for stochastic approximation under noisy conditions can be found in [Chin 93].

[9] The variation of optimal learning rate across the parameter space suggests an adaptive setting of the learning rate $\mu^{(k)} = \mu(\mathbf{p}^{(k)})$, locally optimized for maximum efficiency along the learning process [Styblinsky 83], [Yu 93].

perturbation vector $|\boldsymbol{\pi}^{(k)}|^2$ in (2.14) reaches toward the central limit

$$|\boldsymbol{\pi}^{(k)}|^2 = \sum_{i=1}^{P} {\pi_i^{(k)}}^2 \to P\,\sigma^2 \tag{2.15}$$

as $P \to \infty$[10]. The other term in (2.14), corresponding to the differential perturbed error $\widehat{\mathcal{E}}^{(k)}$, is specific to the instance of the perturbation. More relevant than the stochastic fluctuations of $(\widehat{\mathcal{E}}^{(k)})^2$ in (2.14) are its statistical properties. For a meaningful comparison with gradient descent, we therefore consider the expected value of (2.14). Substituting the expansion (2.6) and the limit (2.15) in (2.14), and using the expectation value for (2.5),

$$E(|\triangle \mathbf{p}^{(k)}|^2_{\text{SD}}) \approx P\,\mu^2\sigma^4 \left|\frac{\mathrm{d}\mathcal{E}}{\mathrm{d}\mathbf{p}}\right|^2 . \tag{2.16}$$

The similarity between (2.13) and (2.16) is not surprising, since on average the stochastic method performs gradient descent with an effective learning rate $\eta_{\text{eff}}^{\text{SD}} = \mu\sigma^2$. The optimal values for the learning rates of both methods can be compared directly by combining (2.13) and (2.16)[11], yielding to first order

$$\eta_{\text{max}} \approx \sqrt{P}\,\mu_{\text{max}}\sigma^2 = \sqrt{P}\,\eta_{\text{eff max}}^{\text{SD}} . \tag{2.17}$$

Hence the convergence speed of the parallel stochastic method, under optimal settings, is about a factor $\sqrt{P}$ slower than that of optimally tuned gradient descent.

Judging merely by considerations of raw speed in the context of (2.17), in terms of the number of updates needed to reach convergence, the stochastic method performs rather poorly in comparison to rigorous gradient descent methods. There are however other considerations which, depending on the application environment, may dominate the comparison of performance in favor of the stochastic method. An important factor therein, as mentioned before, is the complexity of the computations involved in obtaining

[10]The approximate limit (2.15) becomes exact, regardless of the size of P, for a symmetric binary distribution of the perturbations ${\pi_i}^{(k)} = \pm\sigma$.

[11]For simplicity, the spread of the distribution of $|\triangle \mathbf{p}^{(k)}|^2_{\text{SD}}$ around its mean is not considered in the derivation of the optimal learning rates.

values for the incremental parameter updates, affecting the learning speed as well. In particular, for applications in real-time environments with recurrent dynamics as in (1.1), the stochastic method offers a net computational speed comparable to that of gradient descent[12], in addition to the advantages specific to the model-free nature of the method outlined in Section 2.1. The reason behind the favorable net efficiency of the stochastic method, despite the reduction factor in (2.17), follows from the simple requirements on the amount of information needed from the network to generate the updates (2.2). According to (2.3) and (2.4), the algorithm only needs two evaluations of the scalar error index $\mathcal{E}(\mathbf{p})$ on the network per update. On the contrary, complete information about the local gradient of the error $\mathrm{d}\mathcal{E}/\mathrm{d}\mathbf{p}^{(k)}$ needs to be supplied for every update with gradient descent. In this context, the scaling factor $\sqrt{P}$ in relative speed can be interpreted as follows: the stochastic method only requires $2\sqrt{P}$ scalar observations (two error evaluations per update) to obtain the necessary information to construct a parameter update of equivalent quality as that of an update constructed using full gradient information with gradient descent. Since direct observation of the full gradient on the network would require at least $P + 1$ scalar observations of the error[13], the stochastic method is fairly efficient in that respect. In fact, finite difference (FD) methods [Kushner 78], [Jabri 92] effectively perform gradient descent in a cyclic sequence comprising P steps, in each step perturbing a single parameter which is then updated according to the probed gradient component. Formally, finite difference methods comply to expressions (2.2)-(2.4) for the stochastic method, but instead of (2.5) specify perturbations ${\pi_i}^{(k)} = \sigma\, \delta_{ij^{(k)}}$ with $j^{(k)} = k$ modulo P. Since a full effective gradient cycle using finite differences spans P iterations, the finite difference method is in principle

[12]The scaling mechanism causing a comparable net efficiency for both methods will be demonstrated in detail further below.

[13]Clearly, gradient descent methods derive the value of the gradient assuming a given network model using vectorial information gathered from a single network evaluation, rather than probing the gradient components directly on the network. See the discussion regarding model dependence in Section 2.1.

a factor P slower than gradient descent under optimal tuning of the learning rates[14],

$$\eta_{\max} \approx P\, \eta^{\mathrm{FD}}_{\mathrm{eff}\,\max} \ . \tag{2.18}$$

Therefore, the parallel stochastic method gains in speed over finite difference methods (in principle by a factor $\sqrt{P}$) by specifying uniform, mutually uncorrelated random components (2.5) for the perturbations rather than a sequential activation of individual perturbation components.

A few simplifying assumptions were made in the above analysis, ignoring second-order effects specific to the shape of the error functional. In practice, to some extent the relative learning performance of the different incremental methods is expected to vary with the particular application, beyond the simple dependence on the dimensionality P. To validate the analysis in the context of learning in dynamical recurrent systems, we verified the relationship between optimal learning rates, on the same trajectory learning example used before, for simulation runs on the network with gradient descent, the parallel stochastic error-descent method, and the sequential finite-difference method. The values used for the learning rates with the stochastic and finite-difference methods are given in terms of the corresponding effective rates η_{eff} in the mean-path gradient descent approximation (2.8), therefore encoding a measure for the learning speed consistent for all three methods. To provide a quantitative measure for the degree of instability in the updates at a given learning speed, the update efficiency is recorded as a function of the effective learning rate, shown in Figure 2.3 for the three incremental methods. The update efficiency at a given point $\mathbf{p}$ is defined as the ratio of the effective decrease in error $\triangle\mathcal{E}^{(k)}$ over its anticipated value $\triangle\mathcal{E}^{(k)}{}_{\mathrm{ant.}}$ obtained by ignoring the nonlinearities in the error surface. The approximate linear projection $\triangle\mathcal{E}^{(k)}{}_{\mathrm{ant.}}$ is derived from (2.10) in conjunction with (2.1) and (2.2), by ignoring the nonlinear terms in $\mathcal{E}(\mathbf{p})$ corresponding to the nested Taylor

[14]Unlike GD, the FD method produces an individual update for every component of $\Delta\mathbf{p}^{(k)}$ in sequence. Therefore, the optimal learning rate for the FD method may be effectively higher than that for GD, and consequently the speed reduction factor may be less than P. The factor P remains in effect if one may assume that the stability limit on the increment amplitude in (2.10) applies to the components of the update vector *individually*, that is $|\Delta p_i{}^{(k)}| < |\Delta p_i{}^{(k)}|_{\max}$, $\forall i$ to ensure stability.

expansions with respect to the perturbations $\pi_i^{(k)}$ and the parameter updates $\triangle p_i^{(k)}$. For consistency in the results, the update efficiency for the stochastic method and the sequential method is averaged over several incremental updates with different instances of the perturbation vector. Figure 2.3 (a) contains a recording of the global update efficiency, obtained by averaging contributions from random samples of $\mathbf{p}$ across the parameter space ($-1 \leq p_i \leq 1$). Likewise, a local recording of the update efficiency, obtained using the parameter values at convergence which correspond to a minimum of the error, is given in Figure 2.3 (b). As expected, the efficiency degrades for all methods under an increase in the learning rate, with the highest effective rate achievable by the gradient descent method, and the lowest rate by the finite-difference method. Since at the location of a minimum the linear terms (*i.e.*, the first-order derivatives) in the error vanish and the nonlinear terms are relatively more significant, the degradation of the update efficiency at convergence in Figure 2.3 (b) is more severe than the global average in Figure 2.3 (a)[15]. While the region near the convergence values for the parameters is obviously important for practical purposes, there are in general other regions in the parameter space with relatively strong nonlinearities as well, with a stronger than average effect of the learning rate on the frequency and intensity of the update instabilities. Despite the local variations in instability levels, the relative scaling of the critical learning rates between different methods at equal levels of the update efficiency seems to be rather constant. From the uniform displacement (on a logarithmic scale) of the curves in Figure 2.3 (b), an estimate of the relative factors for critical learning rate with gradient descent, the parallel stochastic method, and the sequential finite-difference method, $\eta_{\text{eff}}^{\text{FD}} : \eta_{\text{eff}}^{\text{SD}} : \eta^{\text{GD}}$, yields $0.08 : 0.4 : 1$. The obtained numbers do not quite match the theoretical predictions in (2.17) and (2.18) which for $P = 42$ yield $0.024 : 0.15 : 1$. The discrepancy reflects the simplifying assumptions in the above analysis regarding the statistics and directionality of the incremental updates by the different methods, and suggests that in practice the relative performance of the stochastic

[15] An additional effect of the dominating nonlinearities in the error at the location of a minimum is the further degradation of the update efficiency at higher values of the perturbation amplitude σ for the stochastic and sequential methods, as demonstrated in Figure 2.3 (b).

and finite-difference methods is actually better than that predicted by the initial estimate using simple scaling arguments in (2.17) and (2.18)[16].

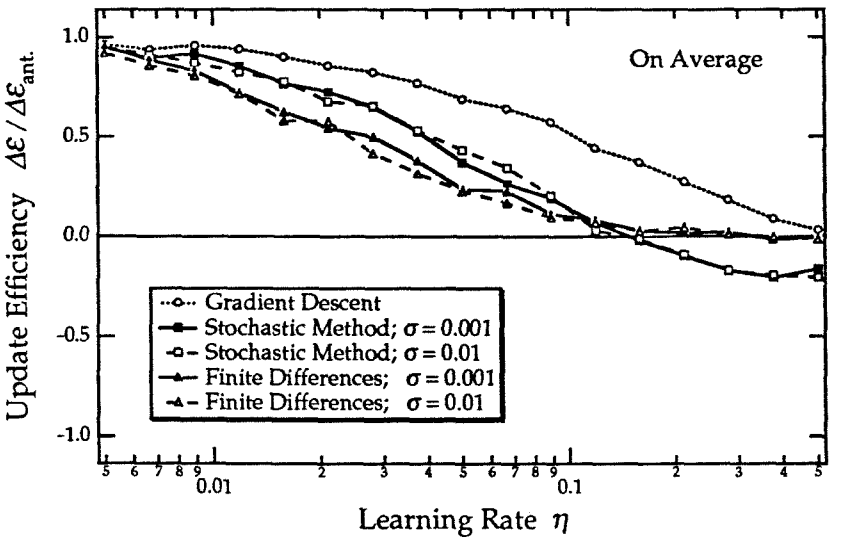
2.2.3 Complexity and Scaling Issues

The above analysis of the convergence properties of the stochastic algorithm, in relation with other incremental methods, is general in scope and does not account for factors specific to the system under optimization. In the following, we investigate the net computational complexity of the stochastic method, specifically for optimization in dynamical recurrent systems. By comparing the complexity of the stochastic method with that for the most efficient on-line gradient descent implementation under equivalent conditions, it will be shown that computationally the stochastic method, while functionally quite simpler, scales with the network size P equally well and possibly better than on-line gradient descent.

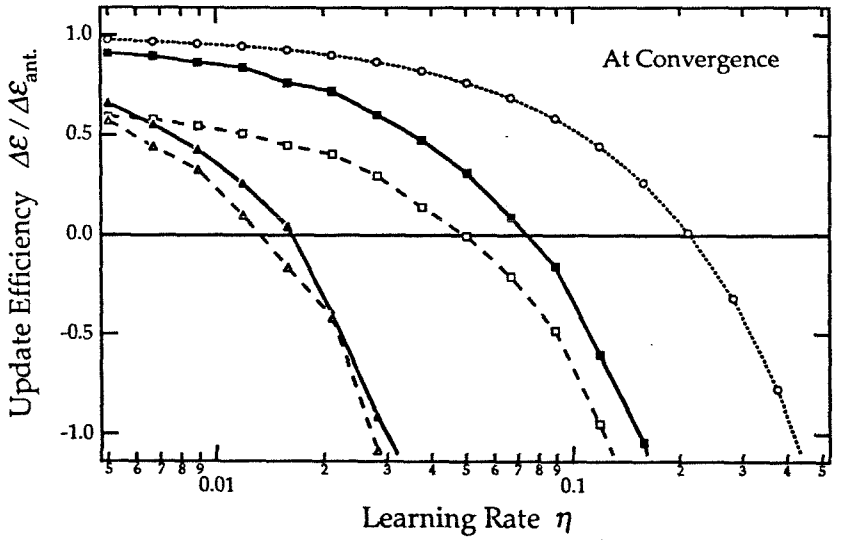
The computational complexity of an algorithm can be generally defined as the number of elementary operations required to complete the computational task. In the context of the stochastic algorithm and other incremental optimization methods, the number of operations is ill-defined since the iterative process towards convergence takes in principle an infinite number of steps to achieve the result at unlimited precision. Therefore, the complexity of the stochastic method cannot be defined without specifying a level of accuracy, unless relative to another method under identical conditions. The analysis of the convergence properties above provides a relationship between the stochastic algorithm and other incremental optimization methods[17]. In particular, expressions (2.17) and (2.18) formulate the convergence speed of the parallel stochastic method and the finite-difference method relative to gradient descent. The relative measure of speed as used in (2.17) and (2.18) refers to the number of updates required for a given decrease in error, and does not account for the amount of computation needed to obtain the values of the update

[16]Since the empirical observation of the relative performance for the different methods in Figure 2.3 is conducted on a particular network with fixed size P, technically no conclusion can be drawn with regard to the scaling with P.

[17]In comparing different incremental methods for optimization in dynamical systems, we so far assumed a batch format for the error functional (1.6), identical for all methods. The natural extension to real-time implementations follows in the next section.



(a)



(b)

Figure 2.3: Update efficiency as a function of effective learning rate, for three incremental optimization methods. *(a)* Global averaged recording. *(b)* Local recording at convergence.

increments. A complete evaluation of the computational complexity for the three methods therefore needs to include the number of operations per update for each method, as well as the relative convergence factors in (2.17) and (2.18).

For gradient descent, the computational complexity per update depends on the model of the network. For dynamical recurrent systems, the complexity becomes obviously more elaborate than that for model-free methods. This is because for dynamical recurrent systems extensive operations are required to calculate the gradient vector from the network outputs and the given model information, whereas for model-free methods the update vector can be constructed from a few direct observations of the error on the network, by means a formula independent of the internal model specifics. Different approaches to calculate the gradient of the error (1.6), each with a different computational complexity, were discussed in Section 2.1 and summarized in table 2.1. As stated before, off-line variational methods to construct the gradient offer the lowest complexity, but are not practical for real-time implementation. Among several versions of on-line gradient methods, the best complexity achieved so far scales as $\mathcal{O}(N^3)$ for a fully connected recurrent network with N neurons. The corresponding implementation requires at least $\mathcal{O}(N^3)$ storage as well, and the instructions are functionally fairly complex, involving inverse matrix operations.

The computational complexity per update for the parallel stochastic method and the sequential finite-difference method can be derived without reference to the model of the network and embedded system, since both methods are model-free. For a system containing P parameters being adjusted by either of the two model-free methods, the operations required for every update basically consist of two global error evaluations (2.3) on the system, and P scalar updates according to (2.2). Additionally, the stochastic method requires the generation of P random perturbation components, while the sequential finite-difference method activates a single perturbation component at a time in deterministic fashion. Each observation of the error (2.3) on the system, which implicitly depends on the P parameters through the network outputs, usually involves a computational effort of

order $\mathcal{O}(P)$ in P[18]. Likewise, the remaining operations for constructing the updates and generating the perturbation scale linearly with the parameter count P as well. The total computational complexity per update for both stochastic and finite-difference methods therefore scales as $\mathcal{O}(P)$, linear in the number of parameters.

In a dynamical, fully interconnected recurrent network with N neurons as defined in (1.2), the dominant fraction of parameters consist of the N^2 connection strengths between neurons. The remaining parameters consist of the thresholds and time constants in the network, and account for a fraction linear in the number of neurons N. Hence, the number of parameters P for recurrent dynamical networks scales predominantly as $\mathcal{O}(N^2)$. With reference to the formalism used for the gradient descent methods in the context of table 2.1, the computational complexity per update for the stochastic and finite-difference methods then transforms into $\mathcal{O}(N^2)$, in contrast with the $\mathcal{O}(N^3)$ scaling for the most efficient on-line gradient calculation.

The net computational complexity of either method is obtained in relative terms by multiplying the complexity per update by the relative convergence factor of the method, which defines a measure for the minimum number of updates required to achieve a given decrease in error. By normalizing this convergence factor to unity for gradient descent, the corresponding values of this factor for the stochastic and finite-difference methods are obtained directly from (2.17) and (2.18) as approximately $\sqrt{P}$ and P respectively[19]. With inclusion of the relative convergence factors (1, N and N^2), the net computational complexities of the three methods comparatively scale as $\mathcal{O}(N^3)$, $\mathcal{O}(N^3)$ and $\mathcal{O}(N^4)$, for gradient descent, the parallel stochastic method, and the sequential finite-difference method, respectively. Therefore, the stochastic method applied to dynamical recurrent networks scales equally well or even slightly better than gradient descent in terms of computational efficiency, in spite of the model-free algorithmic form of the stochastic method with the associated benefits in accordance with the discussion in Section 2.1. The simple model-

[18]This is certainly true for conventional neural networks, and includes dynamical recurrent networks of the generalized functional form (1.1), provided every parameter p_i enters only once in the equations of motion (1.1) unless the parameter count P is incremented once for every extra occurrence.

[19]These relative factors are actually quite smaller in practice, as suggested by the empirical study in Figure 2.3.

free format of the stochastic algorithm also provides for a fairly modular, scalable and general-purpose architecture for hardware implementation, as will be demonstrated in Section 3.1.

The favorable efficiency of the stochastic method relative to gradient descent owes to the relative inefficiency of gradient descent methods specifically for dynamical recurrent systems. In contrast, for network structures with a simple model specification, the complexity of calculating the gradient usually becomes relatively smaller, while the corresponding complexity of the stochastic method is mostly invariant to the network model. In particular, the back-propagation algorithm [Rumelhart 86b], which directly implements gradient descent in feedforward networks, is computationally more efficient than the stochastic method applied to the same network structure. In fact, the relative popularity of feedforward networks, void of recurrent connections supplying continuous-time dynamical features, may partly be attributed to the computational efficiency of the back-propagation algorithm, which only requires one additional computational evaluation on the network (propagating through the network structure in the reverse direction) to construct the gradient. Recurrent networks, on the other hand, are quite broader in scope and more generally applicable than feedforward multi-layered structures, by virtue of the arbitrary connectivity and the potential to generate complex dynamics. With the stochastic method, the advantages of using a recurrent dynamical network instead of a feedforward structure come at basically no extra computational cost. The only complication with applying the stochastic method to learning in dynamical systems concerns the timing constraints with an on-line implementation in real-time settings, which constitute the subject of the section below.

2.3 On-Line Schemes for Real-Time Learning

In the analysis of the stochastic method and its comparative performance in the previous section, aspects related to the timing of the learning process with regard to the dynamics of the system were not considered. Implicitly, a batch format was assumed for the error

functional as in (1.6), with given integration boundaries $[t_0, t_f]$ and initial conditions $\mathbf{x}_0$ implying an effective reset of the system time at every "epoch" when evaluating the error (1.6). Below we investigate on-line variants of the stochastic method, which adjust the parameters while the system runs continuously, from observations of the error on the system without interrupting the continuous flow of time or forcing values for the state variables. To relax the batch format of the error functional, an attempt is initially made to approximate the theoretical error measure (1.4) by partitioning the infinite time horizon into finite time observation intervals, each of a form similar to (1.6). Conceptually, expression (1.4) can be expanded into an infinite sum of contributions from adjacent integration intervals (1.6) as

$$\mathcal{E}(\mathbf{p}) = \sum_{k=-\infty}^{+\infty} \mathcal{E}(\mathbf{p}; \mathbf{x}_k; t_k, t_{k+1}) \ , \tag{2.19}$$

with a monotonically increasing sequence of time instances $\{t_k\}$ at the boundaries between consecutive integration intervals, and with corresponding boundary values for the state variables $\mathbf{x}_k = \mathbf{x}(t_k)$. An on-line version of the stochastic method is obtained, in principle, by iteratively decreasing the error in individual intervals, one at a time in the natural consecutive sequence increasing with time and synchronous with the network dynamics, rather than including the complete ensemble (2.19) at every update. Such iterative on-line process is therefore similar to stochastic gradient techniques[20], which continuously update the parameters with partial knowledge of the error gradient obtained from instantaneous data samples on the network, rather than using the accumulated gradient over the complete data set. The analogy with stochastic gradient descent is limited, since the effect of the partitioning of the integration interval reaches further than merely inflicting stochastic fluctuations in the direction of the updates. The motivation for the partitioning and the validity of the suggested on-line variant of the stochastic method is elaborated next.

[20]The term "stochastic" here refers to the construction of the error functional with regard to the training data, and should not be confused with the context of the stochastic method in the remainder of the text, which relates to the functional form of the algorithm itself.

2.3.1 Optimal Partitioning of the Error Functional

The effect of the partitioning of the error functional on the error descent process, with regard to the incremental updates under gradient descent or any related optimization method, can be best described by analyzing the gradient of the partitioned error. Since the incremental decrease of the error under the parameter updates, for any of the three incremental methods considered here, is directly related to the local gradient of the error, either explicitly through expression (2.1) or implicitly through direct observation on the network as approximated by (2.12), the error gradient is indeed a representative indicator applicable to all three methods. From (2.19), the gradient of the partitioned error is then obtained as

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{dp}}\mathcal{E}(\mathrm{p}) &= \sum_{k=-\infty}^{+\infty} \frac{\mathrm{d}}{\mathrm{dp}}\mathcal{E}(\mathrm{p};\mathrm{x}_k;t_k,t_{k+1}) \\ &= \sum_{k=-\infty}^{+\infty} \left(\frac{\partial\mathcal{E}_k}{\partial\mathrm{p}} + \frac{\partial\mathcal{E}_k}{\partial\mathrm{x}_k}\frac{\mathrm{dx}_k}{\mathrm{dp}}\right) \end{aligned} \tag{2.20}$$

where $\mathcal{E}_k$ denotes $\mathcal{E}(\mathrm{p};\mathrm{x}_k;t_k,t_{k+1})$. Evidently, for every partition k of the error in the sum (2.19), the corresponding contribution to the error gradient in (2.20) consists of two distinct parts, each with a different origin. The first term $\partial\mathcal{E}_k/\partial\mathrm{p}$ represents the gradient of the error partition $\mathcal{E}_k$ with respect to the parameters explicitly, and therefore encodes the isolated contribution directly from the partition interval $[t_k, t_{k+1}]$ discarding external factors which implicitly depend on the parameters. The second part accounts for the indirect contribution by a change in the parameter values to the error $\mathcal{E}_k$ through a change in the initial state x_k, due to the propagation of the network dynamics prior to t_k. In other words, the first part corresponds to the direct contribution to $\mathcal{E}_k$ by a change in the parameters, originating exclusively from within the time interval of the partition, while the second part accounts for the remaining indirect contributions to the present interval $\mathcal{E}_k$ originating from previous time intervals. The indirect contributions from the past to the present error interval are due to the long-ranging effect of the network dynamics, and reflect the conceptual result of a change in parameters applied on the network well in

advance of the present time interval, in principle reaching infinitely far back in the past.

From a practical perspective, the long-ranging indirect terms in (2.20), while certainly significant for a correct representation of the complete error gradient in correspondence with (2.19), are not guaranteed to converge properly under general conditions for the network and target signals, leading to singularities in the gradient and corresponding discontinuities in the error. In particular, singularities are caused by the factors $\mathbf{dx}_k/\mathbf{dp}$ in (2.20), which denote the dependence of the state variables at certain instances in time on a change in parameters in the past. In network configurations which display persistent transients or instabilities in the dynamics of the state variables, as is not uncommonly the case due to the recurrence of the network connections, the sensitivity of the network state with respect to the parameters $\mathbf{dx}(t)/\mathbf{dp}$ grows steadily in amplitude as time increases due to the cumulative effect of the network dynamics[21].

The discontinuities on the error surface in parameter space can be understood as well from simple intuitive considerations of the network dynamics. A relatively small change in the parameters can cause a drastic change in the long-term asymptotics of the network dynamics, beyond the short-term transients. Abrupt changes in the long-term dynamics under small changes in the parameters result from the transition between slightly damped and amplified small-signal dynamics of the state variables around stationary points, defining sharp boundaries separating stable from unstable regions in the parameter space. In contrast, the short-term transient behavior from given initial conditions for the state variables does not depend quite as strongly on the parameters, even at the instability boundaries of the long-term asymptotics. By specifying an infinite time window for the observation of the error, the transients are effectively eliminated and the error undergoes the same discontinuities in parameter space as the asymptotic long-term dynamics. Discontinuities in the error surface due to the sensitivity to the parameters and its implication on gradient descent learning were suggested in [Baldi 92]. Similarly, the deteriorating effect of a long time window for the error gradient on the quality of the learning

[21]The term "sensitivity" here refers to the same context as that in which the original on-line gradient descent algorithm for recurrent dynamical neural networks [Williams 89] was introduced.

process with gradient descent was noted in [Williams 90]. Beside the discontinuities in the error surface, additional factors that lead to a degraded learning performance, due to the increased parameter sensitivity under large time windows for the error, include the presence of local minima in the error and a strong disparity in the size of the gradient across different regions in parameter space.

The partitioning of the integrated error over finite time intervals offers an attractive alternative for on-line incremental learning, which both avoids the ill-conditioning of the error descent process due to large integration times and in addition provides for a simple implementation compatible with the stochastic method. The conventional methodology for on-line learning with gradient descent, on the contrary, aims at establishing the infinite horizon of the error measure in (1.4), by accumulating instantaneous contributions to the error gradient obtained from the network outputs and training signals. In effect, such on-line methods define the error gradient over a gradually expanding time window $[t_0, t_f]$, whereby the upper time limit t_f of the integrated gradient expands synchronously with the network dynamics. Since the growing integration interval includes all history of the network dynamics, the recursive scheme to obtain the gradient needs to account for the long-term contributions to the gradient, corresponding to the equivalent of the indirect part in (2.20). In fact, most of the computational burden with on-line algorithms for gradient descent learning in recurrent dynamical systems arises exactly because of these indirect long-term contributions, which are relatively difficult to obtain through on-line operations with no direct access to prior history of the state variables.

A tempting solution to the problems associated with the long-term indirect contributions is to do away with them completely, and construct simple on-line methods which only employ instantaneous information on the error while discarding any long-term effects. Such short-cut solution may work satisfactorily in some situations, but will as well fail miserably in other cases. The reason of failure is quite obvious, since the complete spectrum of the internal dynamics within the network cannot be fully addressed when only the instantaneous response of the network outputs is evaluated. Such causes a bias

in the learning process favoring solutions which map the dynamics exclusively onto state variables directly feeding into the output units, ignoring the potential of internal dynamics involving internally coupled state variables whose response to external stimuli requires a longer time to reach the network outputs. A viable compromise, therefore, consists of schemes which allow for a certain finite time window in the evaluation of the network output response to a change in parameters, including only the most recent history of the network dynamics.

On-line schemes which sequentially observe the parameter dependence of the error over successive time windows, and which construct incremental updates in the parameter values correspondingly, can be derived directly in relation to the partitioning of the error over adjacent intervals in (2.19), and the corresponding partitioned representation of the error gradient in (2.20). From the arguments stated above, it is clear that the indirect terms in (2.20), corresponding to the long-term dependence on prior history through the dynamics of the network sensitivity $\mathrm{d}\mathbf{x}_k/\mathrm{d}\mathbf{p}$, are undesirable regarding both the ill-conditioning of the error descent process and the increased computational burden caused by these terms. For practical purposes, therefore, the indirect terms can as well be discarded from the error gradient in (2.20). The remaining terms in (2.20), retained in the conceptual representation of the "desirable" gradient of the error, encode the partial gradients of the individual errors $\mathcal{E}_k$ with respect to the parameters directly. Formally, the term $\partial\mathcal{E}_k/\partial\mathbf{p}$ is equivalent to the parameter dependence of the error measure (1.6), defined over the interval $[t_k, t_{k+1}]$ with initial conditions determined by the history of the network dynamics, $\mathbf{x}_0 \equiv \mathbf{x}(t_k)$[22]. Therefore, on-line versions of the incremental methods outlined in the previous section can be obtained directly, by reformulating the error measure used in the incremental updates. In particular, the rigid batch-mode formulation of the error measure in (1.6), with given time boundaries t_0 and t_f and initial conditions $\mathbf{x}_0$ fixed for all updates

[22]Strictly speaking, the initial values $\mathbf{x}(t_k)$ are obtained from the evolution of the network state variables generated under constant parameter values p_i. In a practical on-line implementation, the parameter values change incrementally under activation of the updates at the end of every interval, affecting all subsequent initial states $\mathbf{x}(t_k)$. The impact of the changing parameters on the learning process is a minor issue, typical for other on-line situations as well.

alike, can be freely extended into on-line format by continuously adapting the time window and initial conditions at every new update, to reflect the time lapse which occurred in the last step and the corresponding evolution of the network dynamics. By specifying $t_0 \equiv t_k$, $t_f \equiv t_{k+1}$, and $x_0 \equiv x(t_k)$ for the construction of the update $\triangle \mathbf{p}^{(k)}$, continuity in time is enforced for the values of the network state variables in between consecutive updates of the parameters. Consequently, it is no longer necessary to reset the time and state variables on the network to obtain a new update, and the learning can proceed truly on-line, without interruption of the network dynamics.

On-line methods obtained in this way effectively perform error descent optimization directly on the individual error intervals in sequence, rather than over the complete time horizon at once for every update, as with off-line implementations. In the formalism of the error partitioning in (2.19) and (2.20), each of the individual parameter updates with the above on-line variant accounts only for the parameter dependence of the error over the corresponding interval $[t_k, t_{k+1}]$, excluding all contributions from the history of the network dynamics but those originating from within the particular interval itself. With reference to the above discussions, the finite time window used in the simple on-line formulation based on the error partitioning therefore avoids the problems associated with both the infinite window for the corresponding on-line formulation with accumulated dynamics, and the instantaneous (empty window) format for the error functional which was suggested earlier as a preliminary alternative.

The finite time window format, on the other hand, introduces some degrees of freedom in the error formulation which need to be tuned for optimal learning performance. Obviously, the values obtained for the updates depend on the positioning and size of the partition intervals in time with respect to the evolution of the network error. For a better understanding of the effect of the interval geometry on the error descent process under learning, it proves instructive, as before, to evaluate the gradient of the partitioned error measure specified by the on-line format. The direct gradient of the error within each

interval $[t_k, t_{k+1}]$ can be further expanded to read

$$\begin{aligned} \frac{\partial \mathcal{E}_k}{\partial \mathbf{p}} &= \frac{\partial}{\partial \mathbf{p}} \int_{t_k}^{t_{k+1}} e(\mathbf{z}(t), \mathbf{z}^T(t))\, \mathrm{d}t \\ &= \int_{t_k}^{t_{k+1}} \frac{\partial e}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{p}}\, \mathrm{d}t \ , \end{aligned} \tag{2.21}$$

where the term $\partial e/\partial \mathbf{x}$ encodes the instantaneous contribution from the state of the network variables to the output error, and $\partial \mathbf{x}/\partial \mathbf{p}$ denotes the partial parameter sensitivity of the network variables, under the restriction imposed by the initial condition $\mathbf{x}(t_k) \equiv \mathbf{x}_k$[23]. Unlike the complete derivative format used for the parameter sensitivity in expression (2.20), which includes the entire history of the network dynamics in the account of the parameter dependence, the partial format expressed in (2.21) only accounts for contributions from the network dynamics originating after the initial time t_k of the interval. Specifically, the value of the sensitivity term $\partial \mathbf{x}/\partial \mathbf{p}$ at time t in (2.21) corresponds to the change in the network state $x(t)$ at time t caused by a unit step change on the parameter vector p introduced at time t_k. Obviously, the sensitivity vanishes at the origin of the interval $t = t_k$, where no time is available to propagate the parameter change through the network towards the outputs. On the other hand, the sensitivity at the end of the interval $t = t_{k+1}$ is significant, containing the cumulative contributions from the parameter change as those propagate through the network over the complete interval $[t_k, t_{k+1}]$. Consequently, contributions from the instantaneous network error $e(\mathbf{z}(t), \mathbf{z}^T(t))$ to the error gradient (2.21) are weighted unequally, with a strong bias obstructing earlier error contributions and inflating later ones.

The asymmetry in the weighting of the error contributions to the gradient is intrinsic to the causality of the dynamical system, and of course reaches back to the same origins as those causing the indirect terms from the network history in expression (2.20) for the gradient of the partitioned error. Rather than trying to avoid this asymmetry through dif-

[23] A direct implementation of (2.21), employing coupled differential equations to derive the time evolution of the sensitivities $\partial \mathbf{x}/\partial \mathbf{p}$, leads to one of the on-line algorithms for gradient descent learning in recurrent dynamical networks, as presented in [Williams 89].

ferent schemes, we retain the simple on-line format of the error measure, compatible with the stochastic method, but seek to optimize the error formulation by applying simple principles regarding the structure of (2.21). Little can be said in general about the long-range time dependence of the parameter sensitivity $\partial \mathbf{x}/\partial \mathbf{p}$ over the partition interval, beyond its vanishing initial state at time t_k. Nevertheless, the time dependence of the sensitivity contains dynamical characteristics similar to those for the network state variables themselves, as can be verified by straight differentiation of the evolution equations governing the network dynamics [Williams 89]. Therefore, the effective time range, over which the effect of the sensitivity is expected to prevail within the partition interval, can be directly related to the characteristic time of the network dynamics. Correspondingly, a choice of the length of the interval $\triangle t = t_{k+1} - t_k$ can be made to accommodate timing constraints expressed earlier above.

One such constraint follows directly from the earlier requirement of a finite time window for the error observations, to avoid the disadvantages associated with the sharp parameter dependence of the error measure due to long-range effects in the dynamics. The extent of this effect is represented explicitly by the sensitivity term in (2.21), since it exactly accounts for the cumulative effect of a prolonged initial change in the parameters on the state of the network at later times. The long-range contributions can therefore be avoided by limiting the length of the partition intervals $\triangle t$ below a certain critical level, as determined by the characteristic time for the evolution of the parameter sensitivity, which in turn is given by the typical time range of the network dynamics itself. The other timing constraint complements the other, expressing a lower limit on the length of the partition intervals. As stated before, an instantaneous format used for the error measure to generate the parameter updates generally does not yield satisfactory results, due to the bias introduced in the learning by neglecting the internal dynamics of the network. Again, the cause for this bias can be attributed to the structure of (2.21), as affected by the properties of the sensitivity term. In particular, for times close to the origin t_k of the interval, the sensitivity remains near zero while growing slowly. The initial phase of the

growth proceeds approximately with a linear time dependence, whereby the individual parameters contribute almost exclusively to those network state variables to which they are directly connected [Williams 89]. Since the instantaneous error $e(\mathbf{z}(t), \mathbf{z}^T(t))$ is constructed explicitly on the states of the observable output units, parameters interconnecting internal nodes within the network are not represented in the initial contributions to the integral (2.21). Hence an instantaneous format of the error measure, with a short length $\triangle t$ for the partition intervals, would exclude all internal network parameters from the learning process. This forces learning solutions where only the state variables directly connected to the outside actively participate in the network dynamics, as obtained by straight mapping of the time derivatives of the external units onto their instantaneous states. Obviously, such phase diagram representation of the dynamics, directly onto the observable state variables, has generally much poorer approximation capability than a representation of higher dimensionality, including the additional internal state variables to enrich the potential complexity of the dynamics. To fully exploit the internal dynamics of the network, including active participation of the hidden network variables, the interval length $\triangle t$ must exceed the time necessary for the parameter sensitivity to develop all components equally, including those referring to interaction between internal nodes. The amount of time needed again depends on the dynamical characteristics of the network, and a lower bound on the interval length $\triangle t$ can be formulated with regard to the typical time range of the network dynamics as well.

As a practical guideline, the two above timing constraints can be reasonably satisfied by specifying an interval length $\triangle t$ somewhat larger than the dominant characteristic time $\tau_{\max}$ in the dynamical system, say larger by a factor of three. Such provides ample time for the internal state variables to produce an observable response to a change in parameters, therefore actively contributing in the observations of the network error used to construct the parameter updates. On the other hand, since the integration time $\triangle t$ for the network dynamics in the error measure spans a few multiples of the largest time constant $\tau_{\max}$ in the system, the integrated error still contains a significant fraction of transients which are

needed to assist in damping the sharp features of the error functional in parameter space. The particular choice of the interval length, in relation to the network dynamics, may prove difficult in practice, especially since the characteristic time scale of the dynamics changes constantly as the network adapts to the training sequence. Fortunately, the choice of the interval length is not quite as critical to the learning performance, provided the length $\triangle t$ is larger but not too much larger than the typical time scale $\tau_{\max}$ of the network dynamics. If no information about the timing of the dynamics can be obtained from the network or projected from the training signals, the best way to circumvent the choice of the interval length would be to start with initially short intervals, and to gradually increase the length along the learning until the rough dynamical features of the network outputs fairly resemble that of the training signals. Doing so, the learning process initially explores solutions making exclusive use of the external units, thereafter slowly involving the internal nodes into the network dynamics until the interval is long enough to include all of those equally. Such approach may as well prove useful in avoiding potential overfitting of the noise in the training data, by only incorporating as many internal parameters into the dynamics of the network as needed to smoothly approximate the dynamics of the training signal.

The particular choice for the length of the time intervals aside, there is one further geometrical consideration in the error partitioning of some importance to on-line learning. Specifying a fixed length $\triangle t$ may cause interference of the error observations with the spectral content of the training signals, due to the regular periodical spacing of the intervals. This particularly applies to periodic training sequences, with periodicity equal to or very close to the interval length $\triangle t$. The risk of interference, affecting the uniformity of approximation in the learned network output signals, arises from the above-mentioned asymmetry of the observed parameter dependence of the error across the time interval of the partitions. Since the earlier contributions to the error within the intervals carry relatively little weight in the constructed parameter updates, the corresponding repetitive components in the training signals, positioned in time just after the boundaries of

the partition, are expected to yield relatively poor approximation quality in the network outputs after learning. This problem can simply be alleviated through dithering of the partitioning structure, by selecting for each update a different value for the interval length $\triangle t_k = t_{k+1} - t_k$, say from a random distribution with given spread around the optimal length $\triangle t$. Since under the random partitioning every time instance of the training sequence is positioned equally likely near the beginning or the end of one of the intervals, a fairly uniform representation of the training data in the error measure is hence guaranteed.

The above line of thoughts involved some level of complication in considering several issues, each related to a different aspect of learning in dynamical systems under real-time requirements. Altogether, the above material served to bring forth a rather simple technique which extends the batch-like format of the error, implicitly considered in the previous section, directly into an on-line formulation suitable for real-time learning. The technique simply substitutes the fixed batch-like format of the error measure (1.6) by one which for every update $\triangle \mathbf{p}^{(k)}$ adjusts its timing boundaries and network initialization to reflect the actual state of the network as it evolves with time,

$$\begin{aligned} \mathcal{E}_k(\mathbf{p}) &= \mathcal{E}(\mathbf{p}; \mathbf{x}_k; t_k, t_{k+1}) \\ &= \int_{t_k}^{t_{k+1}} e(\mathbf{z}(t), \mathbf{z}^T(t))\, \mathrm{d}t \ , \end{aligned} \tag{2.22}$$

with the initial states $\mathbf{x}_k = \mathbf{x}(t_k)$ as determined by the actual dynamics of the network. Stated otherwise, the adjusted error measure (2.22) corresponds to the conceptual partitioning of the infinite integrated error (1.4) over successive non-overlapping intervals in time, whereby every parameter update attempts to minimize the individual error contained within the corresponding partition interval only. Simplistically, the partitioning of the error for on-line learning can thus be viewed as a means for distributing the parameter update process over the time domain of the training sequence, in a way quite similar to stochastic gradient techniques. However, due to the long-range time dependence of the observed output error on a previous change in parameters, inherent to the recurrent dynamics in the network, the expectation value for the partitioned error metric (2.22) does

not quite follow the same parameter dependence as the complete integrated error measure (1.4). Nevertheless, the above timing analysis shows that with an appropriate range chosen for the interval length $\triangle t_k$, just sufficiently exceeding the characteristic time scale $\tau_{\max}$ of the network dynamics as to include the relevant time range of the parameter dependence in the error observations, the on-line partitioned error measure actually yields a superior learning performance.

In principle, on-line variants based on time partitioning of the error can be constructed, through the above procedure, for any incremental optimization method which minimizes an error functional of the type (1.6), including various implementations of gradient descent[24], and the gradient-free methods studied in the last section. Below, we apply the on-line error formulation specifically to the parallel stochastic optimization method for real-time learning in recurrent dynamical systems, and discuss two practical implementations which each address different requirements on the timing organization of the training sequence and the configuration of the network within the system under optimization.

2.3.2 Gradient-Free Implementations

Like with other incremental error descent methods, the extended on-line format for the error measure $\mathcal{E}(\mathbf{p})$ in (2.22) can be directly used with the parallel stochastic error descent method, outlined in equations (2.2)-(2.4). In doing so, there is one potential source of concern, depending on the application domain the method is targeted to, which follows from the requirement of two complementary error observations to construct the parameter updates. In principle, both observations should be carried out simultaneously and independently on the system, under application of two different sets of parameters. Strictly, such is not possible in a physical implementation, since only one instance of the parameter vector can be supplied to the network at any given time.

In principle, the dilemma of simultaneous error observations can be partially avoided

[24]Incidentally, a partitioning of the total error integration interval over smaller subintervals was recently proposed in [Lopez 93] for gradient-based learning of coupled time series in recurrent polynomial networks, and in [Catfolis 93] for an improved convergence of the original real-time recurrent network algorithm [Williams 89].

by applying perturbations and performing error observations on the system at a rate much faster than the characteristic time scale of the system dynamics and the training signals. Therefore, successive error observations are expected to yield approximately the same results as obtained by simultaneous observations. In fact, an on-line gradient-free realization based on this principle, using a one-step-ahead error measure, was recently reported in [Spall 92b] for direct adaptive control of discrete-time dynamical systems. While functionally quite elegant and easy to implement, this approach may in certain cases suffer from the problems mentioned earlier concerning a short time interval for the on-line error measure in (2.22). With the time interval $\triangle t$ for the error observations chosen much shorter than the time scale of the system dynamics τ_{max}, the parameters relating to the internal state variables are effectively masked in the optimization process, with the consequences of potentially poor learning quality stated before[25].

To circumvent the timing problem, we suggest and analyze two alternative realizations below, based on the on-line error measure defined in (2.22) without making assumptions on the relative time scale of the observations. One realization employs several identical replicas of the same dynamical system, for synchronous observation of the error under different instances of the parameter vector. The other makes use of time-multiplexed repetitive error observations on the same system, under different parameter settings but otherwise under identical circumstances, thus assuming a periodical sequence for the training signals. While neither of the two realizations is perfectly suitable under the most general real-time operating conditions, they are complementary in nature and each address one specific class of applications. The consequences of the different assumptions on the network configuration and timing of the training signal are discussed below, along with the details of the two realizations.

[25]The masking of the internally coupled parameters biases the learned solution towards a mere mapping of the dynamics of the training signal exclusively onto the output units. This yields acceptable results only if the set of output units represents the full dimensionality of the internal dynamics of the system.

Concurrent Realization Using System Replicas

In case exact copies of the dynamical system under optimization can be provided along with the original system, the error observations under the complementary perturbed states of the parameter vector can be performed concurrently on two separate replicas, sharing the same input and target training signals in time as those supplied to the master system. For a correct and consistent observation of the perturbed error on each of the two replicas, conform to the format in (2.22), both replica systems need to be periodically reset to the unperturbed state $\mathbf{x}_k = \mathbf{x}(t_k)$ at the beginning of every observation interval. To this purpose, the master continuously tracks the system dynamics under unperturbed activation of the parameter vector, thereby serving as a reference for the periodic initialization of the replica system variables. In particular, let $\mathcal{N}$, $\mathcal{N}^+$, and $\mathcal{N}^-$ denote the master system, the first replica, and the second replica, respectively. During the interval $[t_k, t_{k+1}]$, the master is supplied with parameters $\mathbf{p}^{(k)}$ while the replicas are assigned corresponding perturbed values $\widehat{\mathbf{p}}^{+(k)}$ and $\widehat{\mathbf{p}}^{-(k)}$, according to (2.4). Then observation of the interval error (2.22) on the perturbed replica systems $\mathcal{N}^+$ and $\mathcal{N}^-$ yields

$$\begin{aligned} \mathcal{E}_{\mathcal{N}^+}{}^{(k)} &= \mathcal{E}(\mathbf{p}^{(k)} + \boldsymbol{\pi}^{(k)}; \mathbf{x}_k; t_k, t_{k+1}) \\ \mathcal{E}_{\mathcal{N}^-}{}^{(k)} &= \mathcal{E}(\mathbf{p}^{(k)} - \boldsymbol{\pi}^{(k)}; \mathbf{x}_k; t_k, t_{k+1}) \ , \end{aligned} \tag{2.23}$$

with $\mathbf{x}_k = \mathbf{x}_{\mathcal{N}}(t_k)$ as observed on the master system. The remaining operations to construct the parameter update further proceed as before in (2.3) and (2.2), with $\mathcal{E}(\widehat{\mathbf{p}}^{+(k)}) \equiv \mathcal{E}_{\mathcal{N}^+}{}^{(k)}$ and $\mathcal{E}(\widehat{\mathbf{p}}^{-(k)}) \equiv \mathcal{E}_{\mathcal{N}^-}{}^{(k)}$.

For proper functioning of this realization, the replica systems must resemble the original system almost exactly, and need to provide the capability of setting values for the parameters independently of those supplied to the master system. Furthermore, explicit read access to the internal state $\mathbf{x}_{\mathcal{N}}$ of the master system is required, and the states $\mathbf{x}_{\mathcal{N}^+}$ and $\mathbf{x}_{\mathcal{N}^-}$ of the replica systems need to be reinitialized accordingly at the beginning of every update cycle. These stringent requirements on the nature of the system obviously

restrict the practical scope of the concurrent realization, which thus excludes optimization of "black-box" dynamical systems otherwise fully supported by the model-free approach of the stochastic method. The second alternative realization, described next, supports optimization of such black-box systems, by means of a time-interlaced approach not requiring any explicit knowledge about the structure and functional parameter dependence of the dynamical system. On the whole, the virtue of the concurrent realization with system replication as just described is relatively limited. Nevertheless, the restrictions on the system configuration imposed by the concurrent realization are usually satisfied for those applications which cannot be supported by the second alternative realization, further analyzed below.

Time-Interlaced Realization under Repetitive Training

Alternatively, in case the time sequence for the input and target training signals $(\mathbf{y}(t), \mathbf{z}^T(t))$ can be rendered in a strictly periodical format, then the error observations can be organized in time-multiplexed fashion on the same system, without the need of replicas along with the physical system. The realization proposed here does not assume any knowledge on the model of the dynamical system under observation, nor does it interfere with the internal dynamics of the system as it evolves in time. Therefore, the realization is well suited for optimization of unknown dynamical systems, for which the parameter dependence of the performance measure $\mathcal{E}(\mathbf{p})$ can only be obtained through direct observation on the physical system.

In principle, the two complementary error observations required for the differential perturbed error (2.3) can be obtained directly in sequence, under successive alternating perturbation of the parameters according to (2.4), with each error observation interval spanning one period of the training sequence. One problem with following this procedure is the potential inconsistency in the initial state $\mathbf{x}_k$ among both error observations, with regard to the on-line format specified in (2.22). This inconsistency may arise since no mechanism is available to enforce equal values for the internal state variables at dif-

ferent instances in time. Nevertheless, one may reasonably expect that, under constant activation of fixed parameter values onto the system, the sequence of state variables $\mathbf{x}(t)$ asymptotically approaches a limit cycle with same periodicity as the training signals[26]. In other words, a fair consistency in the initial state variables, as needed for the error observations, is intrinsically achieved under fixed parameter settings. The two perturbed error observations in (2.3), on the other hand, require each different parameter settings.

In an attempt to undo the disturbance in the periodicity of the state variables, caused by activation of alternating parameter perturbations in time, the procedure to obtain the differential error measure (2.3) is extended to include two additional cycles of error observations, each with the unperturbed settings for the parameter vector. Specifically, let T be the period of the input and target training signals $(\mathbf{y}(t), \mathbf{z}^T(t))$. Then the perturbation schedule for the parameter settings onto the system is given the time sequence

$$\mathbf{p}(t) = \begin{cases} \mathbf{p}^{(k)} & ; \quad t_k < t \leq t_k + T \\ \mathbf{p}^{(k)} + \boldsymbol{\pi}^{(k)} & ; \quad t_k + T < t \leq t_k + 2T \\ \mathbf{p}^{(k)} & ; \quad t_k + 2T < t \leq t_k + 3T \\ \mathbf{p}^{(k)} - \boldsymbol{\pi}^{(k)} & ; \quad t_k + 3T < t \leq t_k + 4T \end{cases} , \tag{2.24}$$

and the corresponding error observations yield

$$\begin{aligned} \mathcal{E}_0^{+(k)} &= \mathcal{E}(\mathbf{p}^{(k)};\, \mathbf{x}(t_k);\, t_k,\, t_k + T) \\ \mathcal{E}^{+(k)} &= \mathcal{E}(\mathbf{p}^{(k)} + \boldsymbol{\pi}^{(k)};\, \mathbf{x}(t_k + T);\, t_k + T,\, t_k + 2T) \\ \mathcal{E}_0^{-(k)} &= \mathcal{E}(\mathbf{p}^{(k)};\, \mathbf{x}(t_k + 2T);\, t_k + 2T,\, t_k + 3T) \\ \mathcal{E}^{-(k)} &= \mathcal{E}(\mathbf{p}^{(k)} - \boldsymbol{\pi}^{(k)};\, \mathbf{x}(t_k + 3T);\, t_k + 3T,\, t_k + 4T) \ . \end{aligned} \tag{2.25}$$

The differential perturbed error, corresponding to expression (2.3), is estimated from the

[26] In certain cases involving special memory effects in the dynamics, the state variables may not converge to a periodic sequence, or the period of the limit cycle may exceed that for the training signals. Such exceptions are not commonly encountered under "normal" conditions, and can usually be avoided by proper initialization of the parameters prior to learning.

four above error observations as

$$\widehat{\mathcal{E}}^{(k)} = \frac{1}{2}\left(\mathcal{E}^{+(k)} - \mathcal{E}^{-(k)} - \mathcal{E}_0^{+(k)} + \mathcal{E}_0^{-(k)}\right) \ , \tag{2.26}$$

and subsequently the value for the parameter increment is derived as before, using (2.2).

The extra terms in (2.26), corresponding to the intermediate unperturbed error observations ${\mathcal{E}_0}^{+(k)}$ and ${\mathcal{E}_0}^{-(k)}$, relate to the disturbance of the error measure caused by the drift in initial conditions $\mathbf{x}_k$. In principle, this drift could be reduced to arbitrary small levels, by letting the unperturbed intermissions between the perturbed observations span over a sufficient number of training cycles. This would allow the system dynamics to settle towards the unperturbed limit cycle, providing consistent initial conditions $\mathbf{x}_k$ for the subsequent perturbed error observations. To minimize the number of cycles needed in every update of the parameters, only one cycle of the training signal is allowed for the unperturbed intermissions in (2.25), leaving a residual drift in the initial conditions $\mathbf{x}_k$ due to incomplete settling of the dynamics. Nevertheless, the information gained from observations of the error during the intermediate unperturbed cycles can be used to indirectly compensate the disturbance due to the drift. The unperturbed error terms ${\mathcal{E}_0}^{+(k)}$ and ${\mathcal{E}_0}^{-(k)}$ included in (2.26) serve this purpose, though the amount of compensation provided usually exceeds what is strictly needed, as will be clarified below. The over-compensation by the included unperturbed terms in (2.26) actually proves beneficial to the learning process, by favoring parameter solutions which enhance the stability of the system dynamics, as shown below as well.

For simplicity of notation, define ${t_k}^i$ as the initial times $t_k + iT$ of the error observation intervals, $i = 0, \ldots 3$. Similarly, let ${\mathbf{x}_k}^i$ denote the initial states $\mathbf{x}({t_k}^i)$ for the error observations (2.25), and let $\mathbf{x}_k$ represent the asymptotic initial state under unperturbed parameter settings $\mathbf{p}(t) = \mathbf{p}^{(k)}$. Under ideal circumstances, the initial states would not be affected by the alternating perturbations, ${\mathbf{x}_k}^i \equiv \mathbf{x}_k$, and each of the error observations in (2.26) would faithfully correspond to the on-line error measure as defined in (2.22), whereby conceptually the partition interval length spans one training period, $\triangle t_k \equiv T$. Under

such conditions, the unperturbed error terms are obviously redundant, and expression (2.26) simply reduces to the original formulation of the differential perturbed error in (2.3). In practice, the limit cycle cannot be maintained under activation of the perturbations in (2.25), and at least some of the $\mathbf{x}_k{}^i$ are expected to deviate from $\mathbf{x}_k$. The corresponding deviation in the error observations (2.26), relative to the conceptual on-line error measure (2.22), can be expanded to first order as

$$\mathcal{E}(\mathbf{p}; \mathbf{x}_k^i; t_k^i, t_k^{i+1}) \approx \mathcal{E}_k(\mathbf{p}) + \frac{\partial \mathcal{E}_k}{\partial \mathbf{x}_k}(\mathbf{x}_k^i - \mathbf{x}_k) \tag{2.27}$$

whereby the parameter dependence of the latter term can be discarded in the further analysis. Thus, expression (2.26) transforms into

$$\begin{aligned} \widehat{\mathcal{E}}^{(k)} \;\approx\; & \frac{1}{2}\left(\mathcal{E}_k(\mathbf{p}^{(k)} + \boldsymbol{\pi}^{(k)}) - \mathcal{E}_k(\mathbf{p}^{(k)} - \boldsymbol{\pi}^{(k)})\right) \\ & + \frac{1}{2}\frac{\partial \mathcal{E}_k}{\partial \mathbf{x}_k}(-\mathbf{x}_k^0 + \mathbf{x}_k^1 + \mathbf{x}_k^2 - \mathbf{x}_k^3) \;, \end{aligned} \tag{2.28}$$

of which the first part is directly identified as the equivalent of (2.3). The second part, therefore, corresponds to the impact of the variation in the initial states $\mathbf{x}_k{}^i$.

The sequence of initial states is hard to predict in general and depends on the specific details of the system dynamics (1.1) and (1.3). Rather than attempting to derive the dependence of the sequence $\{\mathbf{x}_k{}^i\}$ in exact functional form, some immediate quantitative insight can be gained by applying simple concepts from qualitative arguments. First, it is postulated that, for fixed unperturbed parameter settings, the sequence of successive initial states $\{\mathbf{x}_k{}^i\}$ asymptotically converges to $\mathbf{x}_k$ for $i \to \infty$, assuming a globally stable limit cycle for the dynamics of the system in response to the periodic inputs, with the same periodicity. To first order, the relaxation of the initial states towards the asymptote can be approximated by damped linear recurrence relations. Next, it is reasonable to assume that small perturbations in the parameters give rise to linear increments in the relaxation of the initial states, driving the states $\mathbf{x}_k{}^i$ away from the asymptote $\mathbf{x}_k$. Let $\boldsymbol{\pi}_k{}^i$ be the perturbation applied to the parameter vector in the interval $[t_k{}^i, t_k{}^{i+1}]$, as given in the

timing schedule of (2.25). Then the above arguments can be combined in the recurrence relation

$$\begin{aligned} \triangle \mathbf{x}_k^i &= \mathbf{x}_k^{i+1} - \mathbf{x}_k^i \\ &\approx -\alpha\,(\mathbf{x}_k^i - \mathbf{x}_k) + \beta\,\boldsymbol{\pi}_k^i \end{aligned} \tag{2.29}$$

where the matrix α characterizes the relaxation dynamics of the sequence of initial states, and the matrix β defines the disturbance caused by the perturbation $\boldsymbol{\pi}_k^i$. From (2.29), the relaxation sequence of the states $\{\mathbf{x}_k{}^i\}$ towards their asymptote $\mathbf{x}_k$, in absence of perturbations, undergoes the recurrence

$$\mathbf{x}_k^{i+1} - \mathbf{x}_k \approx (1-\alpha)\,(\mathbf{x}_k^i - \mathbf{x}_k) \tag{2.30}$$

with the dominant terms in the matrix α defining the time scale of the relaxation dynamics[27]. Obviously, the relaxation time scale for the sampled states $\mathbf{x}_k{}^i$ corresponds to that for the continuous-time dynamics of the system. In particular, with $\tau_{\max}$ the characteristic time in the dynamics of the state variables $\mathbf{x}(t)$ and T the interval length of the error observations, the attenuation matrix $1-\alpha$ in (2.30) contains dominant eigenvalues of approximate amplitude $\exp(-\tau_{\max}/T)$.

For a systematic understanding of the purpose of the unperturbed error terms $\mathcal{E}_0{}^{+(k)}$ and $\mathcal{E}_0{}^{-(k)}$ included in (2.26), we first consider the specific case $\alpha \equiv 0$, corresponding to a characteristic time scale $\tau_{\max}$ much larger than the interval length T. In this limit, the change in state $\mathbf{x}_k{}^i$ over one cycle for the intermediate unperturbed intervals $\mathcal{E}_0{}^{+(k)}$ and $\mathcal{E}_0{}^{-(k)}$ becomes negligible, and approximately $\mathbf{x}_k{}^1 = \mathbf{x}_k{}^0$ and $\mathbf{x}_k{}^3 = \mathbf{x}_k{}^2$. Then according to (2.28) the estimate of the differential perturbed error (2.26) reduces to the intended on-line variant of (2.3), and the compensation of the drift in initial states is complete. Effectively, the unperturbed error observations $\mathcal{E}_0{}^{+(k)}$ and $\mathcal{E}_0{}^{-(k)}$ then serve as a reference

[27] Stable relaxation requires that all eigenvalues of $1-\alpha$ be enclosed within the circle with unit radius centered around the origin in the complex plane.

to the subsequent perturbed error observations $\mathcal{E}^{+(k)}$ and $\mathcal{E}^{-(k)}$, sharing common initial states.

In the general case, for arbitrary time scales of the dynamics $\tau_{\max}$ relative to the interval length T, the initial state is no longer retained after one unperturbed cycle, but rather relaxes towards the asymptote $\mathbf{x}_k$ according to (2.30). Directly applying the recurrence relation (2.29) to the states $\mathbf{x}_k{}^i$ for $i = 0$ and 2 with $\pi_k{}^i \equiv 0$, the second part $\widehat{\mathcal{E}}^{(k)}_{\text{drift}}$ of expression (2.28), representing the disturbance in the differential perturbed error due to the drift in initial states, transforms into

$$\widehat{\mathcal{E}}^{(k)}_{\text{drift}} \approx \frac{1}{2} \frac{\partial \mathcal{E}_k}{\partial \mathbf{x}_k} \alpha \, (\mathbf{x}_k^2 - \mathbf{x}_k^0) \ . \tag{2.31}$$

The residuals present in the term $\widehat{\mathcal{E}}^{(k)}_{\text{drift}}$, beyond the intended first part of (2.28) according to expression (2.3), affect the error descent process under iteration of the stochastic incremental updates (2.2) with the values for $\widehat{\mathcal{E}}^{(k)}$ obtained from (2.26).

According to (2.31), the impact of the drift in initial states on the estimate (2.26) of the differential perturbed error grows with the size of the matrix α, which becomes larger as the dominant time constant in the dynamics of the system decreases. Regardless, the extra term of (2.31) tends to enhance the stability of the system dynamics under learning, biasing the convergence towards solutions with reduced sensitivity to variations in the parameters and to perturbations in the state variables. To understand the stabilizing effect of the extra term (2.31) included in the parameter updates (2.2), the following intuitive reasoning is conducted. For simplicity, the matrix α will first be treated as a positive scalar, which then can be extended to the general case using similar principles. With α being a positive scalar quantity, the expression (2.31) for $\widehat{\mathcal{E}}^{(k)}_{\text{drift}}$ can be interpreted, except for a positive scaling factor, as the change in error $\mathcal{E}_k$ corresponding to a change in initial states from $\mathbf{x}_k{}^0$ to $\mathbf{x}_k{}^2$. On the other hand, this change in initial states is caused by activation of the perturbation vector $\pi^{(k)}$ during one cycle, preceded by one unperturbed cycle. [28]

[28] For a meaningful interpretation, it is implicitly assumed that the first initial state $\mathbf{x}_k{}^0$ coincides with the asymptote $\mathbf{x}_k$, such that $\mathbf{x}_k{}^1 \equiv \mathbf{x}_k{}^0$ and (2.31) truly represents the parameter dependence of the error through

Therefore, the term (2.31) represents the indirect effect of the parameter perturbation $\pi^{(k)}$ on the error $\mathcal{E}_k$ through a change in initial states, unlike the first term in (2.28) which encodes the direct dependence of the error $\mathcal{E}_k$ on a change in parameters under fixed initial settings for the state variables. Analogous to the error descent property obtained by the updates (2.2) with regard to the direct part (2.3) of the differential perturbed error, the indirect part (2.31) biases the parameter updates towards minimal sensitivity of the system dynamics $\mathbf{x}(t)$ to changes in the parameters. Specifically, in case the perturbation $\pi^{(k)}$ applied onto the parameters affects a change in initial states causing an increase in error $\mathcal{E}_k$, the indirect term (2.31) biases the parameter update towards the direction opposite to that perturbation vector. Likewise, perturbation directions yielding a decrease in error under the change caused to the initial states are partially reinforced by the updates through the indirect term (2.31). Therefore, the presence of the indirect term (2.31) in the estimate of $\widehat{\mathcal{E}}^{(k)}$ used for the parameter updates biases the convergence in favor of parameter settings supporting invariance in the error measure $\mathcal{E}$ under any change of state variables caused by perturbing the parameters. In practice, this bias in the learning process provides for an enhanced stability of the system dynamics under perturbations of the state variables, whether originating from variations in the parameters or from other sources.

The above reasoning still approximately holds in case the matrix α does not reduce to a scalar, provided the response of the dynamical system to the periodic input signals follows a stable limit cycle with same periodicity as that of the inputs. Then all eigenvalues necessarily contain positive real components[29], and any vector $\mathbf{x}$ points in approximately the same direction as the corresponding vector $\alpha\mathbf{x}$ under the transformation α. Therefore, the term (2.31) still more or less corresponds to the change in error $\mathcal{E}_k$ caused by a change in initial states from ${\mathbf{x}_k}^0$ to ${\mathbf{x}_k}^2$, and the above arguments extend to the general case, though with limited validity. In case the spread in amplitude of the eigenvalues $|\lambda_i|$ is not too large, the approximation is generally fair for most random directions selected

a change in initial states.

[29] In particular, as follows directly from the observation made in the second previous footnote, the eigenvalues λ_i of α are contained within the circle with radius one centered around the point $1 + 0 \cdot i$ in the complex plane.

for the perturbations, and on average the parameter updates still assist in decreasing the sensitivity of the dynamics $\mathbf{x}(t)$ under perturbation of the parameters.

The tendency to stabilize the dynamics can be taken as a fortunate side effect of the added unperturbed terms in (2.26), in an attempt to completely compensate for the drift in initial state for the error observations. As a matter of fact, the extra terms added in (2.26) normally tend to compensate more than is strictly necessary, thereby reversing the natural tendency to create instability in absence of the compensating terms. To exactly assess the degree of compensation offered by including the unperturbed terms, it is instructive to compare the above findings with those obtained in the case where the compensating terms ${\mathcal{E}_0}^{+(k)}$ and ${\mathcal{E}_0}^{-(k)}$ are deleted from the estimate of $\widehat{\mathcal{E}}^{(k)}$ in (2.26). Such corresponds to omitting the terms ${\mathbf{x}_k}^0$ and ${\mathbf{x}_k}^2$ from expression (2.28), thereby transforming the residual part (2.31) into

$$\widehat{\mathcal{E}}^{(k)}_{\text{drift}} \approx -\frac{1}{2}\,\frac{\partial \mathcal{E}_k}{\partial \mathbf{x}_k}\,(\mathbf{x}_k^3 - \mathbf{x}_k^1)\ . \tag{2.32}$$

The most significant difference between the compensated and uncompensated versions of the residual part, in (2.31) and (2.32) respectively, concerns the polarity. The expression (2.32) can be directly interpreted as the negative of the change in error $\mathcal{E}_k$ due to a change in initial states, from ${\mathbf{x}_k}^1$ to ${\mathbf{x}_k}^3$. In turn, this change in states is accomplished by activating a perturbation $\boldsymbol{\pi}^{(k)}$ onto the parameters during one cycle. From a similar perspective as used in the interpretation of (2.31), the term (2.32) again corresponds to the indirect effect of the perturbation $\boldsymbol{\pi}^{(k)}$ on the error $\mathcal{E}_k$ through a change in initial states, though including an inversion of polarity and excluding the scaling performed by the transformation α in (2.31). Consequently, the uncompensated indirect part (2.32) has an averse effect on the stability of the system dynamics during learning, since it biases the parameter updates towards directions of greater variability in initial states under perturbation of the parameters. The inversion of the sign in (2.31) with respect to (2.32) indicates over-compensation of the residual indirect part $\widehat{\mathcal{E}}^{(k)}_{\text{drift}}$ of the estimate (2.26) through inclusion of the unperturbed error terms, thereby reversing the destabilizing effect of the parameter updates beyond what is necessary to provide regenerative dynamics of the initial states ${\mathbf{x}_k}^i$ during learning.

In the limit $\alpha \rightarrow 0$, corresponding to a slow dynamical system response, the degree of compensation achieved is complete, as noted before. Only then does the error descent learning process not interfere, neither stabilizing nor destabilizing, with the dynamical characteristics of the system, other than to the extent of the dynamics implicitly suggested by the time evolution of the training signals.

A practical timing arrangement for the above formulated on-line realization of the stochastic method is obtained directly with regard to the perturbation schedule in (2.24), by a proper choice of the separation $\Delta t_k = t_{k+1} - t_k$ between consecutive updates in relation to the period of the training signal T, which also defines the interval length of the error observations in (2.25). Since four error observations are required to construct every parameter update, the separation between updates needs to exceed four training cycles, $\Delta t_k \geq 4T$. Beyond this strict time margin, it is usually necessary to include a few extra cycles between consecutive updates, in order to undo some of the transients in the dynamics $\mathbf{x}(t)$ occurring immediately after the incremental change in parameters under the last update. Stated alternately, with the updated parameters $\mathbf{p}^{(k)}$ valid over a sufficiently long time interval $[t_{k-1}{}^4, t_k{}^0]$ preceding the error observations (2.25), the initial state $\mathbf{x}_k{}^0$ approaches the asymptote $\mathbf{x}_k$ to the extent necessary for proper compensation of the drift in (2.26)[30].

Rather than fixing a given value for the separation Δt_k, the amount of separation is instead chosen randomly for every update iteration (k) individually, subject to the timing constraint just mentioned. This achieves the same effect as the random partitioning of the error observation intervals suggested before, to improve the uniformity of the quality of approximation over the entire time interval of the training sequence. Since the training signals are periodic, the spread of the random fluctuations in Δt_k can be constrained to span exactly one period T. A practical rule for selecting the time separation between

[30]The excess spacing $t_k{}^0 - t_{k-1}{}^4$ can be safely chosen as short as a few cycles T, since the transients need not be completely extinguished. Particularly, the case of a significant transient $\mathbf{x}_k{}^0 - \mathbf{x}_k$ remaining at time $t_k{}^0$ after several cycles T indicates slow dynamics, hence $\alpha \approx 0$ and the drift compensation in (2.26) becomes effective regardless of the initial state $\mathbf{x}_k{}^0$.

updates is then given by

$$\triangle t_k = (4 + \chi + \zeta_k)\, T \ , \tag{2.33}$$

with χ a fixed small positive integer (*e.g.*, 2) defining the minimum excess spacing between updates and successive error observations, and with ζ_k a random fraction of one period as chosen from a uniform distribution, $0 < \zeta_k \leq 1$.

The above on-line realization of the stochastic method is particularly suited for optimization of uncharacterized dynamical systems, whereby the parameter dependence of a given error measure $\mathcal{E}(\mathbf{p})$, defined on the dynamical response of the outputs, can only be obtained through direct observation on the system. In essence, the above procedure to obtain the parameter updates only involves four error observations (2.25) on the system under different parameter settings (2.24), carried out consecutively in time without interfering with the internal state of the dynamical system. On the downside of this approach, the input and target training signals supplied to the system during learning need to satisfy stringent timing conditions, to warrant consistency in the error measure $\mathcal{E}(\mathbf{p})$ across subsequent observations of the error under different parameter settings. As stated before, both $\mathbf{y}(t)$ and $\mathbf{z}^T(t)$ are assumed periodic, with period T matching the interval length of the error observations. This restriction obviously precludes applications where the training signals are completely assigned by the learning problem, such as for prediction of time-varying processes from recorded data sequences in time. The periodicity requirement for the training signals in the time-interlaced realization can be viewed as a necessary compromise to avoid the problems with the previous concurrent realization, which allows arbitrary training signals but which requires an explicit model specification for the dynamical system, in conjunction with two exact model replicas. The compromise reflects the fact that, in real-time situations, several error observations under different parameter settings cannot be performed simultaneously on the same dynamical system.

Rather than attempting to circumvent the dilemma of simultaneous error observations and unify the concurrent and time-interlaced realizations into one form which is generally valid, we elaborate below on the specific strengths of both alternative realizations for

a given application environment. In particular, it will be shown that the two above realizations support two distinct and mutually complementary classes of applications, each assuming a different configuration for the dynamical system with regard to the embedded network and the training signals.

System Configuration and Application Environment

Depending on the function to be performed by the dynamical system in relation to its environment, the set of applications supported by supervised learning can be roughly divided in two main classes. The first class covers applications for identification and prediction of time-varying processes, from a set of representative signals extracted from the environment. A recording of the observed time series then serves as a set of training signals $(\mathbf{y}(t), \mathbf{z}^T(t))$ supplied to a parameter-driven dynamical network, which is trained to produce outputs $\mathbf{z}(t)$ regenerating the time series as it is observed from the process. In effect, the network is trained to identify the quantitative causal relationship between the signals $\mathbf{y}(t)$ and $\mathbf{z}(t)$. The second class of applications aims at training given dynamical systems to exhibit pre-described dynamical characteristics, by adjusting a set of continuous parameters governing the dynamics. The desired dynamical characteristics of the system can then be represented by a target training sequence for the outputs $\mathbf{z}^T(t)$ in response to an exemplary input sequence $\mathbf{y}(t)$ supplied to the system. Provided the input sequence $\mathbf{y}(t)$ during training represents the typical spectrum of inputs under which the system is intended to operate in practical situations, the desired dynamical response can be established onto the system by supervised learning under the supplied training signals $(\mathbf{y}(t), \mathbf{z}^T(t))$.

While both classes make use of the same supervised learning mechanism, forcing the outputs $\mathbf{z}(t)$ towards the target training signal $\mathbf{z}^T(t)$ by adjustment of the parameters, the configuration of the system under optimization in relation to the environment supplying the training signals is essentially different, and reflects the distinction made between the two system configurations depicted in Figure 1.2. For the first class, the system under

optimization consists of a synthesized network which is intended to reproduce the time-varying signals recorded from the process environment, thereby offering a versatile tool for prediction independent of a physical context. The network is configured as shown in Figure 1.2 (a), with both input and target output training signals being supplied by the same reference source, representing the process data extracted from the environment. For the second class, the system under optimization itself comprises an integral part of the physical environment, and may consist of a parameter-driven dynamical network driving a nonlinear plant, or any general dynamical system of which the functional parameter dependence does not need to be known *a priori*. The sequence of training signals supplied to the system is constructed externally, with reference to the pre-described response characteristic. As illustrated in Figure 1.2 (b), the target outputs $\mathbf{z}^T(t)$ are obtained from a reference system, representing the desired dynamical characteristics, in response to the externally supplied inputs $\mathbf{y}(t)$.

In the context of the above on-line realizations of the stochastic optimization method, the essential difference between both classes concerns the restrictions they implicitly assume on either the access to the internal system dynamics or the timing of the training signals. While the first class specifies a strict training signal as determined by the observed process data, the second class allows for some structural freedom in formulating the training sequence. On the other hand, the second class does not generally support full knowledge of the system nor access to its internal dynamics, whereas for the first class the system is constructed externally according to a prescribed model, therefore providing full control over the internal state variables. Consequently, the concurrent and time-interlaced on-line realizations presented above are specifically tuned to cover the first and the second class of applications, respectively. In particular, the first class allows to duplicate the constructed dynamical system and control the internal states of its replicas, as required with the concurrent realization[31]. Likewise, the second class allows to specify a periodic input sequence $\mathbf{y}(t)$ under training, thereby providing the periodicity in the training signals

[31]Exact replicas can be practically obtained if the implementation platform for the constructed systems is digital with discrete-time dynamics.

$(\mathbf{y}(t), \mathbf{z}^T(t))$ required for the error observations in the time-interlaced realization[32].

Usually, any given application of supervised learning in dynamical systems can be classified under at least one of the two above categories, and an appropriate on-line realization with corresponding system configuration can be formulated accordingly. Possible exceptions, if any, would be found in situations both requiring a non-periodic format for the training signals and denying access to the internal state of the system dynamics. A hypothetical example would be a physical dynamical system trained to produce a certain output sequence in response to a *particular* non-periodic input sequence. Practical applications of training the dynamical response of physical systems do not usually specify particular inputs when defining the desired outputs, but rather formulate a conglomerate output behavior in response to general inputs, analogous to the concept of a transfer function for linear systems. The training input sequence can therefore be chosen periodic, as long as its frequency content covers a sufficiently wide spectrum to guarantee proper generalization of the training data under arbitrary inputs after learning.

Below, we present some simulations on the same Lissajous trajectory learning example as studied before, to check different regimes explored by the on-line approach, and to demonstrate the validity of the two alternative on-line realizations presented above. Though technically this learning example belongs to the first class of applications, supported by the concurrent realization using system replicas, it also extends to the time-interlaced approach since coincidentally the training sequence of the trajectory is periodic. Using the same empirical example for both alternative realizations provides a common base for direct comparison.

2.3.3 Simulations

A first series of simulations, on the trajectory learning example outlined in the introduction section, was performed to verify the general approach followed in the on-line formulation, in particular with regard to the choice of the interval length $\triangle t$ for the error observations

[32]Since the input sequence is periodic, in all situations of interest the corresponding desired response of the system outputs is periodic as well, with same periodicity.

and its anticipated effect on the quality of learning. To test the validity of the general on-line error measure formulated in (2.22), the initial learning simulations were performed using exact gradient descent on the partitioned error (2.22), in three different regimes for the length of the partition intervals relative to the time scale of the trajectory training signal.

At first, the range of interval lengths Δt_k was chosen much shorter than the time scale of the target trajectory, $\Delta t_k \approx 0.1$ in comparison with the period $T = 2\pi$ of the trajectory cycle (1.7). In all trial cases from different initial conditions for the parameters, the training settled towards the same sub-optimal solution at convergence, with the network regenerating only the first lowest-frequency sinusoid output of the target signal (1.7) while failing to reproduce the other sinusoid target output at twice the fundamental frequency. Instead, the second output exhibited low-amplitude sinusoidal oscillations at the fundamental frequency, attempting to approximate the double-frequency target component as closely as possible. This deficiency follows directly from the short interval length Δt_k, which as seen before is inadequate to train internal dynamics of state variables not directly represented in the output units of the system. Since the phase diagram of the target outputs $({x_1}^T(t), {x_2}^T(t))$ for the trajectory in Figure 1.3 (b) contains a singularity in the derivative at the origin, the two output state variables x_1 and x_2 are clearly not sufficient to generate the trajectory dynamics, and at least one or more internal state variables need to be included in the dynamics along with the output units. These internal state variables are masked in the evaluation of the parameter dependence and therefore are effectively excluded in the error descent learning process, due to the short interval length Δt_k of the error measure.

To test the other extreme for the range of the interval length, similar simulations were performed with relatively long intervals Δt_k, in excess of several periods of the trajectory training signal. With a choice $\Delta t_k = 2T$ or larger, the simulation runs indicated the expected symptoms of a sharp and non-uniform parameter dependence of the error measure. Depending on the initial conditions selected for the parameters, qualitatively different solutions were obtained at convergence, quite frequently sub-optimal with similar

deficiencies in the second output $x_2(t)$ as encountered in the previous test case. The teacher forcing mechanism (1.8) used throughout all the simulations, terminally attracting the network dynamics towards that of the targets, did not prove successful in escaping these local minima on the error surface. In addition, the error descent process proceeded in rather non-uniform fashion, alternating between regions of fast and slow progress towards lower error levels[33].

Satisfactory and consistent learning results were obtained for simulations with interval lengths chosen in close proximity of the trajectory period, $\triangle t_k \approx T$. This agrees with the practical guideline formulated above for the choice of the interval length, $\triangle t_k \approx \tau_{\max}$, as the characteristic time scale $\tau_{\max}$ of the network dynamics under learning necessarily adapts towards the time scale of the training signal, with period T. Accordingly, the remaining simulations in this section specify a mean value for the interval lengths $\triangle t \equiv T$, and cover other aspects of the on-line learning methodology not yet empirically explored. Specifically, the simulations described below investigate the performance of the two presented on-line realizations of the stochastic method.

The simulation results corresponding to the concurrent realization with system replicas are presented in Figures 2.4 through 2.6. The simulation sessions were performed on the trajectory learning example (1.7) for $10,000$ stochastic update iterations using the procedure (2.23), with perturbation strength $\sigma = 0.001$ and effective learning rate $\eta = 0.02$ ($\mu = 2 \times 10^4$) [34]. The interval lengths were selected randomly at every iteration, in the range $\triangle t_k = (1 \pm 0.1)\, T$. Figure 2.4 shows the evolution of the error during learning, for four different sessions from identical initial conditions for the parameters. The free-running network dynamics resulting at convergence, obtained from all four sessions, are illustrated in Figures 2.5 and 2.6, showing the free-running trained output waveforms and the phase diagram trajectories of the corresponding limit cycles, respectively.

For direct comparison, the corresponding simulations results for the time-interlaced

[33] Such can be partially alleviated by using an adaptive biasing of the learning rate.

[34] As in [Toomarian 92], the teacher forcing amplitude was gradually decreased towards convergence to avoid a residual bias in the network dynamics, using the formula $\lambda = \lambda_0/(1 + \mathcal{E}(\mathbf{p}^{(k)})/\mathcal{E}_{\text{crit}})$, with $\lambda_0 = 1$ and $\mathcal{E}_{\text{crit}} = 0.005$. However, this did not prove very essential to obtain good results in this case.

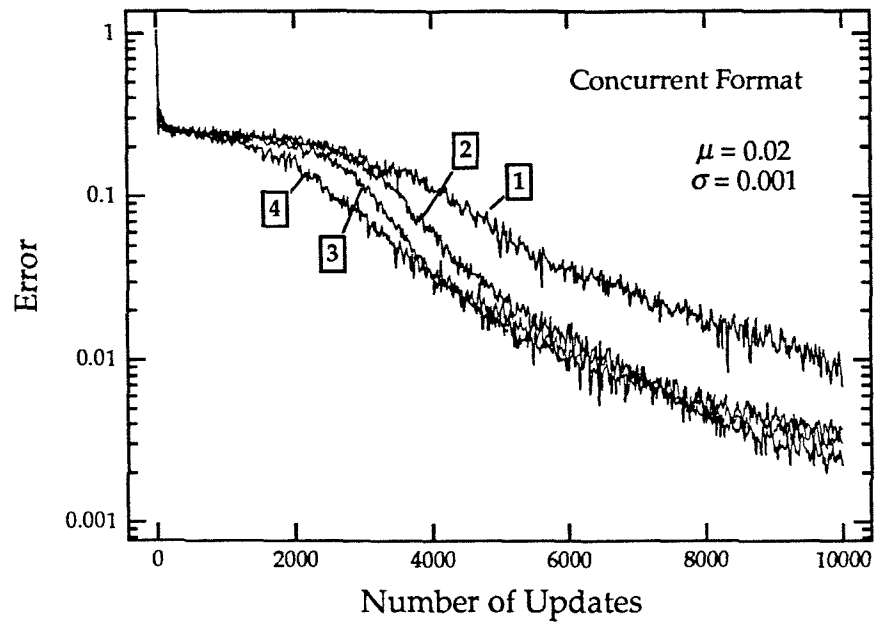


Figure 2.4: Error descent profiles of four sessions with the stochastic method using the concurrent format for the error observations.

realization with repetitive training, under identical or otherwise equivalent conditions as just specified for the concurrent case, are given in Figures 2.7 through 2.9. The parameter updates were obtained from (2.26) using the procedure (2.25) under perturbation schedule (2.24) and timing scheme (2.33), with $T = 2\pi$ and $\chi = 2$. The other settings of learning constants were retained from the previous case. In similar fashion, the error descent profile and the free-running network dynamics at convergence, including output waveforms and phase diagram trajectories, are illustrated in Figures 2.7, 2.8, and 2.9, respectively. While the underlying mechanism and supporting network configuration to generate the parameter updates is essentially different for both on-line realizations, the over-all error descent profile and the network dynamics obtained at convergence are evidently quite similar. One apparent difference between both concerns the relatively strong fluctuations in the error descent profile for the concurrent realization in Figure 2.4. However, the presumed discrepancy is merely an artifact in the formulation of the error functional. The fluctuations in the concurrent format of the on-line error result from the random positioning

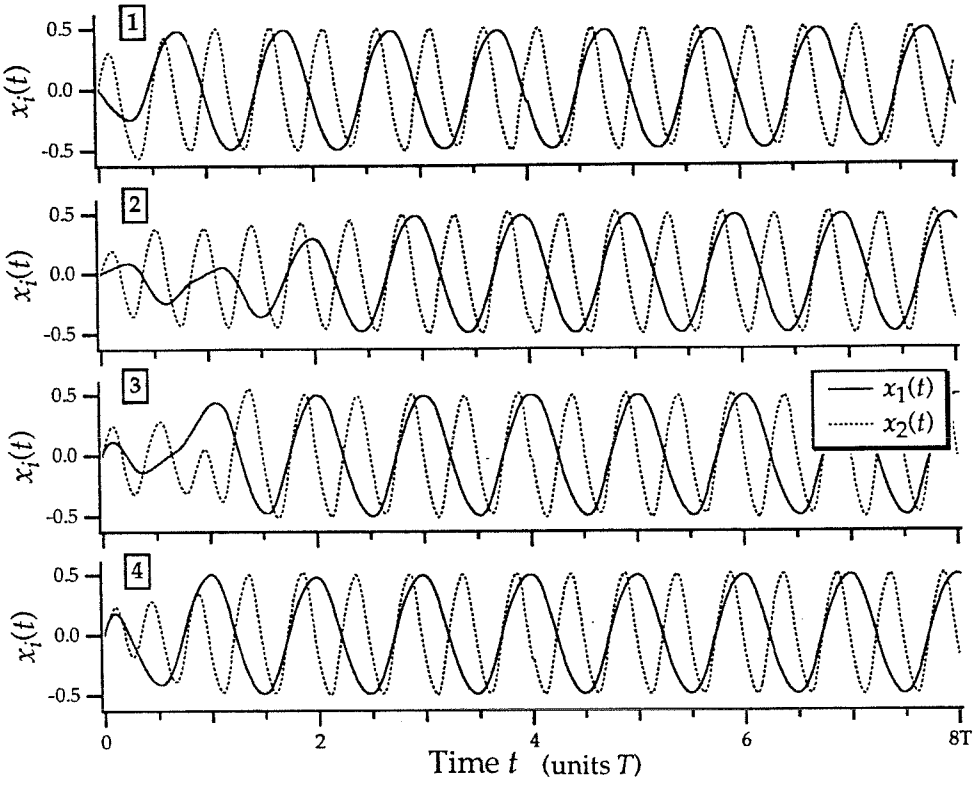


Figure 2.5: Free-running network dynamics obtained from four sessions of the stochastic method using the concurrent format: transient output waveforms.

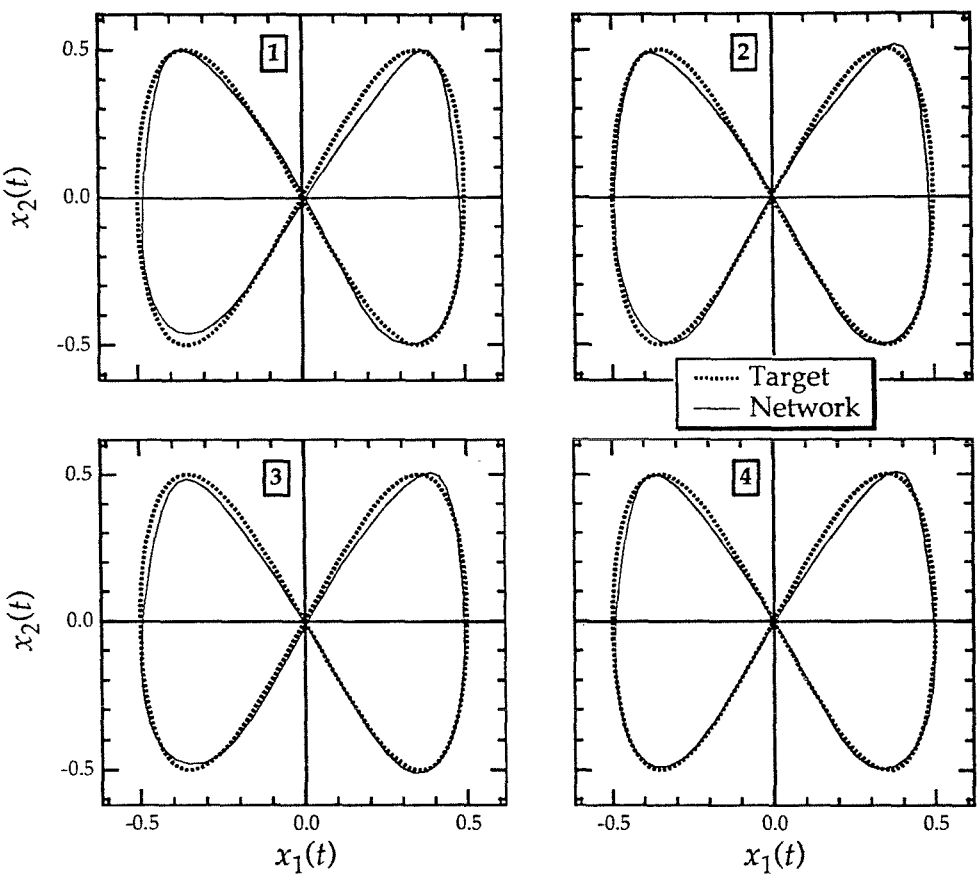


Figure 2.6: Free-running network dynamics obtained from four sessions of the stochastic method using the concurrent format: limit-cycle trajectory phase diagrams.

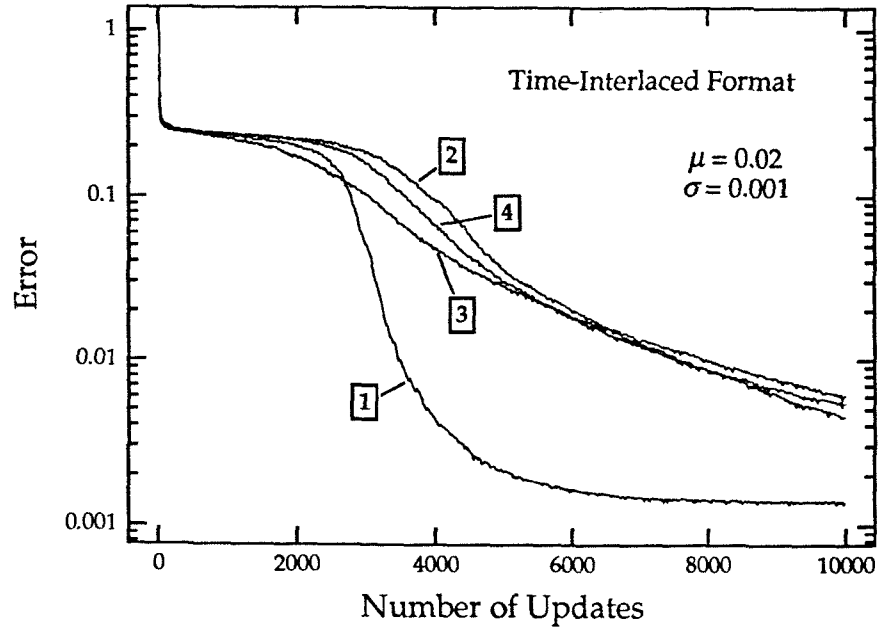


Figure 2.7: Error descent profiles of four sessions with the stochastic method using the time-interlaced format for the error observations.

of the observation partition intervals relative to the training signal waveforms[35] and do not affect the convergence process on a macroscopic scale. Otherwise, the quantitative agreement between both realizations indicates that the compensation mechanism used in the time-interlaced realization, to undo the effect of the drift in initial states on the error observations, is effective.

[35]The error measure displayed in Figure 2.4 is normalized to the length of the observation intervals Δt_k. However, fluctuations still arise because of the non-uniformity of the error contributions in time.

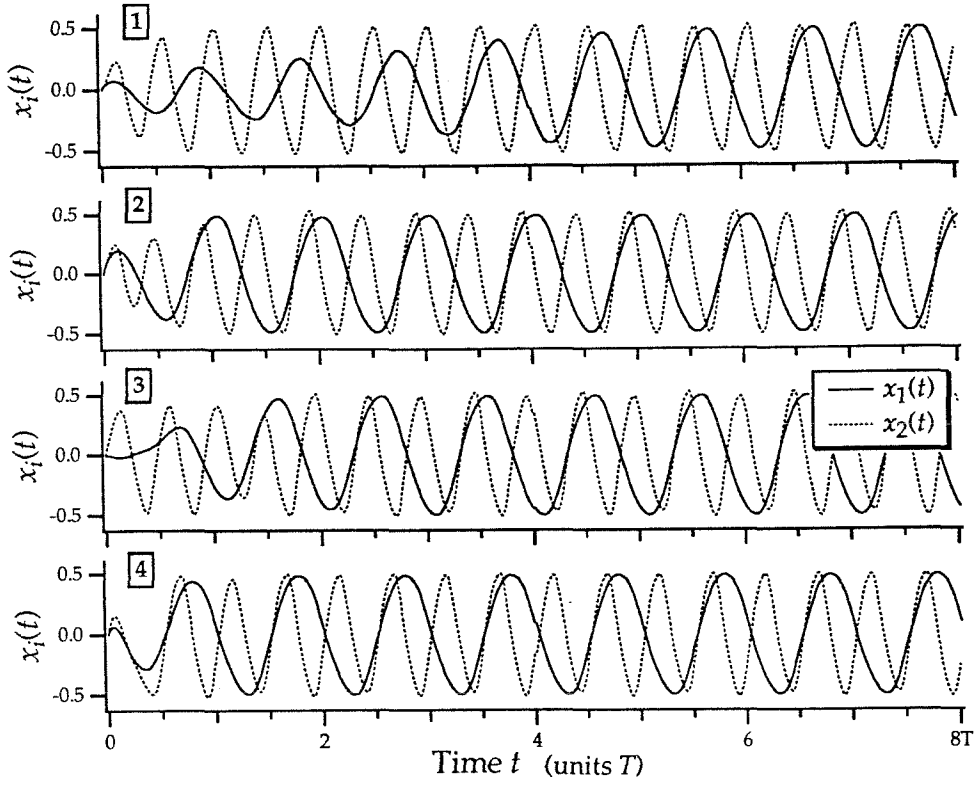


Figure 2.8: Free-running network dynamics obtained from four sessions of the stochastic method using the time-interlaced format: transient output waveforms.

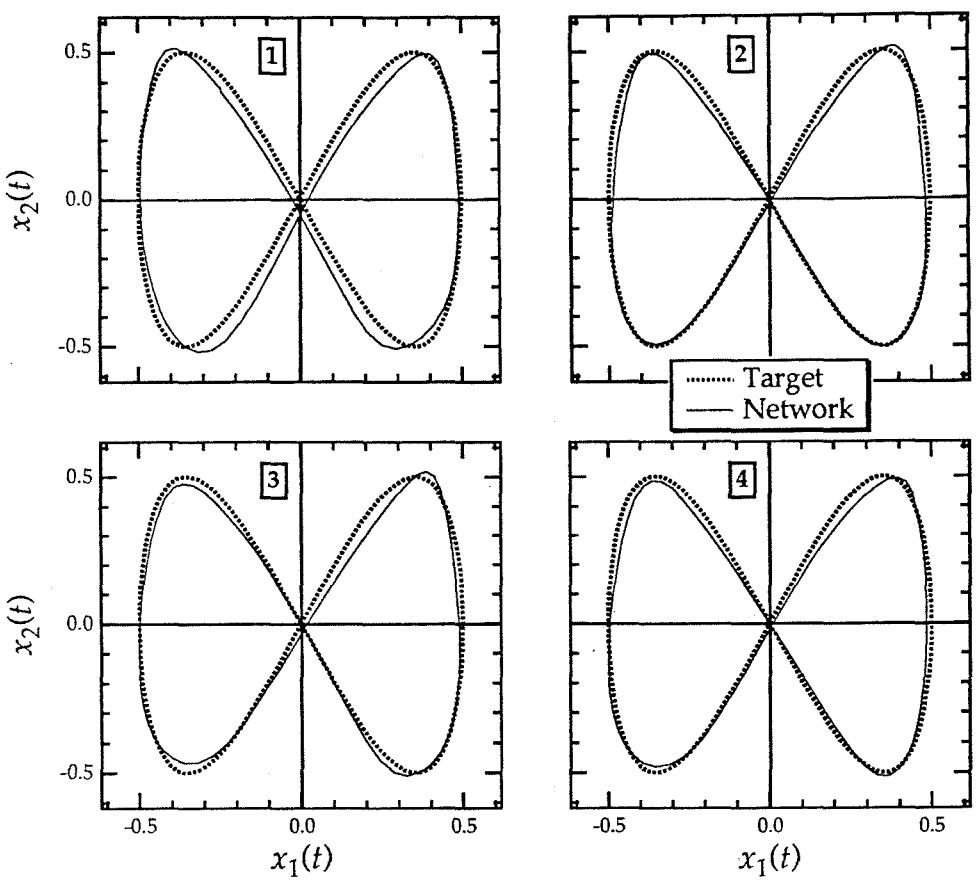


Figure 2.9: Free-running network dynamics obtained from four sessions of the stochastic method using the time-interlaced format: limit-cycle trajectory phase diagrams.

Chapter 3

Implementation Architectures

The simple structure and robust character of the stochastic method is particularly suited for special-purpose analog hardware implementations. In this chapter, we show that the method supports a modular and parallel implementation architecture with local units serving the update and perturbation functions for the individual parameters, thereby achieving a high computational efficiency for dedicated applications. Furthermore, we show that for analog VLSI implementations the resources providing the local update functions in the parallel structure can be used as well to dynamically store the volatile parameter values whenever the learning is disabled, using an autonomous and fault-tolerant refresh scheme which does not necessitate external storage.

Section 3.1 addresses architectural issues for the hardware implementation of the stochastic learning method in real-time application environments, with a discussion of design trade-offs arising in analog and digital implementation technologies. Of particular interest to analog VLSI systems is the time-interlaced on-line realization, which unlike the concurrent realization retains the model-independent nature of the stochastic method. Section 3.2 describes the refresh technique for dynamic multi-valued storage of the volatile parameter values.

3.1 On-Line Error-Descent Learning

The on-line gradient-free optimization methodology for dynamical systems presented in the previous chapter can be implemented on a variety of technology platforms, including general-purpose processors and dedicated digital or analog hardware, depending on the needs and the constraints of the application environment in which the system under optimization is embedded. Strictly judging from the arguments of scaling and computational complexity presented in Section 2.2, the stochastic method offers an efficient alternative for gradient descent and other incremental optimization methods on any of the available platforms. On certain platforms, however, the anticipated efficiency of the stochastic implementation may not necessarily be so obvious. On general-purpose processors, in particular, the provision of large memory storage, if available, allows for alternative, more computationally efficient off-line gradient descent implementations [Pearlmutter 89], [Sato 90a], [Werbos 90], which otherwise would be prohibitive due to timing constraints imposed by the real-time application environment. Likewise, in case model independence is essential for the application, more effective and intrinsically more sophisticated gradient-free alternatives for the stochastic method could be developed on such flexible platforms, involving specialized algorithms [Brent 73].

Consequently, the benefits of the stochastic method are utilized most on special-purpose platforms which only provide for limited functionality, leading to simple but effective low-cost implementations. Owing to their specialized nature, dedicated implementations tuned specifically to the application environment are intrinsically more efficient than equivalent implementations on general-purpose platforms, and are therefore able to achieve significantly higher bandwidths at modest levels of functional complexity. The specific advantages offered by dedicated implementations of the stochastic method and its on-line variants, in terms of functional simplicity, parallelism, and architectural modularity, are analyzed below. The general architecture of the stochastic optimization method, in relation to the configuration of the system and its parameters, is described first.

3.1.1 General Architecture

A simple and modular parallel architecture implementing the stochastic optimization method is naturally obtained by identifying local and global operations needed to construct and activate the incremental parameter updates[1]. A clear separation between both is indeed feasible, since the global functions specified by the stochastic method only involve a few scalar variables constructed from the error observations, contributing to all local functions in exactly the same manner.

Global and local functions for the stochastic method can be readily identified from the structure of expressions (2.2) through (2.5), and are symbolically represented in the architecture of Figure 3.1. Global operations include the perturbed and unperturbed error observations, and the corresponding estimate of the differential perturbed error $\widehat{\mathcal{E}}^{(k)}$ in (2.3). They proceed at a level surpassing that of the individual parameters and their location in the configuration of the system. Local operations then comprise the generation of perturbation components $\pi_i{}^{(k)}$ according to (2.5), and the construction and activation of the incremental parameter update components $\triangle p_i{}^{(k)}$ in (2.2). The local operations each pertain to one particular value of the component index i, thus representing one single parameter in the system and the corresponding provisions for constructing its update increments from the global error signals.

With reference to the general learning architecture of Figure 3.1 and the formulas (2.2) through (2.5), the alternating sequence of global and local operations implementing the stochastic method proceeds as follows. Initially, local random values are generated for the perturbation components $\pi_i{}^{(k)}$, one for every parameter $p_i{}^{(k)}$ in the system. The obtained perturbations $\pi_i{}^{(k)}$ are applied to the parameters $p_i{}^{(k)}$ of the system in complementary format, $\widehat{\mathbf{p}}^{+(k)}$ and $\widehat{\mathbf{p}}^{-(k)}$ respectively. Corresponding error observations on the outputs of the system then yield the perturbed errors $\mathcal{E}(\widehat{\mathbf{p}}^{+(k)})$ and $\mathcal{E}(\widehat{\mathbf{p}}^{-(k)})$ respectively[2]. Next, the

[1]Local operations are defined as functions implemented at the micro-level on a limited subset of variables, without requiring communication with other variables at the same level, except in the immediate neighborhood. Global operations interconnect the local functions at a higher level and involve communication across the complete set of participating variables.

[2]The practical organization of the complementary perturbations and corresponding error observations,

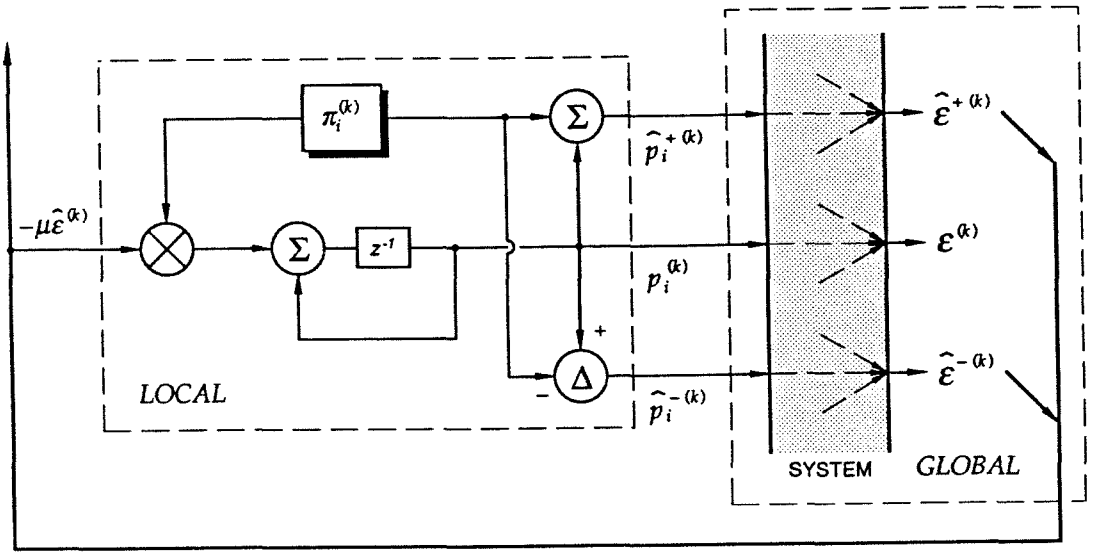


Figure 3.1: General architecture implementing the stochastic method.

obtained values for the complementary errors are combined to produce an estimate for the differential perturbed error $\widehat{\mathcal{E}}^{(k)}$, serving as a global signal used along with the local perturbation component values $\pi_i{}^{(k)}$ to locally construct the update increment components $\triangle p_i{}^{(k)}$. Subsequently, the parameter values are locally updated according to the obtained increment components. This sequence completes one update cycle, and the process is repeated for the next updates.

Two main factors contribute to the particularly high degree of computational efficiency and functional modularity of the implementation of the stochastic method and its corresponding architecture. First, the local operations relating to the perturbation components and parameter increments can proceed in full parallel fashion, by means of a set of independent and identical functional units provided locally, one for each individual parameter p_i[3]. This is because the local component-wise operations specified in (2.2) and (2.5) are functionally independent and identical for each of the parameter components, each interfacing to the global signal $\widehat{\mathcal{E}}^{(k)}$ in exactly the same functional way. The independent

regarding the concurrent and time-interlaced on-line realizations, will be dealt with in more detail below.

[3]For simplicity, only one local processing unit corresponding to a single parameter is represented in the architecture of Figure 3.1.

and identical format for the local update units supports a highly modular and integrated structure for the learning architecture, wherein update units can be freely added and removed in conjunction with adding and removing the corresponding parameters in the system. The added update units can then be integrated locally, directly interfacing with the embodiment of the corresponding parameter, into the physical system. In addition, the parallelism of the operations provides a large bandwidth for high-speed operation, unlike equivalent sequential implementations on general-purpose platforms.

The second factor relates to the simple implementation of the global functions. Usually, global operations require extensive interconnections between functional blocks across the structure, leading to scaling problems and a hardware overhead, typically causing excessive wiring. The stochastic method does not suffer from this interconnect problem, owing to the simple stochastic procedure to estimate the parameter dependence through scalar error observations on the system, in contrast to the elaborate calculations involving complex multi-dimensional operations usually required when estimating the full error gradient from a functional model. Instead, by using direct error observations on the system, as specified with the stochastic method, no additional global interconnect structures are required other than that already provided by the system itself[4]. The hidden global character effectively performed by the error observations follows from the underlying parameter dependence of the error measure used in the observations, which implicitly includes the combined effect of all individual parameters. The post-processing of the perturbed error observations, constructing the estimate $\widehat{\mathcal{E}}^{(k)}$ according to (2.3) involving only a few scalar variables, does not require significant functional resources in the implementation either. Likewise, the subsequent distribution of the obtained value for $\widehat{\mathcal{E}}^{(k)}$ to the entire set of local units, performing the individual parameter updates (2.2), does not require special provisions for interconnectivity in the implementation, since just one single signal line, shared by all local units across the whole structure, suffices to serve the purpose.

Therefore, the computational efficiency and modularity offered by the parallel im-

[4]In the case of the concurrent on-line realization, however, an extra overhead is imposed due to the required system replicas. See the further discussion in the text below.

plementation of the stochastic method are attained at relatively low levels of functional complexity, only involving simple operations with few operands in both local and global functions. With regard to actual digital and analog hardware implementations, the functional complexity of the implementation can be further reduced by simplifying the computational operations to be performed by the local update units, comprising the core of the implementation hardware. The computational ingredients required by the local operations, with regard to (2.5) and (2.2), include random number generation, addition or subtraction, and multiplication. Those are relatively easy to implement in general form; furthermore, significant simplifications result in the specific case of a symmetric binary distribution for the perturbations, $\pi_i^{(k)} = \pm\sigma$. While functionally very elegant, the symmetric binary distribution, with equal probabilities for both polarities, offers optimal efficiency in the stochastic updates among other distributions satisfying (2.5), complying to the divisive format of related stochastic approximation methods covered in Section 2.1[5]. Sequences of binary values with pseudo-random properties can be generated through elementary deterministic operations involving binary shift registers [Golomb 67], and simple extensions on this scheme offer a compact means of providing parallel multi-channel streams of pseudo-random bit sequences in VLSI environments [Alspector 91]. Furthermore, the binary format for the perturbations obliterates the need for full multiplication in the local operations, since the size of the update increments is fixed and identical for all parameters, determined exclusively by global factors and given by $\mu\sigma\widehat{\mathcal{E}}^{(k)}$. By directly providing this quantity globally to all local update units, instead of the global error signal $\widehat{\mathcal{E}}^{(k)}$, explicit multiplication is no longer needed in the local operations, and only addition or subtraction remains to be performed locally to obtain the increment values. The polarity assigned to the fixed-size update increments, representing what remains locally of the multiplications in (2.2), is then obtained from the local value of the perturbation bit $\pi_i^{(k)}$.

The amount of computational effort needed at the global level to obtain the size of

[5]A binary format for the perturbations is not appropriate in case an annealing schedule of the perturbations is used for non-convex global optimization, as presented in [Styblinski 90]. Then the simplifications regarding binary perturbations do not apply.

the update increments $\mu\sigma\widehat{\mathcal{E}}^{(k)}$ is to a large extent irrelevant, since the global operations generating this quantity are performed strictly once, common for all local update units. In fact, by providing some extra functionality and flexibility in the global operations, which account for a small fraction of the total implementation complexity anyway, a significant improvement in overall performance and efficiency of the optimization process may be achieved. Examples of potential schemes to improve learning performance, through additional complexity for the global operations, include adaptive biasing of the learning rate μ as determined by the estimated worst-case curvature of the error surface [LeCun 93], efficient line-search techniques [Brent 73] to find the minimum of the error along the given random direction of the perturbation, and one-dimensional extensions to second-order methods for convex optimization functionals[6]. In addition, the global operations can be extended to include measures guarding against instabilities in the error descent process, identifying unlikely conditions for the updates and formulating safe actions accordingly for recovery.

For practical implementation, the wide variety of the above procedures and other global functions supervising the optimization process are most conveniently incorporated on a separate general-purpose implementation platform, interfacing globally with the dedicated parallel hardware implementing the local update operations. General-purpose platforms, such as programmable micro-processors, offer the degree of flexibility and the spectrum of complex instructions essential to implement global operations with advanced high-level features, including *e.g.*, conditional branching for event-driven control functions. Since the global functions required for the stochastic method only involve a few scalar variables, obtained from error observations on the system, the intrinsic sequential nature of processing operations on a typical general-purpose platform does not create a significant bottleneck in the performance bandwidth. On the other hand, a dedicated parallel and scalable implementation for the local update units, integrated in the system architecture, is essential for achieving an efficient and inherently fast throughput of the pro-

[6]For instance, $\mu^{(k)} = 2/(\mathcal{E}(\widehat{\mathbf{p}}^{+(k)}) - 2\mathcal{E}(\mathbf{p}) + \mathcal{E}(\widehat{\mathbf{p}}^{-(k)}))$ approximating the inverse of the local second derivative along the direction of the perturbation, as obtained from Newton's method in one dimension.

cessed information. Consequently, the split configuration of specialized and configurable hardware for the local and global functions, respectively, establishes an optimal balance between functional efficiency and quality of performance achieved by the implemented architecture.

Implementation issues concerning on-line realizations of the stochastic method, specifically for real-time supervised learning in dynamical systems, are covered next. Since the concurrent and time-interlaced variants outlined above each support a different system configuration and application environment, their implementation is essentially different. The distinction between both is therefore carried through below, dealing with concurrent and time-interlaced implementations separately.

3.1.2 Concurrent On-Line Implementation

The concurrent on-line realization of the stochastic method, as outlined and discussed above, is aimed at applications where a well-defined dynamical system, in the form of a synthesized network controlled by a set of parameters, is required to generate time-varying outputs approximating given arbitrary signals, as those extracted from an external unknown process. The concurrent realization requires two exact replicas of the constructed network to be provided along with the original system, allowing independent settings for the parameter values in the three networks, and supporting direct access to the internal state variables. The latter is required to replicate the initial state $\mathbf{x}_k$ of the master system onto the replica systems, prior to the error observations (2.22).

Since the three networks need to be synthesized and considerable freedom is allowed in their implementation, an efficient concurrent on-line implementation of the stochastic method is obtained by integrating the local learning functions along with the implementation of the three networks, superimposed on the same substrate. As exact system replica are required, the preferred technology of the implementation is digital, to completely avoid imprecisions arising with analog-domain implementations. A fixed-point arithmetic is appropriate for this purpose, simplifying the implementation complexity.

The triple network structure, including the master system as well as the replicas into the same architecture, implies an overhead of roughly a factor two in the implementation, beyond that of the actual network itself. For a practical arrangement supporting the local format of the update and perturbation operations of the integrated learning functions, the functional elements representing one particular parameter in the three networks need to be physically adjacent to each other, near to the corresponding update unit. This is because the values $\mathbf{p}^{(k)}$, $\mathbf{p}^{(k)} + \boldsymbol{\pi}^{(k)}$ and $\mathbf{p}^{(k)} - \boldsymbol{\pi}^{(k)}$ supplied to the three instances of the same parameter need to be obtained from the same local update and perturbation unit. In addition, the functional elements corresponding to the same state variables in the three networks need to be located at adjacent positions in the implemented architecture as well, providing an efficient means for replicating the state of the master system onto the replica systems when so required, thereby avoiding excessive communication across cells. To achieve the closest possible distance between corresponding identical elements in the three networks, the intended structure of the master network can be physically replicated twice and superimposed onto the original structure. In a two-dimensional implementation environment, such as VLSI circuitry, the superposition of the three networks can be accomplished without problems of interference, by carefully displacing the integrated identical cells of the three structures relative to each other, and interlacing the corresponding interconnections. The local update units are then integrated at appropriate positions in the intertwined structure, directly interfacing with the three cells representing the corresponding parameter in the master and replica networks.

3.1.3 Time-Interlaced On-Line Implementation

The time-interlaced on-line realization of the stochastic method, in contrast, supports optimization of arbitrary dynamical systems of which the parameter dependence is not necessarily known from the start. By means of the four-fold alternating timing scheme for the perturbations and error observations under periodic training signals, as outlined and discussed above, the time-interlaced realization avoids the need for supplying separate

identical replica systems and accessing internal state variables. As a result, this approach is most suited for optimization of dynamical systems integrated in a physical environment, which deny direct access to the internal structure and underlying state variables, except implicitly through experimental observation of a dynamical performance index $\mathcal{E}(\mathbf{p})$ for given parameter settings $\mathbf{p}$ supplied onto the system. Examples already mentioned include direct adaptive control of nonlinear plants by means of parameter-driven dynamical networks, and more generally the optimization of a set of parameter values affecting the dynamical characteristics of an unknown but observable physical system. The class of applicable systems supported by the time-interlaced approach also includes cases whereby the model uncertainty does not arise from a lack of knowledge but rather from technical limitations in the implementation, such as mismatches found in analog electronic or mechanical systems implementing a dynamical model of the type (1.1).

Since virtually no constraints and requirements are placed on the structure of the system, the implementation of the learning functions supporting the physical system configuration is fairly general and straightforward. At the local implementation level, the update units each interface directly with the element representing the respective parameter value activated onto the physical system. Depending on the internal system structure, this parameter element may for example consist of a storage device containing a synaptic strength interconnecting artificial neural nodes or a filter coefficient evoking a dynamical response, or may supply a constant bias level interfacing with a general unspecified electrical or mechanical circuit. The local update units, providing the incremental updates and activating the perturbations to the corresponding parameter element, comply to the general structure specified above for the implementation of the stochastic method. More specifically, the unperturbed values $\mathbf{p}^{(k)}$ and complementary perturbed values $\mathbf{p}^{(k)} \pm \boldsymbol{\pi}^{(k)}$ for the parameter settings are supplied through a single channel in alternating phases, following the time-interlaced perturbation schedule (2.24). The timing of the perturbation is controlled at the global level, together with the synchronization of the error observations (2.25) in relation to the periodic training sequence, and with the remaining global

operations to construct the global update increment amplitude for further distribution to the local update units[7].

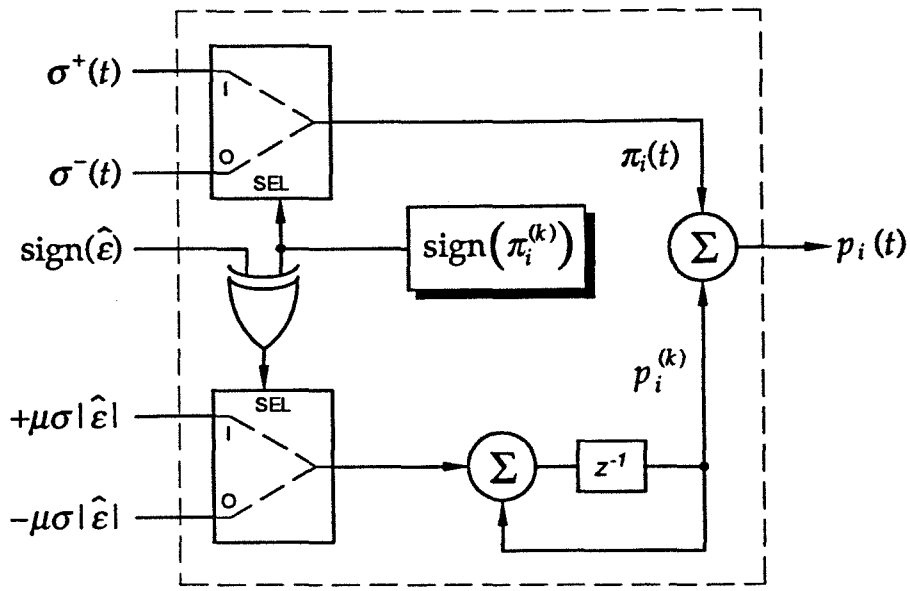
The parameters provided by the update units need to be supplied in some analog format, compatible with the accepted input format for the parameter elements in the physical system. A digital implementation of the local update units would therefore imply the need of digital-to-analog conversion to interface between the two different parameter representations in the update units and in the corresponding parameter elements. Provisions for data conversion at the local level are quite expensive, and can be avoided by implementing the local update functions in the analog domain instead. An analog implementation technology for the update units offers additional advantages, besides the matching format at the interface between parameter representations. In order to perform simple functions as those specified for the updates, analog implementations tend to require significantly less device elements than equivalent digital implementations. While the increased implementation efficiency of an analog representation necessarily comes at the expense of a loss in precision, the inherent fault tolerance of the stochastic method allows to a large extent for imperfections of various kinds in its implementation. Imperfections typically encountered with analog implementations include systematic offsets arising from variations in the fabrication process and random fluctuations caused by noise. The tolerance to random, non-repetitive errors is evident from the principles of stochastic approximation, on which the method is based. Systematic errors in the update and perturbation functions are recovered by the stochastic method as well, to an extent which depends on the specific details of the implemented architecture.

A robust architecture, virtually immune to systematic offsets, is obtained by implementing the update functions in hybrid analog-binary technology, owing to the simplifications of the multiply operations by the binary format for the perturbations. The analog-binary architecture for the update units is illustrated in Figure 3.2 (a), and the corresponding

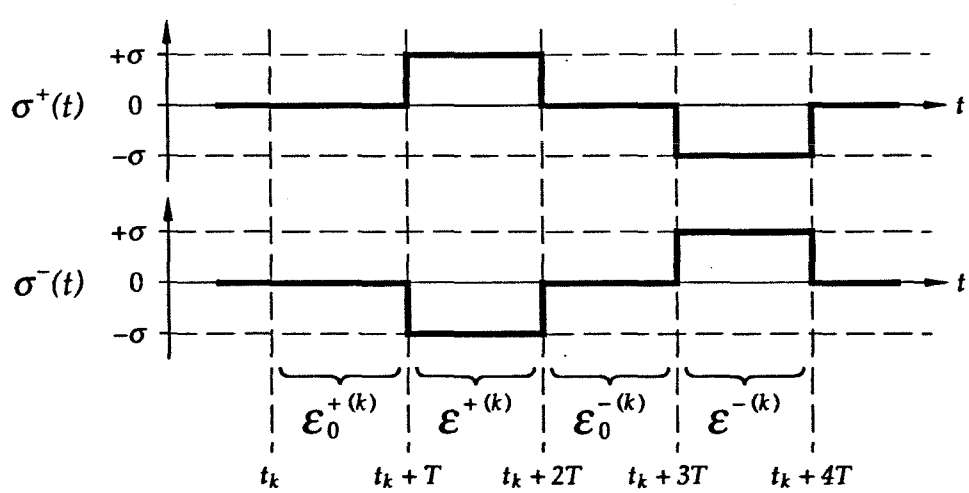
[7]In case a binary format for the perturbation components is not admissible, the "stripped" global value $\mu\widehat{\mathcal{E}}^{(k)}$ is distributed instead, and the update increments are constructed locally through multiplication with the corresponding perturbation value.

timing and waveform diagram, with time-interlaced activation of the perturbations and observation of the error, is given in Figure 3.2 (b). Conceptually, the architecture performs all local operations involving the perturbation values in rigorous binary format, while retaining the full analog format for the parameter values and their increments. Correspondingly, the value of the global increment size $\mu\sigma\widehat{\mathcal{E}}^{(k)}$ is decomposed into its amplitude $\mu\sigma|\widehat{\mathcal{E}}^{(k)}|$ and polarity $\mathrm{sign}(\widehat{\mathcal{E}}^{(k)})$, which are distributed separately to the local update units in the form of three global signals $\mu\sigma|\widehat{\mathcal{E}}|$, $-\mu\sigma|\widehat{\mathcal{E}}|$ and $\mathrm{sign}(\widehat{\mathcal{E}})$. Therefore, the incremental updates in the parameters $p_i^{(k)}$ are obtained by selecting the appropriate update value, among the two global signals $\mu\sigma|\widehat{\mathcal{E}}|$ and $-\mu\sigma|\widehat{\mathcal{E}}|$, for addition with the previous parameter value. As shown in Figure 3.2 (a), the selection between both is determined by an exclusive-or (XOR) operation of the local binary perturbation $\pi_i^{(k)}$ and the global binary signal $\mathrm{sign}(\widehat{\mathcal{E}})$, yielding the negative increment in case both polarities are identical, and the positive increment otherwise. Likewise, the two possible instances for the time-varying perturbation components $\pi_i(t)$, according to (2.24) for either polarity of $\pi_i^{(k)}$, are provided separately in the form of two globally distributed signals, with waveforms $\sigma^+(t)$ and $\sigma^-(t)$ of opposite polarity shown in Figure 3.2 (b). Consequently, the local parameter values $p_i(t)$ supplied to the system are obtained by addition of the updated parameter value $p_i^{(k)}$ and the perturbation signal $\pi_i(t)$, selected from the two global signals $\sigma^+(t)$ and $\sigma^-(t)$ by the polarity of the local perturbation $\pi_i^{(k)}$. The binary selection operations for the local generation of the analog update and perturbation values significantly simplifies the implementation structure, which beyond the analog multiplexers only requires simple analog addition elements.

Besides simplifying the structure of the implementation, the binary format for the perturbations improves the fault tolerance to systematic errors in the implementation, as a result of carefully separating the binary and analog components in the representation of the local variables affecting the parameter updates and error observations. The decomposition of the operands into analog and binary parts obliterates the need for analog multiplication, thereby avoiding potential sources of both systematic and random error due to analog



(a)



(b)

Figure 3.2: Analog-binary time-interlaced implementation of the stochastic method. *(a)* Local cell structure. *(b)* Waveform and timing diagram.

offsets and noise in the implementation. Systematic offsets are avoided in the analog-binary implementation, since the binary variables are able to rigorously control the polarity of the increments, regardless of the size of the differential observed error in (2.2)[8]. As a matter of fact, the rigor in the sign of the incremental updates is the main determining factor in the quality of convergence for the implementation of the method, more important than the exact analog amplitude of the update increments. This allows substantial room for analog imprecision in the implementation without drastically affecting the learning performance. In effect, the dual analog-binary implementation manages to combine the advantages of efficiency and precision from both analog and digital worlds, resulting in a simple and fault-tolerant architecture for the update units.

3.2 Fault-Tolerant Dynamic Multi-Level Storage

While the analog representation of the parameter values in the latter implementation allows for a compact and simple structure of the update units, the volatile nature of the capacitive medium used to locally store the parameters necessitates a mechanism to counteract the spontaneous drift in the analog values due to junction leakage and other drift phenomena. Below, we present a fault-tolerant refresh technique for dynamic multi-level analog storage of the parameter values, which can be implemented locally using virtually the same functional elements as needed for the above local learning update functions.

3.2.1 Partial Incremental Refresh

Methods for dynamic analog storage in VLSI essentially quantize a given stored analog parameter value to one of a discrete set of voltage levels, and repeatedly refresh the analog value towards the identified discrete level to counteract the drift of the volatile storage medium. As covered before in Section 1.3, the typical scheme employed for refresh

[8]This assumes a proper circuit-level implementation of the update accumulation function in Figure 3.2. A real-time learning system based on the time-interlaced approach, with specific analog VLSI implementation of the update units, is described and demonstrated in the next chapter.

consists of identifying the discrete level closest to the stored analog parameter value, and then updating the analog value by loading the identified discrete level onto the storage device [Terman 81], [Hochet 89], [Hochet 91], [Vittoz 91]. The refresh scheme used here [Cauwenberghs 93c] is more robust to random errors, by incorporating functions which exploit redundancy and statistical averaging to overcome the sudden effect of random errors in the quantization.

Specifically, the refresh method repeatedly applies small fixed-size increments in the parameter values in the direction of the nearest discrete level, rather than completely substituting the stored analog value with the identified level. The degree of redundancy in the refresh updates offered by the method depends on the amplitude of the fixed-size increments and decrements, which is specified to be much smaller than the separation between adjacent discrete analog levels. For a small refresh step amplitude, the identification of the exact nearest level on an individual base is not critical, which may even be strongly erroneous as long as the fraction of occurring errors is small enough. Furthermore, the information required from the analog value to define a refresh action is binary, since only the *polarity* of the refresh action needs to be specified, and the nearest level is not explicitly required in the refresh. The polarity to be determined depends uniquely on the position of the analog parameter value relative to that of the nearest discrete value, and can be rendered in the form of a binary quantization function defined on the analog input variable, alternating between opposite polarities according to the distance of the analog value from nearby discrete levels:

$$Q(.) : \mathcal{R} \rightarrow \{-1, +1\} \ . \tag{3.1}$$

Essentially, the binary quantization function $Q(.)$ encodes the direction of the analog value towards the nearest discrete level, though in principle a general quantization function can be specified for implementation without explicit reference to given discrete levels, thereby defining rather than employing the set of discrete levels. An example illustrating the general shape of the binary quantization function $Q(.)$ is shown in Figure 3.3. The arrows on the graph indicate the refresh action taken according to the polarity obtained

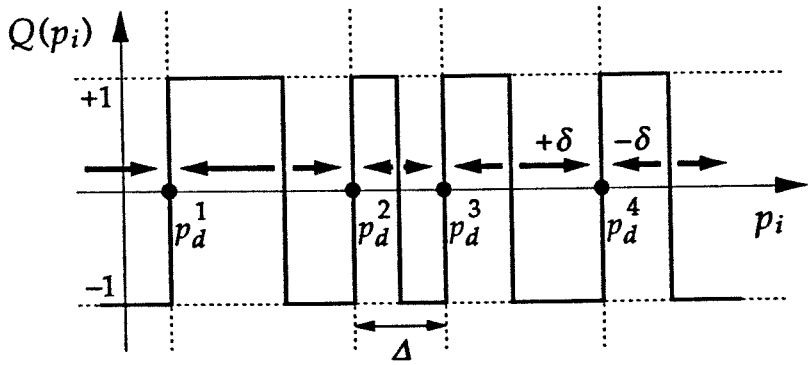


Figure 3.3: Example illustrating the binary quantization function $Q(.)$.

from the binary quantization. The analog value is decreased by an amount δ for a positive polarity of the quantization bit, and is increased by that amount otherwise. Formally,

$$p_i^{(k+1)} = p_i^{(k)} - \delta\, Q(p_i^{(k)}) \ , \tag{3.2}$$

with $p_i^{(k)}$ the instance of the analog parameter value prior to the update cycle k, and with δ the amplitude of the refresh step. Under periodic iteration of this method, the analog value p_i undergoes strong attraction locally towards a nearby boundary region where the quantization makes a positive transition, from a -1 to a $+1$ value. These transition boundaries define the positions of the discrete memory levels $p_d{}^j$ as indicated in Figure 3.3. The binary quantization and the incremental refresh hence define alternating regions of attraction and repulsion, creating and isolating the discrete levels for stable memory operation.

For a small refresh amplitude δ, the strength of attraction and repulsion in the alternating regions remains virtually unattenuated under noise contaminating the binary quantization and causing occasional errors in $Q(.)$. Stated differently, a large number of consecutive erroneous quantization bit values, steering the analog value in the wrong direction, are required to cause a transition from one stable state to another, such transition becoming exponentially less likely to occur as the refresh amplitude δ is decreased. A small value for δ is desirable as well to reduce the dynamic refresh noise due to the persistence of

the finite update increments at equilibrium, which ultimately limit the analog resolution of the stored value. Nevertheless, the value for δ cannot be decreased beyond the nominal amplitude of the drift due to leakage over one refresh time interval, to avoid a runaway unstable condition for the stored value. Thus, an appropriate value for δ needs to satisfy the condition

$$|r_d|T < \delta \ll \Delta \ , \tag{3.3}$$

with r_d the leakage drift rate, T the refresh time interval, and $\Delta = \min_j(p_d{}^{j+1} - p_d{}^j)$ the minimum separation between adjacent discrete memory levels. Because the leakage is temperature dependent and is hardly expected to be uniform across different instances of memory cells, a safety margin for the lower bound in (3.3) is advisable, such as $\delta = 10|r_d|T$. The time scale of refresh T in the latter then needs to be adjusted to accommodate the other condition $\delta \ll \Delta$. Clearly, the increased fault-tolerance of the modified refresh procedure comes at the expense of an increase in the minimum refresh rate required to control the drift of the volatile medium, with the amount of increase corresponding to the degree of redundancy Δ/δ. Nevertheless, in practical analog VLSI situations with modest leakage rates of capacitive storage, the requirement (3.3) is still fairly easy to accommodate.

3.2.2 Implementation Structure

The functional diagram of the partial refresh method, corresponding to the binary quantization and incremental update operations of Eqns. (3.1) and (3.2), is shown in Figure 3.4. The drift of the storage medium and the noise affecting the quantization are represented symbolically as additive components, included in the dashed-line inset of Figure 3.4. The structure of the diagram comprises a loop of quantizing and integrating elements somewhat reminiscent of the technique of delta-sigma modulation [Candy 92], which similarly incorporates redundancy and statistical averaging to achieve high accuracy, mainly for applications of data conversion. Obvious differences between the refresh method and single-bit delta-sigma modulation concern the alternating format of the binary quantization function of Figure 3.4 and the absence of a difference ("delta") unit.

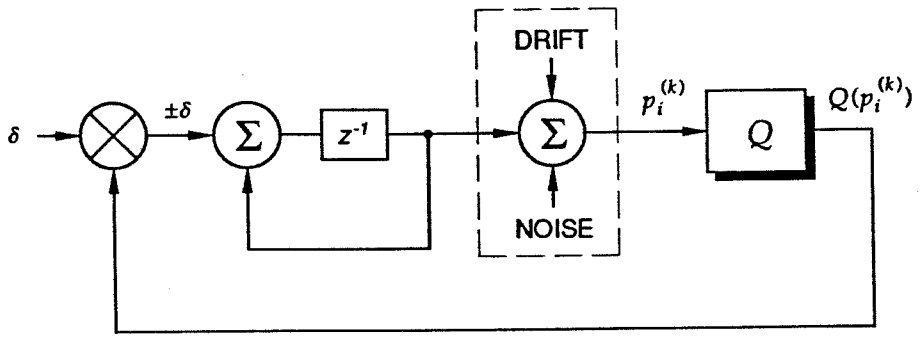


Figure 3.4: Functional diagram of the partial refresh method.

More relevant to the scope of this chapter is the similarity between the update functions of the above on-line analog-binary learning architecture and the present refresh method, both activating either increments or decrements of a fixed amplitude onto the parameter values as determined from the polarity of a locally supplied binary value. Indeed, as shown in Section 3.1 and illustrated in Figure 3.1, the learning increment amplitude is a globally defined analog quantity, supplied uniformly to all update cells, and the polarity of the learning increment is selected from a locally generated binary value. Therefore, the implementation of the learning and refresh functions can be efficiently combined in a single integrated architecture, with both functions sharing the same local elements supplying the incremental parameter updates.

In the remaining of this chapter, we describe the system-level configuration and circuit-level structure of the functional elements providing the binary quantization and the update increments, for realizing analog VLSI storage in general. The implementation of a recurrent dynamical neural network with integrated learning and storage architecture, employing the same incremental update elements, is described in the following chapter.

Two architectures for implementing the refresh procedure are given in Figure 3.5. Both configurations comprise the same functional elements, a binary quantizer Q and an increment/decrement device I/D, which interface with a capacitive storage device C to

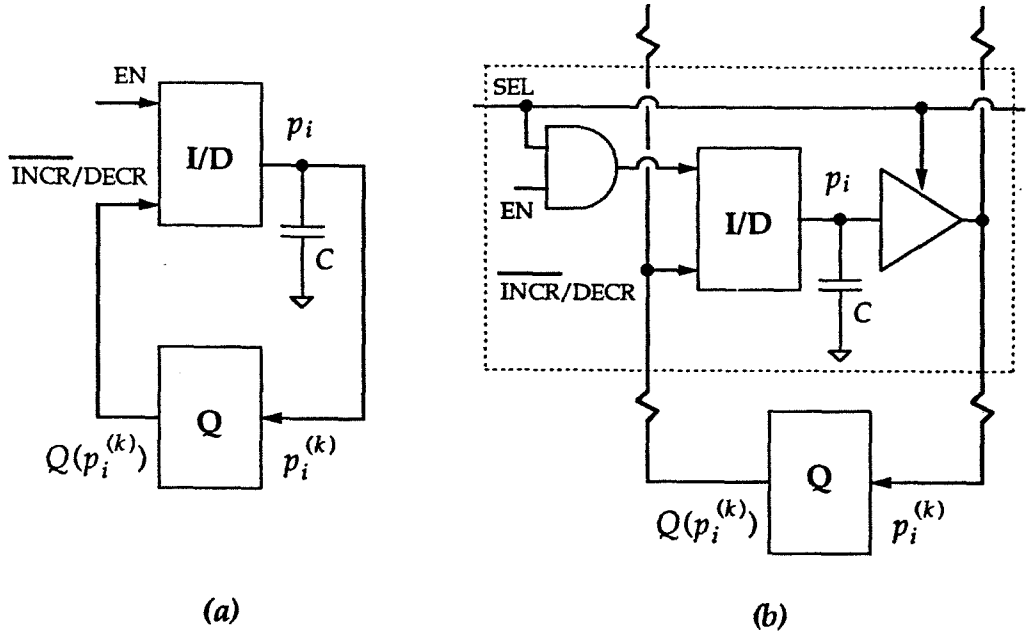


Figure 3.5: Architectures implementing the partial incremental refresh method. *(a)* Standard configuration of the analog memory cell. *(b)* Array configuration of analog memory cells with time-multiplexing of a common quantizer.

generate a binary quantization value and to produce the corresponding refresh increment, respectively. The first configuration of Figure 3.5 (a) employs dedicated instances of elements Q and I/D for every individual storage device C. In the second configuration of Figure 3.5 (b), one binary quantizer Q is shared among several memory cells, each containing a storage device C and an I/D element, in a multiplexed arrangement for sequential refresh. The particular memory cell to be refreshed at a given moment is identified by means of a select signal SEL, driving the output multiplexing and update enabling circuitry schematically depicted in Figure 3.5 (b). In a practical multiplexed arrangement, the select signal cycles through all connecting memory cells in sequence for a uniform and periodic refresh cycle among all cells. While it requires a quantizer of sufficient bandwidth to support the entire array of memory cells at the desired refresh rate, the multiplexed configuration may result in a significant reduction of the cell complexity, depending on the complexity of the circuitry implementing the binary quantizer.

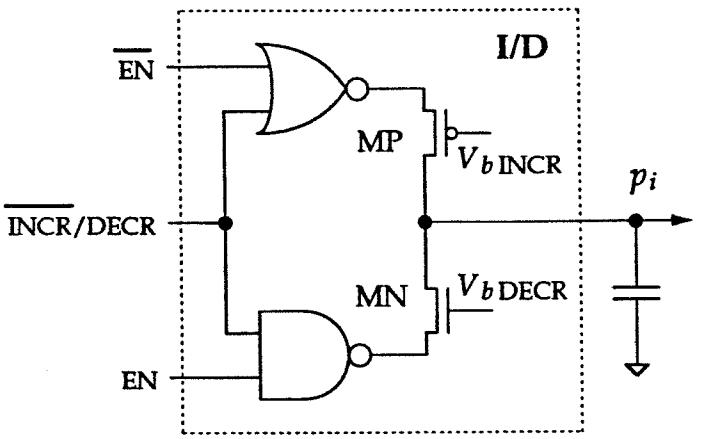


Figure 3.6: A CMOS charge-pump implementation of the I/D device.

Bipolar Incremental Update Element

Figure 3.6 shows a binary controlled analog CMOS charge pump, providing an increment/decrement device I/D offering a fine resolution of the increment amplitude ranging over several decades. The charge pump device, driving a capacitive storage device C at its output, dumps either a positive or a negative update current, of equal amplitude, onto the storage capacitor whenever it is activated, effecting either a given increment or decrement on the parameter value p_i respectively.

The positive and negative currents of the device are supplied by complementary MOS transistors MP and MN, and their amplitudes are controlled by the bias voltage levels $V_{b\,\mathrm{DECR}}$ and $V_{b\,\mathrm{INCR}}$ on the gates of the transistors, respectively. The particular transistor, supplying the current for either increment or decrement action, is selected by the input binary signal $\overline{\mathrm{INCR}}$/DECR, obtained from the quantizer, and the selected current is activated onto the storage capacitor at the output depending on the logic state of the complementary enable control signals EN and $\overline{\mathrm{EN}}$, respectively. For increment action, the injecting current source MP is activated by driving EN active high and $\overline{\mathrm{EN}}$ active low. Likewise, for decrement action, the sinking current source MN is activated by driving both EN and $\overline{\mathrm{EN}}$ active high. The selection and activation of the currents from the logic control levels is accomplished by means of NAND and NOR gates, which drive the sources of the MOS

transistors MN and MP.

Note that the switching of the current supplied by transistors MN and MP is controlled by driving the source voltage on the transistors as opposed to driving the gate voltage. Such strictly avoids switch injection noise, a typical phenomenon affecting switched capacitor circuitry [Gregorian 86], resulting in unpredictable offsets induced on the voltage across a capacitor when a switch coupled to it is opened. Because the gate voltages of MN and MP are kept at constant bias, no such parasitic injection of charge can reach the storage capacitor when either current is deactivated. By virtue of the clean switching transients, the I/D device is therefore able to resolve reliable small size increments and decrements $\pm\delta$.

To attain small values for the increment amplitude $\pm\delta$, the gate voltages of transistors MN and MP are biased in the subthreshold range [Vittoz 77], [Mead 89], allowing for exponential control of the current amplitude over several decades down to pA levels. While a large dynamic range of $\pm\delta$ is not essential for the refresh increments, it is essential for the implementation of the learning update increments as stipulated earlier in Section 3.1. Furthermore, the subthreshold characteristics are desirable for a robust implementation of the refresh method as well, for reasons of physical origin. Both the subthreshold current across a MOS transistor and the leakage current across a reverse biased p-n junction follow the same $\propto$ É $\exp(-1/kT)$ temperature dependence, such that the subthreshold refresh amplitude and the spontaneous leakage of the storage capacitor exhibit matched thermal characteristics. Therefore, in a practical analog VLSI context the requirement for stability imposed by the left-hand side of (3.3) can be met over a wide temperature range.

A/D/A Binary Quantizer

In principle, any device constructing a binary output value from an analog input value in fairly consistent manner can perform the function of binary quantizer Q, provided the transfer characteristic contains sufficient and uniformly distributed bit transitions to accommodate the desired analog resolution of the memory. Practical realizations can be

obtained as direct extensions on the quantizers used with the previous refresh methods, for identifying the discrete level closest to the analog value.

We describe one realization, based on the concept of combined dual A/D (analog-to-digital) and D/A (digital-to-analog) conversion as explored for analog quantization purposes in [Terman 81]. The A/D/A approach offers an attractive implementation for the quantizing element in general, since it additionally provides digital access to the stored analog parameter values in both read and write formats, owing to the available A/D and D/A conversion functions. The particular bit-serial implementation of the A/D/A converter, as specified below and described in detail in Appendix A, is especially attractive for high-resolution dynamic analog storage, since it does not require resources to physically supply or trace the complete set of discrete analog levels in the implementation.

The schematic representation of the bit-serial bi-directional A/D and D/A converter used for binary quantization is depicted in Figure 3.7, conform to description of a practical implementation in Appendix A. The core of the A/D/A converter comprises an algorithmic bit-serial D/A converter, processing the bit sequence of the digital word in the order from most significant bit (MSB) to least significant bit (LSB), algorithmically generating the analog output value from successive intermediate values corresponding to the partial bit-accumulated conversion results. With the addition of a latched comparator, the device furthermore operates as a successive approximation A/D converter which employs the intermediate values of D/A conversion as successive approximations for comparison with the analog input, to determine the bit sequence of the digital output in the order of MSB to LSB. The combination of D/A and A/D conversion within a single device provides a means of bi-directional access between the analog and digital domains, with matched transfer characteristic in both directions of conversion by virtue of sharing the same hardware. The implementation of the A/D/A converter as further specified in the Appendix A additionally provides guaranteed monotonicity in the conversion characteristics regardless of component mismatches and nonlinearities.

The A/D/A converter performs binary quantization of the analog input in algorithmic

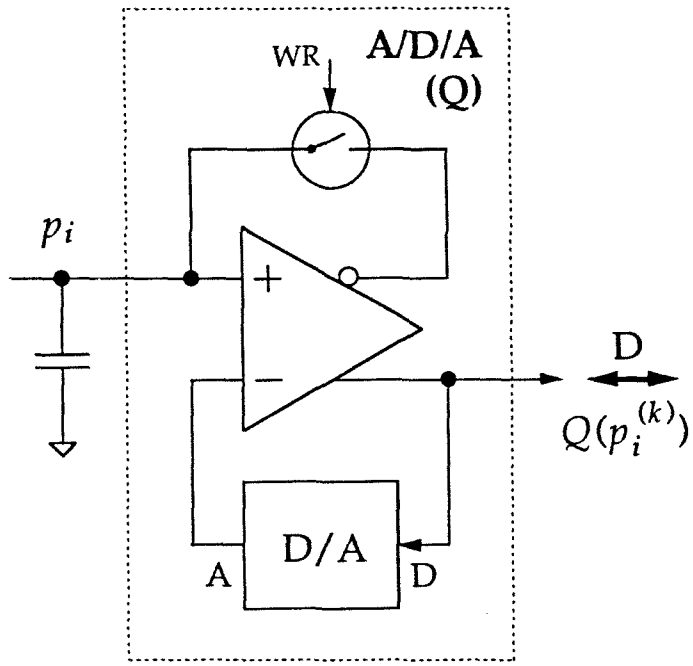


Figure 3.7: Binary quantizer comprising a bit-serial A/D/A converter.

fashion. For an analog memory supporting n-bit resolution, *i.e.*, containing 2^n discrete levels, the quantization bit is obtained from the least significant bit for $(n+1)$-bit A/D conversion, identified as the last bit in the bit-serial sequence of the digital output. The LSB of the $(n+1)$-bit conversion follows the desired regular profile of alternating polarities of which the positive transitions, marking the positions of the equilibrium discrete memory levels under periodic refresh, coincide with the 2^n analog levels corresponding to the n-bit partial D/A conversion. Therefore, the multi-valued state of the memory is uniquely identified in digital format by the partial n-bit sequence preceding the LSB. While the method of dynamic refresh only retains the LSB as information necessary for the binary quantization, the remaining bits preceding the LSB provide the key to the digital access to the stored parameter values, exploiting a different operation mode provided by the A/D/A converter. Specifically, the following modes of operation are supported with the A/D/A embodiment of the binary quantizer, for a 2^n-level analog memory:

a. **partial refresh,** by obtaining the quantization bit Q from the LSB of (n+1)-bit A/D conversion and subsequently activating the I/D device with the polarity of Q;

b. **read access in digital format,** by obtaining the digital word identifying the memory state by means of n-bit A/D conversion. Alternatively, to support uninterrupted dynamic refresh in the background, the digital word can be derived from the partial n-bit result obtained during the (n+1)-bit conversion needed for the binary quantization in procedure *a*;

c. **write access in digital format,** by D/A conversion of the n-bit digital data to be written into the analog memory, and by subsequently activating the comparator to yield a binary value for Q, which reflects the result of comparing the analog memory value with the constructed analog conversion value. The assignment procedure for the binary value of Q is followed by activation of the I/D device with the polarity of Q, in order to effect a partial update for the analog memory value in the direction of the D/A conversion result.

Periodic iteration of the update produced from the write procedure *c* establishes the desired value, the discrete level corresponding to the digital word, onto the storage memory element, in a fault-resistant and noise-tolerant manner similar to the mechanism of dynamic partial refresh. To speed up the writing process, the analog value on the storage device can be preset to a coarse approximation to the desired value, prior to the fine-tuning under the iterative process of the updates under procedure *c*. Such is achieved by temporarily closing the WR switch in Figure 3.7 after the initial D/A conversion, which forces the analog value of the D/A result onto the storage element by means of high-gain negative feedback. To that purpose, an auxiliary inverted transconductance output is provided with the comparator circuitry, as indicated in Figure 3.7 and further documented in Appendix A.

Test results obtained on an integrated array of storage devices containing the above specified binary quantizers and I/D elements are included in the following chapter.

Chapter 4

Analog VLSI Systems

The present chapter describes the implementation and experimental performance of a complete learning and storage system, integrated with a recurrent dynamical neural network on a single analog VLSI chip. While the system incorporates all of the local functions needed for the parameter updates and the perturbation values, the global functions and higher-level instructions are performed off-chip to allow for a greater flexibility in the experimental set-up, as needed to characterize the performance at different levels. Results obtained on a separate but compatible system are also included, to demonstrate autonomous operation of dynamic storage using on-chip integrated A/D/A binary quantizers.

A brief specification of the implemented network model and learning algorithm is given in the next section. Section 4.2 describes the architecture of the integrated network and learning system, and the circuit implementation of its functional elements. Experimental results on learning a periodical continuous-time trajectory are analyzed in Section 4.3. Results on the autonomous dynamic refresh system with integrated quantizers are presented in Section 4.4.

4.1 System Architecture

While the implemented network and learning model follow the specific structure outlined in Chapters 1 and 2 and used for the simulations in Chapter 2, there are some differences mainly motivated by the resulting simplifications in the analog VLSI implementation. Owing to the model-independent nature of the implemented stochastic perturbative learning algorithm, none of the applied network changes is expected to affect the learning performance. The impact of the functional simplifications carried out in the implementation of the learning algorithm, especially regarding the timing of the error observations, will be discussed further below. The modified structure and functionality of the network and the learning algorithm are reformulated next.

The implemented network contains six fully interconnected recurrent neurons with continuous-time dynamics,

$$\tau \frac{\mathrm{d}}{\mathrm{d}t} x_i = -x_i + \sum_{j=1}^{6} W_{ij}\, \sigma(x_j - \theta_j) + y_i \ , \tag{4.1}$$

with $x_i(t)$ the neuron state variables constituting the outputs of the network, $y_i(t)$ the external inputs to the network, and $\sigma(.)$ a sigmoidal activation function. The free parameters, to be optimally adjusted by the learning process, constitute the 36 connection strengths W_{ij} and the 6 thresholds θ_j. The time constant τ is kept fixed in the present implementation, and has an identical value for all neurons. As in the previous chapters, the weights W_{ij} and thresholds θ_j will be considered below as components of the same parameter vector p.

The network output consists of the two neuron signals $x_1(t)$ and $x_2(t)$, which are trained in supervised mode with target output signals $x_1^T(t)$ and $x_2^T(t)$ presented to the network. For the specific trajectory learning example used in the experiments, no inputs $y_i(t)$ are specified along with the target outputs $x_i^T(t)$. However, the implemented architecture allows for more general learning tasks typical in applications of identification and control, for which the training signals $x_i^T(t)$ comprise the desired dynamical response of

the network under activation of inputs $y_i(t)$.

While the integrated learning system fully supports the on-line time-interlaced format of the error observations as described in Section 3.1, a modified on-line scheme is used instead, significantly simplifying the timing organization of the experimental set-up. Though a time-interlaced format would definitely have granted faster learning speed and more uniform convergence behavior, the simplified version still allows to demonstrate the general principle. Essentially, the modified format of the error functional used for supervised learning consists of the long-term time average

$$\mathcal{E}(\mathbf{p}) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} e(\mathbf{x}^T(t), \mathbf{x}(t))\mathrm{d}t \tag{4.2}$$

of the network output error

$$e(\mathbf{x}^T(t), \mathbf{x}(t)) = \sum_{k=1}^{2} |x_k^T(t) - x_k(t)|^\nu \ , \tag{4.3}$$

with a distance metric of norm ν. The infinite time window for integration of the network error (4.2) is not practical for real-time implementation, but a fair practical approximation to the error average is obtained by replacing the integral (4.2) by the output of a low-pass filter operating on the instantaneous network error (4.3). The approximation is particularly valid in case the training sequence of input and target signals is periodic, with a repetition rate significantly higher than the cut-off frequency of the filter. The condition of periodicity is needed to ensure consistency in the outcome of the error measure taken at different instances in time, just as needed with the time-interlaced version as well. We adopt a periodic format for the training signals in the learning experiment.

The stochastic perturbative method, conforming to the specifications (2.2)-(2.4) in Section 2.2, is implemented for error descent learning. Specifically, the learning algorithm iteratively specifies incremental updates in the parameter vector p as

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \mu\, \widehat{\mathcal{E}}^{(k)}\, \boldsymbol{\pi}^{(k)} \tag{4.4}$$

with the differentially perturbed error

$$\widehat{\mathcal{E}}^{(k)} = \frac{1}{2}\left(\mathcal{E}(\mathbf{p}^{(k)} + \boldsymbol{\pi}^{(k)}) - \mathcal{E}(\mathbf{p}^{(k)} - \boldsymbol{\pi}^{(k)})\right) \tag{4.5}$$

obtained from a two-sided parallel activation of fixed-amplitude random perturbations $\pi_i^{(k)}$ onto the parameters $p_i^{(k)}$; $\pi_i^{(k)} = \pm\sigma$ with equal probabilities for both polarities. As shown in Section 2.2, the update rule (4.4)-(4.5) of the stochastic method applied to on-line learning in recurrent dynamical systems provides a net computational efficiency rivaling that of alternative on-line techniques based on exact gradient descent, at a much reduced complexity of implementation.

The teacher forcing signal, injected into the network as an external network input $\mathbf{y}$, is specified according to

$$y_i(t) = \lambda\,\gamma(x_i^T(t) - x_i(t))\ ,\ \ i = 1,2\ , \tag{4.6}$$

conforming to the general format of (1.8). As stated in Section 1.2, the teacher forcing is needed to attract the network outputs towards the targets during learning in case external synchronization between the network and the training signal is not available, such as for the trajectory learning experiment conducted here. The teacher forcing amplitude λ is gradually attenuated along the learning process, as to suppress any bias in the network outputs that might result from residual errors at convergence [Toomarian 92].

4.2 Analog VLSI Implementation

The network and learning circuitry are implemented on a single analog CMOS chip. While most learning functions, including generation of the random perturbation bits, are integrated on-chip along with the implemented network, some global and higher-level learning functions of low dimensionality, such as the evaluation of the error (4.2) and construction of the perturbed error (4.5), are performed outside the chip. The off-chip implementation of the higher-level global functions allows for greater flexibility in tailoring the learning process. Conversely, the lower-level learning functions involving the full

dimensionality of the parameter vector are implemented locally on-chip, where they are performed in parallel for optimal efficiency. The implemented array structure of learning cells, providing locally for the parameter updates, additionally serves to refresh the volatile parameter values for long-term storage after learning is completed. The structure and functionality of the implemented network and learning circuitry are described below.

4.2.1 Implementation Floor Plan

The floor plan of the VLSI network of synapses is organized in the usual two-dimensional array configuration for interconnecting two layers of neurons, with input and output lines running across the array of synapses in orthogonal directions, whereby two particular input and output lines intersect at the location of the synapse cell interconnecting the corresponding input and output neurons [Graf 89]. The 2-D array of synapses interfaces at the outside boundary with a linear array of output neuron cells, collecting synaptic contributions corresponding from the output lines for further processing to construct the neuron outputs. With synapse cells supporting analog current-mode outputs, the synaptic contributions are collected at the output simply by dumping the analog output currents supplied by the synapse cells directly onto the output lines. This current-mode output arrangement combined with the 2-D array configuration of synapse cells offer a simple, modular and scalable VLSI architecture for implementing a fully interconnected neural network, adopted here. Continuous-time recurrent dynamics in the neuron state variables are obtained by feeding a smoothly delayed version of the neuron outputs back into the neuron input layer [Cauwenberghs 90].

To maintain the scalable and modular architecture of the implemented network while integrating local learning functions onto the chip, the added complexity of global on-chip interconnects necessitated by the implemented learning algorithm cannot exceed the N^2 limit imposed by the 2-D arrangement of cells. For feed-forward steady-state neural networks, the typically used incremental outer-product learning rules, such as the delta rule and back-propagation for supervised learning and Hebb-type rules for

unsupervised learning, support a scalable and integrated 2-D architecture intertwined with the implemented network [Cauwenberghs 92]. However, no scalable on-line extensions to the outer-product rules are available for on-chip learning of time-varying features in dynamical recurrent networks.

The stochastic perturbative algorithm, in contrast, supports a simple implementation structure integrated locally with the network, with basically no requirements for global interconnects other than those already supplied by the network itself. The implementation structure, which directly corresponds to the on-line time-interlaced realization of the stochastic method outlined in Section 3.1, comprises an array of identical local learning cells superimposed onto the array structure of network cells, each instance connecting locally to the analog storage node of one particular parameter W_{ij} or θ_j in the network. Except for a few global learning signals, distributed in identical format to all local learning cells, no communication across different cells is needed to perform the learning functions, with the learning cells basically operating autonomously to supply the parameters updates from the locally generated parameter perturbations. Consequently, the implementation structure providing the learning functions extends directly to other parameter-driven networks, say with non-standard or adjustable configuration of the processing cells and their interconnections [Satyanarayana 92], for which the implemented learning functions essentially retain the complexity and adopt the structure of the implemented network.

Figure 4.1 shows the structural organization of the integrated network and learning functions, comprising an array of synapse and threshold parameter cells each including local learning and storage facilities, and peripheral linear array interfaces comprising neuron output cells. Auxiliary provisions to support local generation of the parameter perturbations during learning mode, and to refresh the volatile parameters during storage mode, are also indicated. Global connections for distributing certain scalar signals uniformly to all cells in the array are omitted from Figure 4.1 for clarity. The functional specification and circuit implementation of most of the elements in Figure 4.1 are further elaborated in the text below.

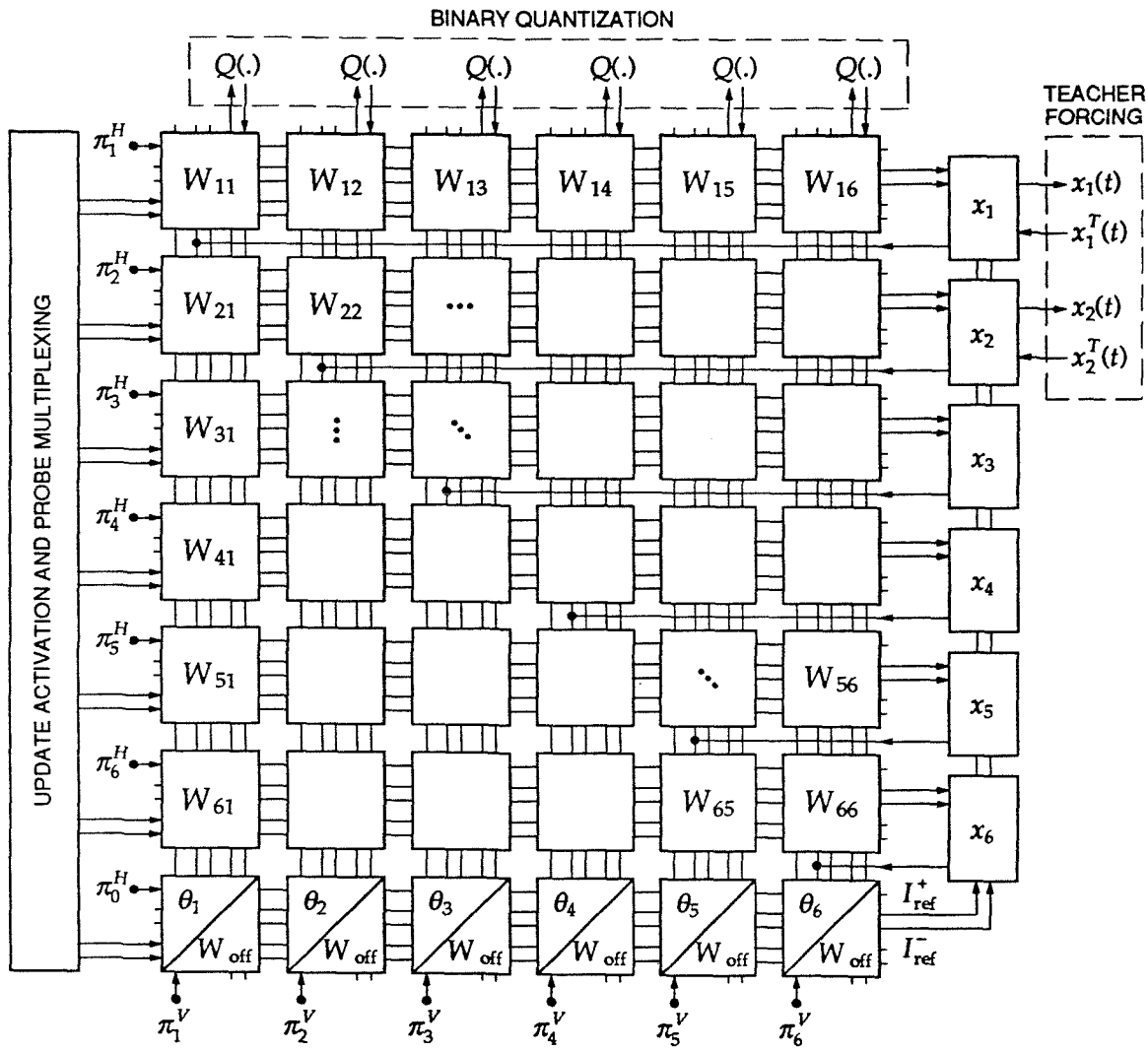


Figure 4.1: Array structure of the network, containing parameter cells with integrated learning and storage functions.

4.2.2 Network Circuitry

A transconductance current-mode approach is adopted for the implementation of the network, allowing continuous-time evolution of the network state variables. Through dedicated transconductance circuitry, which includes regulated cascode triode structures and double-stack differential pairs described below, a wide dynamic range is achieved for the neuron voltages and parameter values at relatively low levels of power dissipation.

A high-output impedance transconductance element with wide voltage range, which is used in both the synapse and neuron cell circuitry, is shown in Figure 4.2 (a), together with its symbolic representation for further reference in subsequent figures. The device comprises a MOS transistor MT biased in the linear triode region, connected to a cascode transistor MC which by means of a high-gain feedback circuit [Fattaruso 93] provides high output impedance while forcing the drain of the triode transistor MT to a constant voltage level. The triode drain voltage V_d^{MT} is primarily set by the control voltage V_c of the regulated cascode circuit, and is largely independent of the triode gate voltage and cascode drain voltage. Therefore, the supplied output current I_{out} is proportional to the input voltage V_{in} while invariant to the output voltage V_{out}, implementing a linear transconductance element operating over a fairly wide voltage range of V_{in}. A bias circuit to generate the control voltage V_c from a desired level for V_d^{MT}, specified by an externally supplied voltage V_d, is shown in Figure 4.2 (b). The bias circuit allows for approximately linear control of the transconductance value of the element by means of the voltage V_d. Obviously, only one instance of the bias circuit needs to be provided for every family of elements with a common transconductance value.

An active floating resistive element with wide voltage range is effectively obtained by combining two instances of the triode transconductor with a current mirror, shown in Figure 4.3 (a). The variable resistive element, injecting a current into the V_{out} terminal proportional to the voltage difference $V_{\mathrm{off}} - V_{\mathrm{out}}$ and the control voltage V_d, is used particularly for conversion between voltage and current formats of the neuron state variables of the network. The measured *I-V* characteristics of the implemented variable resistor are given

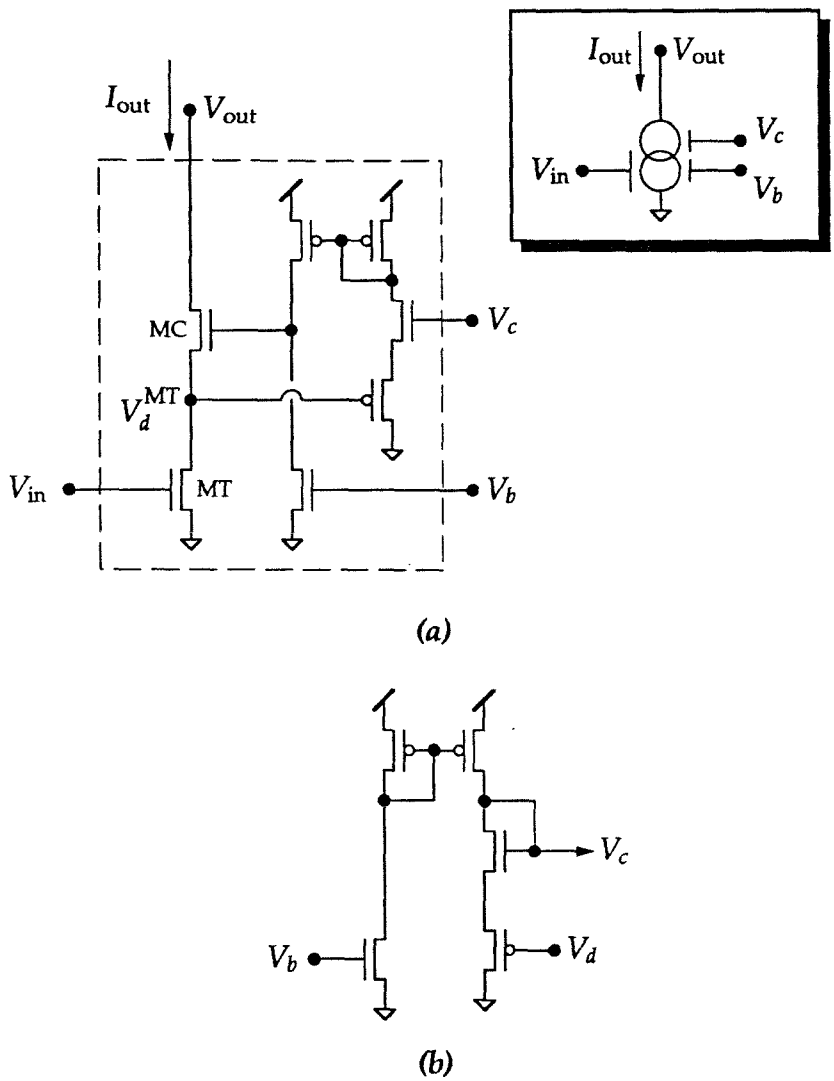
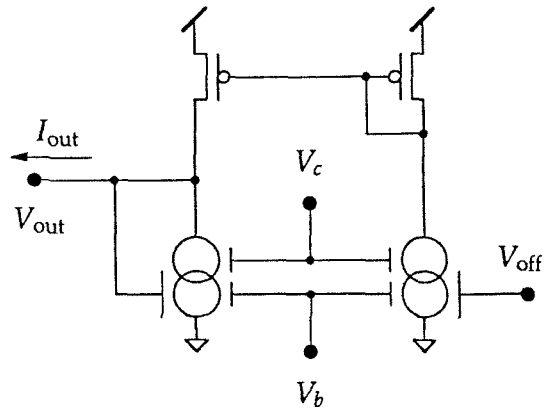


Figure 4.2: Wide range CMOS triode transconductance element with regulated cascode high impedance output. *(a)* Circuit diagram. *(b)* Bias generating circuit.

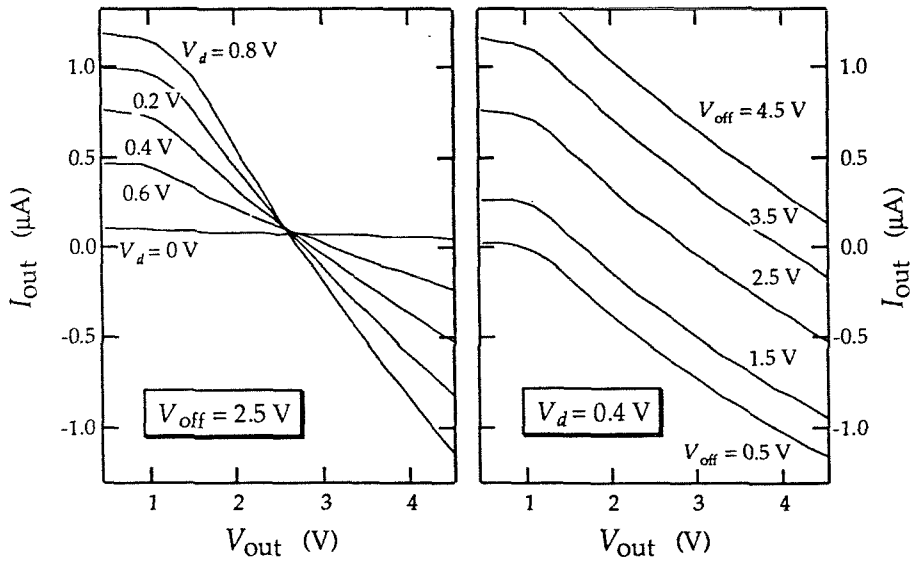
in Figure 4.3 (b), indicating fairly strong nonlinearities in the implemented resistance. Certainly, other CMOS circuit implementations of variable transconductance and resistive elements, with better linear *I-V* characteristics, are available, *e.g.*, [Czarnul 86]. However, linearity is not an absolute requirement in the implementation of an intrinsically nonlinear system. Nonlinearities and other errors in the implemented network furthermore provide an opportunity to experimentally verify the robustness of the learning performance in the presence of model mismatches. A primary motivation for using the transconductance element of Figure 4.2 (a), besides the high-impedance current output and the wide dynamic range of the voltage input, is the ohmic and capacitive decoupling between the voltage input and current output nodes, allowing for fairly insulated capacitive analog storage on the V_{in} terminal as needed to implement volatile adjustable parameters.

Figure 4.4 shows the schematics of the main synapse and neuron cell circuitry employed in the network array and its peripherals. A synapse cell of single polarity is shown in Figure 4.4 (a). A constant current I_{ij}, linear in the voltage W_{ij} over a wide range, is provided by an instance of the triode multiplier of Figure 4.2 (a). The synaptic current I_{ij} feeds into a differential pair, injecting the current $I_{ij}\ \sigma(x_j - \theta_j)$ differentially into the diode-connected I^+_{out} and I^-_{out} output lines. The double-stack transistor configuration of the differential pair offers an expanded linear sigmoid range at modest I_{ij} current levels [Watts 92].

The summed output currents I^+_{out} and I^-_{out} of a row of synapses are collected in the output cell of the corresponding neuron x_i, Figure 4.4 (b). The diode connection of the load transistors on the output lines of Figure 4.4 (a) provides normalization of the collected output currents to an appropriate level, regardless of the number of participating synaptic cells feeding into the neuron output, thereby supporting scalable expansion of the network architecture [Satyanarayana 89]. The neuron output cells also receive two common reference currents I^+_{ref} and I^-_{ref} obtained from one separate row of reference synapses, identical in structure to the other synaptic cells. The reference synapses, represented in the bottom row of the array of Figure 4.1, are supplied uniformly with a synaptic strength W_{off}, defining a



(a)



(b)

Figure 4.3: Wide range active CMOS resistive element. *(a)* Circuit diagram. *(b)* Measured $I - V$ characteristics.

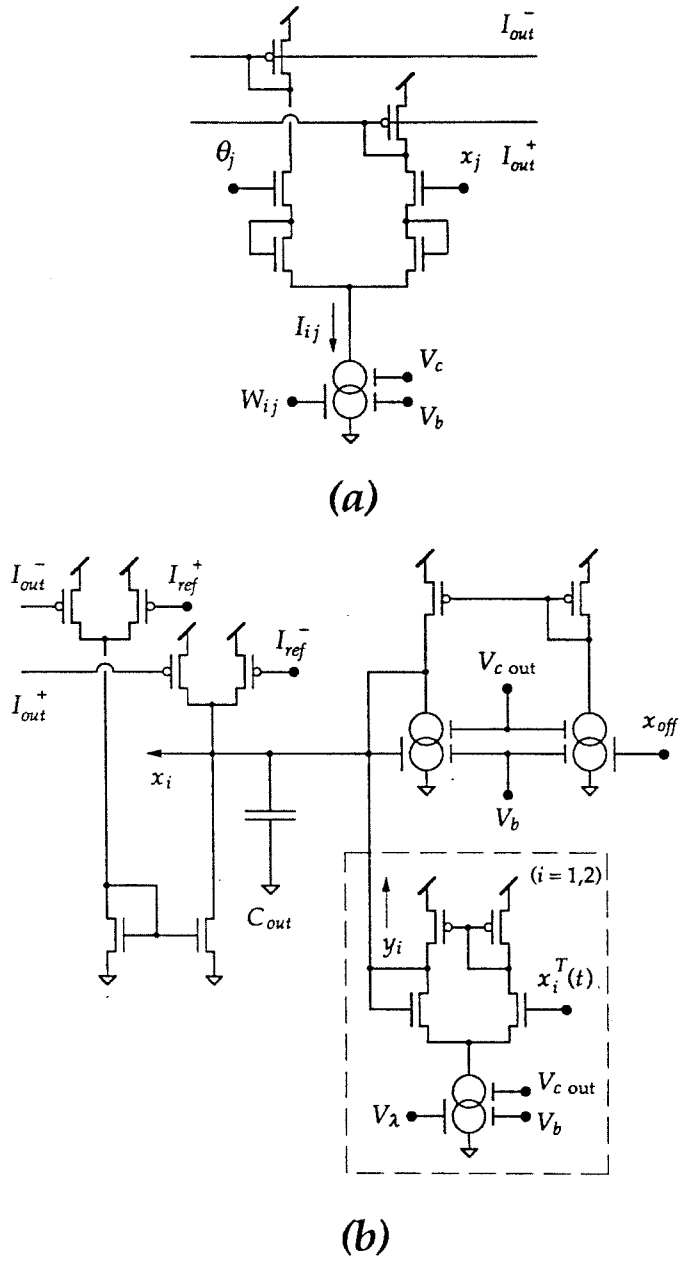


Figure 4.4: Schematics of synapse and neuron network circuitry. *(a)* Synapse cell of single polarity. *(b)* Neuron output cell with current-to-voltage converter.

common offset for the synaptic values W_{ij} serving as an effective zero level reference for four-quadrant operation of the synapses [Cauwenberghs 92]. By combining the currents I^+_{out}, I^-_{out}, I^+_{ref} and I^-_{ref} through mirror and summing operations, the neuron cell constructs the output current

$$\begin{aligned} I_{\text{out}} &= (I^+_{\text{out}} - I^-_{\text{out}}) - (I^+_{\text{ref}} - I^-_{\text{ref}}) \\ &= \sum_{j=1}^{6} g_c\, W_{ij}\, \sigma(x_j - \theta_j) - \sum_{j=1}^{6} g_c\, W_{\text{off}}\, \sigma(x_j - \theta_j) \\ &= g_c \sum_{j=1}^{6} (W_{ij} - W_{\text{off}})\, \sigma(x_j - \theta_j) \ . \end{aligned} \tag{4.7}$$

Apart from a factor g_c, which accounts for the triode element transconductance and current scaling factors on the output line, the current (4.7) corresponds to the summed synaptic contributions in (4.1). Additionally, the contribution of the external input y_i to neuron x_i is included in (4.7) by injecting a current proportional to y_i into the output node. With the particular network configuration of the present implementation, for demonstration of trajectory learning, the external inputs only contain the teacher forcing signals (4.6) applied to the first two neurons, x_1 and x_2. A differential transconductance element with inputs x_i and x_i^T, for forced teacher action in accordance with (4.6), is shown connected to the neuron output in the dashed inset of Figure 4.4 (b), applicable to the forced neurons x_1 and x_2 only. The transconductance element implements the teacher forcing characteristics of (4.6), with the function $\gamma(.)$ performed by the sigmoidal transfer function of the input differential pair, and the amplitude λ determined by the tail current of the device, equal to the product of the triode transconductance $g_{c\,\text{out}}$ and the control voltage V_λ.

The combined output current I_{out} is converted to the neuron output voltage x_i by means of an active resistive element, as described above with reference to Figure 4.3 (a). Besides serving to convert between current and voltage formats, the resistive element also implements the dynamics for the neuron state variables, with the time constant τ in (4.1) corresponding to the RC product of the resistance value $1/g_{c\,\text{out}}$ and a parallel capacitance C_{out}. With $C_{\text{out}} = 5pF$, the delay ranges between 20 and 200μsec, adjustable

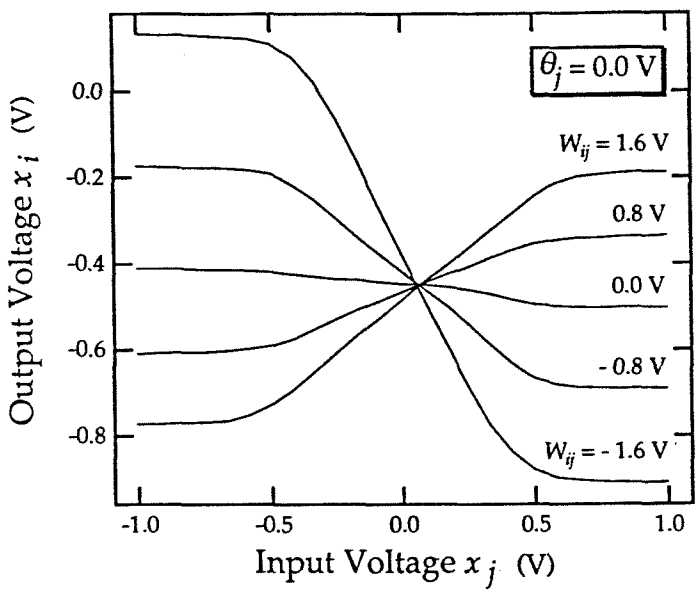
by the control voltage $V_{c\,\mathrm{out}}$ of the regulated cascode defining the conductance $g_{c\,\mathrm{out}}$ of the resistive element.

Figure 4.5 shows the measured static characteristics of the synapse and neuron functions for different values of W_{ij} and θ_j $(i = j = 1)$, obtained by disabling the neuron feedback and driving the neuron inputs of the synapse array externally. Effects of random and systematic errors in the implementation on the shape of the curves are clearly visible. In particular, the characteristics for different threshold values θ_j in Figure 4.5 (b) show a significant distortion due to saturation effects at the boundaries of the synapse dynamic range. Nevertheless, the discrepancy between expected and realized network characteristics is largely irrelevant, since the stochastic perturbative learning process does not assume a particular form of the network structure, or perfect model knowledge of the implemented structure. Significantly more important than the shape of the implemented network characteristics is the analog resolution it supports for the network parameters and state variables. As demonstrated in Figure 4.5, the implemented network provides a dynamic range of 2 V for the neuron voltages, and 3 V of fairly linear voltage range for programming of the synapse and threshold parameters.

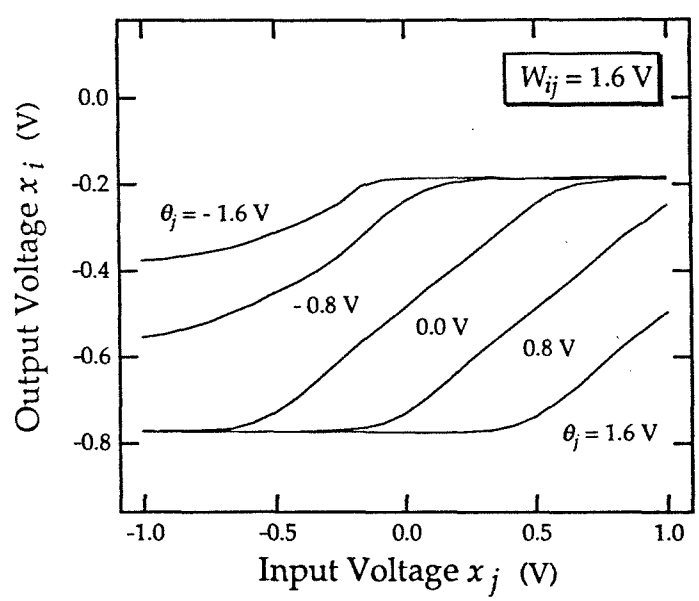
4.2.3 Learning Circuitry

Figure 4.6 (a) shows the simplified schematics of the learning cell circuitry, replicated locally for every parameter W_{ij} and θ_j in the network array of Figure 4.1. The circuitry of Figure 4.6 (a) directly implements the high-level schematic of the learning cell depicted in Figure 3.2, and therefore complies with the system-level description regarding the on-line time-interlaced architecture in Section 3.1. The description of the learning system therefore proceeds with direct reference to the general architectural issues addressed in Section 3.1.

As indicated before, most of the variables relating to the operation of the learning cell are local, with exception of a few global signals communicating to all cells. Global signals include the sign and the amplitude of the perturbed error $\widehat{\mathcal{E}}$ and predefined control signals. The stored parameter p_i and its binary perturbation π_i are strictly local to the cell, in that

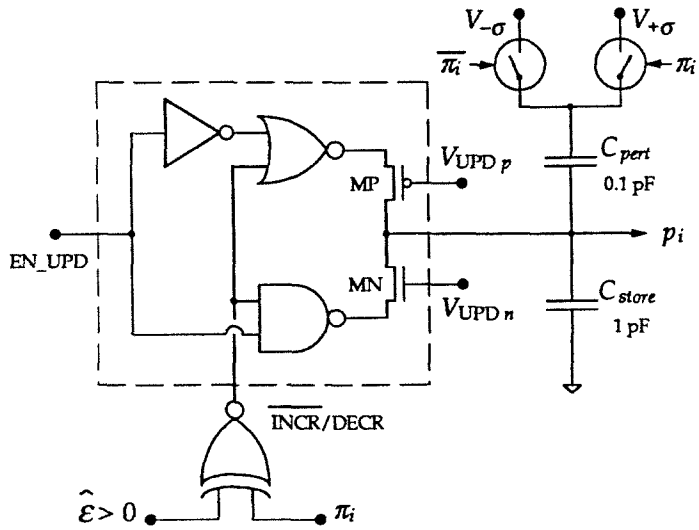


(a)

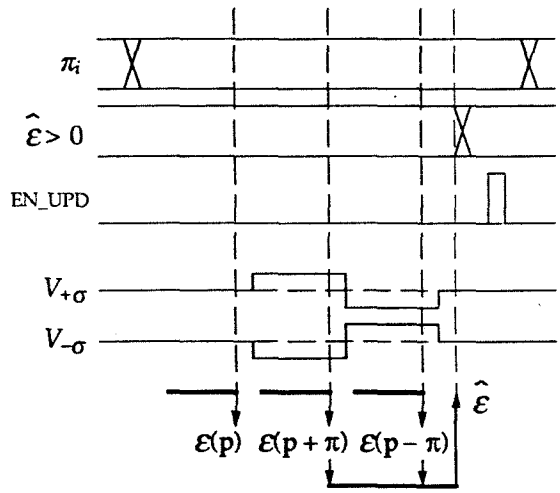


(b)

Figure 4.5: Measured static synapse and neuron characteristics, for various values of *(a)* the connection strength W_{ij}, and *(b)* the threshold θ_j.



(a)



(b)

Figure 4.6: Learning cell circuitry. *(a)* Simplified schematics. *(b)* Waveform and timing diagrams.

they do not need to communicate explicitly to outside circuitry, except trivially through the neural network it drives. This simplifies the structural organization and interconnection of the learning cells, which are integrated with the synapse and threshold cells in the network array of Figure 4.1.

The parameter voltage p_i is stored on the capacitor C_{store}, and interfaces directly with the corresponding network circuitry providing the synapse or threshold function. The storage capacitor C_{store} furthermore couples to a second capacitor C_{pert} for activation of the perturbation. Since the perturbation can only take one of two values $\pm\sigma$, it is locally represented and processed in binary format. The binary perturbation is generated locally, once at the beginning of every update cycle, by means of a procedure described further below. The perturbation bit π_i selects either of two complementary signals $V_{+\sigma}$ and $V_{-\sigma}$, supplied globally to all learning cells, for driving the coupling capacitor C_{pert}. $V_{+\sigma}$ is selected for $\pi_i = 1$, and $V_{-\sigma}$ is selected otherwise. With the specific shape of the waveforms $V_{+\sigma}$ and $V_{-\sigma}$ depicted in Figure 4.6 (b), the proper sequence of perturbations activated onto the parameter is established, for successive observations of the complementary error terms in (4.5). In particular, both $V_{+\sigma}$ and $V_{-\sigma}$ represent the perturbation sequence $0, \pi_i, -\pi_i$, with $\pi_i = +\sigma$ for $V_{+\sigma}$ and $\pi_i = -\sigma$ for $V_{-\sigma}$. Successive observations of the network error, synchronized with the three phases of the perturbation sequence, yield estimates of the unperturbed error $\mathcal{E}(\mathbf{p})$ and the complementary perturbed errors $\mathcal{E}(\mathbf{p} + \boldsymbol{\pi})$ and $\mathcal{E}(\mathbf{p} - \boldsymbol{\pi})$, respectively. An estimate of the differential perturbed error $\widehat{\mathcal{E}}$ is then constructed externally by combining the obtained values for $\mathcal{E}(\mathbf{p} + \boldsymbol{\pi})$ and $\mathcal{E}(\mathbf{p} - \boldsymbol{\pi})$, in accordance with (4.5). By virtue of the rigorous scheme used to activate the perturbations onto the parameters, involving mainly binary selection operations at the local implementation level, the perturbed error observations and the corresponding estimate of $\widehat{\mathcal{E}}$ can be expected of reasonably high quality, provided the outcomes of the error observations are systematic and consistent in the respective parameter settings.

The obtained global value for $\widehat{\mathcal{E}}$ is then used, in conjunction with the local perturbation bit π_i, to locally update the parameter value p_i according to (4.4). As suggested in

Section 3.1, the update increment in (4.4) is decomposed into its amplitude and polarity components, each further being processed and activated independently. The motivation for separating both follows from accuracy considerations applying to incremental update learning rules in general. While the correct implementation of the polarity of the intended learning increments is crucial to warrant proper convergence of the parameter values, the amplitudes of the implemented increments or decrements are allowed relative errors within certain margins. Incidentally, formula (4.4) reveals that the increment polarity can be obtained from a binary exclusive-or operation on the polarities of the perturbed error $\widehat{\mathcal{E}}$ and the perturbation bit π_i, with the parameter value being decremented if both polarities are equal, and incremented otherwise. Likewise, the increment amplitude is given by the amplitude of $\widehat{\mathcal{E}}$. The binary operation to determine the increment polarity is implemented virtually error-free with simple digital CMOS circuitry. The obtained value for the polarity then activates either of a given analog increment or decrement, with amplitude proportional to $|\widehat{\mathcal{E}}|$, onto the parameter value. The analog device serving the increments and decrements does not need to be excessively precise, other than required to satisfy the requested increment polarity. Requirements on relative accuracy of the increment size, scaling with the amplitude, are therefore more stringent than requirements on absolute, uniform accuracy.

The binary controlled charge pump described in Section 3.2, offering a fine resolution of the charge increment amplitude ranging over several decades, is used for this purpose. The charge pump circuitry of Figure 3.6 is shown essentially replicated in the dashed-line inset of Figure 4.6 (a), interfacing with the storage capacitor C_{store} to update the analog parameter p_i according to the supplied analog and binary signals defining the increment amplitude and polarity. The polarity of the increment or decrement action is determined by the control signal DECR/$\overline{\text{INCR}}$, obtained from an exclusive-or gate operating on the polarities of the perturbed error $\widehat{\mathcal{E}}$ and the perturbation bit π_i. The amplitude of the incremental update, set proportionally to $|\widehat{\mathcal{E}}|$, is controlled by the globally supplied $V_{\text{UPD}\,n}$ and $V_{\text{UPD}\,p}$ bias levels onto the gates of MN and MP, respectively. Both transistors MN

and MP are biased in the subthreshold region [Vittoz 77], [Mead 89], allowing exponential control of the current amplitude ranging down to pA levels. Consequently, the charge pump device is able to supply reliable increments and decrements of almost arbitrarily small size, as needed in the final stages of the learning near convergence. The learning cycle is completed by activating the update with a high pulse on the enable line EN_UPD. The next learning cycle then starts with a new random bit value for the perturbation π_i.

4.2.4 Local Generation of Random Perturbations

The random bit streams $\pi_i^{(k)}$ are generated on-chip using a variant on the well-known technique to obtain pseudo-random bit sequences by means of linear feedback shift registers [Golomb 67]. For optimal performance, the perturbations need to satisfy certain statistical orthogonality conditions, and a rigorous but elaborate method to generate a set of uncorrelated bit streams in VLSI has been derived [Alspector 91]. To preserve the scalability of the learning architecture and the local nature of the perturbations, we have chosen a simplified scheme which drastically reduces the implementation complexity but which does not aversely affect the learning performance to first order, as verified experimentally. The array of perturbation bits, configured in a two-dimensional arrangement as prompted by the location of the learning cells near the parameters in the network array, is constructed by an outer-product exclusive-or operation from two generating linear sets of uncorrelated row and column bits, π_j^{H} $(j = 0 \ldots 6)$ and π_i^{V} $(i = 1 \ldots 6)$, shown in Figure 4.1. In particular, the values of the perturbation bits are obtained locally, inside the corresponding learning cells, by means of exclusive-or gates whose inputs connect to the locally intersecting horizontal and vertical bit lines. While obviously deterministic relationships exist among the thus constructed bit values, statistically the obtained bits are mutually uncorrelated with equal probabilities for both binary outcomes, provided the generating row and column bits π_j^{H} and π_j^{V} are statistically uncorrelated and unbiased as well.

The row and column bits themselves are obtained from exclusive-or combinations of

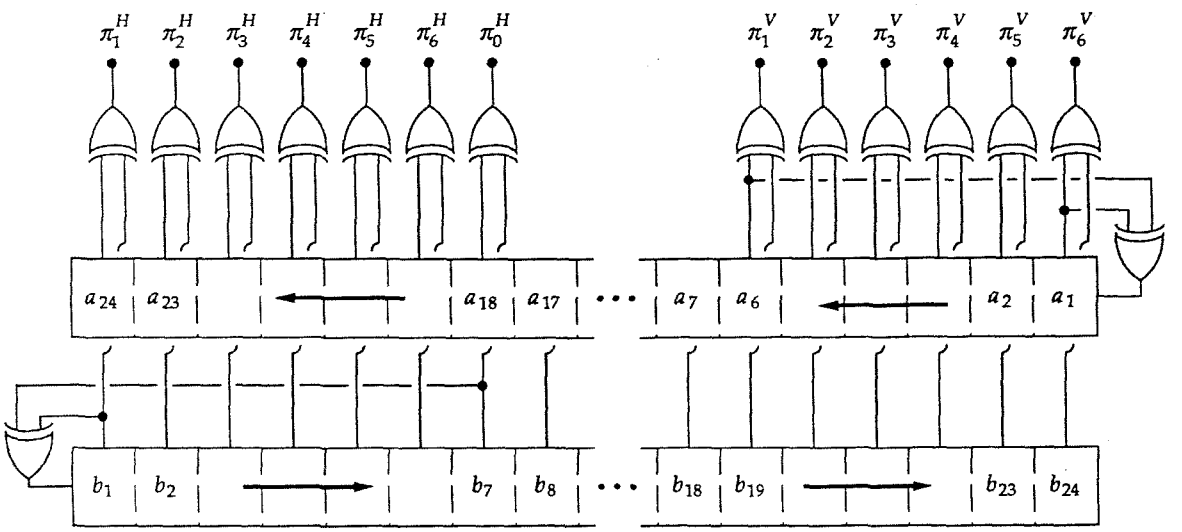


Figure 4.7: Multi-channel pseudo-random bit generation using linear feedback shift registers.

the parallel outputs of two counter-propagating linear feedback shift registers, with polynomial degrees six and seven respectively, generating two independent pseudo-random bit sequences in time and space. The configuration of the two registers and the linear array of exclusive or-gates, combining the register outputs into the row and column bits, is shown in Figure 4.7. In principle, the row and column bits could have been obtained *from the parallel outputs of one single feedback register only, without the XOR operations*, though such would imply obvious correlations between the bit values over time, as any two given taps of the parallel output of a shift register represent the same bit sequence merely displaced in time. Since the parallel output bit sequences of the two registers generating the row and column bits are mutually counter-propagating and independent, propagation effects within the set of generated bits are avoided. With polynomial degrees of six and seven for the two shift register bit sequences, the combination of the register outputs delivers a periodicity of $(2^6 - 1) \times (2^7 - 1) = 8,001$ [Golomb 67], which is appropriate for the perturbative learning scheme. While longer random sequences could be obtained by choosing higher degrees for the generating polynomials of the registers, the particular choice was made to allow for a simple implementation of the linear feedback circuitry.

4.2.5 Long-Term Volatile Storage

After learning, it is desirable to retain the parameter settings established onto the network. The implemented chip includes local provisions for refreshing the parameters, using the dynamic analog storage technique described in Section 3.2 of the previous chapter, with the function form of the updates specified by Eqns. (3.2) and (3.3). The parameter refresh is performed in the background, and does not interfere with the continuous-time network operation.

For practical reasons, the refresh circuitry implementing the above analog memory technique is integrated with the learning circuitry driving the parameter storage capacitor. As suggested before in Chapter 3, the charge pump used for the learning updates can serve directly to supply the incremental updates in the parameter values for partial refresh as well. To that purpose, probing and multiplexing circuitry are added to the learning cell of Figure 4.6 (a), activated in the refresh mode of operation while the learning circuitry is disabled, to access the stored parameter value for binary quantization and drive the charge pump correspondingly. The simplified schematics of the refresh circuitry, discarding the learning portion of the cell sharing the same charge pump, are shown in Figure 4.8. To reduce the complexity of implementation, the binary quantization of the stored analog values is performed sequentially outside the array of cells, in a multiplexed arrangement along each of the columns in the array, as illustrated generally in Figure 3.5 (b) and more specifically in Figure 4.1. Since the refresh rates required for the volatile storage medium are rather modest, typically in the 100 Hz range, a small set of binary quantizers, one per column in the array, provides sufficient bandwidth to accommodate multiplexing of a fairly large number of storage cells. A binary SEL signal selects the particular row of cells in the array to be refreshed at a given time, connecting the buffered stored value of a selected cell to the binary quantizer on the corresponding column. A logic high pulse on the EN_UPD line of the selected row finally establishes the partial increments (3.2), activating the charge pumps of the selected storage cells with polarity DECR/$\overline{\text{INCR}}$ given by the binary quantization values of the corresponding columns.

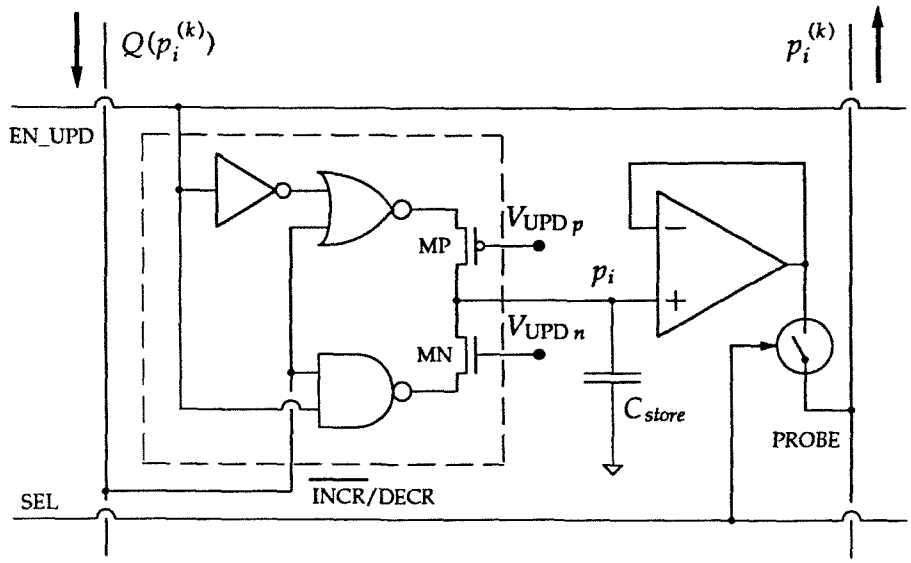


Figure 4.8: Simplified schematics of the storage cell circuitry.

The complete circuit-level schematics of the synaptic cell W_{ij} used in the array of Figure 4.1, including the local functions of synaptic contribution, generation of the perturbations, stochastic error-descent learning, and long-term storage of the volatile parameters, is given in Figure 4.9. The layout of the synapse cell, measuring 170μm $\times$ 180μm in 2μm CMOS technology, is depicted in Figure 4.10. The reference synaptic cells θ_j/W_{off} shown in the bottom row of Figure 4.1, which supply the value of the locally stored threshold parameter θ_j to all synapses on the same column while providing the W_{off} synaptic contributions on the reference row, follow a circuit and layout structure almost identical to that of Figures 4.9 and 4.10. Table 4.1 summarizes the chip features, and a micrograph of the implemented chip is shown in Figure 4.11.

4.2.6 Global Supervision of Learning and Storage Functions

Besides the local learning and storage functions, of which the implementation was described in detail above, the coordination and supervision of those local functions requires some global functions, which operate at a higher level surpassing the individual structure of parameter cells in the array. The global functions involve a dimensionality much

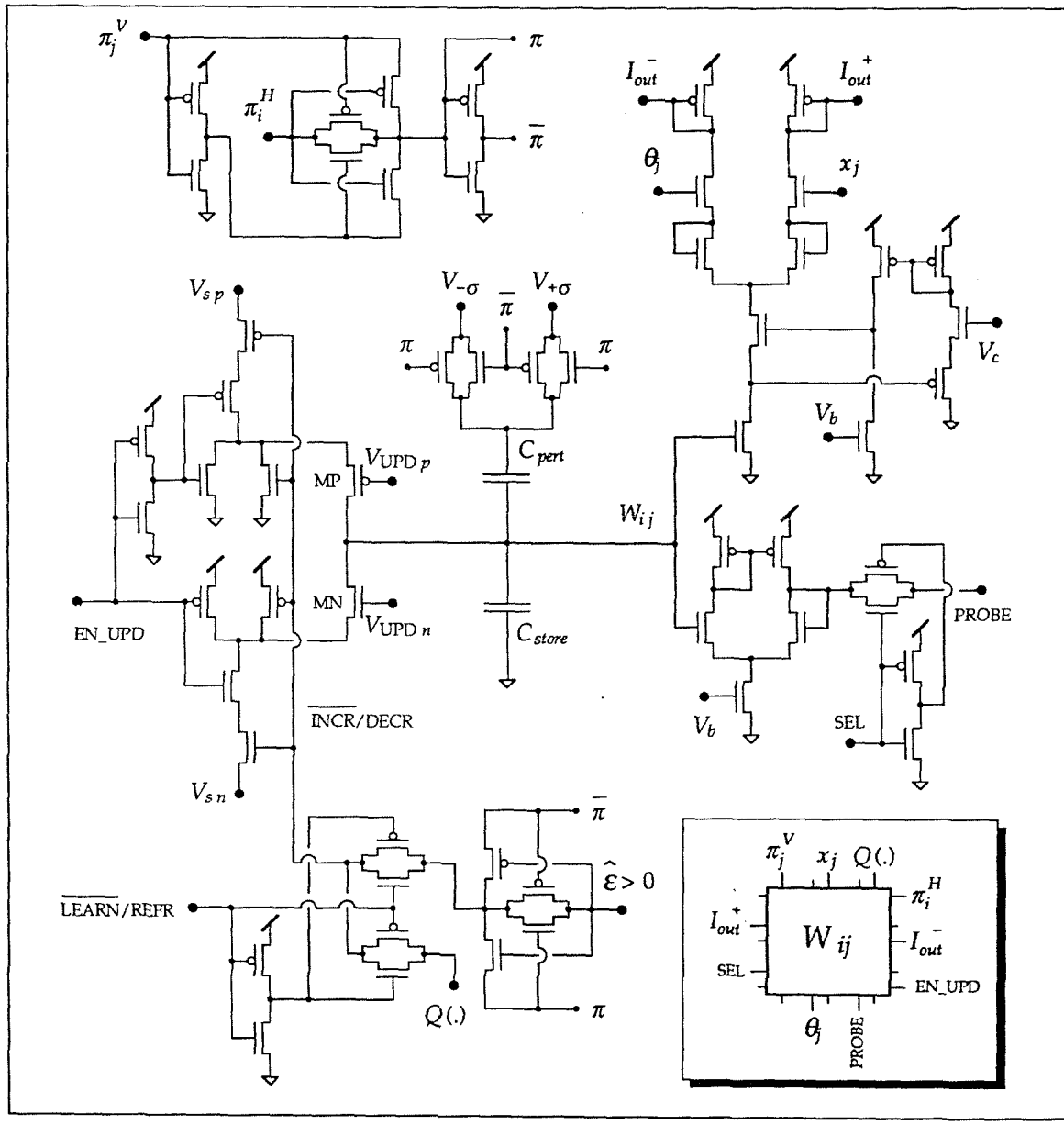


Figure 4.9: Complete schematics of the synapse cell including learning and storage functions.

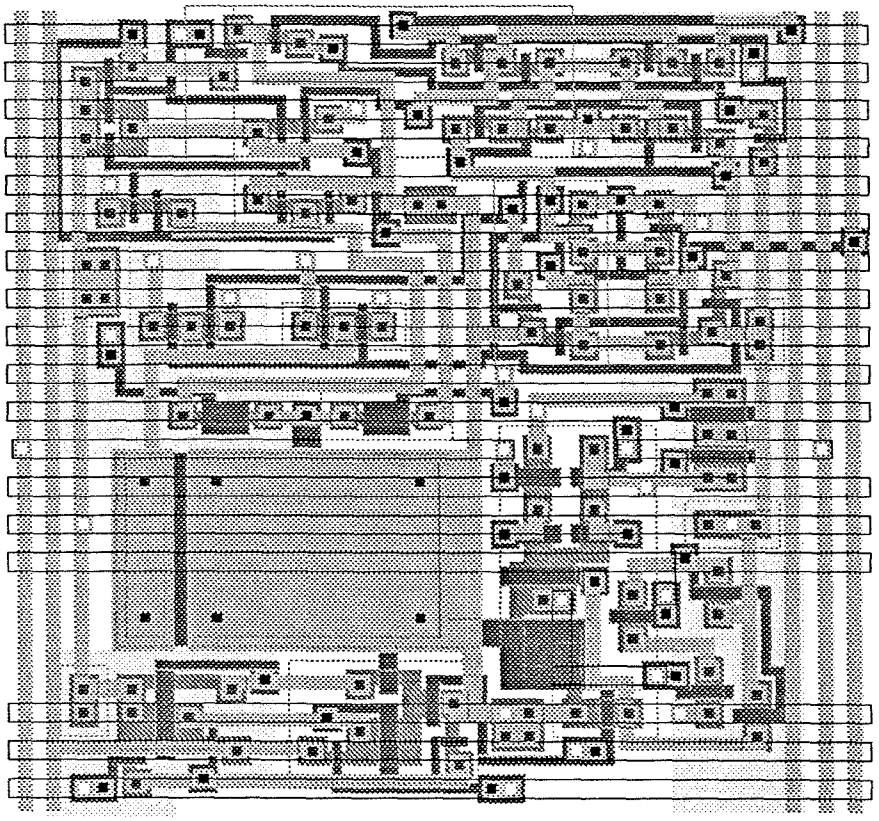


Figure 4.10: Physical layout of synapse cell with integrated learning and storage functions.

Table 4.1: Chip features of the implemented learning network.

Technology	2 μm p-well double-poly CMOS
Supply voltage	+ 5 V
Power dissipation	
total	1.2 mW
Area	
active die	2.2 mm X 2.2 mm
synapse cell	170 μm X 182 μm
Transistor count	
total	3885
synapse cell	56

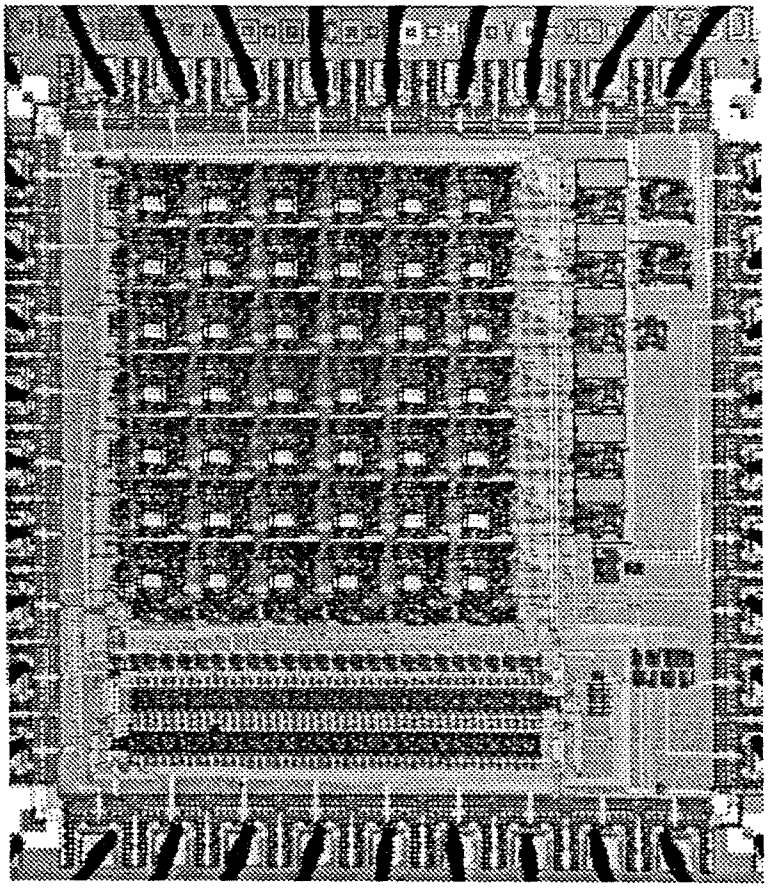


Figure 4.11: Chip micrograph. Center: network array of synapse and neuron cells. Bottom: multi-channel random bit generator.

smaller than that of the local functions, such that a dedicated hardware implementation tuned towards optimal efficiency is not needed. As suggested in Section 3.1, the low-dimensional global functions can instead be implemented on a general-purpose platform such as a micro-controller, offering a wide spectrum of functionality which can be tuned towards optimal performance at virtually no cost of hardware efficiency. For the stochastic perturbative learning algorithm in particular, the global functions operate on scalar variables relating to error observations on the network, evaluating the network error under complementary perturbations, and combining the two results to construct an estimate of the differential perturbed error (4.5). Possible functional extensions for increased performance, which can easily be incorporated in a general-purpose implementation, include adaptive biasing techniques for dynamically optimizing the learning rate, and conditional tests on the range of unperturbed and perturbed error observations (or various combinations of all three observed values $\mathcal{E}(\mathbf{p})$, $\mathcal{E}(\mathbf{p}+\boldsymbol{\pi})$ and $\mathcal{E}(\mathbf{p}-\boldsymbol{\pi})$) to guard and remedy against potential instabilities in the updates.

In the present implementation, all global learning functions are performed off-chip, by means of analog and digital hardware interfacing with a general-purpose computer. The evaluation of the error functional (4.2) on the network is performed with discrete analog components, leaving some flexibility to experiment with different formulations of error functionals that otherwise would have been hardwired. A mean absolute difference ($\nu = 1$) norm is used for the metric distance, and the time-averaging of the error (4.3) is achieved by a fourth-order Butterworth low-pass filter. The cut-off frequency is adjusted to accommodate an AC ripple smaller than 0.1%, giving rise to a filter settling time extending 20 periods of the training signal. Since three observations of the error (4.2) on the network are needed to obtain the unperturbed and complementary perturbed error estimates, a single learning iteration spans at least 60 periods of the training signal.

In principle, the refresh of the parameters does not require higher-level global supervision at all, since the timing and multiplexing of local update and binary quantization functions can be coordinated by means of predefined cyclic control sequences, which can

be generated on-chip driven by a periodic clock signal. Such allows the refresh operations to run autonomously in the background, transparent to the operation of the network. In the present realization, the binary quantization functions are not included on the chip circuitry and need to be performed externally. While the on-chip integration of appropriate quantizer architectures is technically possible, the main focus in the experiment conducted here is to characterize the learning performance of the network chip, to which the supporting storage method is secondary. A second related chip, containing an isolated array structure of storage cells incorporating on-chip which does incorporate binary A/D/A quantizers, has been developed and tested separately. The results, demonstrating long-term dynamic analog storage with autonomous operation, are reported in Section 4.4.

For simplicity of the conducted learning experiment, the parameters p_i are stored externally once the learning is completed, and thereafter refreshed sequentially by supplying values for the quantization bits $Q(p_i{}^{(k)})$ as defined by the polarity of the observed deviation between internally probed $p_i{}^{(k)}$ and externally stored p_i parameters values. External storage and generation of the binary quantization values are performed and controlled by the computer supervising the experiment. The simplified refresh scheme with external storage does not involve internal quantization feedback loops, and therefore allows error-free operation at slower refresh rates. The parameter refresh is performed in the background with a 100msec cycle, whenever the learning process is disabled or interrupted.

4.3 Experimental Learning Results

As a proof of principle, the network is trained to generate outputs following a circular target trajectory, in absence of externally supplied inputs. The target output signals are defined by the quadrature-phase oscillator

$$\begin{cases} x_1^T(t) &= A\cos(2\pi f t) \\ x_2^T(t) &= A\sin(2\pi f t) \end{cases} \tag{4.8}$$

with amplitude $A = 0.8$ V and frequency $f = 1$ kHz. In principle a recurrent network of two neurons suffices to generate quadrature-phase oscillations, and the extra neurons in the network serve to accommodate the particular amplitude and frequency requirements and assist in reducing the nonlinear harmonic distortion.

Training a recurrent neural network to exhibit prescribed oscillatory behavior by means of error descent in the parameter space is a harder problem than might be anticipated at first thought. As indicated in the timing analysis of Section 2.3, the shape of the error surface in parameter space, defined by the time-averaged format of (4.2) or its equivalent as implemented by low-pass filtering of (4.3), contains local minima and abrupt discontinuities, which lead to unpredictable learning results when the network is initialized arbitrarily. Incidentally, we found that randomly initialized learning sessions usually fail to generate oscillatory behavior at convergence, the network being trapped in a local minimum defined by a strong point attractor. Even with strong teacher forcing, these local minima persist. As shown in Section 2.3, the sharp parameter dependence of the error functional, fueling the convergence problems, can be mostly avoided by using shorter integration time intervals for averaging of the error (4.2), matching the characteristic time scale of the network dynamics and training signal. Such requires a modified scheme of coordinating the perturbations and error observations to allow practical on-line implementation, which could not be incorporated in the present experimental set-up.

The convergence problems due to the ill-shaped structure of the error functional can be circumvented through a rational choice of the initial conditions. Clearly a particular choice of initial conditions for the parameter values may artificially convert a hard learning problem to a trivial one, and to warrant meaningful results we have derived initial conditions based on physical considerations generally valid for other trajectory learning tasks on recurrent neural networks as well. We obtained consistent and satisfactory results with the following initialization of network parameters: strong positive diagonal connection strengths $W_{ii} = 1$, zero off-diagonal terms $W_{ij} = 0$; $i \neq j$ and zero thresholds $\theta_i = 0$. The positive diagonal connections W_{ii} are needed to free the neuron state variables from the

point attractor at the origin, corresponding to the spontaneous decay term $-x_i$ in (4.1). This allows the network outputs to better follow the target signals under strong initial teacher forcing, for fast and robust learning. The zero initial values for the cross connections W_{ij} ; $i \neq j$ are required to avoid any bias in the dynamics, due to coupling between neurons, during the initial phase of the learning. Gradual relaxation of the teacher forcing strength afterwards allows to establish the desired coupling strengths needed to maintain the neuron output waveforms closely near those of the target signals.

Figure 4.12 shows recorded error sequences under training of the network with the target oscillator (4.8), for four different sessions of 1,500 learning iterations each starting from the above initial conditions. The learning iterations span 60 msec each, for a total of 100 sec per session. The teacher forcing amplitude λ is set initially to $V_\lambda = 3$ V, and thereafter decays logarithmically over one order of magnitude towards the end of the sessions. Fixed values of the learning rate and the perturbation amplitude are used throughout the sessions, with $\mu = 25.6\,\mathrm{V}^{-1}$ and $\sigma = 12.5\,\mathrm{mV}$. All four sessions show a rapid initial decrease in the error under stimulus of the strong teacher forcing, and thereafter undergo a region of persistent flat error slowly tapering off towards convergence as the teacher forcing is gradually released. Notice that this flat region does not imply slow learning; instead the learning constantly removes error as additional error is adiabatically injected by the relaxation of the teacher forcing.

Near convergence, the bias in the network error due to the residual teacher forcing becomes small. Figure 4.13 shows the network outputs and target signals at convergence, with the learning suspended and the parameter refresh activated, illustrating the minor effect of the residual teacher forcing signal on the network dynamics. The oscillogram of Figure 4.13 (a) is obtained under a weak teacher forcing signal, and that of Figure 4.13 (b) is obtained with the same network parameters but with teacher forcing disabled. In both cases the oscilloscope is triggered on the network output signals. Obviously, in absence of teacher forcing, the network does no longer run synchronously with the target signal. However, the discrepancy in frequency, amplitude and shape between either of the

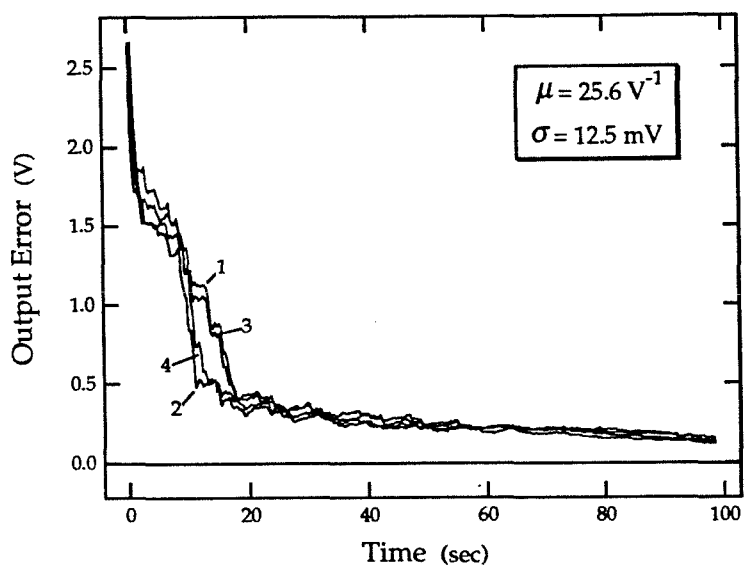
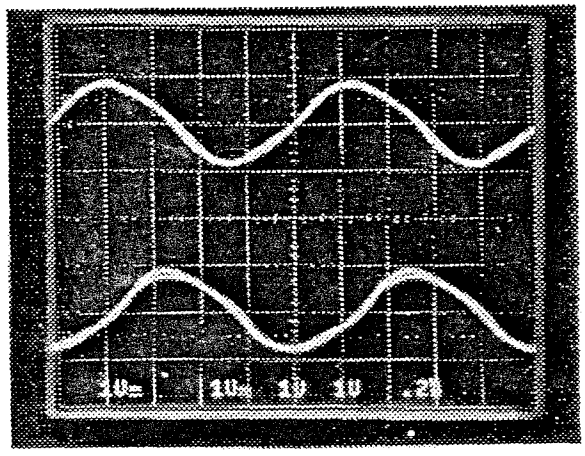


Figure 4.12: Recorded evolution of the error during learning, for four different sessions on the network chip.

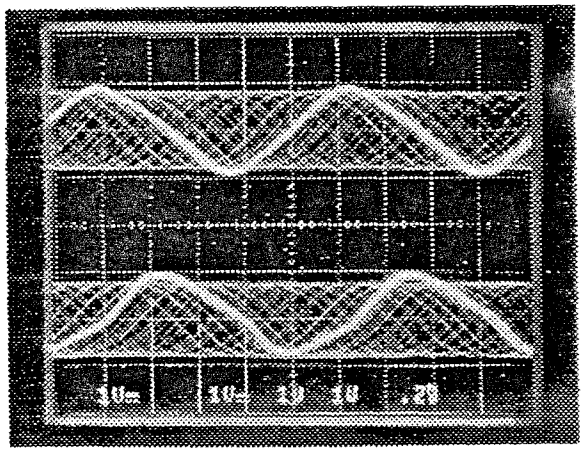
free-running and forced oscillatory output waveforms and the target signal waveforms is evidently small.

It would have been instructive to observe the internal dynamics of the network other than that of the two output neurons x_1 and x_2. Unfortunately, the output voltages of internal neurons x_3 through x_6 are not accessible off-chip in the present chip implementation, due to pin-out limitations. Instead, we recorded the evolution of the complete set of network parameters, which are accessible through the probing and multiplexing circuitry supporting the binary quantization, during each of the above four learning sessions. The recorded curves for the synapse and threshold parameters, sampled every 50 learning update cycles, are shown separately for all four sessions in Figure 4.14. To ease interpretation, the parameter curves relating to either of the four internal neurons are visually distinguished from those pertaining exclusively to the two output neurons. Both families of curves clearly display a significant change in the parameter values from the initial settings occurring along the learning sessions, which testifies that all neurons, including the internal neurons, are actively engaged in training the dynamics of the output neurons. The strong similarity between sets of parameter curves across different learning sessions, shown in the four separate plots of Figure 4.14, further indicates that the involvement of the internal neurons during learning is a systematic rather than a purely diffusive process, whereby the complete spectrum of dynamics achievable by the fully configured network is explored to produce the desired output neuron waveforms.

Closer examination of the parameter values at convergence, given in Table 4.2, reveals that all four sessions actually result into approximately the same network, with the neurons configured in a particular order. In principle, the outputs of the network are invariant to topological interchanging of internal neurons, with corresponding permutations of rows and columns in the parameter matrix. Therefore, the learning task contains multiple degenerate solutions, all of which should equally likely be obtained under learning from the unbiased initial values for the parameters. In practice, analog offsets and mismatches in the implementation of the network break the symmetry existing among the



(a)



(b)

Figure 4.13: Oscillograms of the target signals and network outputs after training, *(a)* under weak teacher forcing, and *(b)* with teacher forcing disabled. Top traces: $x_1(t)$ and $x_1^T(t)$. Top traces: $x_2(t)$ and $x_2^T(t)$.

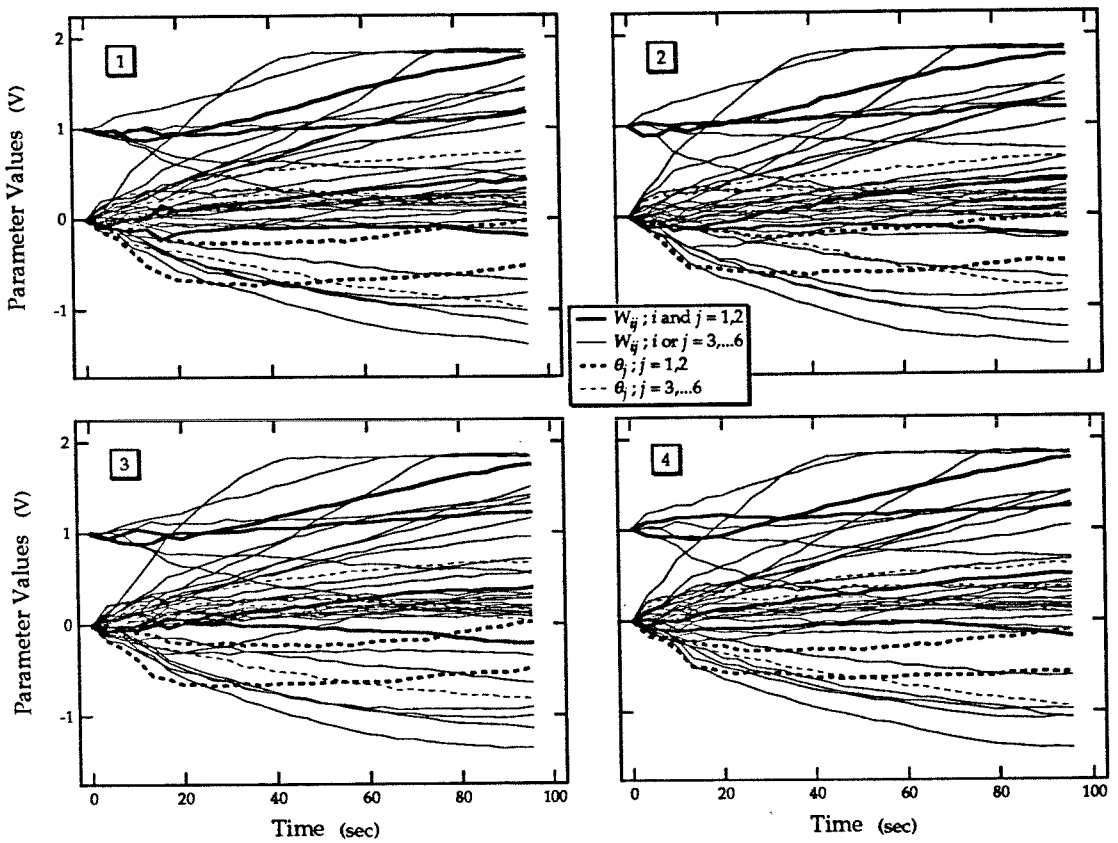


Figure 4.14: Recorded evolution of the network parameters during learning, for four different sessions on the network chip. (Bold curves: external-related parameters; thin curves: internal parameters. Solid curves: weight parameters; dashed curves: threshold parameters.)

neurons, and introduce a bias in the network dynamics at the initial parameter settings, favoring a particular solution. The fact that the implemented system is able to consistently regenerate the same network solution from different learning sessions, following more or less the same trajectory in parameter space despite different instances for the random perturbation values, illustrates the global deterministic error descent property of the stochastic scheme, resembling the characteristics of gradient descent on a macroscopic scale. More importantly, learning sessions on different fabricated instances of the same chip design all consistently converged to network solutions of comparable quality, despite fairly large differences in the parameter values obtained at convergence due to the randomness of the implementation errors. The robustness of the implemented learning system to random offsets and nonlinearities induced at fabrication is a major advantage of the model-independent approach followed here, using statistical techniques based on direct observations of the error to obtain the gradient information needed for learning.

4.4 Autonomous Dynamic Analog Storage

Performance tests on the autonomous storage aspects of the described integrated architecture are carried out on a separate chip, implementing an array of 128 storage cells interfacing with on-chip binary quantization elements configured for autonomous refresh operation. Strictly speaking, experimental results obtained from the reduced chip implementation cannot be directly extrapolated towards the original system with embedded network and learning functions. We note that nevertheless both implemented architectures are fully compatible, complying to the analog-binary structure outlined in Section 3.1, and using topologically identical instances for the charge-pump incremental update elements. Therefore, the experience gained from the experiments presented below is applicable to future generations of learning chips incorporating on-chip binary quantization for autonomous refresh operation. The reason for not integrating quantizing functions in the present learning network implementation was partly driven by practical considerations of pin count and die size, and was also partly intended to clearly separate different aspects

Table 4.2: Parameter values observed at convergence, for four different sessions.

Session 1

j	1	2	3	4	5	6
$W1j$	1.18	-0.21	0.61	-0.68	-1.17	1.06
$W2j$	0.44	1.80	0.47	-0.79	1.86	-1.40
$W3j$	0.16	0.29	1.18	0.15	0.26	0.06
$W4j$	1.61	0.16	-0.04	0.45	-0.12	0.19
$W5j$	0.68	0.33	1.46	1.83	1.84	0.30
$W6j$	0.27	1.42	0.25	1.23	-1.02	0.15
θj	-0.00	-0.51	-0.99	0.78	0.18	0.16

Session 2

j	1	2	3	4	5	6
$W1j$	1.17	-0.24	0.48	-0.69	-1.24	1.05
$W2j$	0.40	1.80	0.34	-0.84	1.86	-1.45
$W3j$	-0.05	0.37	1.26	0.13	0.17	0.04
$W4j$	1.40	0.11	-0.26	0.73	0.00	0.06
$W5j$	0.69	0.19	1.56	1.82	1.85	0.40
$W6j$	0.21	1.43	0.28	1.31	-1.08	0.24
θj	0.01	-0.47	-0.78	0.66	0.16	0.09

Session 3

j	1	2	3	4	5	6
$W1j$	1.21	-0.24	0.62	-0.66	-1.16	0.99
$W2j$	0.39	1.76	0.45	-0.90	1.86	-1.38
$W3j$	0.14	0.29	1.32	0.09	0.19	0.14
$W4j$	1.54	0.31	0.11	0.54	-0.37	0.08
$W5j$	0.73	0.26	1.43	1.82	1.83	0.35
$W6j$	0.21	1.41	0.29	1.19	-1.02	0.20
θj	0.02	-0.46	-0.81	0.68	0.12	0.03

Session 4

j	1	2	3	4	5	6
$W1j$	1.25	-0.20	0.54	-0.61	-1.06	1.06
$W2j$	0.49	1.79	0.47	-0.98	1.86	-1.42
$W3j$	0.05	0.21	1.24	0.32	0.15	0.17
$W4j$	1.46	0.25	-0.16	0.70	-0.10	0.08
$W5j$	0.67	0.32	1.43	1.83	1.85	0.41
$W6j$	0.20	1.44	0.21	1.31	-1.07	0.13
θj	-0.08	-0.57	-0.96	0.64	0.35	0.06

of performance for systematic testing.

The array structure of storage cells contains four rows of 32 capacitive memory cells each, every row driven by a single binary quantizer in a time-multiplexed configuration conform to that of Figure 3.5 (b). A micrograph of the 4 mm^2 2μm double-poly CMOS chip is shown in Figure 4.15. The incremental refresh device complies with the I/D circuit structure of Figure 3.6. For the binary quantizers, the A/D/A architecture of Figure 3.7 was chosen, with specific circuit implementation described in Appendix A. The size of the capacitive storage elements is 1 pF. With special care in the design to minimize the parasitic junction leakage current induced by the MOS transistors coupled to the storage capacitors, the drift rate of the stored voltages was reduced to typically less than 1 mV/sec at room temperature.

Figure 4.16 shows the results of experiments conducted on the chip to demonstrate the robustness of autonomous analog memory operation in especially noisy and imprecise environments. While the embodiment of the A/D/A converter used for the quantization was specifically designed and rated for a resolution of 8 bit, the device was required to operate at 9-bit resolution in A/D conversion mode to support 8-bit effective quantization (256 discrete levels) of the analog memory. Likewise, no precaution was taken to shield the circuitry from noise originating in the power supply and from various other external noise sources. This resulted into relatively poor accuracy and noise performance of the binary quantization obtained from the LSB of the conversion. Figure 4.16 (a) shows the measured probability distribution of the quantization bit as a function of input voltage, in the neighborhood of the most critical bit-transition of the converter. The distribution, which ideally follows sharp and regularly spaced transitions, shows signs of distortion due to noise and conversion nonlinearities, giving rise to rather frequent errors in the individual updates under refresh. Nevertheless, for a very small amplitude of the refresh updates (20 μV) relative to the separation between the discrete levels (13 mV), the memory experimentally demonstrated long-term storage with perfect data retention over time spans exceeding 10^9 refresh cycles. This robustness was observed even for relatively "weak" memory states

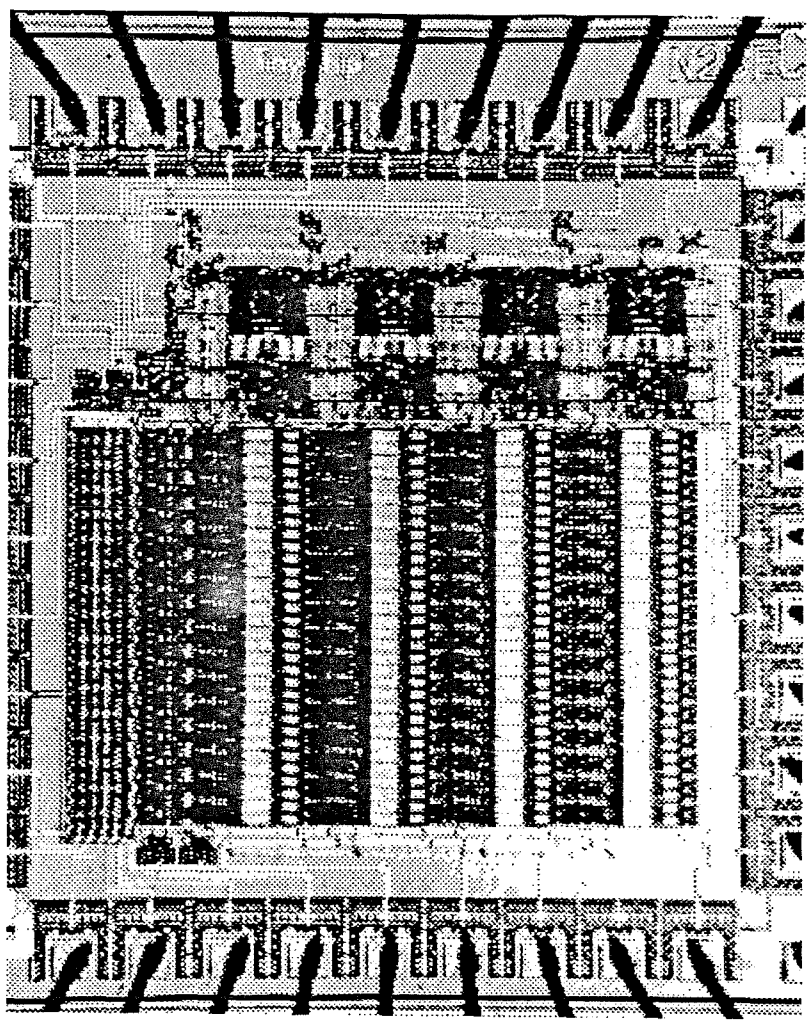


Figure 4.15: Chip micrograph of the 128-element integrated analog memory.

such as the one corresponding to "10000000," for which Figure 4.16 (a) predicts an individual error rate higher than 30% at its boundary toward the state "01111111." Even at an elevated 30% error rate, it is quite unlikely that an ensemble of consecutive quantization bits will produce a consistent drift away from the correct memory level, which intuitively explains the long-term stability of the memory despite the high quantization error rate. Figure 4.16 (b) shows the measured histogram distribution for the voltage level stored on an analog memory cell, preset initially to either of four adjacent memory states, under periodic refresh at 5 msec intervals and with 20 μV update steps. By virtue of the redundant refresh method of the invention, the excursion of the memory value is confined to a narrow band with observed width less than 2 mV (limited by the resolution of the voltmeter), significantly better than the intrinsic voltage discrimination capability supported by the quantizer due to the noise and the conversion nonlinearity.

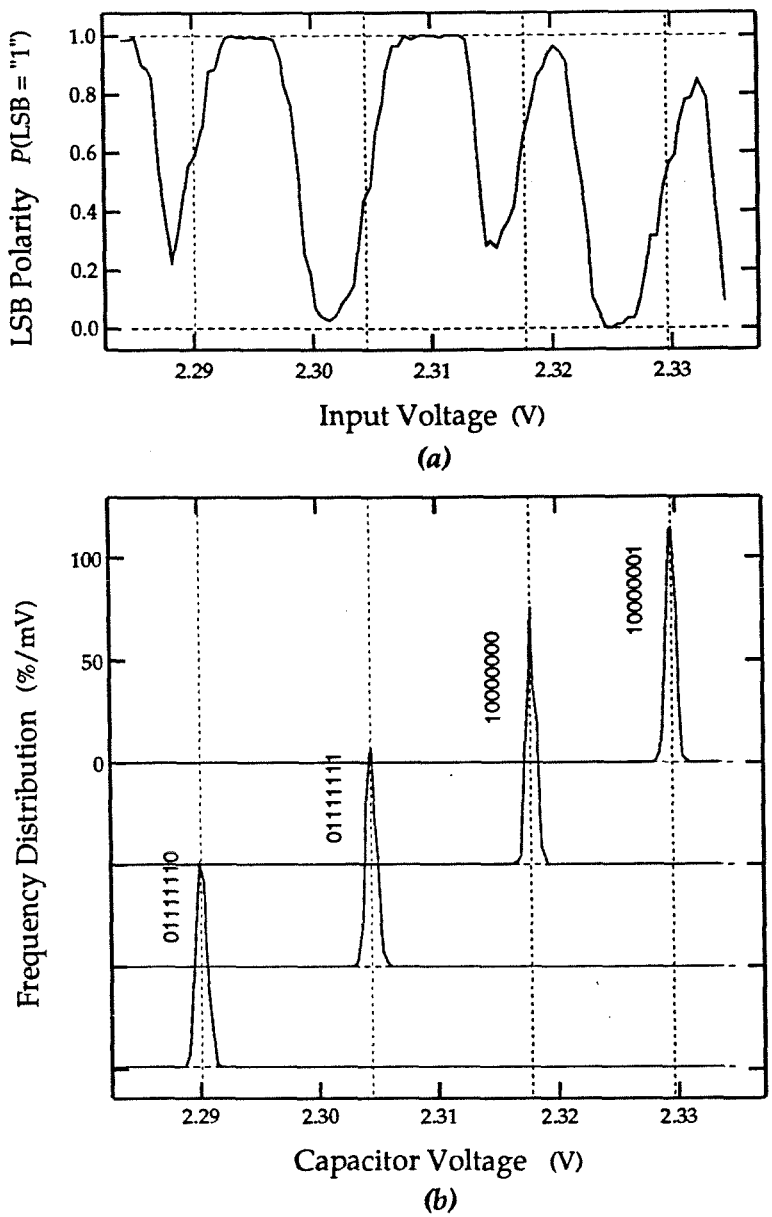


Figure 4.16: Experimental observation of quantization and autonomous refresh. *(a)* Measured probability of LSB polarity at 9-bit quantization. *(b)* Voltage distribution of quantized states observed under periodic refresh with parameters $\delta = 20\ \mu V; T = 5\ msec.$

Chapter 5

Conclusions

5.1 Stochastic Error Descent Optimization

The parallel stochastic method for error-descent optimization offers an attractive gradient-free alternative for gradient-based optimization, such as the widely used method of gradient descent and its various derivates. In particular, the stochastic alternative does not require explicit information about the functional dependence of the system under optimization, unlike gradient-based methods which assume a given functional model for the system to calculate the parameter updates. In contrast, the stochastic method obtains the values for the updates from direct evaluation of the optimization error index at select values for the parameters only. The method therefore supports optimization of physical systems which lack explicit quantitative information about the internal structure with regard to the parameters and their impact on the error measure, but which support direct observation of the error under arbitrary settings of the parameters. Likewise, the stochastic method avoids the usual degradation of optimization performance encountered in the presence of mismatches between the conceptual and the true models of an otherwise well-defined system, such as systematic errors in a constructed system arising from impurities in the implementation.

From a computational perspective, the efficiency of the stochastic method for error-

descent optimization is obviously hindered by the limited scalar information obtained from the error observations, only revealing part of the full error gradient per iteration. Nevertheless, owing to the uncorrelated statistics of the random parameter perturbations used for the error observations, it was shown that the efficiency of the stochastic method compares favorably to that of gradient descent which usually requires an extensive amount of computation to arrive at a value for the error gradient from the model information. As a result, the method becomes computationally more efficient than gradient descent for optimization in complex systems, for which the required computational resources to obtain full gradient information usually scale strongly with the number of parameters to be optimized. In addition, the favorable efficiency of the method is achieved using relatively simple operations independent of the intricacies of the system model, which are therefore more amenable to special-purpose hardware implementations.

5.2 On-Line Learning in Dynamical Systems

While the stochastic method can be generally applied to the optimization of a broad class of parameter-driven systems, it is especially suited for applications of supervised learning in recurrent dynamical systems, for which the model complexity of the system dynamics leads to fairly elaborate schemes to compute the error gradient, especially when real-time operation is required. On-line schemes for the stochastic method, supporting real-time supervised learning, were derived through partitioning of the integrated optimization error functional in successive non-overlapping time intervals. It was shown that optimal convergence results are obtained when the time spacing between consecutive intervals matches the typical time scale of the system dynamics and the training signals. Particularly long time intervals for the error observations cause a strong and non-uniform parameter dependence of the error functional. Likewise, very short time intervals lead to incomplete activation of the internal system dynamics during learning, effectively only involving parameters which relate to those state variables directly connected to the output units.

Practical on-line realization of the method in principle requires two error observations

be performed simultaneously on the system, for two different settings of the parameters. A generally applicable realization, under arbitrary configurations of the system under optimization and the training signal, was not found feasible due to the relatively long time spans required for the error observations. For practical purposes, two alternative on-line realizations of the method, each supporting a distinct class of applications, were formulated and analyzed with regard to the implied constraints on the system configuration and training signals. One realization, for prediction of time-varying processes, employs replicas of the system to perform the error observations concurrently, thereby requesting full knowledge of the system model and requiring access to the internal state variables. The other realization, for establishing a given dynamical response onto an arbitrary system, basically unfolds the two error observations in time on the same system, and implies a periodic format for the training signals without requiring any knowledge about the system characteristics. While the first concurrent realization may allow for gradient-based alternatives, the second time-interlaced realization truly reflects the model-free nature of the stochastic method unattainable with gradient based methods, thereby supporting the optimization of black-box dynamical systems for which the parameter dependence can only be obtained through direct observation.

5.3 Analog VLSI Implementation

The model-free nature of the stochastic method, and its on-line time-interlaced variant, is particularly suited for learning and optimization in analog VLSI systems, of which exact models cannot be formulated due to analog imprecisions in the implementation. By virtue of the simple functional form of the method, the learning functions themselves can be efficiently implemented in analog VLSI, organized into a scalable and modular architecture, whereby local units serving parameter updates and perturbations are replicated for every parameter, and whereby global functions pertaining to the error observations coordinate the local functions at a higher level. The resulting architecture supports full parallel and mutually independent operation of the local functions to attain a large bandwidth, and

implies a minimal activity of global interconnect signals across the local cells.

Considerations of accuracy in the implementation of the local learning functions have prompted a hybrid analog-binary scheme for the activation of the perturbations and the generation of the incremental parameter updates. The resulting analog-binary architecture can essentially be used as well to periodically refresh the parameter values using a fault-tolerant incremental scheme, for long-term dynamic multi-valued storage of the parameter values whenever the learning is disabled. A practical and fairly accurate realization for the learning and refresh increments is obtained using a binary controlled CMOS charge pump circuit, supplying increments of reliable size over a large dynamic range.

Because of the common functional form of the incremental parameter updates under both learning and refresh, it is tempting to look for deeper origins which connect both, if any. Both the stochastic perturbative method for learning and the partial incremental refresh method for learning and optimization can be viewed as belonging to a common class of robust techniques, which exploit redundancy and statistical averaging to enhance system performance in an otherwise imprecise and noisy analog VLSI implementation medium. In particular, the redundancy in the partial incremental format of the refresh updates, specifying small fixed-amplitude increments in the direction towards the nearest discrete memory level, avoids the sensitivity to quantization errors typical of alternative refresh techniques. Likewise, the statistical averaging of the linear gradient estimates of the stochastic method under perturbations in random directions produces error descent characteristics which resemble those of an error-free gradient descent implementation, at a fraction of the implementation complexity. One standardized technique which can be counted as belonging to the same robust class is the method of delta-sigma modulation, exploiting redundancy and statistical averaging to extract high-resolution analog or digital information of low frequency content.

5.4 Experimental Verification

We have implemented and characterized an analog recurrent neural network chip incorporating the integrated learning and storage architecture. Experimental tests on the chip, trained with a circular dynamic trajectory, confirmed the robustness of learning performance in the presence of analog offsets in the implementation due to imprecision in the fabrication. While the time-averaged formulation of the functional used for error-descent learning is known to cause convergence problems due to local minima and discontinuities in the error surface, consistently successful results were obtained using strong initial teacher forcing and unbiased initial settings for the parameters. The network repeatedly succeeded to learn 1 kHz quadrature-phase oscillatory waveforms in 1,500 update cycles, spanning 100 sec total. Tests on a related structure, comprising a 128- element array of capacitive storage devices interfacing with integrated A/D/A quantizers, have demonstrated stable and autonomous storage of analog volatile information at 8-bit effective resolution.

5.5 Efficiency and Complexity

A final remark to the gradient-free approach, as it was presented here by means of the stochastic method, pertains to the trade-off between the implementation complexity and the attainable over-all computational efficiency. Indeed, one may argue rightly that more efficient methods can be derived to perform the same computational task through significantly less iterations and consequently through less computational effort. The same argument applies to ordinary gradient descent methods as well, for which significantly more efficient extensions have been formulated in the past, most but not all of those implying an increase in complexity for the functions to be performed. One of the important advantages of the gradient-free and real-time methodology presented here for supervised learning in dynamical systems is the simple and modular implementation structure that it supports, for which presently no alternatives exist. On the contrary, extended variants

of the stochastic method which incorporate a higher efficiency at the expense of increased complexity may as well be substituted for even more efficient methods employing first or higher derivatives in the parameters, in case model information on the system is available.

Nevertheless, the efficiency of the stochastic method can be improved almost without affecting the implementation complexity by reorganizing the global functions only, which supervise the error observations and corresponding parameter updates at a higher level and thereby affect the error descent performance directly. Simple extensions for increased efficiency directly supported at the global level can be obtained by adjusting the learning rate according to second-order information gathered from the error observations, thereby speeding the convergence process. Extensions for an increased functionality may include safety measures in the updates to avoid breakdown of the method under ill-defined conditions, and aspects of robustness and consistency in non-convex optimization may be addressed through global adjustment of the perturbation amplitude. In principle, other extensions at the global level yielding further improvements in efficiency or functionality may be derived as well, possibly involving additional simple alterations in the structure of the local update units. The degree of freedom provided in the organization of the global and local operations leaves ample room for further exploration towards advanced variations on the stochastic method, rivaling the performance of complex optimization methods at a fraction of their implementation cost.

Appendix A

Bit-serial A/D/A Conversion

Below, we present a compact and fault-tolerant CMOS circuit implementing a bi-directional bit-serial A/D/A (analog-to-digital and digital-to-analog) converter, which can be used to perform the function of binary quantization for multi-valued dynamic analog storage while allowing for additional read/write digital access as described in Section 3.2. Specifically, the device comprises a monotonic algorithmic D/A converter which processes the MSB first and the LSB last, and a successive approximation A/D converter employing the intermediate conversion results of the D/A converter.

The MSB-first approach for D/A conversion has two distinct advantages for large-scale integrated signal processing systems. First, it supports the bit sequence order necessary for successive approximation A/D conversion, and hence allows the integration of both A/D and D/A functions in a single bi-directional device with matched transfer characteristics between analog and digital formats in both directions. Second, the algorithm used for MSB-first D/A conversion [Cauwenberghs 93b] warrants conversion monotonicity regardless of component mismatches and nonlinearities, avoiding special compensation schemes for traditional algorithmic converters to improve accuracy beyond the intrinsic precision granted by the fabrication process [Shih 86].

A.1 Conversion Algorithm

Figure A.1 schematically illustrates the MSB-first D/A conversion algorithm, in the case of 4-bit conversion for an input code "1011." The intermediate states U_i each correspond to the partial conversion of the i most significant bits of the n-bit input code, and their values are constructed from previous intermediate values conditional on the state of the sequenced bit, leading to the final conversion value in n successive steps. Two reference values, U^+_{ref} and U^-_{ref}, define the analog conversion range and roughly correspond to the upper and lower extremes of the analog spectrum respectively. The algorithm employs two analog "registers" U^+_{ref} and U^-_{ref} for storage and later recall of selected intermediate conversion values, and one unit for averaging the two values contained in both registers to construct the next intermediate conversion value. The following sequence of symbolic instructions specifies the algorithm [Cauwenberghs 93b]:

a. Initialization($i = 0$):

Preset U_0^+ to U_{ref}^+ and U_0^- to U_{ref}^- ;

Set U_0 halfway in between U_0^+ and U_0^- ;

b. Algorithmic iteration ($i - 1$ to i , from $i = 1$ to n):

If bit i is "1": set U_i^+ to U_{i-1}^+ and U_i^- to U_{i-1};

"0": set U_i^+ to U_{i-1} and U_i^- to U_{i-1}^- ;

Set U_i halfway in between U_i^+ and U_i^- ;

c. Termination ($i = n$):

The final D/A conversion result is given by U_n.

Because this algorithm implements a tree structure for the intermediate values which naturally preserves the order of the input codes, a monotonic conversion is warranted even in the case of a bias or nonlinearity in the averaging. Nevertheless, errors in the recall of the stored register values $U^+{}_i$ and $U^-{}_i$ distort the tree structure and affect the monotonicity directly. The algorithm extends to successive approximation A/D conversion by successively comparing an analog input with the intermediate conversion values, and if the input is larger assigning a "1" to the succeeding bit to be converted, and otherwise

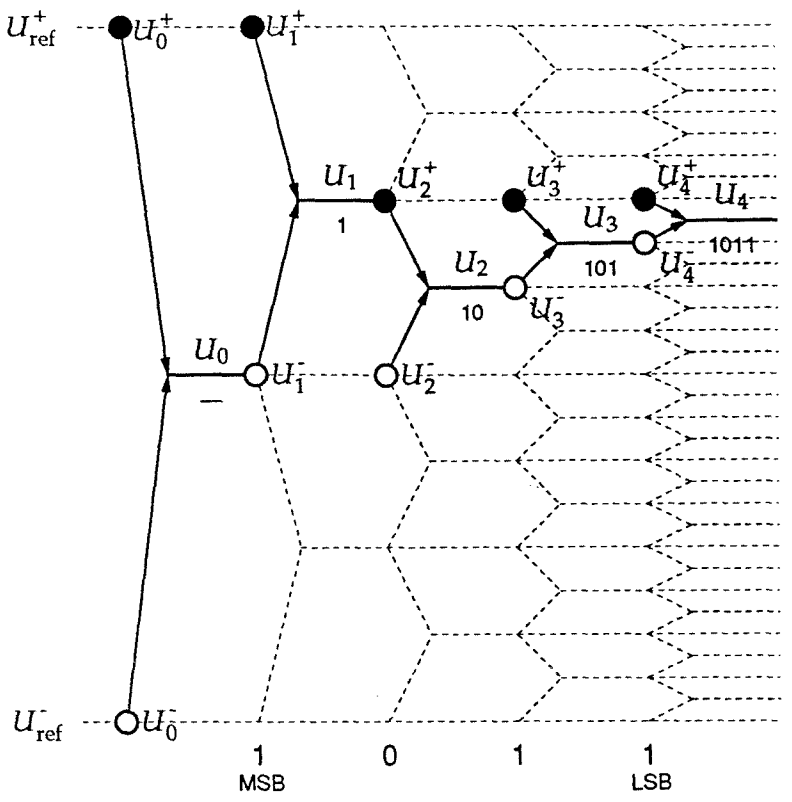


Figure A.1: D/A conversion algorithm.

assigning a "0." The digital output is then given by the sequence of bits obtained, from MSB to LSB.

A.2 A/D/A Converter Block Diagram

The block diagram of the analog portion of the bi-directional A/D/A converter, implementing the above algorithm with CMOS-C elements, is shown in Figure A.2. The analog circuit comprises four capacitors C^+, C^-, $C^{+\prime}$ and $C^{-\prime}$, two differential transconductance amplifiers OTA^+ and OTA^- with both inverting and non-inverting outputs, one latched comparator with additional inverting transconductance output, and a series of switches [Cauwenberghs 93b]. Switch SH performs the averaging function by sharing and redistributing charge preset on C^+ and C^-. $C^{+\prime}$ and $C^{-\prime}$ represent the storage registers U^+

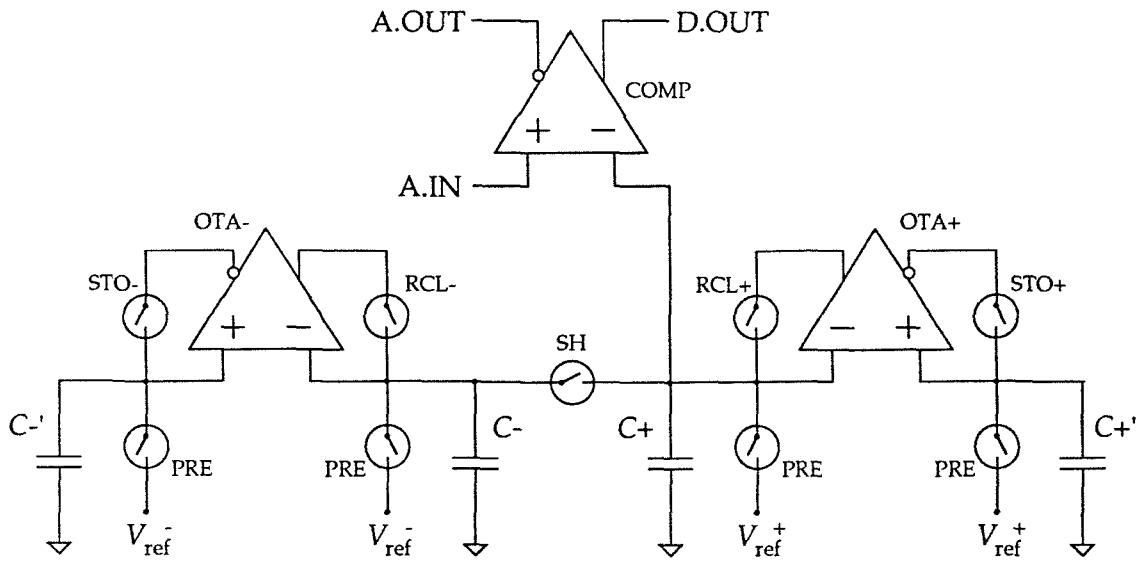


Figure A.2: A/D/A converter block diagram.

and U^- respectively. OTA$^+$ and OTA$^-$, in conjunction with switches STO$^+$, RCL$^+$, STO$^-$ and RCL$^-$, direct the storage and recall operations by triggered non-destructive replication of charge from one terminal to the other, in either direction. To implement offset-free registers, a zero replication error for successive store and recall operation is required, which is satisfied if the input-referred offsets for the inverting and non-inverting outputs of the corresponding OTA are equal. Comparator COMP serves as a successive approximation discriminator. The inverting output A.OUT of COMP is included to provide negative feedback to the A.IN input, in order to construct a representative D/A output reflecting the input offset of the comparator, thereby allowing close matching between A/D and D/A conversion transfer characteristics.

With this set of functional elements, the above conversion algorithm translates as [Cauwenberghs 93b]:

a) Initialization:

Preset C^- and $C^{-\prime}$ to V_{ref}^-, and C^+ and $C^{+\prime}$ to V_{ref}^+ with $V_{ref}^- < V_{ref}^-$.

Then share the charge on C^- and C^+ (activate switch SH).

b) Algorithmic iteration / Successive approximation:

For n successive steps, iterate:

If D/A conversion:

Obtain the next input bit (from MSB to LSB)

If A/D conversion:

Compare V^+ across C^+ (or V^- on C^-) with A.IN

(Evaluate the output D.OUT from the comparator);

set the bit to D.OUT;

If the bit is "1": Activate BRE^- in STO mode (activate STO^-) and

activate BRE^+ in RCL mode (activate RCL^+);

"0": Activate BRE^- in RCL mode (activate RCL^-) and

activate BRE^+ in STO mode (activate STO^+);

then share the charge on C^- and C^+ (activate switch SH).

c) Termination:

If D/A conversion:

The converted voltage is represented by either of

V^- across C^- and V^+ across C^+.

If A/D conversion:

The digital word is given by the sequence of the bits from b)

in the order processed, the MSB first and the LSB last.

A.3 Detailed Circuit Structure

The detailed structure of the analog portion of the A/D/A circuit is given in Figure A.3. The circuit methodology has been derived as to reduce the circuit complexity and the sensitivity to process variations to the absolute minimum, yielding a compact, process-robust and low-power design. The OTA's and the comparator use a standard cascoded current mirror stage, rather than a folded cascode stage, for low-power operation and a reduced sensitivity to transistor mismatches. The configuration of the switches internal to the OTA's and the comparator, for selection of the operational mode, has been chosen as to provide close

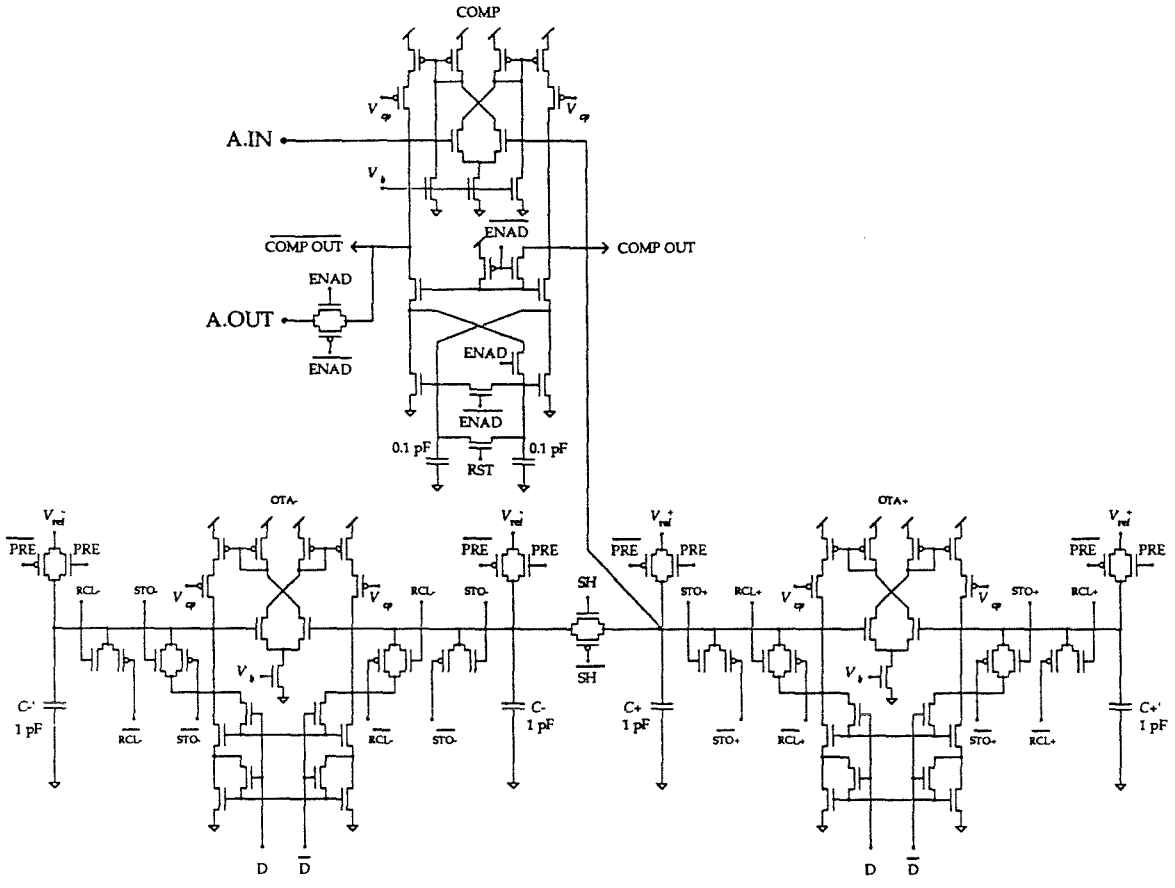


Figure A.3: Detailed schematics of the A/D/A analog circuitry.

matching between the input-referred offsets of the inverting and non-inverting outputs. Dummy switches are added to the capacitors for differential compensation of the charge injection due to clock feedthrough. Such, the switch injection noise appears as a common-mode input component at the OTA's, and disappears at both sides after deactivation of the selected pair of functional and dummy switches. The comparator, when activated in comparison mode (ENAD active), contains a cross-coupled output stage for latched operation. An active-high pulse on RST triggers and fixes the binary output. A source follower isolates the capacitor C^+ from the input of the comparator to avoid capacitive coupling to the A.IN analog input. An additional source follower is added to C^- to preserve symmetry for improved matching.

Logic circuitry is included in the converter, integrated along with the analog circuitry, to control operation and drive the STO and RCL switch waveforms. The control signal

ENAD selects the operational mode of the A/D/A converter. A/D conversion mode is achieved with ENAD in a high state, and D/A conversion mode is established otherwise. ENAD controls the tri-state D and $\overline{D}$ nodes, which serve as both digital input and output. ENAD also selects the activation state of the inverting and non-inverting outputs of the comparator. All STO and RCL switches, including the dummies, are disabled when SEN is in a low state; otherwise the STO^- and RCL^+ switches are activated for a "1" state of the converted bit, and the STO^+ and RCL^- switches are activated for a "0" bit. The STO and RCL switching waveforms are slew-rate controlled for the switch-off transition to reduce residual clock feedthrough.

The A/D/A converter interfaces with five individual terminals for operation control and I/O: ENAD, A.IN, A.OUT, D and $\overline{D}$. Figure A.4 shows some possible I/O connection configurations of the A/D/A converter, operating as a one-way or bi-directional converter interfacing with a hybrid analog-digital VLSI system. In addition, the converter requires 13 global connections, common for all instances, to supply two reference sources ($V_{ref}{}^+$ and $V_{ref}{}^-$), five bias levels (V_f, V_b, V_{cp}, SR_n and SR_p), and six clock waveform signals (PRE, SH, RST, SEN and the complements of PRE and SH). The adjustable bias levels are provided to accommodate diverse accuracy, power level and speed requirements. The required format of the clock waveform signals for proper D/A and A/D conversion is given in Figure A.5.

A.4 Results and Discussion

An array of eight of the described A/D/A converters has been fabricated on a MOSIS 'Tiny'- chip in a 2μm double-poly CMOS process. The layout of the converter cell is shown in Figure A.6. The 11 common supply lines run across horizontally, to enable abutting of the cell with neighboring cells for the compact construction of linear arrays. A scope-plot displaying the binary tree of intermediate D/A conversion voltages captured from the chip is given in Figure A.7. The devices were tested at 8-bit resolution for conversion cycle speeds ranging from 20 μsec to 2 msec. For each given conversion speed, the bias levels were adjusted to obtain a global optimal A/D and D/A conversion performance (in terms

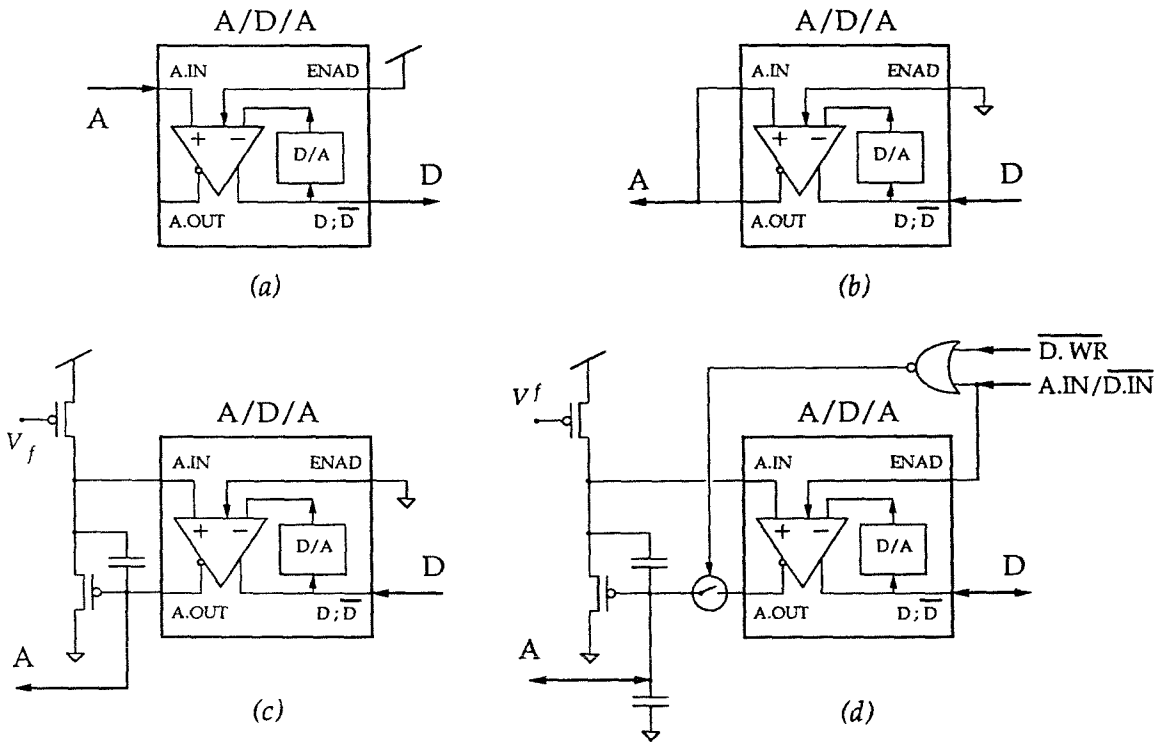


Figure A.4: I/O connection configurations. *(a)* A/D converter; *(b)* Fast settling nonlinear D/A converter; *(c)* D/A converter; *(d)* bi-directional A/D/A converter implementing analog memory with A and D random access.

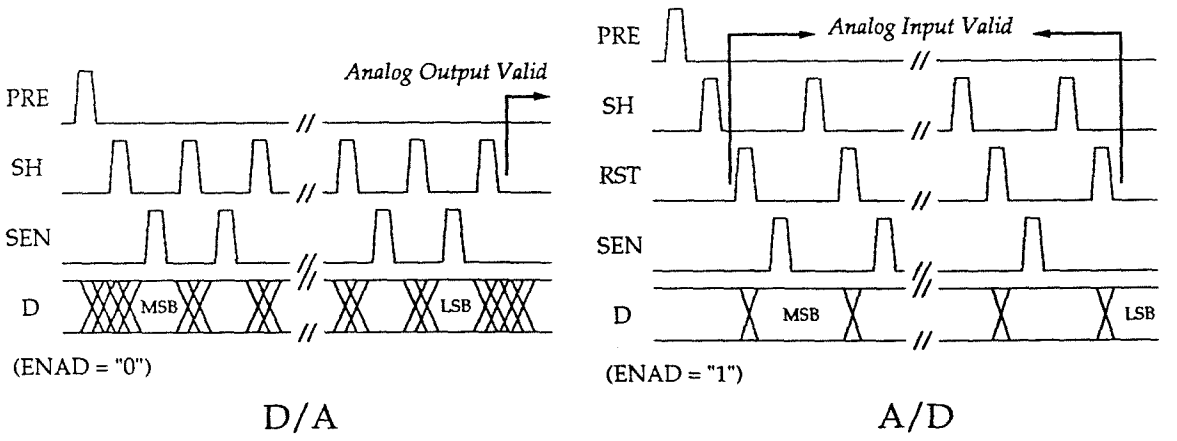


Figure A.5: Format of clock waveforms and I/O timing.

of differential linearity). Subsequently, the bias currents were further decreased until the point of perceptible performance degradation. Figure A.8 shows typical plots of A/D and D/A conversion nonlinearities at 2 msec conversion cycle and 8-bit length. Whereas a uniform, low (0.2 LSB) differential nonlinearity was observed for all converters under equal biasing and timing conditions, the variance and typical amplitude of the integral nonlinearity was recorded to be quite higher. This effect is attributed to the statistical variations of capacitance ratios and comparator offsets across the chip, which by virtue of the conversion algorithm do not affect the monotonicity, but distort the integral conversion characteristic into a rather smoothly curved shape. The latter should not be considered a disadvantage *per se*; many signal acquisition applications only require monotonicity. Furthermore, as Figure A.8 suggests, the integral nonlinearity for both A/D and D/A conversion are closely matched because of the successive approximation A/D scheme, and the integral nonlinearities in successive A/D and D/A conversions cancel out to yield a near-identity characteristic. Such is useful, *e.g.*, for the implementation of addressing and auto-refresh schemes for random access capacitive analog memories.

The characteristics of the A/D/A converter and the performance results at different cycle speeds are summarized in Table A.1 and A.2. As expected, speed can be traded for power economy, but with better accuracy at lower speeds and higher processing efficiency at higher speeds. The meaning of the term "efficiency" in this context relates to the number of conversions per unit time and unit power. Interestingly, at 20 μsec cycle speed, the A/D/A converter performs the equivalent of 250 Msamples/sec at 1 W of power. Unlike high-speed converters, an array of A/D/A converters achieves this bandwidth through parallelism and distributed processing rather than through dedicated and localized high-speed circuit techniques.

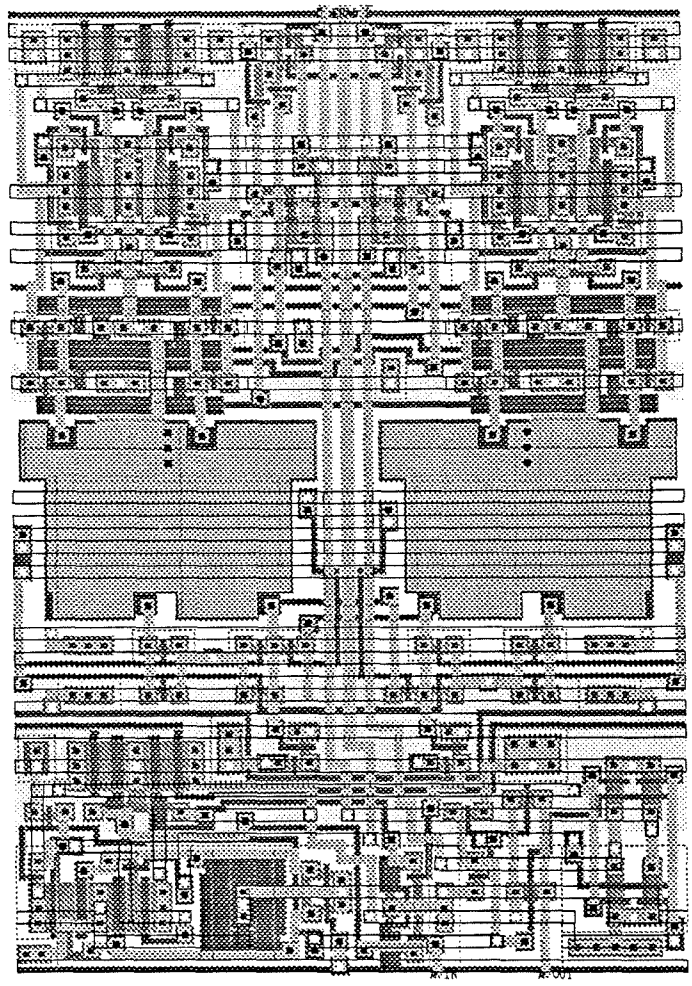


Figure A.6: Layout of the A/D/A converter cell.

Table A.1: A/D/A cell characteristics.

Technology	2 µm p-well double-poly CMOS
Cell Size	216 µm X 315 µm
Transistor Count	114
Supply	+ 5 V
Analog Conversion Range	1 V — 4 V
Resolution	8 bits
Cycle Time	≥ 20 µsec

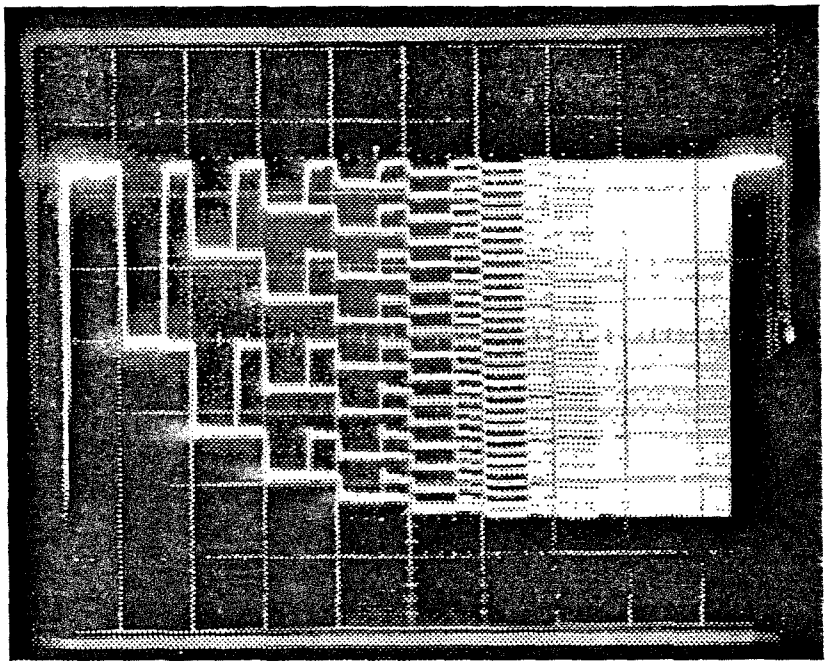


Figure A.7: Recorded algorithmic D/A conversion tree. Horizontal: 10 μsec/div; vertical: 0.5 V/div.

Table A.2: A/D/A cell performance at various conversion speeds.

Conversion Cycle Speed	20 µsec	200 µsec	2 msec
D/A Linearity			
DNL	0.5 LSB	0.4 LSB	0.2 LSB
INL	< 2 LSB	< 2 LSB	< 2 LSB
A/D Linearity			
DNL	0.6 LSB	0.5 LSB	0.3 LSB
INL	< 2 LSB	< 2 LSB	< 2 LSB
Supply Current	40 µA	25 µA	7.5 µA
Power Dissipation	200 µW	125 µW	38 µW

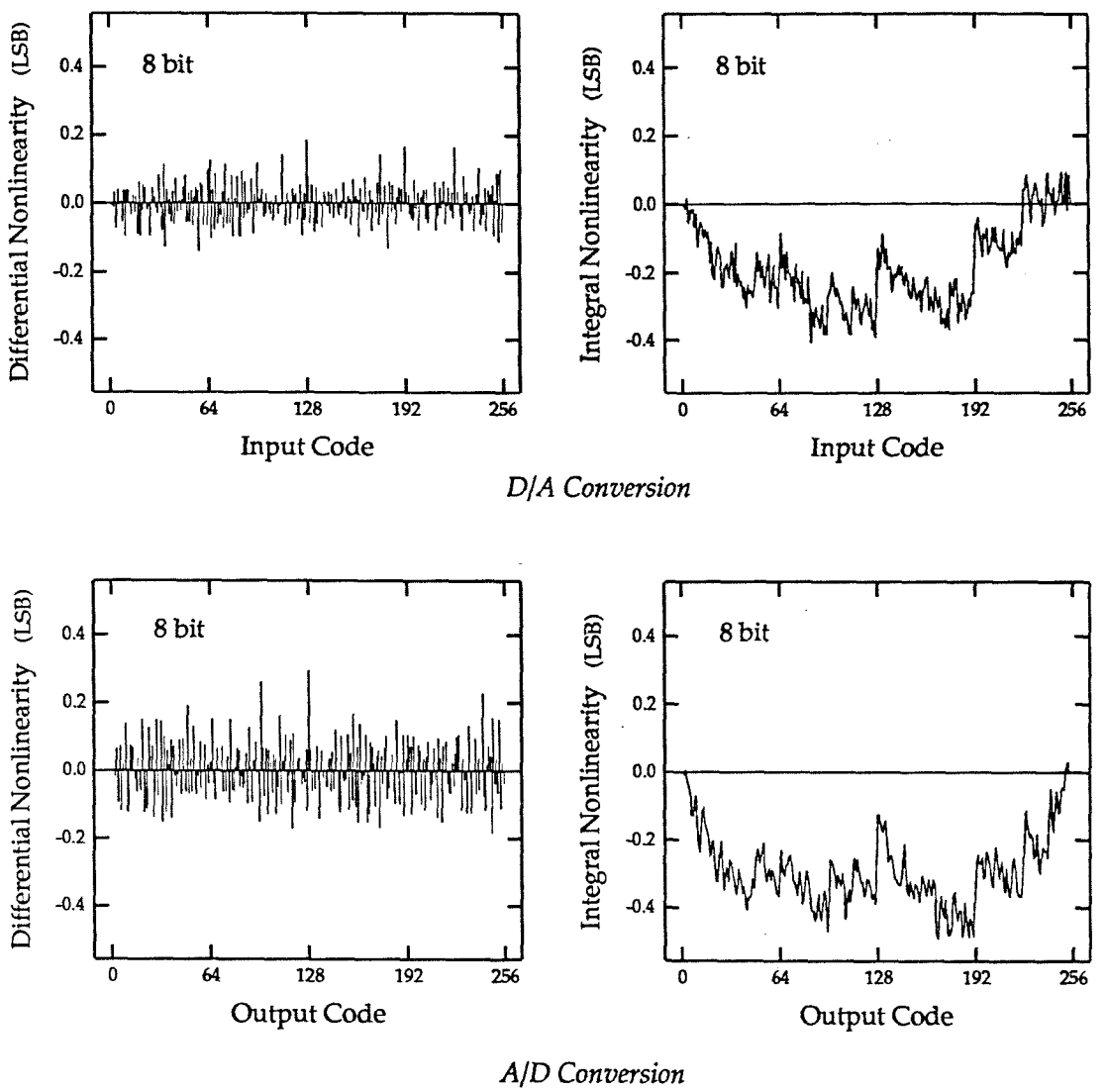


Figure A.8: Differential and integral nonlinearity of D/A and A/D conversion by the A/D/A converter, at 2 msec conversion cycle.

References

[Alspector 91] J. Alspector, J.W. Gannett, S. Haber, M.B. Parker, and R. Chu, "A VLSI-Efficient Technique for Generating Multiple Uncorrelated Noise Sources and Its Application to Stochastic Neural Networks," *IEEE T. Circuits and Systems*, **38** (1), pp 109-123, 1991.

[Alspector 89] J. Alspector, B. Gupta, and R.B. Allen, "Performance of a Stochastic Learning Microchip," in *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman, vol. **1**, pp 748-760, 1989.

[Alspector 93] J. Alspector, R. Meir, B. Yuhas, and A. Jayakumar, "A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks," in *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman, vol. **5**, pp 836-844, 1993.

[Andreou 91] A.G. Andreou, K.A. Boahen, P.O. Pouliquen, A. Pavasovic, R.E. Jenkins, and K. Strohbehn, "Current-Mode Subthreshold MOS Circuits for Analog VLSI Neural Systems," *IEEE Transactions on Neural Networks*, vol. **2** (2), pp 205-213, 1991.

[Baldi 93] P. Baldi, "Gradient Descent Learning Algorithms: A General Dynamical Systems Perspective," to appear in *IEEE Transactions on Neural Networks*.

[Baldi 92] P. Baldi, "Learning in Dynamical Systems: Gradient Descent, Random Descent and Modular Approaches," JPL Technical Report, California Institute of Technology, 1992.

[Benson 93] R.G. Benson, Ph.D. Dissertation, California Institute of Technology, 1993.

[Brent 73] R.P. Brent, "Algorithms for Optimization without Derivatives," Englewood Cliffs, NJ: Prentice Hall, 1973.

[Candy 92] J.C. Candy and G.C. Temes, "Oversampled Methods for A/D and D/A Conversion," in *Oversampling Delta-Sigma Data Converters: Theory, Design and Simulation*, NJ: IEEE Press, pp 1-25, 1992.

[Catfolis 93] T. Catfolis, "A Method for Improving the Real-Time Recurrent Learning Algorithm," *Neural Networks*, vol. **6** (6), pp 807-821, 1993.

[Cauwenberghs 90] G. Cauwenberghs, C.F. Neugebauer, A. Agranat, and A. Yariv, "Large Scale Optoelectronic Integration of Asynchronous Analog Neural Networks," in *Dig. Int. Neural Network Conf.* (INNC-90 Paris), Kluwer Academic, vol. **2**, pp 551-554, 1990.

[Cauwenberghs 92] G. Cauwenberghs, C.F. Neugebauer, and A. Yariv, "Analysis and Verification of an Analog VLSI Outer-Product Incremental Learning System," *IEEE Transactions on Neural Networks*, vol. **3** (3), pp 488-497, 1992.

[Cauwenberghs 93a] G. Cauwenberghs, "A Fast Stochastic Error-Descent Algorithm for Supervised Learning and Optimization," in *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman, vol. **5**, pp 244-251, 1993.

[Cauwenberghs 93b] G. Cauwenberghs and A. Yariv, "Method and Apparatus for Monotonic Algorithmic Digital-to-Analog and Analog-to-Digital Conversion," U.S. patent **5,258,759**, 1993.

[Cauwenberghs 93c] G. Cauwenberghs and A. Yariv, "Method and Apparatus for Long-Term Multi-Valued Storage in Dynamic Analog Memory," U.S. patent pending, filed 1993.

[Chin 93] D.C. Chin, "Performance of Several Stochastic Approximation Algorithms in the Multivariate Kiefer-Wolfowitz Setting," in *Proc. of the 25th Symp. on the Interface Computing Science and Statistics*, 1993.

[Cohen 92] M.H. Cohen and A.G. Andreou, "Current-Mode Subthreshold MOS Implementation of the Jutten-Herault Autoadaptive Network," *IEEE J. Solid-State Circuits*, vol. **27** (5), pp 714-727, 1992.

[Czarnul 86] Z. Czarnul, "Modification of the Banu-Tsividis Continuous-Time Integrator Structure," *IEEE Transactions on Circuits and Systems*, vol. **33** (7), pp 714-716, 1986.

[Dembo 90] A. Dembo and T. Kailath, "Model-Free Distributed Learning," *IEEE Trans. Neural Networks*, vol. **1** (1), pp 58-70, 1990.

[Donald 93] J. Donald and L. Akers, "An Adaptive Neural Processor Node," *IEEE Transactions on Neural Networks*, vol. **4** (3), pp 413-426, 1993.

[Fattaruso 93] J.W. Fattaruso, S. Kiriaki, G. Warwar, and M. de Wit, "Self-Calibration Techniques for a Second-Order Multibit Sigma-Delta Modulator," in *ISSCC Technical Digest*, IEEE Press, vol. **36**, pp 228-229, 1993.

[Flower 93] B. Flower and M. Jabri, "Summed Weight Neuron Perturbation: An $\mathcal{O}(n)$ Improvement over Weight Perturbation," in *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman, vol. **5**, pp 212-219, 1993.

[Frye 91] R.C. Frye, E.A. Rietman, and C.C. Wong, "Back-Propagation Learning and Nonidealities in Analog Neural Network Hardware," *IEEE Transactions on Neural Networks*, vol. **2** (1), pp 110-117, 1991.

[Funahashi 93] K.-I. Funahashi and Y. Nakamura, "Approximation of Dynamical Systems by Continuous Time Recurrent Neural Networks," *Neural Networks*, vol. **6** (6), pp 801-806, 1993.

[Furman 88] B. Furman, J. White, and A.A. Abidi, "CMOS Analog IC Implementing the Back Propagation Algorithm," in *Abstracts 1st Annual INNS Meeting*, vol. **1**, p 381, 1988.

[Golomb 67] S.W. Golomb, "Shift Register Sequences," San Francisco, CA: Holden-Day, 1967.

[Graf 89] H.P. Graf and L.D. Jackel, "Analog Electronic Neural Network Circuits," IEEE Circuits and Devices Mag., vol. **5** (4), pp 44-49, 1989.

[Gregorian 86] R. Gregorian and G.C. Temes, "Analog MOS Integrated Circuits for Signal Processing," New York, NY: John Wiley, 1986.

[Hochet 89] B. Hochet, "Multivalued MOS Memory for Variable-Synapse Neural Networks," Electronic Letters, vol. **25** (10), pp 669-670, 1989.

[Hochet 91] B. Hochet, V. Peiris, S. Abdot, and M.J. Declercq, "Implementation of a Learning Kohonen Neuron Based on a New Multilevel Storage Technique," *IEEE J. Solid-State Circuits*, vol. **26** (3), pp 262-267, 1991.

[Horio 92] Y. Horio, and S. Nakamura, "Analog Memories for VLSI Neurocomputing," in *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations*, IEEE Press, pp 344-363, 1992.

[Jabri 92] M. Jabri and B. Flower, "Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayered Networks," *IEEE Trans. Neural Networks*, vol. **3** (1), pp 154-157, 1992.

[Kirk 93] D. Kirk, D. Kerns, K. Fleischer, and A. Barr, "Analog VLSI Implementation of Gradient Descent," in *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman, vol. **5**, pp 789-796, 1993.

[Kirkpatrick 83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vechi, "Optimization by Simulated Annealing," *Science*, vol. **220** (4598), pp 671-680, 1983.

[Kushner 78] H.J. Kushner, and D.S. Clark, "Stochastic Approximation Methods for Constrained and Unconstrained Systems," New York, NY: Springer-Verlag, 1978.

[LeCun 93] Y. LeCun, P.Y. Simard, and B. Pearlmutter, "Automatic Learning Rate Maximization by On-Line Estimation of the Hessian's Eigenvectors," in *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman, vol. 5, pp 156-163, 1993.

[Lee 91] E.K.F. Lee, and P.G. Gulak, "Error Correction Technique for Multivalued MOS Memory," Electronic Letters, vol. 27 (15), pp 1321-1323, 1991.

[Linares-Barranco 93] B. Linares-Barranco, E. Sanchez-Sinencio, A. Rodriguez-Vazquez, and J.L. Huertas, "A CMOS Analog Adaptive BAM with On-Chip Learning and Weight Refreshing," *IEEE Transactions on Neural Networks*, vol. 4 (3), pp 445-455, 1993.

[Lopez 93] V. Lopez, R. Huerta, and J.R. Dorronsoro, "Recurrent and Feedforward Polynomial Modelling of Coupled Time Series," *Neural Computation*, vol. 5 (5), pp 795-811, 1993.

[Mead 89] C.A. Mead, "Analog VLSI and Neural Systems," Reading, MA: Addison-Wesley, 1989.

[Narendra 90] K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1 (1), pp 4-27, 1990.

[Narendra 91] K.S. Narendra and K. Parthasarathy, "Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks," *IEEE Transactions on Neural Networks*, vol. 2 (2), pp 255-262, 1991.

[Narendra 93] K.S. Narendra and K. Parthasarathy, "Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization," *IEEE Transactions on Neural Networks*, vol. 4 (2), pp 192-206, 1993.

[Pearlmutter 89] B.A. Pearlmutter, "Learning State Space Trajectories in Recurrent Neural Networks," *Neural Computation*, vol. **1** (2), pp 263-269, 1989.

[Robins 51] H. Robins and S. Monro, "A Stochastic Approximation Method," *Annals of Mathematical Statistics*, vol. **22**, pp 400-407, 1951.

[Rumelhart 86a] D.E. Rumelhart and J.L. McClelland, Eds., "Parallel Distributed Processing, Explorations in the Microstructure of Cognition," vol. **1**, Cambridge, MA: MIT Press, 1986.

[Rumelhart 86b] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, vol. **1**, D.E. Rumelhart and J.L. McClelland, eds., Cambridge, MA: MIT Press, 1986.

[Sato 90a] M. Sato, "A Real Time Learning Algorithm for Recurrent Analog Neural Networks," *Biological Cybernetics*, vol. **62**, pp 237-241, 1990.

[Sato 90b] M. Sato, "A Learning Algorithm to Teach Spatiotemporal Patterns to Recurrent Neural Networks," *Biological Cybernetics*, vol. **62**, pp 259-263, 1990.

[Satyanarayana 89] S. Satyanarayana, Y.P. Tsividis, and H.P. Graf, "Analogue Neural Networks with Distributed Neurons," *Electronics Letters*, vol. **25** (5), pp 302-303, 1989.

[Satyanarayana 92] S. Satyanarayana, Y.P. Tsividis, and H.P. Graf, "A Reconfigurable VLSI Neural Network," *IEEE Journal on Solid-State Circuits*, vol. **27** (1), pp 67-81, 1992.

[Schmidhuber 92] J. Schmidhuber, "A Fixed Size Storage $O(n^3)$ Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks," *Neural Computation*, vol. **4** (2), pp 243-248, 1992.

[Shih 86] C.-C. Shih, and P.R. Gray, "Reference Refreshing Cyclic Analog-to-Digital and Digital-to-Analog Converters," *IEEE J. Solid-State Circuits*, vol. **21** (4), pp 544-554, 1986.

[Smith 90] M.J.S. Smith, "An Analog Integrated Neural Network Capable of Learning the Feigenbaum Logistic Map," *IEEE Transactions on Circuits and Systems*, vol. **37** (6), pp 841-844, 1990.

[Sompolinsky 86] H. Sompolinsky and I. Kanter, "Temporal Association in Asymmetrical Neural Networks," *Phys. Rev. Letters*, vol. **57** (22), pp 2861-2864, 1986.

[Spall 92a] J.C. Spall, "Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation," *IEEE Trans. Automatic Control*, vol. **37** (3), pp 332-341, 1992.

[Spall 92b] J.C. Spall, and J.A. Cristion, "Direct Adaptive Control of Nonlinear Systems Using Neural Networks and Stochastic Approximation," in *Proc. IEEE Conf. Dec. Control*, pp 878-883, 1992.

[Styblinsky 83] M.A. Styblinsky and A. Ruszcynski "Stochastic Approximation Approach to Statistical Circuit Design," *Electronics Letters*, vol. **19** (8), pp 300-302, 1983.

[Styblinski 90] M.A. Styblinski and T.-S. Tang, "Experiments in Nonconvex Optimization: Stochastic Approximation with Function Smoothing and Simulated Annealing," *Neural Networks*, vol. **3** (4), pp 467-483, 1990.

[Sun 92] G.-Z. Sun, H.-H. Chen, and Y.-C. Lee, "Green's Function Method for Fast On-Line Learning Algorithm of Recurrent Neural Networks," in *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman, vol. **4**, pp 333-340, 1992.

[Szu 87] H. Szu, "Nonconvex Optimization by Fast Simulated Annealing," *IEEE Proceedings*, vol. **75** (11), pp 138-1400, 1987.

[Terman 81] L.M. Terman, Y.S. Yee, R.B. Merrill, L.G. Heller, and M.B. Pettigrew, "CCD Memory Using Multilevel Storage," IEEE J. Solid-State Circuits, vol. **16** (5), pp 472-478, 1981.

[Toomarian 92] N.B. Toomarian and J. Barhen, "Learning a Trajectory using Adjoint Functions and Teacher Forcing," *Neural Networks*, vol. **5** (3), pp 473-484, 1992.

[Vittoz 77] E. Vittoz and J. Fellrath, "CMOS Analog Integrated Circuits Based on Weak Inversion Operation," *IEEE Journal on Solid-State Circuits*, vol. **12** (3), pp 224-231, 1977.

[Vittoz 89] E. Vittoz and X. Arreguit, "CMOS Integration of Herault-Jutten Cells for Separation of Sources," in *Analog VLSI Implementation of Neural Systems*, Norwell, MA: Kluwer, pp 57-83, 1989.

[Vittoz 91] E. Vittoz, H. Oguey, M.A. Maher, O. Nys, E. Dijkstra, and M. Chevroulet, "Analog Storage of Adjustable Synaptic Weights," in *VLSI Design of Neural Networks*, Norwell MA: Kluwer Academic, pp 47-63, 1991.

[Watts 92] L. Watts, D.A. Kerns, and R.F. Lyon, "Improved Implementation of the Silicon Cochlea," *IEEE Journal on Solid-State Circuits*, vol. **27** (5), pp 692-700, 1992.

[Werbos 90] P.J. Werbos, "Backpropagation Through Time: What It Does and How to Do It," *IEEE Proceedings*, vol. **87** (10), pp 1550-1560, 1990.

[Williams 89] R.J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. **1** (2), pp 270-280, 1989.

[Williams 90] R.J. Williams and J. Peng, "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories," *Neural Computation*, vol. **2** (4), pp 490-501, 1990.

[Yu 93] X.-H. Yu, G.-A. Chen, and S.-X. Cheng, "Acceleration of Backpropagation Learning Using Optimised Learning Rate and Momentum," *Electronics Letters*, vol. **29** (14), pp 1288-1290, 1993.

[Zak 89] M. Zak, "Terminal Attractors in Neural Networks," *Neural Networks*, vol. **2** (4), pp 259-274, 1989.