Stochastic Neural Computation I: Computational Elements

Bradley D. Brown, *Member*, *IEEE*, and Howard C. Card, *Fellow*, *IEEE*

Abstract—This paper examines a number of stochastic computational elements employed in artificial neural networks, several of which are introduced for the first time, together with an analysis of their operation. We briefly include multiplication, squaring, addition, subtraction, and division circuits in both unipolar and bipolar formats, the principles of which are well-known, at least for unipolar signals. We have introduced several modifications to improve the speed of the division operation. The primary contribution of this paper, however, is in introducing several state machine-based computational elements for performing sigmoid nonlinearity mappings, linear gain, and exponentiation functions. We also describe an efficient method for the generation of, and conversion between, stochastic and deterministic binary signals. The validity of the present approach is demonstrated in a companion paper through a sample application, the recognition of noisy optical characters using soft competitive learning. Network generalization capabilities of the stochastic network maintain a squared error within 10 percent of that of a floating-point implementation for a wide range of noise levels. While the accuracy of stochastic computation may not compare favorably with more conventional binary radix-based computation, the low circuit area, power, and speed characteristics may, in certain situations, make them attractive for VLSI implementation of artificial neural networks.

Index Terms—Pulsed neural networks, stochastic arithmetic, computational elements.

1 Introduction

Stochastic arithmetic principles have been known for many years [1]. The original motivation for considering computation using stochastic arithmetic was the simplicity of the computational elements involved. Pioneers in the areas of machine computation were faced with hardware implementations that were large, power hungry, and relatively unreliable. Stochastic arithmetic held out the possibility of carrying out complex computations with very simple hardware. Modern technology is now capable of fabricating systems with millions of computational elements at extremely low costs. While this might appear to make stochastic processing irrelevant, there is a class of systems which requires virtually unlimited numbers of processing elements and is prepared in exchange to sacrifice accuracy of the individual components. Artificial neural networks (ANNs) are massively parallel systems which can benefit from technology which allows implementation of an unusually large number of simple computational elements on a single integrated circuit. Unlike traditional computation devices such as microprocessors, certain ANN models are characterized by a tolerance for much less accurate individual computations [2]. Stochastic arithmetic may be implemented by computational elements which are both very small and compatible with modern VLSI design and manufacturing technology.

• *The authors are with the Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, Manitoba, Canada R3T 5V6. E-mail: hcard@ee.umanitoba.ca.*

Manuscript received 19 May 2000; revised 29 Nov. 2000; accepted 2 Apr. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 112140.

Stochastic arithmetic provides a number of benefits over other computing techniques:

1. very low computation hardware area,
2. fault tolerance,
3. simple communications over one wire per signal,
4. simple hardware implementations allowing very high clock rates, and
5. capability to trade off computation time and accuracy without hardware changes.

There are some disadvantages, such as the variance inherent in estimating the value of a stochastic signal and the increased number of clock cycles required to accomplish a given computation. The potential of massive parallelism driven by the small circuit areas involved may alleviate some of these disadvantages. Of particular importance is that, unlike binary radix arithmetic, the signal format in stochastic arithmetic is robust in the presence of noise/single bit faults and the hardware complexity of the computational elements is generally low. Also, unlike binary radix arithmetic, accuracy may be controlled using the time dimension. Also retiming (pipelining) may be applied easily in order to maximize system clock rate. Comparisons of stochastic systems with conventional computational hardware and with analog pulse stream systems are provided in Section 4.

A binary stochastic pulse stream is a sequence of binary digits, or bits, in which the information is contained in the primary statistic of the bit stream <X> or the probability of any given bit in the stream being a logic 1. Observation of the information carried on a given signal line is therefore a probabilistic process itself. All of the bits have the same significance (unary coding). Higher precision computation is accomplished by observing the sequence of bits for a longer time in order to allow a lower variance estimate of

the average to be made. Due to the uniform significance of the bits in the sequence, the significance of a single bit error is small and decreases with higher precision computations. Stochastic arithmetic has been applied to the implementation of ANNs by several groups, for example, [3], [4], [5], [6]. In this paper, stochastic pulse streams are used to carry signals coded in two formats, bipolar and unipolar. These formats are essentially the same as those described by Gaines [1] as representation I (unipolar) and representation III (single line bipolar). In the unipolar format, the information carried in a stochastic stream of bits X is

$$X = P(X = 1) = P(X), \tag{1}$$

whereas, in the bipolar format,

$$X = 2P(X = 1) - 1 = 2P(X) - 1. \tag{2}$$

The practical difference is best understood by noting that average arrival rates, or pulse duty cycles of 0, 0.5, and 1 represent, in the unipolar format, the values of 0, 0.5, and 1. In the bipolar format, these same pulse duty cycles represent values of -1, 0, and 1.

We continue in Section 1.1 with a summary of other work on pulsed neural networks. This is followed, in Section 2, by the design and analysis of a number of stochastic computational elements, including circuits for multiplication, squaring, division, addition, and subtraction. Special techniques to improve the response of the division operation are included. We also describe computational elements based on finite state machines configured as generalized stochastic counters, which implement linear and nonlinear activation functions. Discussion of the higher order statistics includes methods for restoring outputs to a Bernoulli sequence. One of these methods is a consequence of stochastic multiplexing of the outputs by the next layer or stage in the network, which therefore comes with no additional cost. In Section 3, we introduce the application in the companion paper. Section 4 compares the stochastic computations of the present paper from a hardware point of view with alternative serial binary implementations, as well as with analog circuits for pulsed neural networks.

1.1 Earlier Work on Pulsed Neural Networks

Artificial neural networks normally employ continuous signals [2], [3] but much of the recent work on these systems has emphasized the use of signals encoded by streams of pulses. A summary of these approaches appears in a recent text [4]. Analog techniques to implement neural networks have varied from methods which attempt to capture most of the biological details of biological neurons [5], [6] through models which incorporate only certain features of their biological counterparts [4], [7], [8], [9] to implementations of pulsed networks which exploit standard analog VLSI circuit techniques [10], [11], [12], [13], [14]. In the latter category, the best known methods, such as those of Murray et al. [10], [11], [12] typically sum the charge from a series of pulses on a capacitor and use the capacitor voltage to generate an output pulse stream using a voltage-controlled oscillator. Analog information may be encoded in the pulse rate of standard pulses or in variable heights and widths of pulses, as in traditional analog communications systems. These methods, and mixed analog-digital variations by other groups, such as [13], [14], produce efficient implementations from the point of view of silicon area and power dissipation. More discussion of these methods and their comparisons with the methods of the present paper are presented in Section 4.

Since the early work of Gaines [1], which was concerned with stochastic arithmetic for general signal processing, a number of groups have employed stochastic arithmetic in pulsed neural networks. These are digital networks, but the value of signals is encoded in the average pulse rate or primary statistic of a stochastic stream. As such it is a unary representation, as distinct from the binary radix representation of conventional digital systems, including those which employ serial signal processing and serial arithmetic. Many methods of incorporating stochastic computation into neural networks have been described recently, including probabilistic random-access memories [15], [16] which are related to Aleksander's classical Wisard machine, and other approaches [17], [18], [19], [20], [21], [22], [23], [24] more closely related to the present paper. The emphasis in most of this work has been on the forward classification or recall operation, though some of these papers do include stochastic implementations of the learning operation. Other reports of learning computations with stochastic arithmetic include [25], [26], [27], [28]. Cellular automata [29], [30] have been employed in the parallel generation of pseudorandom numbers required by stochastic neural networks. These are discussed in later sections. Some of the limitations arising from the variance in the stochastic signals and the potential for doubly stochastic statistics have been presented in [31]. Potential targets for stochastic neural networks are Boltzmann and Helmholtz machines [32], [33].

2 Design and Analysis of Stochastic Processing Elements

This section deals with a number of signal processing elements required in various artificial neural network circuits. The analysis of all of the processing elements are predicated on the assumption that the input signals are Bernoulli sequences. This means that the probability of a given bit being a 1 is independent of the values of any previous bits. While all of the processing elements are analyzed with the assumption that their inputs are Bernoulli sequences, note that the elements' processing functions are evaluated only with respect to their outputs' primary statistic. The outputs are not, in general, Bernoulli sequences. It is also assumed, in the case of a processing element with multiple inputs, that the inputs are uncorrelated with each other. We briefly present circuits for multiplication, squaring, addition, subtraction, and division. Although these circuits are known, for example, through the early work of Gaines [1], their bipolar formats may not be as familiar. We also employ a novel approach to parallel random number generation based upon the use of cellular automata. The simplest cellular automata employ 1D or 2D locally connected arrays of identical processors, each consisting of simple logic gates with a single bit of memory. These arrays have been shown to produce parallel

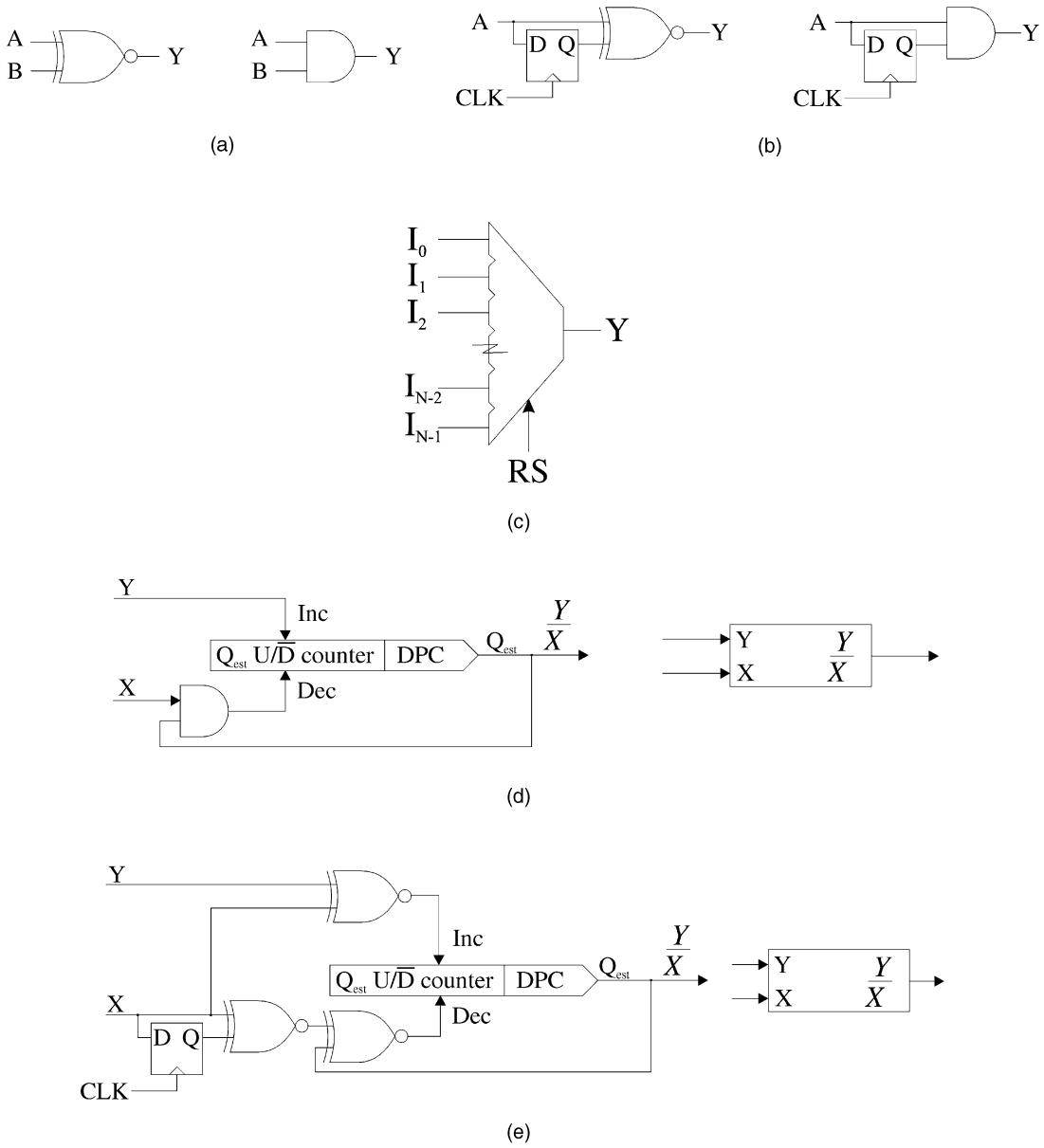


Fig. 1. Stochastic arithmetic circuits. (a) Bipolar and unipolar multipliers. (b) Squaring circuits. (c) Adder. (d) Unipolar divider and its circuit symbol. (e) Bipolar divider and circuit symbol. (f) Digital to stochastic element and circuit symbol. (g) Probability estimator (PDC) and circuit symbol.

pseudorandom numbers with correlations that disappear rapidly with distance from a given site, unlike the correlations in linear feedback shift registers [29]. They also require only local wiring resources and have efficient circuit implementations [30].

2.1 Multiplication

Multiplication of two Bernoulli sequences is actually one of the simplest operations. In the bipolar stochastic pulse stream coding format, an XNOR gate performs a full four quadrant multiplication. In the unipolar stochastic pulse stream coding format, an AND gate performs a one quadrant multiplication. These are shown in Fig. 1a. The AND gate has a I output only when both inputs are 1. The XNOR (equivalence) gate has a 1 output when the two inputs are either both 1 or both 0, the latter case corresponding to -1 x -1. For the bipolar configuration:

$$P(Y) = P(A) \cdot P(B) + \bar{P}(A) \cdot \bar{P}(B). \tag{3}$$

Since $\bar{P} = 1 - P, P(A) = [A+1]/2, P(B) = [B+1]/2$, we have that

$$\begin{aligned} Y &= 2P(Y) - 1 \\ &= \frac{[(A+1)\cdot(B+1) + (1-A)\cdot(1-B)]}{4} \\ &= \frac{(A \cdot B + 1)}{2} \end{aligned} \tag{4}$$

and

$$Y = 2 \cdot P(Y) - 1 = A \cdot B. \tag{5}$$

In this case, the output bit stream Y will be a Bernoulli sequence. In the event that only one of the inputs is not a

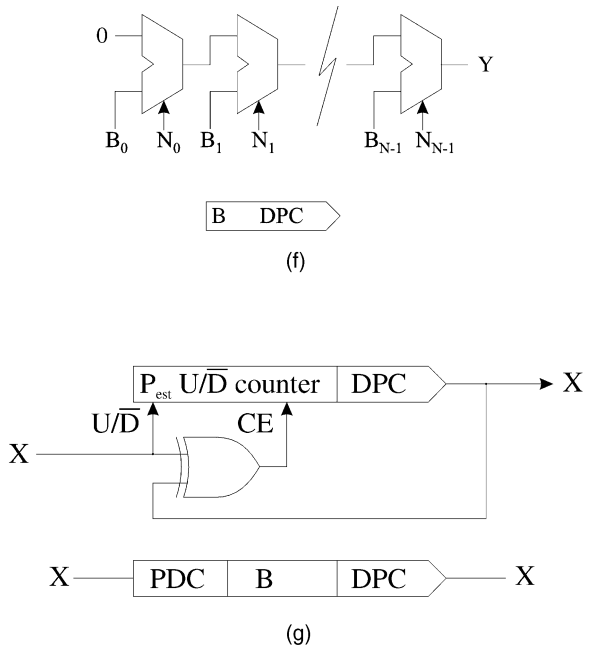


Fig. 1 (continued). Stochastic arithmetic circuits. (a) Bipolar and unipolar multipliers. (b) Squaring circuits. (c) Adder. (d) Unipolar divider and its circuit symbol. (e) Bipolar divider and circuit symbol. (f) Digital to stochastic element and circuit symbol. (g) Probability estimator (PDC) and circuit symbol.

Bernoulli sequence, (5) still holds. The output will not, however, be a Bernoulli sequence.

For the unipolar configuration,

$$Y = P(Y) = P(A) \cdot P(B) = A \cdot B. \quad (6)$$

2.2 Squaring Circuit

Squaring a signal is very closely related to the process of multiplying two signals together. The catch is in the requirement that the two input sequences for a multiplier are uncorrelated. Clearly, attempting to square a signal A by connecting it to both inputs of an XNOR gate will result in an output signal equal to 1 always, while connecting it to both inputs of an AND gate will result in an output signal equal to A. This is because the two inputs to the multiplier are correlated with each other. In the case of an input stream which is a Bernoulli sequence, making a "copy" of the input sequence which is uncorrelated with it is very simple—delay it by one clock cycle. Squaring circuits are shown in Fig. 1b. Note the D-type flip flop which acts as the delay element. In cases where the input sequence is not a Bernoulli sequence, but has an autocorrelation function with a limited width spike, creation of an uncorrelated "copy" may be accomplished by delaying the sequence with a shift register containing one stage for every two bits of width in the autocorrelation spike.

2.3 Addition and Subtraction

Addition and subtraction are very different operations to multiplication and squaring in that they are not closed operations on the interval [-1, 1] for bipolar signals or [0, 1] for unipolar signals. As such, it is not possible to perform them exactly, independent of a scaling operation. Addition with scaling is very simple to perform. A multiplexer which randomly selects a given input i with some probability S_i such that $\sum_i S_i = 1$ will generate an output with a probability which is a scaled sum of the input probabilities. Such a multiplexer is shown in Fig. 1c, where RS refers to the random selection and is supplied by pseudorandom numbers in hardware. We have that

$$P(Y) = \sum S_i P(I_i) \quad (7)$$

and, for bipolar signals,

$$Y = 2P(Y) - 1 = 2\left[\sum_i S_i(I_i + 1)/2\right] - 1 = \sum_i S_i \cdot I_i. \quad (8)$$

If a bipolar input is required to be subtracted, as opposed to added, then it can be multiplied by -1 before it enters the multiplexer (simple logic inversion). Generation of an appropriate random selection is relatively trivial if the number of inputs to the adder is a power of two. If the number of inputs is not a power of two, there are a few design possibilities. One is to increase (pad) the number of inputs by adding in extra input connections to a single 0 signal (random "noise" bit for bipolar 0, logic 0 for unipolar). This option will reduce the output value (decreased scaling constant), leading to sup-optimal performance. The option employed is to generate a random binary variable with the appropriate statistics (P(RS = i) = 1/i) by using a counter with a modulus equal to the number of inputs, driven to either increment or maintain its state each cycle based on a single random bit.

2.4 Computational Elements Based on State Machines

A major objective of this work was to develop stochastic computational elements based on relatively simple state machines. Gaines [1] described the use of an ADDIE (ADaptive Digital Element) for both estimation of the statistics of a stochastic signal and for generation of arbitrary functions. There are similarities between the construction of the function generators presented in this paper and the ADDIE technique reported by Gaines. Both may be viewed as being based on a saturating counter, that is, a counter which will not increment beyond its maximum state value or decrement below its minimum state value. In the ADDIE, however, the state of the counter is controlled in a closed loop fashion. The transition probabilities between the states of the counter are not important. Due to the conversion of the state of the ADDIE into a Bernoulli sequence and its comparison to the input sequence, equilibrium is forced at a counter hyperstate dependent only on the input signal's primary statistic. The use of a closed loop system certainly has advantages, but it does require that the output of the counter is converted into a stochastic pulse stream in order to implement the closed loop feedback, a hardware intensive undertaking. The state machines presented in this section do not use closed loop feedback. While specific output functions may be generated through either deterministic or stochastic functions of the counter state (as in Gaines), no closed loop is present. Certain functions are also generated by changing the

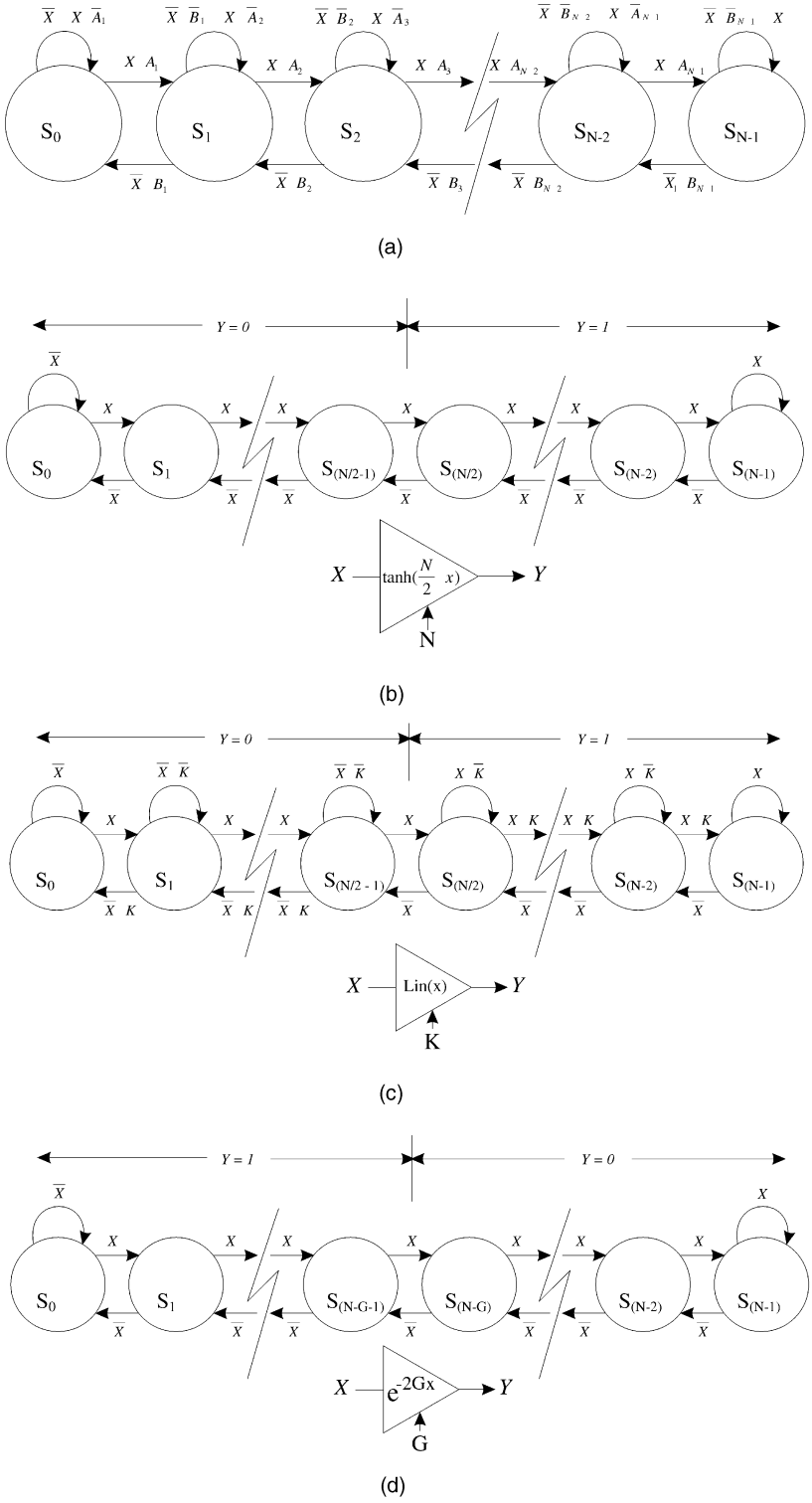


Fig. 2. State machine-based computational elements. (a) Generic linear state machine state transition diagram. (b) Stanh element: state transition diagram and circuit symbol. (c) Linear gain element: state transition diagram and circuit symbol. (d) Exponentiation element: state transition diagram and circuit symbol.

counter from a deterministic state machine (albeit with a stochastic control input) to a stochastic state machine. By making the transition probabilities in the state machine functions of both the control inputs and also a set of stochastic variables, many degrees of freedom are added to the functions generated. The generalized form for the state machine to be analyzed in this thesis is illustrated in Fig. 2a.

The basic form of the state machine is a set of states $S_0 \rightarrow S_{N-1}$ arranged in a linear form (saturating counter). Transitioning from the first state to the last state must occur through a set of transitions through all of the intermediate

states. The state transitions are driven by an input stochastic bit stream X (assumed to be a Bernoulli sequence) and 2N-2 statistically independent control variables, A_n, B_n (also assumed to be Bernoulli sequences). Note that the first and last states have saturating effects: Repeated 0s in State 0 or repeated 1s in State N-1 simply return the system to those states. Given that $A_n, B_n \in R(0,1]$, we know that the system is ergodic and will have one single stable hyperstate. Solving for this stable hyperstate is relatively simple given a few conclusions which may be drawn from the existence of one single stable hyperstate. These conclusions are that the individual state probabilities in the hyperstate must sum to unity, and that the probability of transitioning from state i-1 to state i must equal the probability of transitioning from state i to state i-1.

Using the notation $P_i = P < state = S_i >$ and assuming $A_0 \equiv B_0 \equiv A_N \equiv B_N \equiv 1$, we have

$$P_{i-1} \cdot X \cdot A_i = P_i \cdot \bar{X} \cdot B_i, \tag{9}$$

$$P_i = \frac{X \cdot A_i}{\bar{X} \cdot B_i} \cdot P_{i-1} = \frac{X^i \cdot \prod_{j=0}^{i} A_j}{\bar{X}^i \cdot \prod_{j=0}^{i} B_j} \cdot P_0, \tag{10}$$

and, noting that $\sum_{i=0}^{N-1} P_i = 1$, we find that [28]

$$P_0 = \left[\sum_{i=0}^{N-1} \frac{X^i \cdot \prod_{j=0}^{i} A_j}{\bar{X}^i \cdot \prod_{j=0}^{i} B_j} \right]^{-1}, \tag{11}$$

$$P_i = \left[\frac{X^i \cdot \prod_{j=0}^{i} A_j \cdot \bar{X}^{N-1-i} \cdot \prod_{j=i+1}^{N} B_j}{\sum_{i=0}^{N-1} \left[X^i \cdot \prod_{j=0}^{i} A_j \cdot \bar{X}^{N-1-i} \cdot \prod_{j=i+1}^{Ni} B_j \right]} \right]. \tag{12}$$

Using the probabilities of the individual states P_i, it is possible to synthesize output functions by forming logic functions of the states and, possibly, additional stochastic variables. Due to the mutually exclusive nature of the states, exact sums of their probabilities may be formed through simple use of the logic OR function. It is important to note that one of the strengths of the proposed state machine-based computational elements can also be a significant weakness. Depending on how the output function is formed, there can be significant correlations between the output of the state machine in a given clock cycle and the output in the next clock cycle (i.e., output sequence is not a Bernoulli sequence). While these correlations do not affect the primary statistic of the output, they do affect the variance of estimates made of the primary statistic. There is also the issue of compatibility between the output of the state machine and other computational elements in the event that further processing is required. These effects have not been a problem in our implementations. Reasons for this are explained in Section 2.9.

2.4.1 Stanh Function

A stochastic approximation to the tanh function, Stanh, with both input and output signals coded as bipolar stochastic signals may be implemented using a state machine of the form introduced above, as in Fig. 2b. Note that the generalized state machine of Fig. 2a is specialized by a particular choice of parameters and that the output of the states has also been chosen in a particular way to obtain the tanh characteristic. The associated A_n, B_n are all unity and the output Y is a digital 1 whenever the state machine is in the right half of its possible states (N is even). In this configuration, the approximate transfer function is:

$$S\tanh(N, x) \cong \tanh(xN/2). \tag{13}$$

For small values of N, the approximation to a tanh function is rather poor, but, for large values of N, the approximation is much better. Fig. 3a and Fig. 3b illustrate two examples of the transfer function of the Stanh function and their fidelity with the corresponding tanh function. The results in Fig. 3b for the Stanh of a state machine with 16 states (Stanh 16, x) are considerably closer to the corresponding tanh (8x) than those of Fig. 3a for a system with only four states (Stanh 4, x) are to tanh (2x). Note also that the gain G of the activation function, which is the slope of the characteristic for x = 0, is four times greater in Fig. 3b than in Fig. 3a. The gain is one method of effectively scaling the weights on the incoming signals to the neurons from the [-1, 1] range to the [-G, G] range.

2.4.2 Linear Gain Function

An approximation to a linear gain function (with saturation) with both input and output signals coded as bipolar stochastic signals may also be implemented using a state machine of the form introduced above, as in Fig. 2c. Note that the choice of parameters differs from those chosen in Fig. 2b for the nonlinear tanh characteristic, but the encoding of the output is the same. While multiplication by a factor whose absolute value is less than unity is simple, multiplication by a factor whose absolute value is greater than unity is nontrivial. The specific conditions for this state machine in this case are that

$$A_n = 1 : n \in [0, N/2+1], A_n = K : n \in [N/2+1, N-1],$$

where the total number of states is N, and K is a stochastic control variable with $P_K \in (0,1]$. We also have that $B_n = 1 : n \in [N/2, N], B_n = K : n \in [0, N/2-1]$ and output Y = 1 for all $S_i : i \in [N/2, N-1]$.

In this configuration, the approximate transfer function is:

$$\begin{aligned} &Lin(N,K,x) \cong x \cdot G(N,K) : x \in [-G(N,K)^{-1}, +G(N,K)^{-1}] \\ &Lin(N,K,x) \cong -1 : x \in [-1, -G(N,K)^{-1}] \\ &Lin(N,K,x) \cong +1 : x \in [+G(N,K)^{-1}, +1], \end{aligned} \tag{14}$$

where $G(N,K)$ is the gain, a nonlinear function of the primary statistic of the stochastic control variable K in Fig. 2c. Fig. 4a illustrates a few examples of the variation of $G(N,K)$. The logarithm of $G(N,K)$ is close to a linear function of K. It is also relatively independent of N, the

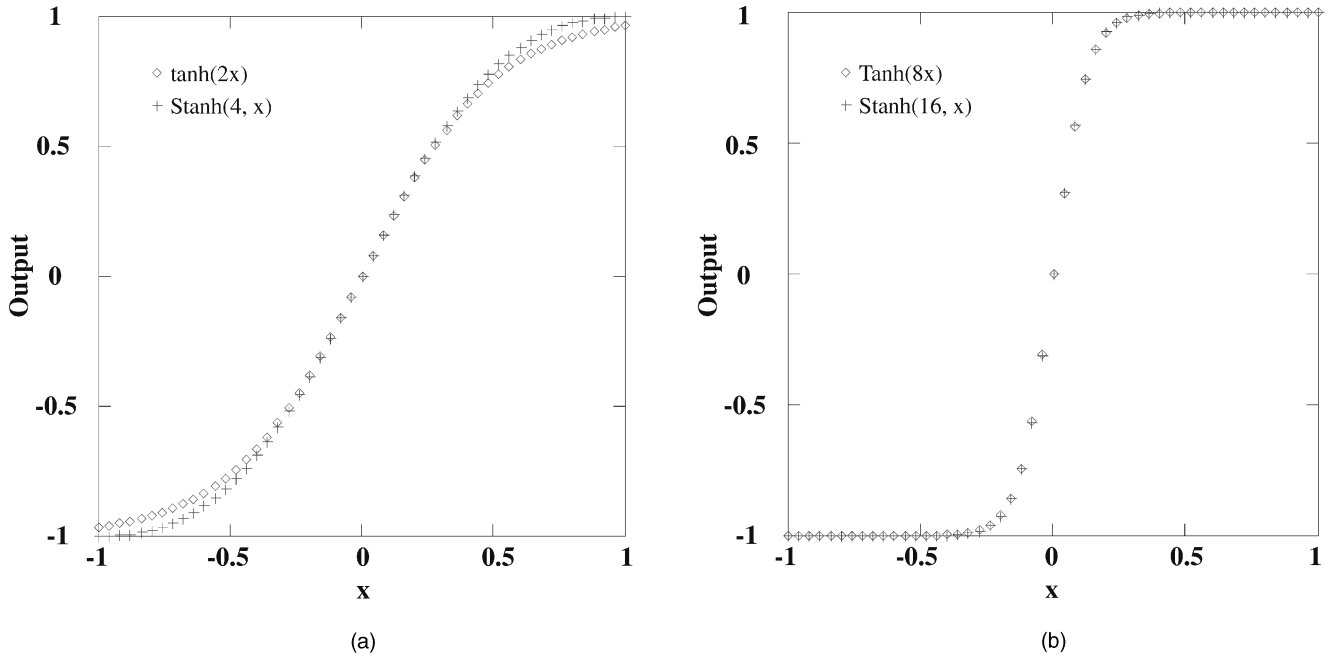


Fig. 3. (a) Stochastic activation function Stanh (4, x) from 4-state machine of Fig. 2b and tanh (2x) for comparison. (b) Stochastic activation function Stanh (16, x) from 16-state machine of Fig. 2b and tanh (8x) for comparison.

number of states in the state machine, except when K is large. As the number of states N in the state machine increases, the linearity in the gain with K in Fig. 4a is simply preserved over a wider range. Some examples of the linear gain function are plotted in Fig. 4b. As the gain $G(N, K)$ increases by increasing K, the number of states N must be increased in order to maintain a reasonably linear transition from Y = -1 to Y = +1. In the limit, when K is unity, the linear gain function becomes the same as the tanh function with the gain factor being dependent on the number of states N.

2.4.3 Exponentiation Function

An approximation to an exponentiation function, with the input coded as a bipolar signal and output signal coded as a unipolar signal, may also be implemented using a state machine of the form introduced above, as in Fig. 2d. Note that the parameters are the same, but the output encoding is opposite to that of the tanh function in Fig. 2b. The range of the function is [0, 1] and, thus, the approximation is valid only for inputs greater than zero. The specific conditions for this state machine are that $A_n = 1, B_n = 1, \forall n$ and the output $Y = 0, \forall S_i : i \in [N - G, N - 1]$ and, for all other S_i, $Y = 1$. The total number of states is N and G is an integer gain parameter. In this configuration, the approximate transfer function is:

$$\begin{aligned} &S\exp(N, G, x) \cong \exp(-2Gx) : x \geq 0 \\ &S\exp(N, G, x) \cong 1 : x < 0. \end{aligned} \tag{15}$$

Fig. 5 illustrates three examples of the Sexp function. For comparison, in two cases, the number of states is the same (32) with different gain (2 vs. 8) and, in two cases, the gain is the same (8) but with different numbers of states (32 vs. 64). Note that, not surprisingly, the approximation is poorest around $x = 0$. As the second and third curve illustrate, it is important that the number of states N be significantly greater than the gain parameter G. In fact, $S\exp(N, N/4, 0) = 0.75$ (ideal would be 1). As N is made larger, the approximation improves. Dropping N to $2G$ would degrade the function to being a specific case of the Stanh function, although one must account for the difference in output signal interpretation (unipolar vs. bipolar).

2.5 Division

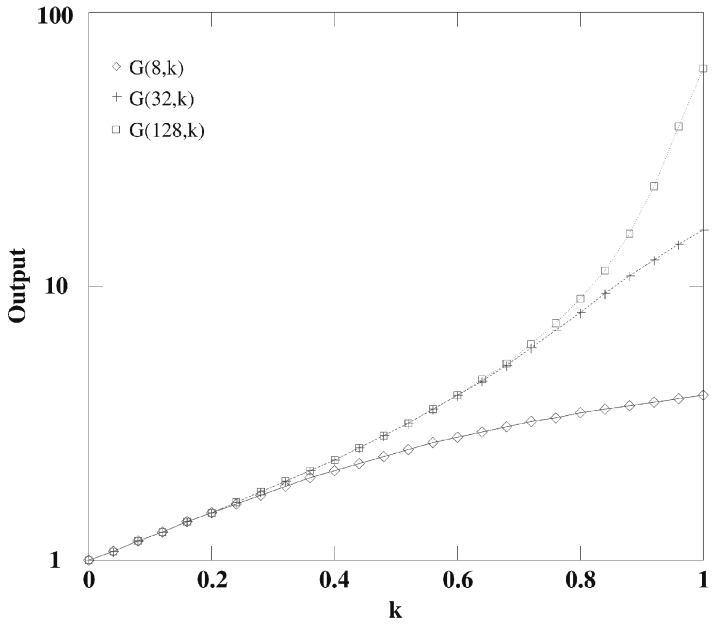
Division may be accomplished in only an approximate form in the stochastic number representation schemes presented. The only technique available is gradient descent using a saturating counter as an integrator. The reason for the technique only being approximate is that, when the estimated quotient is at the limits for the counter state, the probability of a counter increment cannot be balanced with the probability of a counter decrement. A full analysis of the basic principles of stochastic division may be found in [1].

2.5.1 Unipolar Division

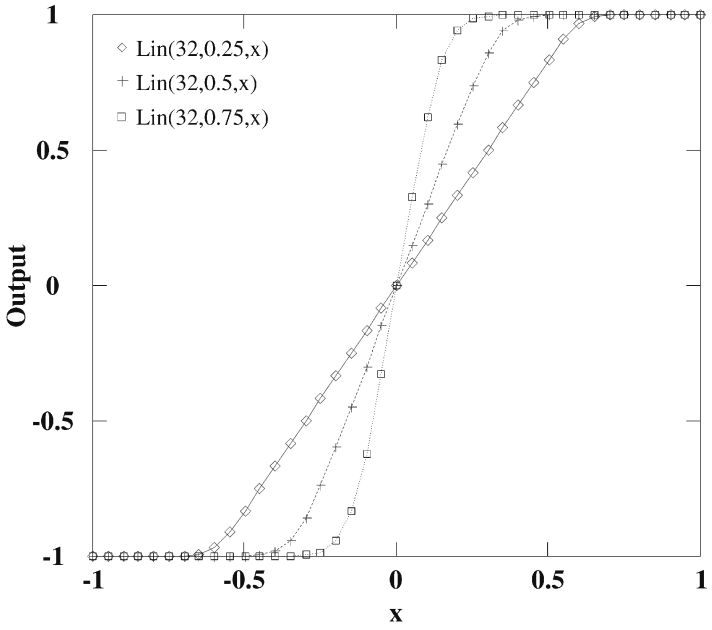
Unipolar division is performed by performing gradient descent on a error function E. Recall that for unipolar signals $X = P(X = 1), Y = P(Y = 1), Q = P(Q = 1)$.

$$\begin{aligned} &E = X \cdot Q - Y \\ &\frac{\partial E^2}{\partial t} = 2 \cdot E \cdot X \cdot \dot{Q} \\ &\dot{Q} = -\alpha \cdot E = -\alpha(X \cdot Q - Y) \\ &\frac{\partial E^2}{\partial t} = -2 \cdot \alpha \cdot E^2 \cdot X < 0, \end{aligned} \tag{16}$$

where α is a positive parameter related to the rate at which the counter state (estimated quotient) changes. Given that



(a)



(b)

Fig. 4. (a) Gain G (N, k) of linear activation function vs. control probability K. (b) Examples of linear gain activation functions Lin (n, K, x) for 32-state machines of Fig. 2c with K = 0.25, 0.5, and 0.75.

E^2 is always positive and that it is bounded below by 0, (16) indicates that E must be driven to zero. Implementation of (16) may be accomplished by incrementing the counter when Y is a one and decrementing the counter when both X, Q are ones. Fig. 1d illustrates the unipolar division component and its symbol. Note the increment and decrement inputs, including the feedback connection of the current estimate for Q to the decrement circuit.

2.5.2 Bipolar Division

Bipolar division is also obtained by performing gradient descent on a error function E.

$$\begin{aligned} E &= X \cdot Q - Y \\ \frac{\partial E^2}{\partial t} &= 2 \cdot E \cdot X \cdot \dot{Q} \\ \dot{Q} &= -\alpha \cdot X \cdot E = -\alpha(X^2 \cdot Q - X \cdot Y) \\ \frac{\partial E^2}{\partial t} &= -2 \cdot \alpha \cdot E^2 \cdot X^2 0. \end{aligned} \tag{17}$$

Implementation of (17) may be observed in the circuit diagram Fig. 1e. This resembles that of Fig. 1d except for the added complexity in the increment and decrement circuits.

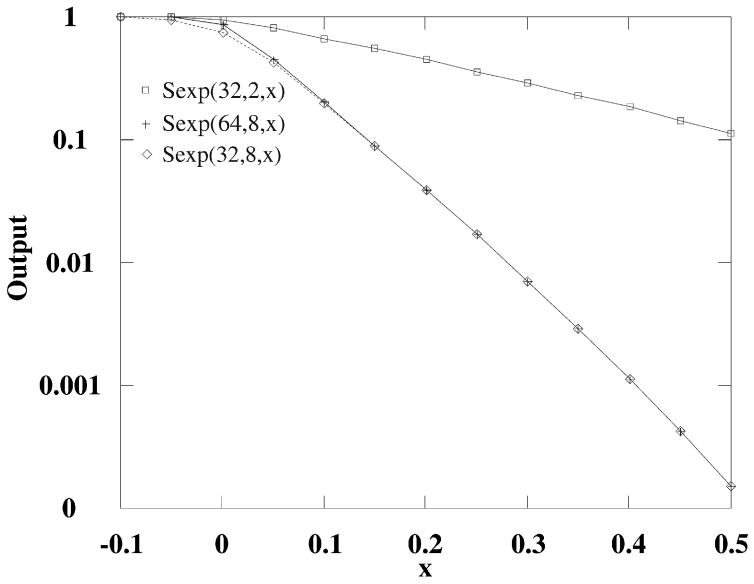


Fig. 5. Examples of exponentiation function Sexp (n, G, x) for 32 and 64-state machines of Fig. 2d and gains G = 2 and G = 8.

2.5.3 Increased Speed of Division Using Stepped Velocity

Computation of a quotient can be time consuming, particularly if the numerator and denominator involved are very small. In this case, many clock cycles can occur in which the quotient estimate is neither incremented or decremented because there is effectively no information available to indicate a discrepancy between the required quotient and the estimated quotient. In these cases, the "velocity" of convergence may be stepped through a decreasing sequence of values, starting with a counter step size of $N/2$ (MSB of the counter) and then decreasing this by a factor of two at each step. The number of clock cycles is successively increased by a factor of two at each step, allowing the counter to traverse the same "amount of ground" at each step of the process. Fig. 6 illustrates the difference in the results when a signal with a probability of 0.2 is divided by the sum of a set of six signals whose probability adds up to 0.8. The curve marked "output with stepped velocity" converges to the target output much more rapidly than the standard output, which does not employ this procedure.

2.5.4 Increased Speed of Division Using Scaled Processing Time

As previously mentioned, computation of a quotient where the numerator and denominator are very small values can be very slow. This creates a problem where the dynamic range of the numerator and denominator is large. Rather than always allowing the quotient to be computed for a fixed time, which is sufficient to achieve acceptable precision with the smallest numerator/denominator which will be encountered, it is much more efficient to scale the processing time as required. This allows the system to spend, on average, much less time calculating quotients. This optimization may or may not be practical in the context of the system in which the computation is taking place. If a number of dividers are operating in parallel with different operands, the potential time savings may be very low. If, however, there is only a single divider in the system, or if all of the dividers in a given layer operate simultaneously with similar operands, the technique can greatly speed computation.

2.6 Digital to Stochastic Converter

Fig. 1f illustrates a simple circuit for implementing a binary digital to probability (stochastic) conversion, together with its circuit symbol. The technique essentially uses a chain of weighted adders implemented with 2-input multiplexers. Each multiplexer select line must be driven by a single pseudorandom number or noise bit (the inputs $N_0 \rightarrow N_{N-1}$ in the figure) with a probability of 1/2. Each adder in the chain generates an output probability that is one half of the sum of the probability from the next earlier stage in the chain and a bit from the digital word to be converted. The result is similar to that of an R-2R ladder digital to analog converter. The transfer function, with a zero terminated chain, is:

$$P(Y=1) = \frac{1}{2}\cdot(B_{N-1} + \frac{1}{2}\cdot(B_{N-2} + ... \frac{1}{2}\cdot(B_0 + 0)..) = \frac{B}{2^N}, \tag{18}$$

where N is the number of bits in B (and stages in the chain).

This circuit benefits from simplicity of structure and, due to the nature of the signals involved, may be pipelined fairly easily in order to maintain very high processing speeds. In the event that the digital value B is a constant, the circuit may be pipelined by adding a flip flop at the output of each multiplexer (or every second one, etc.). In the event that B changes, proper statistics will require extra clock cycles to propagate to the output, based on the number of flip flops in the circuit. Given the number of clock cycles that computations are based on, this delay may not be a significant factor. If the temporary shift in the output statistics cannot be tolerated, an accurate pipelining technique may be used where the bits of B are pipelined as

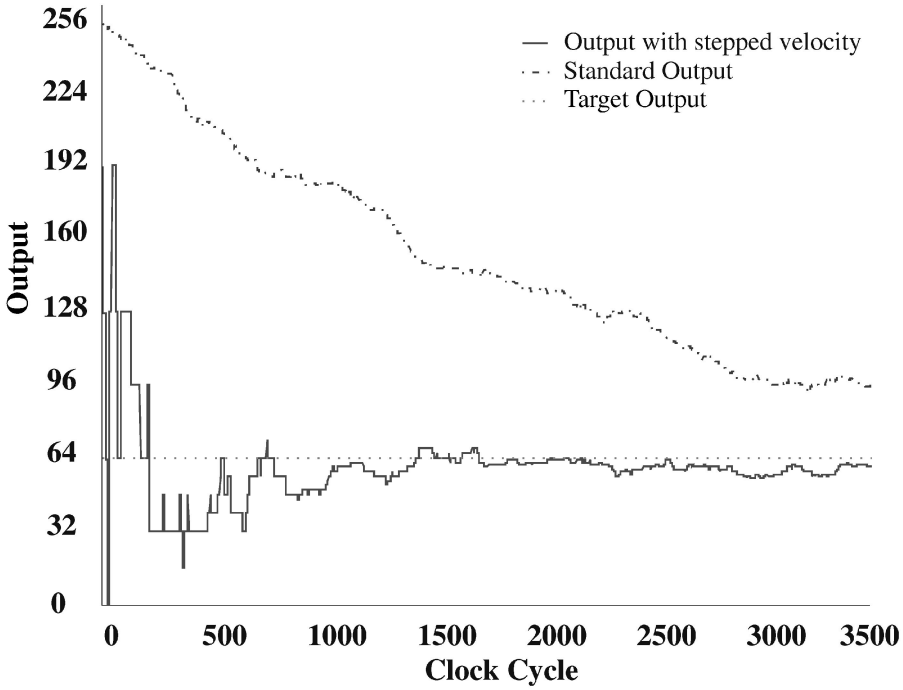


Fig. 6. Comparison of unipolar divider with and without stepped velocity procedure.

well. Doing this requires flip flops to be added in series with each bit of B in order to maintain the same number of flip flops in the path from the bit to the output as for all other bits. Note that the random "noise" bits need not be pipelined.

2.7 Probability to Digital Converter

We really cannot directly convert a probability to a binary digital number, but we can generate an estimate for the probability. In Gaines [1], the ADDIE was presented as an optimal estimator for the primary statistic of a signal when the input distribution is unconstrained. Fig. 1g illustrates an ADDIE configured as a probability estimator, together with its circuit symbol. Use of this circuit in the context of unipolar or bipolar signals involves both differences in the format of the binary coded state of the ADDIE and in the connections to the saturating counter (integrator) inside the ADDIE which generates the estimate. When estimating unipolar coded signal values, the state of the integrator is interpreted as an unsigned number. When estimating bipolar coded signal values, the state of the integrator is interpreted as an offset binary number. Note that the CE signal is a count enable and the U/D signal is an up/down control signal for the saturating counter. The X' output from the circuit is the stochastic output from the current integrator state. It provides a "rerandomized" output bit stream with the same primary statistic (signal value) as the input bit stream X. Its use is desirable when the X input is not a Bernoulli sequence and further processing with the signal is desired.

2.8 Generation of Stochastic Sequences

A central element of a digital stochastic processing system is a source of stochastic bit streams (pseudorandom noise). Many of the processing elements presented require digital noise bits where the probability of each bit being a one is fixed at 1/2. Gaines [1] describes a number of potential sources of digital noise. While noise sources based on quantum mechanical effects may provide noise which meets all of the statistical requirements, generation of noise using these techniques is not amenable to standard VLSI processes.

An alternative source of pseudorandom digital noise, the linear feedback shift register (LFSR), described by Gaines and by many others, may be constructed using standard digital components. The LFSR, however, generates stochastic sequences which have an autocorrelation function which is a set of repeating delta functions with each delta function separated by the number of bits in the LFSR's sequence length. While this is near ideal, there are a few properties of the LFSR which are suboptimal for use in a stochastic processing system. The bits in the state of the LFSR are all correlated with each other with small shifts (one for nearest neighbor bits in the shift register). While the use of XOR combinations of various bits in the LFSR state may be used to generate bit sequences with any desired shift, use of this technique would require either the centralization of a noise generator or the distribution of multiple LFSR state bits to each stochastic processing element that requires the noise. The routing overhead of such techniques could be significant. Distributing the flip flops of the LFSR among the stochastic processing elements would only partially solve the problem. The LFSR requires feedback from the last element in the shift register back to the first element in order to generate a maximal length sequence. Routing this feedback path, along with multiple register neighbor bits, to the physical location where a noise bit is required would take a significant toll on routing resources.

A more appropriate pseudorandom signal generator, based upon the use of cellular automata (CA), has been

described earlier by our group [29]. One of the desirable properties of this noise generator is that the bit sequences from adjacent bits in the state of the CA are not, in general, shifted from each other by only a single clock. This property eliminates the requirement to use sets of the register bits to form a single noise bit. The CA described in [29] is also characterized by the presence of only localized (nearest-neighbor) feedback. These characteristics, taken together, make it possible that the individual flip flops that make up the CA register could be placed in a circuit wherever a noise bit is needed. The various flip flops could then be all tied together after placement in a nearest-neighbor fashion in order to minimize the burden of routing the nearest-neighbor feedback connections. This process is exactly what is commonly done in current full scan test design practices [34]. After the CA register bits are tied together and their total number is known, an appropriate CA rule is configured into the logic surrounding each CA flip flop in order to generate a maximal length pseudorandom sequence. In this paper, a single 32-bit CA based on [29] is used to generate all of the digital noise bits used by the stochastic processing elements.

2.9 Effects of Processing Elements on Higher Order Statistics

The state machine-based processing elements presented above have outputs which, as mentioned earlier, are not in general Bernoulli sequences. While the outputs may not be Bernoulli sequences, they may still be processed by some functions. The multiplier and adder, in particular, do not require all of their inputs to be Bernoulli sequences in order for them to function properly. Ultimately, however, the correlations present in the state machine output sequences will lead to increased variance in any estimates made of the outputs' primary statistic. Note that, while estimates of the primary statistic will have higher variance, the variance of the outputs' higher order statistics will actually be reduced. One approach to reducing the correlations would be to resort to the technique given by Gaines [1], where the output of the state machine is turned into a Bernoulli sequence. As mentioned earlier, this is a hardware intensive approach.

The other approach is to use characteristics of the processing typically performed after a state machine-based computational element to reduce the effects of the correlations. A typical artificial neural network is formed of layers of neurons. Each neuron performs a function on its total input. Typical functions for a neuron activation function are a tanh or linear gain function, functions which may be implemented with state machine-based computational elements. Between the layers of neurons are synapses which form the total input for the next layer of neurons. Each neuron generates a weighted sum of the outputs from its input neurons by using multipliers and a single adder. If this processing of the activations of the outputs of a layer of neurons sufficiently reduces the effects of the neuron output correlations, then it may be justifiable to totally eliminate the conversion of the neuron output to a Bernoulli sequence. Consider a set of neurons with outputs that are not true Bernoulli processors, but instead have correlation effects or memory length in proportion to the number of states in the stochastic counters. If the outputs of these neurons are sampled by a stochastic multiplexer in the next layer of the network, and if the number of inputs to this multiplexer or fan-in exceeds the memory length, then the inputs to the next stage have been rerandomized by the multiplexing process and are again Bernoulli sequences.

2.9.1 Autocorrelation Function of a Bernoulli Sequence

A Bernoulli sequence has, by definition, a sequence of bits whose probability of being a one is independent of any other bit in the sequence. This makes the definition of the autocorrelation function of a Bernoulli sequence very simple. Given that the probability of any given bit in the sequence is a one is P and $\bar{P} = 1 - P\text{th}$ autocorrelation function is

$$\begin{aligned} \Gamma(k) &= 1 : k = 0 \\ \Gamma(k) &= 1 - 2 \cdot P \cdot \bar{P} : k \neq 0. \end{aligned} \tag{19}$$

An experimental example, generated using a DPC to generate the bit sequence and plotting the autocorrelation function $\Gamma(k)$ over a sequence of 16K bits is shown in Fig. 7a. The generating probability for the sequence is $1/2$ and, therefore, the expected value of the autocorrelation function is $1/2$ for $k \neq 0$.

2.9.2 Effects of Addition on State Machine Output Sequences

Fig. 7b illustrates both the autocorrelation function of a the stochastic $S\exp$ function (N = 64, G = 16, X = 0.0156) and of the output of a scaled stochastic summer which has eight inputs being fed by eight similar $S\exp$ functions. Note that the spike in the autocorrelation of the summer output is one clock wide. The x input of all of the $S\exp$ functions have been chosen to drive the probability of its output to be $1/2$ (maximal variance). As the upper plot in Fig. 7b indicates, the autocorrelation function of the $S\exp$ output has a very broad spike centered at $k = 0$. Lower gain leads to a much narrower spike. The plot also indicates that, to a very large extent, the generation of the sum of a similar set of $S\exp$ function outputs does mask much of the correlation at the inputs. In this case, it appears that the fact that the multiplexer which "throws away" seven out of every eight bits in each input stream is discarding many bits that are correlated with each bit taken. Some research has been aimed at ways to avoid this potential loss of information [23], but, in the present situation, the loss of correlated bits is actually desirable as discussed at the end of Section 2.9. Results for the other state machine based computational elements were qualitatively similar.

3 Stochastic Arithmetic in Soft Competitive Learning Application

In order to demonstrate the applicability of the computational elements presented, a sample ANN problem was implemented in both conventional (deterministic, floating-point) and stochastic arithmetic. The intention is to show that stochastic arithmetic is a viable technique for ANN implementation, with results comparable to that for a conventionally implemented solution. The example

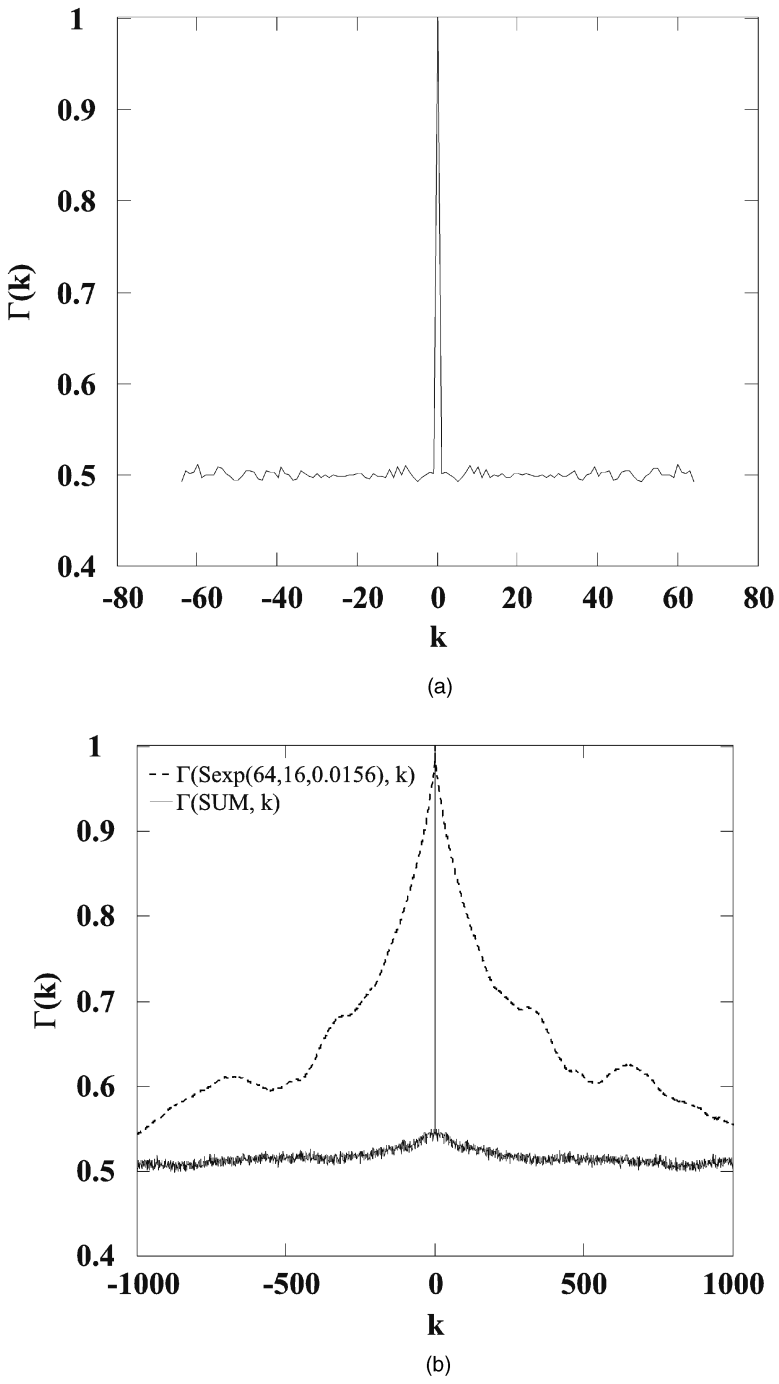


Fig. 7. (a) Autocorrelation function of a Bernoulli sequence. (b) Autocorrelation of Sexp function (Fig. 2d and Fig. 5) (upper curve) and of 8-input stochastic multiplexer or summer (lower curve).

problem considered is the optical recognition of noisy characters from a limited character set. The details of our implementation are deferred to the companion paper [35]. The ANN employed is implemented in two layers. The first layer is an unsupervised soft competitive learning network. The second layer is a simple set of linear neurons which classify the features detected by the first layer into the corresponding characters using supervised learning (delta rule). C++ programs were written for the floating-point implementation, then extended to implement the ANN, using a clock cycle-by-cycle behavioral simulation of a stochastic arithmetic solution designed to implement the same functionality. Implementation of the computations employed the stochastic computational elements discussed above, including the increased "velocity" technique for division. The generalization performance of the networks was evaluated by observing the effects of noisy input patterns on the network MSE. If a network is very good at recognizing corrupted input patterns, its MSE will not be much higher than for a set of clean input patterns. MSE verses noise probability was measured by presenting one hundred sets of input characters for each noise probability.

As we show in the companion paper [35], the learning performance of the stochastic SCL layer trained to within 10 percent of the final squared error of the double precision floating-point implementation. Through the use of a feedback technique to reduce the dynamic range of the neuron activations, the clock cycle count for training the stochastic SCL layer was reduced. The learning performance of the stochastic network as a whole, evaluated through squared error on all of the characters in the font, was within a factor of 1.5 of the conventional implementation. Comparisons of the networks' generalization capabilities indicated very close agreement between the stochastic and the conventional implementations. Computational clock cycle counts were quite reasonable, indicating computational performance more than an order of magnitude greater for the stochastic computation than the conventional radix arithmetic solution, assuming clock frequency parity (an overly conservative assumption).

4 Hardware Comparison with Other Implementations

The serial communication of data in a binary radix integer (BRI) representation is more rapid than unary stochastic representations. Data with a resolution of M bits requires a communication time M for BRI, but a time 2^M for the unary stochastic case. On the other hand, assuming the operation of a simple logic gate requires unity area and time, the area and time required for a multiply-accumulate operation by a single synapse using serial BRI arithmetic are βM and αM^2, respectively, where the parameters β, α are implementation dependent. The parameter $\beta > 1$ is a consequence of the overhead circuitry to implement an M-bit serial binary multiplier and $\alpha < 1$ arises from the carry operation in the accumulate step (which increases the duration of a clock cycle).

In assessing the area required for the hardware implementation of neural networks, it is common practice to consider only the area of the synapses. This is because, for a network with $O(N)$ neurons, there are typically $O(N^2)$ synapses. However, if the neurons have $O(N)$ area themselves, and the synapses are of $O(1)$ area, we cannot neglect their contribution entirely. The neurons in the stochastic systems of this paper are implemented by stochastic counters, which are registers whose length (number of states) may, in some cases, be proportional to the fan-in m and, therefore, be $O(m)$ in area. The neurons also include stochastic multiplexers which, for fan-in m, consist of a $\log_2 m$ to m decoder, driven by $\log_2 m$ pseudorandom number bits. The outputs from this decoder control m transistor switches or transmission gates connecting to the m inputs to the neuron. Since $O(m)$ is at most $O(N)$, the overhead area from the neurons may be regarded as expanding, by at most a small constant factor, the $O(1)$ area of the synapses. Finally, the generation of the parallel pseudorandom numbers using cellular automata [29], [30] contributes a small constant factor overhead at the neuron level, which is amortized over the $O(m)$ synapses per neuron.

The area and time per synapse for the stochastic case are $1 + \gamma$ and 2^M, respectively, since this operation is performed by a simple logic gate of area unity over 2^M regular clock cycles. The factor γ accounts for the incremental overhead of the pseudorandom number generation, which is small if this is implemented using cellular automata. In the stochastic case, we are assuming that the uncorrelated pulse streams representing both the multiplier and the multiplicand (the input signal and its weight) are available as (off-chip) inputs to the multiplier. This is reasonable in a fully stochastic neural system since the inputs and outputs of all chips in the system will be stochastic signals, except, perhaps, the first chip where one converts from binary deterministic information from an independent nonstochastic system, such as a conventional computer.

The area-time product is often used as an (inverse) performance metric in VLSI systems. It is a measure of the energy of a computation if one assumes a constant power dissipation per unit area. Comparing stochastic and BRI systems, we have that

$$\frac{(AT)_S}{(AT)_{BRI}} = \frac{(1+\gamma)\cdot 2^M}{\alpha \cdot \beta \cdot M^3}. \tag{20}$$

This is equal to unity when

$$M = \log_2\left(\frac{\alpha \cdot \beta}{1+\gamma}\right) + 3\log_2 M, \tag{21}$$

which corresponds to $M \approx 10$ bits if we assume a typical value of unity for $\alpha\beta/(1+\gamma)$. The first term is only weakly dependent upon the details of the implementation. In other words, for $M < 10$ bits, the stochastic system has better performance, as measured by the AT product, than the BRI system. For $M > 10$ bits, the BRI system does better.

The other important consideration is the relative error tolerance of the BRI and stochastic representations. If we consider the worst-case situation for a single bit error, the maximum error in the BRI case is $2^{M-1}/2^M = 1/2$ of the full-range value (the same result for signed integers). The maximum error in the stochastic case is only $1/2^M$ of the full-range value. The stochastic systems therefore have much better error tolerance than BRI systems. However, the stochastic systems experience an inherent statistical variation in counting pulses, given by the binomial distribution

$$p(k) = C(n,k)p^k(1-p)^{n-k}, \tag{22}$$

where $C(n,k)$ are the binomial coefficients, p is the Bernoulli probability (an abbreviation of the $P(X)$ of the previous sections) which represents the signal magnitude, and the pulse count takes place over n clock cycles. The mean of this distribution is $\mu = np$ and the variance is $\sigma^2 = np(1-p)$, so the coefficient of variation $CV = \sigma/\mu = \{(1-p)/np\}^{1/2}$. The coefficient of variation is a measure of the precision, so, for a fixed precision, we have a required processing time (clock cycles) given by

$$n = \frac{1-p}{p}\cdot\frac{1}{(CV)^2}. \tag{23}$$

The number of clock cycles required is greater for smaller Bernoulli probabilities. For a minimum $p = 0.1$ and a desired precision of $CV = 0.1$ or 10 percent uncertainty, $n = 900$ clock cycles. Since $n = 2^M$, this also corresponds to $M \approx 10$ bits. For greater values of Bernoulli probability, the required integration times are much shorter. For example, for $p = 0.5$, $n = 100$ and, for $p = 0.9$, $n = 11$ clock cycles are required for this same precision of $CV = 0.1$.

It is more complicated to compare the unary digital stochastic systems of this paper with analog pulse stream implementations. Part of the reason for this is that there are so many different ways of implementing pulsed neural networks in analog form [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. The information can be encoded in the pulse frequency, the pulse height, the pulse width, or any combination of these [11], [12]. We can, however, make certain observations. In the analog case, we expect the signal processing times to be comparable with the stochastic unary representations of this paper. The areas for multiplication will be larger in the analog case because these usually require a multiplying D/A converter (since, typically, the weights in this case are represented in binary form), rather than a simple logic gate, as in the stochastic digital case. Addition is, however, easier in the analog case since one can simply employ Kirchoff's current law.

Another consideration is that new pulse streams must be regenerated in analog circuits after each computation step or network layer. This happens automatically if we use the stochastic counters of this paper to perform the nonlinear activation functions of the neurons. In the analog case, this is most often accomplished using voltage-controlled oscillators. In summary, the relative efficiency of the digital stochastic systems of this paper and that of analog pulse stream implementations will be dependent upon the application, the required bit precision of the learning and recall operations, the acceptable silicon area and power dissipation, and the noise tolerance.

5 Discussion and Further Work

The penalties paid for using stochastic arithmetic, which are increased computation time and the variance in estimating the pulse rates of stochastic streams, were observed to be alleviated by other factors. The massive parallelism in the ANN, supported by the low circuit areas of stochastic synaptic computational elements, compensates for the large number of clock cycles involved in any given single computation. The variance involved in estimating the primary statistic of a stochastic signal was not a significant problem, at least in the context of our chosen application [35]. There are several opportunities for further work on this topic. While simulations of the stochastic circuits did, in most ways, provide accurate models of a realistic hardware implementation, there are a few deviations. The most notable is in the generation of pseudorandom numbers. The stochastic simulations all used a single 32-bit random number generator based on cellular automata (CA). While this is practical to implement in digital hardware, an actual hardware implementation of a stochastic network requires a larger set of random bits in each clock cycle. Simulations reconciled this issue by cycling the CA multiple times per clock cycle of stochastic processing.

The topic of generation of random selections over ranges that are not powers of two (for weighted stochastic summation) is an area that deserves further study. The demonstrated ability of the stochastic summer to reduce unwanted correlations generated by the state machine-based computational elements was shown with a random select. Correlations in the random selects generated over ranges that are not powers of two could have a deleterious effect on this ability. Another significant area for future work addresses the generation of new stochastic computational elements. While the current generic state machine was analyzed for the statistics of its hyperstate, a general procedure for choosing the appropriate stochastic parameters and output generation function has not been developed. A general procedure for designing the computational elements would be very useful for designing stochastic processing systems with enhanced capabilities. Finally, the new stochastic computational elements developed here would benefit from further analysis and hardware comparisons with alternatives to evaluate their performance in terms of circuit area, power consumption, and accuracy.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through an Operating Grant (HCC) and a postgraduate scholarship (BDB). The equipment support of the Canadian Microelectronics Corporation (CMC) is also very much appreciated. Helpful discussions concerning stochastic arithmetic and cellular automata-based pseudorandom number generation with Peter Hortensius, Bob McLeod, Jeff Dickson, and Greg Bridges also contributed to this paper.

References

[1] B.R. Gaines, "Stochastic Computing Systems," *Advances in Information Systems Science,* J.F. Tou, ed., vol. 2, chapter 2, pp. 37-172, New York: Plenum, 1969.

[2] J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation.* Redwood City, Calif.: Addison-Wesley, 1991.

[3] C.M. Bishop, *Neural Networks for Pattern Recognition.* Oxford: Clarendon Press, 1995.

[4] *Pulsed Neural Networks,* W. Maass and C.M. Bishop, eds. Cambridge, Mass.: MIT Press, 1999.

[5] M.A. Mahowald, "Evolving Analog VLSI Neurons," *Single Neuron Computation,* T. McKenna, J. Davis, S. Zornetzer, eds., pp. 413-435, San Diego, Calif.: Academic Press, 1992.

[6] S.R. Deiss, R.J. Douglas, and A.M. Whatley, "A Pulse Coded Communications Infrastructure for Neuromorphic Systems," *Pulsed Neural Networks,* W. Maass and C.M. Bishop, eds., chapter 6, Cambridge, Mass.: MIT Press, 1999.

[7] J.G. Elias, "Artificial Dendritic Trees," *Neural Computation,* vol. 5, pp. 648-663, 1993.

[8] W. Maass, "Fast Sigmoidal Networks via Spiking Neurons," *Neural Computation,* vol. 9, pp. 279-304, 1997.

[9] J. Meador, A. Wu, C. Cole, N. Nintunze, and P. Chintrakulchai, "Programmable Impulse Neural Circuits," *IEEE Trans. Neural Networks,* vol. 2, pp. 101-109, Jan. 1991.

[10] A.F. Murray and A.V.W. Smith, "Asynchronous VLSI Neural Networks Using Pulse Stream Arithmetic," *IEEE J. Solid State Circuits,* vol. 23, pp. 688-697, 1988.

[11] A.F. Murray, D. Del Corso, and L. Tarrassenko, "Pulse Stream VLSI Neural Networks Mixing Analog and Digital Techniques," *IEEE Trans. Neural Networks,* vol. 2, pp. 193-204, 1991.

[12] A.F. Murray, "Pulse Based Computation in VLSI Neural Networks," *Pulsed Neural Networks,* W. Maass and C.M. Bishop, eds., chapter 3, Cambridge, Mass.: MIT Press, 1999.

[13] D. Del Corso, F. Gregoretti, and L.M. Reyneri, "Mixed Analog-Digital Basic Cells for Artificial Neural Systems Using Pulse Rate and Width Modulations," *Parallel Architectures and Neural Networks,* pp. 279-287, Apr. 1988.

[14] D. Del Corso, E. Filippi, F. Gregoretti, C. Pellegrini, L.M. Reyneri, and M. Sartori, "An Artificial Neural System Based on Pulse Stream Neural Chips," *Parallel Architectures and Neural Networks,* pp. 164-171, Apr. 1991.

[15] T.G. Clarkson, D. Gorse, J.G. Taylor, and C.K. Ng, "Learning Probabilistic RAM Nets Using VLSI Structures," *IEEE Trans. Computers,* vol. 41, pp. 1552-1561, 1992.

[16] T.G. Clarkson, Y. Guan, J.G. Taylor, and D. Gorse, "Generalization in Probabilistic RAM Nets," *IEEE Trans. Neural Networks,* vol. 4, pp. 1552-1561, 1993.

[17] J.E. Tomberg and K. Kaski, "Pulse Density Modulation Technique in VLSI Implementation of Neural Network Algorithms," *IEEE J. Solid-State Circuits,* vol. 25, pp. 1277-1286, 1990.

[18] M.S. Tomlinson, D.J. Walker, and M.A. Sivilotti, "A Digital Neural Network Architecture for VLSI," *Proc. Int'l Joint Conf. Neural Networks,* vol. 2, pp. 545-550, 1990.

[19] D.E. Van den Bout and T.K. Miller III, "A Digital Architecture Employing Stochasticism for the Simulation of Hopfield Neural Nets," *IEEE Trans. Circuits and Systems,* vol. 36, pp. 732-738, 1989.

[20] M.S. Melton, T. Phan, D.S. Reeves, and D.E. van der Bout, "The TinMANN VLSI Chip," *IEEE Trans. Neural Networks,* vol. 3, pp. 375-384, May 1992.

[21] A. Torralba, F. Colodro, E. Ibanez, and L.G. Franquelo, "Two Digital Circuits for a Fully Parallel Stochastic Hopfield Neural Network," *IEEE Trans. Neural Networks,* vol. 6, pp. 1264-1268, Sept. 1995.

[22] Y. Kondo and Y. Sawada, "Functional Abilities of a Stochastic Logic Neural Network," *IEEE Trans. Neural Networks,* vol. 3, pp. 434-443, May 1992.

[23] C.L. Janer, J.M. Quero, J.G. Ortega, and L.G. Franquelo, "Fully Parallel Stochastic Computation Architecture" *IEEE Trans. Signal Processing,* vol. 44, pp. 2110-2117, Aug. 1996.

[24] P.S. Burge, M.R. van Daalen, B.J.P. Rising, and J.S. Shawe-Taylor, "Stochastic Bit-Stream Neural Networks," *Pulsed Neural Networks,* W. Maass and C.M. Bishop, eds., Cambridge, Mass.: MIT Press, 1999.

[25] J.A. Dickson, R.D. McLeod, and H.C. Card, "Stochastic Arithmetic Implementations of Neural Networks with In Situ Learning," *Proc. Int'l Conf. Neural Networks,* pp. 711-716, 1993.

[26] J. Zhao, J. Shawe-Taylor, and M. van Daalen, "Learning in Stochastic Bit-Stream Neural Networks," *Neural Networks,* vol. 9, pp. 991-998, 1996.

[27] Y.C. Kim and M.A. Shanblatt, "Architecture and Statistical Model of a Pulse-Mode Digital Multilayer Neural Network," *IEEE Trans. Neural Networks,* vol. 6, pp. 1109-1118, 1995.

[28] B.D. Brown, "Soft Competitive Learning Using Stochastic Arithmetic," MSc thesis, Dept. of Electrical and Computer Eng., Univ. of Manitoba, 1998.

[29] P.D. Hortensius, R.D. McLeod, and H.C. Card, "Parallel Random Number Generation for VLSI Systems Using Cellular Automata," *IEEE Trans. Computers,* vol. 38, no. 10, pp. 1466-1472, Oct. 1989.

[30] H. Zhou, H.C. Card, and G.E. Bridges, "Parallel Pseudorandom Number Generation in GaAs Cellular Automata for High Speed Circuit Testing," *J. Electrical Testing and Theory Applications,* vol. 6, pp. 325-330, 1995.

[31] H.C. Card, "Doubly Stochastic Poisson Processes in Artificial Neural Learning," *IEEE Trans. Neural Networks,* vol. 9, pp. 229-231, 1998.

[32] G.E. Hinton and T.J. Sejnowski, "Learning and Relearning in Boltzmann Machines," *Parallel Distributed Processing: Explorations in Microstructure of Cognition,* D.E. Rumelhart and J.L. McClelland, eds., Cambridge, Mass.: MIT Press, 1986.

[33] G.E. Hinton, P. Dayan, B.J. Frey, and R.M. Neal, "The Wake-Sleep Algorithm for Unsupervised Neural Networks," *Science,* vol. 268, pp. 1158-1161, 1995.

[34] R.S. Fetherston, I.P. Shaik, and S.C. Ma, "Testability Features of the AMD-K6 Microprocessor," *IEEE Design and Test of Computers,* pp. 64-69, July-Sept. 1989.

[35] B.D. Brown and H.C. Card, "Stochastic Neural Computation II: Soft Competitive Learning," *IEEE Trans. Computers,* vol. 50, no. 9, pp. 906-920, Sept. 2001.

Bradley D. Brown obtained the BSC degree in electrical and computer engineering in 1985 and the MSc degree in 1998, both from the University of Manitoba, Canada. He is currently president and founder of IDERS Inc., an electronic engineering and design firm based in Winnipeg, Manitoba, Canada. IDERS has produced turn-key products for organizations like CIBC, Datacard, Centra Energy, and Conviron. He has been responsible for the successful development of many products currently in mass production and martketed internationally. In addition, he is co-inventor of four patents, two issued and two pending, in the areas of measurement, signal processing, and product architecture. His MSc thesis was in the area of stochastic arithmetic and its application to OCR using a neural network. He remains very active in ongoing research and engineering activities and is chiefly responsible for system architecture, as well as high-level design activities such as ASIC and DSP design at IDERS. He is a member of the IEEE.

Howard C. Card obtained the PhD degree from the University of Manchester in 1971. He has held academic appointments at Manchester, Manitoba, Waterloo, and Columbia Universities, industrial appointments at the IBM T.J. Watson Research Center and AT&T Bell Labs, and has spent a sabbatical year at Oxford. He has been an Athlone Fellow and a Mullard Fellow. He has received several teaching awards, including the Stanton Award for Excellence in Teaching and the UMFA-UTS Award. He also has a number of research awards, including the NSERC E.W.R. Steacie Memorial Fellowship, the Ross Medal of IEEE Canada, the Rh. Institute Award for Multidisciplinary Research, the Sigma Xi Senior Scientist Award, the ITAC-NSERC Award for Academic Excellence, Fellow of the Canadian Academy of Engineering, and Fellow of the IEEE. He has worked on device physics and modeling, VLSI systems, parallel processing, and artificial intelligence. His current research is on neural networks and stochastic computation. He is also interested more generally in the relationship between biology, physics, and computation.