Guest Editors' Introduction: Stochastic Computing for Neuromorphic Applications

Ilia Polian
Institute of Computer Architecture and Computer Engineering, University of Stuttgart, 70174 Stuttgart, Germany

John P. Hayes
University of Michigan, Ann Arbor, MI 48109 USA

Vincent T. Lee
Facebook Reality Labs Research, Redmond, WA 98052 USA

Weikang Qian
Shanghai Jiao Tong University, Shanghai 200240, China

Editor's notes:
This extended editorial provides an introduction into stochastic computing (SC) and its usage in emerging neuromorphic applications. It covers SC primitives and the tradeoffs that occur when designing larger SC-based systems. The article shows how SC enables low-cost, low-power, and error-tolerant hardware implementation of neural networks suitable for edge computing. It provides a brief survey about recent proposals in this domain and introduces the articles in this special issue.
—Jörg Henkel, Karlsruhe Institute of Technology

AFTER DECADES OF research on general-purpose computing, the main pathway of computer architecture research has recently shifted to domain-specific concepts. In their Turing lecture in 2018, Hennessy and Patterson [1] have called the transition to domain-specific languages and architectures a "golden age for computer architects." Neuromorphic architectures have raised tremendous interest from researchers and industrial users. We can distinguish two main trends: specialized neural network (NN) processors with size and throughput exceeding that of off-the-shelf CPUs—such as the 400,000-core CS-1 wafer-scale engine by Cerebras [2]—and small NN solutions for use in resource-constrained systems, whose main advantage is their area and power efficiency.

This special issue is devoted to hardware realizations of NNs based on the stochastic computing (SC) paradigm [3], [4]. While intrinsically digital, SC offers several advantages enjoyed by analog computing: very compact and power-efficient realization of certain primitives—including multipliers and adders which are ubiquitous in NNs—and a natural compatibility with sensors and actuators. In addition, SC does not have a notion of bit significance and therefore is comparatively error-tolerant. Machine learning and pattern recognition were major driving forces for the initial development of SC in the 1960s [3], yet the researchers of that time did not manage to achieve scalability nor widescale adoption.

Digital Object Identifier 10.1109/MDAT.2021.3080989
Date of current version: 6 December 2021.

The interest in SC has increased dramatically in the last few years; theoretical understanding of SC has improved a lot and a number of better basic blocks have become available. Perhaps, most importantly, we understand now much better what factors limit the accuracy of stochastic circuits and how to overcome these limitations [5]. Interestingly, the first success stories after SC's reemergance in the mid-2010s were not NNs but rather digital filters [6], image processing [7], and decoding of low-density parity codes [8]. However, the increasing interest in edge computing has driven renewed research in SC to reduce the high power and area costs for NN tasks. Enabling low-power NN tasks at the edge with SC would also obviate the need for transmitting data to the cloud that requires using a potentially unreliable or unsecure wireless communication channel.

This extended editorial aims at making the reader familiar with the basic concepts of SC and SC-based NN realizations. It is not a comprehensive survey and only aims to cover the minimal material required to make the articles in the special issue accessible. For more extensive surveys of SC, we refer the reader to [4] for general SC and [9] for SC NNs.

Stochastic computing

The central concept of SC is a stochastic number (SN). An SN is a sequence of n bits. The numerical value of an SN x that has n_0 zeros and n_1 ones is $x = n_1/n$. In contrast to the conventional binary encoding, the bits of an SN have no positional significance, for example, 0100011010 and 1100010101 have the same value 4/10 = 0.4. The numbers representable as SNs lie in the interval [0, 1] like probabilities and follow certain properties of probabilities; this has led to the name "stochastic computing." Picking a bit at a random position within an SN will result in a 1 with a probability equal to the SN's value. To process numbers outside of [0, 1], they must be scaled into this interval.

The main strength of SC is its area- and power-efficient operations. An AND gate performs a multiplication of two SNs x and y on its inputs and provides the product on its output. To see why this is the case, recall that the probability of encountering a 1 on a randomly picked SN position is its value. If we assume that this holds for *all* positions x_i of x and y_i of y, then the probability of finding a 1 in position $(x \text{ AND } y)_i$ on the output of the AND gate corresponds to the probability of x_i and y_i being 1 simultaneously, or the arithmetic product of the individual probabilities for x_i and y_i. This holds only when x and y are uncorrelated; otherwise, x_i and y_i are not independent of each other. We will discuss the correlation problem and ways to overcome it further below.

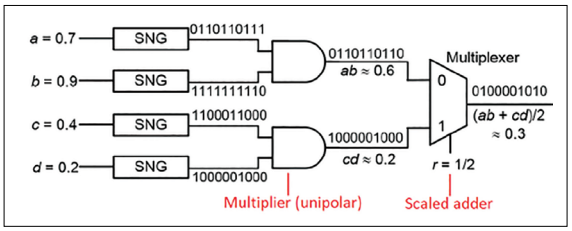

Figure 1. Example stochastic circuit [11].

If, besides multiplication, it could also support addition, SC would be able to compute all polynomial functions and approximate all other functions by polynomials [10]. However, implementing addition directly in SC is difficult because adding two numbers in the interval [0, 1] may overflow this interval. *Scaled addition* is used instead: two SNs x and y are applied to the data inputs of a multiplexer (MUX) and its select input is connected to a random stream of zeros and ones (or, equivalently, an SN of value 0.5). The MUX's output computes the value $(x + y)/2$. If a circuit includes k MUX-based additions in series, the outcome must be multiplied by 2^k to compensate for downscaling.

Figure 1 illustrates a stochastic circuit that computes the polynomial $(ab + cd)/2$ for specific values of the four inputs. An SN generator (SNG) converts a binary number into an SN (a stream of the SN bits), in analogy to an analog-to-digital converter. Once the numbers are in the stochastic domain, efficient arithmetic operations (AND for multiplication, MUX for scaled addition) can be applied. Compare this with binary adders, especially multipliers that need hundreds or thousands of gates for the usual 32- or 64-bit operations!

A typical implementation of SNG is shown in Figure 2. It compares, in each cycle, a random

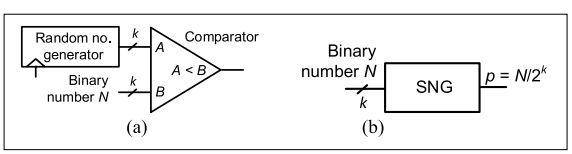

Figure 2. (a) SNG and (b) stochastic-to-binary counter.

number with the number $N \in [0, 1]$ to be converted. This produces a stream of bits with 1-probability equal to N; this is nothing else than an SN with this value. We can use either a pseudorandom number generator (PRNG) or a true random number generator (TRNG) to generate the random number. However, the simplest PRNGs, that is, linear feedback shift registers (LFSRs), can introduce undesired correlations when producing multiple SNs in parallel; improved PRNGs for SN generation have been proposed [12]. Conversion back to binary is simply done by a counter that determines the number of 1's in the SN. An interesting feature of SC is a natural interface to the analog domain; to this end, there are circuits to convert analog values directly into the stochastic representation [3].

Addition and multiplication are not the only SC primitives. Further useful SC blocks are the division circuit [13] and some operations found in image processing [7]. Even more important in the context of SC NNs are circuits for the activation function (e.g., the hyperbolic tangent function [14]) and the maximum function [15] for the max-pooling layer and for the rectified linear unit (ReLU) activation function. These circuits include sequential elements.

A number of different SC encodings have been used in the literature. Among these, the most popular is the *bipolar encoding*, where the value of an SN x is taken to be $x = (n_1 - n_0)/n$. For example, the bipolar interpretation of 0100011010 is $(4 - 6)/10 = -0.2$, as opposed to 0.4 according to the encoding introduced before (that encoding is called *unipolar*). Bipolar encoding represents numbers in the interval $[-1, 1]$, which is useful in NN calculations that can include negative numbers. When computing in the bipolar stochastic domain, multiplications are realized by XNOR gates rather than AND gates, and scaled additions remain MUX based.

SC tradeoffs and challenges

The main advantage of SC is the extremely small footprint of its primitives. The tradeoff is comparatively long SNs, leading to long latencies, and some degree of inaccuracy introduced by approximation. For example, the exact value of ab in Figure 1 is $0.7 \cdot 0.9 = 0.63$, but it cannot be represented exactly using a 10-bit SN, so its approximation 0.6 is used instead. The accuracy is further diminished by random fluctuations. For example, cd in Figure 1 is equal to 0.08, but the computed value is 0.2 and not the closest quantized value 0.1, just because the SN representation of d happens to have two 1's in the same positions as in c.

Several researchers worked on formalizing the notion of accuracy in stochastic circuits and identifying sources of inaccuracies. Qian et al. [10] distinguished between errors due to quantization, random fluctuations, and, for the generic case of nonpolynomial functions, functional approximation. Subsequent works considered further sources of errors, for example, the correlations between bits in different SNs [16], the choice of the RNG used to create SNs [17], and autocorrelation [18]. As an example, the inset "Using Isolators to Overcome the Curse of Correlations" in the following section discusses the correlation problem and its possible remedies.

Several recent approaches automatically evaluate the accuracy of a given stochastic circuit. Out of these, [11] is based on direct Monte Carlo simulation and [19] on Bayesian analysis.

The accuracy of SC increases with SN length, and it is possible to continue running the computation, that is, extending the length of the SNs, until a sufficient accuracy has been attained. This is referred to as *progressive precision* and is illustrated in Figure 3a.

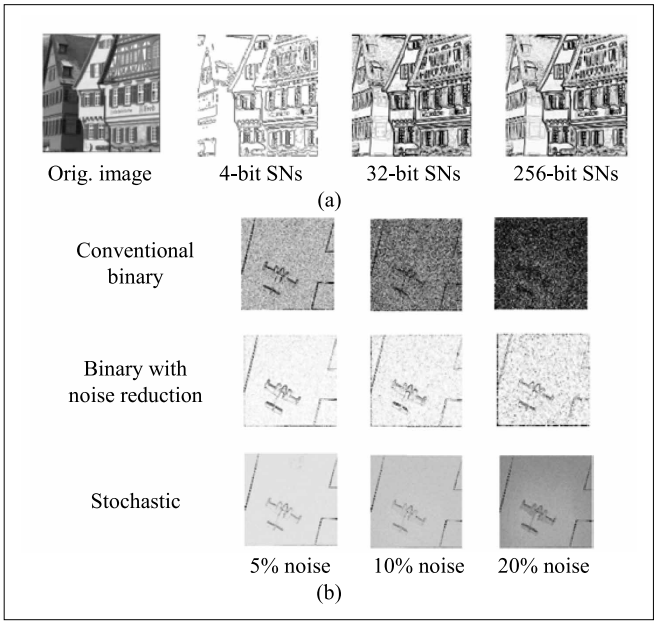

Figure 3. (a) Progressive precision for an edge-detection application and (b) error tolerance of SC [7].

We have seen that SC is affected by inaccuracies due to quantization and random fluctuations. Some sources of inaccuracy are "intrinsic," that is, they stem from SC's nondeterministic nature. However, SC's encoding scheme makes it surprisingly robust against "extrinsic" errors, that is, disturbances due to noise or radiation. Imagine that a bit is flipped in one of the SNs in the circuit of Figure 1. The circuit output will change, but only by a small value, as every bit contributes only $1/n$ to the overall result. Figure 3b shows the superior error tolerance of SC using imaging applications as an example. Multiple bit flips may even cancel out (i.e., 1-to-0 combined with a 0-to-1 bit flip result in no error in SC).

From SC primitives to SC systems

One major advantage of all digital technologies is their scalability: if you have designed good primitives, you can compose many of them into as large a system as can fit onto your circuit area and into your power budget. This is in contrast to, for example, analog circuits where different components are sensitive to disturbances stemming from their neighbors, and a system that integrates too many components no longer operates reliably. As in many other aspects, stochastic circuits fall into an intermediate position between analog and digital circuits when it comes to scaling. They are regular digital circuits and so are unaffected by any technology-level integration issues. However, simply connecting good stochastic primitives will not necessarily make a working system.

The first major obstacle is the presence of correlation among the SNs. Figure 4 illustrates how correlated values degrade the accuracy of SC multiplication; the inputs to the circuit have the same value but different correlations in this example. As a result, the outputs of Figure 4a and b differ substantially. Correlations can occur when, for example, neighboring stages of the same LFSR (or a different PRNG with a high autocorrelation) feed multiple SNGs that

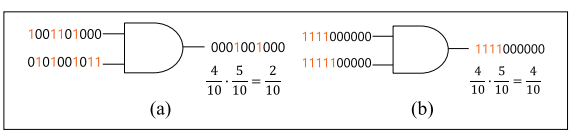

Figure 4. (a) Exact calculation of the product of two SNs and (b) the same calculation made inexact by correlations.

Inset 1: Using Isolators to Overcome the Curse of Correlation

Assume that you need to compute the square of a number, that is, the function $z(x) = x^2$. Observing that $x^2 = x \cdot x$, it appears logical to take an AND gate and connect the SN x to both its inputs. However, as shown in Figure 5a, the calculated output bits will be $z_1 = x_1$ AND $x_1 = x_1$; $z_2 = x_2$ AND $x_2 = x_2$; and so forth, that is, z will be equal to x and not x^2. Obviously, SN x is maximally correlated with its own copy. A remedy is to add an isolator flip-flop i, which will delay one input stream by one cycle. As a consequence, the output bits will be $z_1 = x_2$ AND x_1; $z_2 = x_3$ AND x_2; $z_3 = x_4$ AND x_3; and so forth; note that this stream is produced with one cycle delay. Under the (reasonable) assumption that $x_1, x_2, \ldots$ are not autocorrelated, this computation gives the correct $z(x) = x^2$.

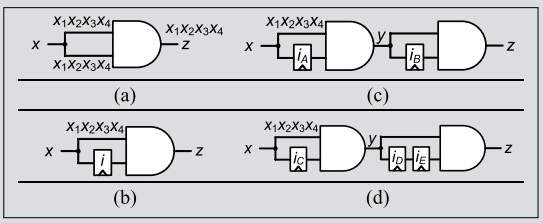

Figure 5. (a) Incorrect (correlated) squarer; (b) correct (decorrelated) squarer; (c) incorrect quartic circuit; and (d) correct quartic circuit [11].

The difficult part about correlations is that they manifest themselves beyond the boundaries of one component. To see this, assume we are now implementing the quartic function $z(x) = x^4$. Since we have designed a correct (decorrelated) squarer, intuitively it makes sense to just cascade two such squarers, using two isolator flip-flops as shown in Figure 5c. However, this will give us the output sequence $z_1 = y_2$ AND $y_1 = (x_3$ AND $x_2)$ AND $(x_2$ AND $x_1) = x_1$ AND x_2 AND x_3; $z_2 = x_2$ AND x_3 AND x_4; and so forth. Obviously, we are calculating $z(x) = x^3$ instead of the correct $z(x) = x^4$. This is because two of the four paths through the circuit have the same number of one isolator flip-flops. Figure 5d shows a design that overcomes this problem.

are used together. A rather expensive technique to eliminate correlations is to *regenerate* an SN: convert it into binary and back into stochastic domain using an unrelated SNG. Special correlation-reduced SNGs have been proposed in the past [12]. There is also a decorrelation technique based on adding isolator flip-flops [16] (see Inset 1 for details).

The second obstacle to designing large SC systems is the prevalence of scaling addition: adding every k numbers leads to an unavoidable scaling by at least a factor of k (the outcome could violate the representable range of [0, 1] for unipolar or [–1, 1] for bipolar numbers otherwise). This scaling deteriorates precision. Unfortunately, many applications, including NNs, include a large number of additions. We will discuss possible NN-specific countermeasures further below. A generic approach is to use a hybrid stochastic-binary circuit for addition. For example, the approximate parallel counter (APC) [20] takes 16 SNs and produces a 4-bit binary output that approximates their sum.

Neuromorphic architectures based on SC

On an abstract level, an NN is a sequence of basic operations, such as additions, multiplications, activation functions, and maximum functions. Given that all these basic functions have good SC primitives, a designer can implement a stochastic NN circuit by simply putting the stochastic primitives together. However, he or she must take care of two problems. First, some inaccuracy issues outlined above, most notably correlations and scaling effects of additions, manifest themselves only in larger systems. Second, SC NNs are much larger than most stochastic circuits discussed previously, which leads to nontrivial control methods to orchestrate all SNGs, stochastic arithmetic units, on-chip memories, and communication protocols.

Before delving into these two challenges, we point out that an NN, at a high level of abstraction, has two basic modes of operations: 1) training and 2) inference. During training, the NN determines (learns) its internal weights; during inference, it processes its input data using the previously learned weights. The typical application scenario of a resource-constrained NN, which is the main target for SC NN, is a pretrained NN where learning is not required, not feasible, and in the case of a safety-critical application such as an autonomous vehicle, even not desired for liability reasons. Therefore, the majority of SC NNs introduced so far [25]–[27], [29], [32]–[34] focus on the inference step done in SC hardware while skipping training. (Interestingly, one notable exception is the first comprehensive SC NN publication [14].)

To keep inaccuracies in check, SC NNs can use all the above-mentioned techniques for generic stochastic circuits: using APC instead of the MUX-based addition; correlation-aware SN generation; adding isolators where needed; and so forth. There are also approaches designed specifically for SC NNs. First, the designer can use hybrid stochastic-binary implementations [21], thus realizing the savings from the

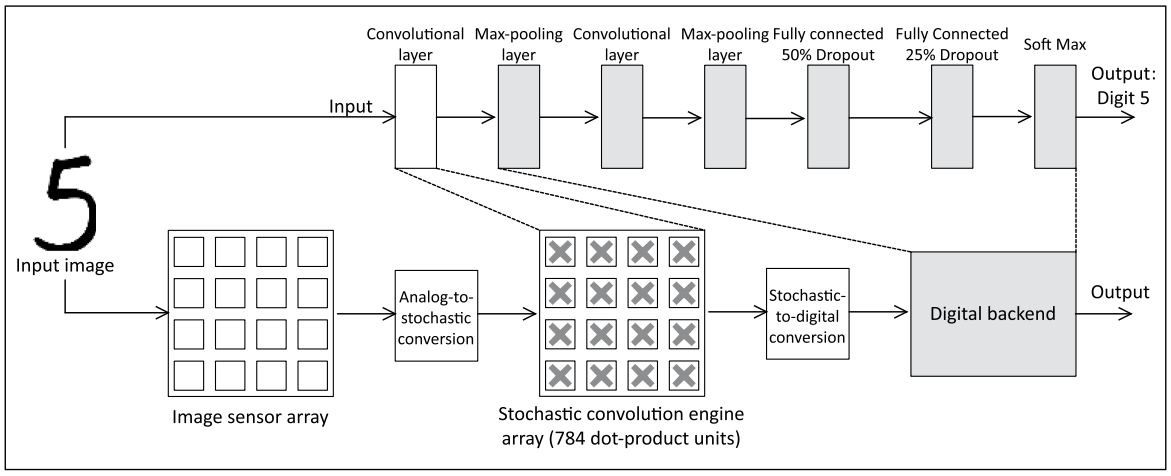

Figure 6. Hybrid binary-stochastic NN (redrawn from [21]). Only the first layer is in SC, while the other layers are binary.

Table 1. Numbers of nonzero weights (out of 16) in four kernels of a stochastic NN before and after regularization and retraining (from [22, Fig. 4]). Lower numbers correspond to fewer numbers to be added and less precision loss due to scaling.

	Before regularization	After regularization	After retraining
Kernel 1	15	5	4
Kernel 2	16	9	4
Kernel 3	16	7	3
Kernel 4	16	6	5

stochastic part while not letting its inaccuracies accumulate beyond some limit (see Figure 6 for an example). Second, the designer can modify the network itself before mapping it to hardware. The NN community has developed the notion of *regularization*, that is, modifying an NN's weights with limited impact on the network's performance. Table 1 shows how regularization with respect to the L_1 norm (followed by retraining) reduces the number of nonzero weights in a simple network [22]. Since we only need to do inference and do not modify the weights during operation, it is safe to ignore the zero weights and only provide addition hardware for nonzero weights, thus reducing the unwanted scaling factor.

To illustrate the complexity of a full-fledged SC implementation of even a comparatively small and simple NN, Figure 7 shows one full design of the seven-layer LeNet-5 network for the Xilinx Zynq XC7Z020-484C FPGA on the ZedBoard development board [23]. This design uses 512-bit unipolar SNs with an extra sign bit, and therefore the training procedure extracts NN's weights and biases as 9-bit binary numbers. A correlation-controlled SNG design [12] makes it possible to use one SNG block to generate up to 36 different SNs. The stochastic neuron in Figure 7 uses an exact binary adder especially designed for sign-extended unipolar SNs to eliminate scaling effects and a sigmoid activation function realized by a modification of the saturating counter from [14]. For max-pooling operations, a 2 × 2 array of neurons is combined with a four-input stochastic maximum circuit ("QuadSC Neuron" in Figure 7).

An SD card stores the images to be classified and the NN parameters (weights and biases). A bare-metal application on the embedded ARM microprocessor controls their reading as well as the interface with the board's pushbuttons and LEDs and communicates the classification result via UART. Rather than processing the whole image at once, all layers except the first and the last one work sequentially, making it necessary to store intermediate results in special output buffers. Together with the frame buffer to store the image under analysis and the parameter buffers to store weights and biases (the values used for the ongoing computation are stored

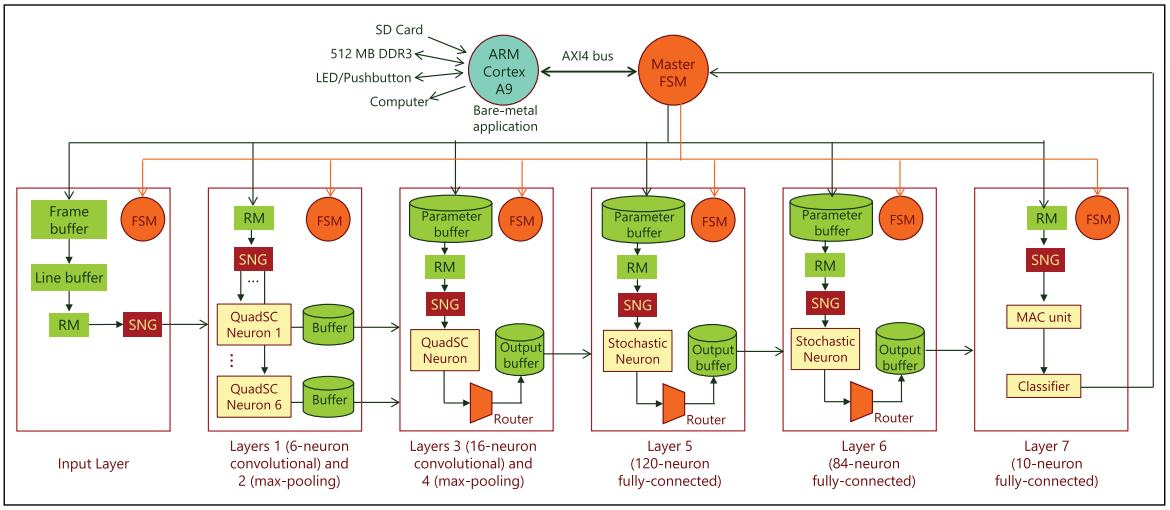

Figure 7. Block diagram of an FPGA implementation of a seven-layer fully stochastic LeNet-5 NN [24] for real-time image classification (redrawn after [23]). Register matrix (RM) blocks store parameters, such as weights, needed for the current computation in binary form to reduce required memory.

in a memory called "register matrix"), the implementation requires 48 out of a total of 140 36-KB block random-access memory (BRAM) blocks available on the FPGA. The design uses a total of 27.8 K lookup tables (LUTs) (out of 53.2 K available) and 26.5 K flip-flops (out of 106.4 K available).

The system achieves 98.13% classification accuracy on the MNIST data set and 83.06% accuracy on the more challenging fashion-MNIST data set. In comparison, the fully binary implementation of the same network achieves 98.67% and 85.97% accuracy, respectively. The binary version is faster, yet the stochastic implementation's latency is sufficient for real-time operation. A binary version with precisely the same structure as the stochastic one would consume approximately nine times its number of LUTs and would not fit on the FPGA used. For example, one SC neuron from layer 1 requires 121 LUTs and 14 flip-flops, while a binary neuron with same parameters would use either 3,862 LUTs and 1,666 flip-flops, or 511 LUTs, 21 flip-flops, and 25 digital signal-processing cores out of 220 available on this FPGA.

Further SC NN designs

While artificial-intelligence tasks were considered in the early days of SC [3], the first comprehensive treatment of an SC NN is the article by Brown and Card [14] from the year 2001. The first (to the best of our knowledge) actual fabrication of an SC NN circuit was reported in 2003 [25]; the implemented network was a three-layer Boltzmann machine with 50 SC neurons. Starting from 2015, a relatively large number of different SC NN designs were proposed. We reiterate that it is not our aim to provide a comprehensive survey of the extensive number of SC NN publications from the last few years and point the interested reader to the survey [9]. Instead, we will briefly explain several recent directions of improvement for SC NNs and refer the reader to a few interesting publications for each class of the discussed improvements.

Improved components

One avenue to improve SC NNs is to improve the performance of their building blocks and/or their integration into the overall system. APCs discussed above overcome downscaling during addition [20]. The HEIF scheme [26] further improved the APC circuit implementation and introduced a new stochastic ReLU activation function. Kim et al. [27] presented a new stochastic activation function design and combined it with a weight rescaling method, which eliminates near-0 weights and scales up the remaining ones. A more accurate hardware implementation of activation functions for SC NNs based on piecewise linearization is found in [28].

SNs beyond unipolar and bipolar

Canals et al. [29] designed an SC NN based on extended stochastic logic (ESL), where the ratio of two SNs determines the value of a number. ESL eliminates the restriction to range [–1, 1], but complicates addition. Ardakani et al. [30] demonstrated a deep belief network (DBN) based on integer SC, where an SN is a stream of integers rather than a stream of bits. Another DBN implementation [31] supports unsupervised learning using the ADAM algorithm; it is reconfigurable and includes an approximate SC activation (SCA) unit that can implement different activation functions. Faraji et al. [32] used deterministic bit stream processing (sometimes also called deterministic SC) with SNs generated from the so-called low-discrepancy sequences instead of RNGs. This makes the operations more accurate and predictable at the cost of additional area.

SC for realizing other types of NNs

A few recent publications consider stochastic implementations of recurrent NNs (RNNs), and specifically long short-term memory (LSTM) networks [33], [34]. RNNs include feedback loops, which can lead to accumulation of error effects, thus necessitating careful control of inaccuracies. LSTM designs proposed so far include the same computational primitives as regular NNs and employ the same optimization techniques, for example, Liu et al. [34] used the integer SC encoding from [29]. An RNN variant with a recent SC implementation is the time delay reservoir [35]. Other types of NNs that can benefit from SC are spiking NNs (SNNs). For instance, Smithson et al. [36] noticed striking similarities between leaky integrate and fire (LIF) neurons of SNNs and finite-state machines (FSMs) for nonlinear functions in SC.

Case studies and practical implementations

Many of the above-mentioned methodology-oriented works included implementations of SC NNs to demonstrate their ideas. A number of articles, however, focus on implementing an SC NN on either an

Inset 2: Stochastic Computing the Day After Tomorrow

This text focuses on SC NN circuits that represent today's state of the art. However, we can speculate about SC laying the foundation for more radical breakthroughs in the future:

Neural-to-SC interfaces: Biologists observed as early as 1926 that the information carried by nerves relates to the rate of electrical spikes in the signal [41]. Almost 100 years later, we still do not fully understand how the human brain represents and transmits information, and yet the principle of *rate encoding* has striking similarities to SNs (see Figure 8). If we extend our knowledge about the meaning of signals produced by brain and other organs, we may develop extremely efficient SC implants to, for example, quickly detect medical conditions with no need for power-hungry analog-to-digital converters. Future biomedical circuits may even be able to generate neural signals and actively feed them into the nervous system to overcome health problem such as epileptic attacks.

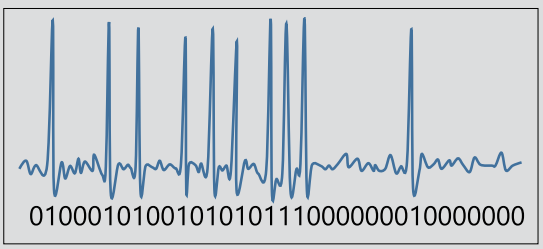

Figure 8. Are neural signals SN?

SC and nanotechnology: Memristive nanodevices can realize stochastic circuit components, both as a TRNG for SN generation and for arithmetic operations [42]. (Indeed, one of the articles in this special issue is about memristive SC!) Nanodevices offer a low area and power footprint and promise an easy integration of logic with nonvolatile memories, further reinforcing the intrinsic advantages of SC. Moreover, SC can compensate for an intrinsic drawback of most nanodevices: their limited reliability and their vulnerability to errors and noise. SC comes with error tolerance on board and may be the architectural principle of choice during the adoption of at least the first few generations of unreliable nanodevices.

SC for secure AI: The usage of NNs and other artificial intelligence techniques is at risk of *adversarial attacks*, where small perturbations in images or audio files can be practically imperceptible but lead to wrong classification outcomes. For example, an adversary could manipulate an intelligent personal assistance device to "misinterpret" the phrase "switch on the light" as "transfer $100 to account XYZ." One defense against adversarial attacks is to inject randomness into NNs, thus making it harder for the adversary to determine the minimal extent of manipulations. SC NNs can provide such randomness naturally, and designers can add special randomness-injection circuits to increase the resistance of NN implementations to adversarial attacks [43].

ASIC or an FPGA. We already discussed [23] (the first ASIC implementation of an SC NN) and [25] (the example of the "Neuromorphic architectures based on SC" section). A further interesting approach is found in [37], where SC components on an FPGA process sensor data in a fault-tolerant manner. Kim et al. [38] presented an implementation which maps parts of a convolutional NN to the FPGA's LUTs while using the on-FPGA microprocessor for the remaining operations. In [39], SC-enabled modules accelerate stochastic gradient descent computations during deep learning. Maor et al. [40] introduced an FPGA implementation of an LSTM network (see the "SC for realizing other types of NNs" section).

Articles in this special issue

This special issue starts with a keynote article by Brian R. Gaines, the inventor of SC. He shares both a view back on the history of neuromorphic computing and a view forward on deep learning as a new information processing technology. Gaines observes that computing has been a *recursive technology*: it supports other technologies that in turn support the progress of computing itself, leading to a positive exponential feedback loop and an exponential growth. He infers that the same holds for deep learning with its ability to *meta-learn* solutions to its own design problems. Gaines specifically focuses on energy aspects of hardware-backed learning and inference and the challenges due to still lacking theoretical foundations.

The first regular article, "Embracing Stochasticity to Enable Neuromorphic Computing at the Edge," by Agrawal et al., goes beyond SC in the sense of "computing with SNs." It discusses emerging nanodevices—resistive RAMs and spintronics—and their use in future neuromorphic systems. The authors discuss construction of stochastic neurons and synapses and their use for (supervised and unsupervised) learning and inference.

"Exact Stochastic Computing Multiplication in Memristive Memory," by Alam et al., continues the theme of using nanodevices for SC. The authors focus on memristors and demonstrate how to convert numbers between binary and stochastic domains and how to perform multiplications using in-memory computations by the memristive logic family "MAGIC." In contrast to earlier works on memristive SC, the authors do not harness the intrinsic stochasticity of memristive devices but rather create deterministic SNs using well-defined operations. They compare their operations with both: earlier non-SC memristive solutions and with off-memory SC.

Ardakani et al. devote their article, "Training Binarized Neural Networks Using Ternary Multipliers," to the under-investigated problem of training SC NNs (as we have pointed out, most existing works focus on inference). They introduce a new *dynamic sign magnitude* representation for symbols in *ternary* format {–1, 0, 1}, instead of the alphabet {0, 1} used for unipolar SNs and {–1, 1} used for bipolar SNs, to facilitate learning while retaining SC's benefits.

In their article "In-Stream Correlation-Based Division and Bit-Inserting Square Root in Stochastic Computing," Wu et al. design improved SC primitives for division and square root operations. Both are nonlinear functions that cannot be reduced to additions and multiplications. The authors make use of the very correlations that are usually considered undesirable in SC; their controlled injection into computations leads to good compromises between convergence time and area requirements.

The article "High-Performance Deterministic Stochastic Computing Using Residual Number System," by Givaki et al., discusses how to reduce the latency of stochastic computations. The authors represent an integer number as a set of remainders with respect to a set of relatively prime moduli, for example, the representation of number 17 with respect to the moduli set $<5, 7, 8>$ is $<2, 3, 1>$, because $2 = 17 \bmod 5$, $3 = 17 \bmod 7$, and $1 = 17 \bmod 8$. Operations such as multiplication, implemented using a deterministic version of SC, work directly on the remainders, thus yielding a partitioning of the original computation and a significant decrease in the number of clock cycles required for computation.

Finally, in their article, "An Area- and Power-Efficient Stochastic Number Generator for Bayesian Sensor Fusion Circuits," Belot et al. introduce a new SNG module with low autocorrelation properties and systematically compare it with earlier proposals regarding its implementation cost, its theoretical properties, and its performance in a specific application: Bayesian sensor fusion. ■

Acknowledgments

We are thankful to Florian Neugebauer of the University of Stuttgart for input on state of the art in the "Further SC NN designs" section and for help with picture material.

The work of Ilia Polian was supported in part by the German Research Council (DFG) under Grant 1220/13. The work of John P. Hayes was supported by the U.S. National Science Foundation under Grant CCF-2006704. The work of Weikang Qian was supported by the National Key R&D Program of China under Grant 2020YFB2205501.

References

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, 2018.

[2] K. Rocki et al., "Fast stencil-code computation on a wafer-scale processor," in *Proc. SC*, 2020, pp. 58:1–58:14.

[3] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*, vol. 2. Boston, MA, USA: Springer, 1969, pp. 37–172, doi: 10.1007/978-1-4899-5841-9_2

[4] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1515–1531, Aug. 2018.

[5] F. Neugebauer, I. Polian, and J. P. Hayes, "On the limits of stochastic computing," in *Proc. ICRC*, 2019, pp. 98–105.

[6] R. Wang et al., "Design, evaluation and fault-tolerance analysis of stochastic FIR filters," *Microelectron. Rel.*, vol. 57, pp. 111–127, Feb. 2016.

[7] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proc. DAC*, 2013, pp. 136:1–136:6.

[8] A. Naderi et al., "Delayed stochastic decoding of LDPC codes," *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5617–5626, Nov. 2011.

[9] Y. Liu et al., "A survey of stochastic computing neural networks for machine learning applications," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 5, 2020, doi: 10.1109/TNNLS.2020.3009047.

[10] W. Qian et al., "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.

[11] F. Neugebauer, I. Polian, and J. P. Hayes, "Framework for quantifying and managing accuracy in stochastic circuit design," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, pp. 31:1–31:21, 2018.

[12] F. Neugebauer, I. Polian, and J. P. Hayes, "S-box-based random number generation for stochastic computing," *Microprocess. Microsyst.*, vol. 61, pp. 316–326, Sep. 2018.

[13] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *Proc. ISVLSI*, 2016, pp. 116–121.

[14] B. D. Brown and H. C. Card, "Stochastic neural computation I: Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.

[15] F. Neugebauer, I. Polian, and J. P. Hayes, "On the maximum function in stochastic computing," in *Proc. CF*, 2019, pp. 59–66.

[16] T.-H. Chen and J. P. Hayes, "Analyzing and controlling accuracy in stochastic circuits," in *Proc. ICCD*, 2014, pp. 367–373.

[17] J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, "Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *Proc. DATE*, 2016, pp. 155–1550.

[18] T. J. Baker and J. P. Hayes, "Impact of autocorrelation on stochastic circuit accuracy," in *Proc. ISVLSI*, 2019, pp. 271–2775.

[19] T. J. Baker and J. P. Hayes, "Bayesian accuracy analysis of stochastic circuits," in *Proc. ICCAD*, 2020, pp. 124:1–124:9.

[20] K. Kim, J. Lee, and K. Choi, "Approximate de-randomizer for stochastic circuits," in *Proc. ISOCC*, 2015, pp. 123–124.

[21] V. T. Lee et al., "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *Proc. DATE*, 2017, pp. 13–18.

[22] J. Oh et al., "Retraining and regularization to optimize neural networks for stochastic computing," in *Proc. ISVLSI*, 2020, pp. 246–251.

[23] P. K. Muthappa et al., "Hardware-based fast real-time image classification with stochastic computing," in *Proc. ICCD*, 2020, pp. 340–347.

[24] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[25] S. Sato et al., "Implementation of a new neurochip using stochastic logic," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1122–1127, Sep. 2003.

[26] Z. Li et al., "HEIF: Highly efficient stochastic computing-based inference framework for deep neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1543–1556, Aug. 2019.

[27] K. Kim et al., "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proc. DAC*, 2016, pp. 124:1–124:6.

[28] V.-T. Nguyen et al., "An efficient hardware implementation of activation functions using stochastic computing for deep neural networks," in *Proc. IEEE Int. Symp. Embedded Multicore/Many-Core Syst.-Chip (MCSoC)*, Sep. 2018, pp. 233–236.

[29] V. Canals et al., "A new stochastic computing methodology for efficient neural network implementation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 551–564, Mar. 2016.

[30] A. Ardakani et al., "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2688–2699, Oct. 2017.

[31] Y. Liu et al., "An energy-efficient online-learning stochastic computational deep belief network," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 454–465, Sep. 2018.

[32] S. R. Faraji et al., "Energy-efficient convolutional neural networks with deterministic bit-stream processing," in *Proc. DATE*, 2019, pp. 1757–1762.

[33] A. Ardakani et al., "The synthesis of XNOR recurrent neural networks with stochastic logic," in *Proc. NeurIPS*, 2019, pp. 8442–8452.

[34] Y. Liu et al., "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 9, pp. 2213–2221, Jun. 2019.

[35] L. Loomis, N. McDonald, and C. E. Merkel, "An FPGA implementation of a time delay reservoir using stochastic logic," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 4, pp. 46:1–46:15, 2018.

[36] S. C. Smithson et al., "Stochastic computing can improve upon digital spiking neural networks," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2016, pp. 309–314.

[37] R. P. Duarte and H. C. Neto, "Stochastic processors on FPGAs to compute sensor data towards fault-tolerant IoT Systems," in *Proc. DSC*, 2018, pp. 1–8.

[38] D. Kim et al., "FPGA implementation of convolutional neural network based on stochastic computing," in *Proc. FPT*, 2017, pp. 287–290.

[39] C. Lammie and M. R. Azghadi, "Stochastic computing for low-power and high-speed deep learning on FPGA," in *Proc. ISCAS*, 2019, pp. 1–5.

[40] G. Maor et al., "An FPGA implementation of stochastic computing-based LSTM," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 2019, pp. 38–46.

[41] E. D. Adrian and Y. Zotterman, "The impulses produced by sensory nerve-endings," *J. Phys.*, vol. 61, no. 2, pp. 151–171, 1926.

[42] S. Gaba et al., "Memristive devices for stochastic computing," in *Proc. ISCAS*, 2014, pp. 2592–2595.

[43] P.-S. Ting and J. P. Hayes, "Exploiting randomness in stochastic computing," in *Proc. ICCAD*, 2019, pp. 1–6.

Ilia Polian is a Professor and the Director of the Institute for Computing Architecture and Computer Architecture at the University of Stuttgart, Stuttgart, Germany. His research interests include hardware-oriented security, test methods, emerging architectures, and quantum computing. Polian has a PhD in computer science from the University of Freiburg, Freiburg, Germany. He is a Senior Member of IEEE.

John P. Hayes is a Professor of EECS and holder of the Claude E. Shannon Chair of Engineering Science at the University of Michigan, Ann Arbor, MI, USA. His teaching and research interests include design and testing of VLSI circuits, reliable computer architecture, and unconventional computing methods. He is a Fellow of IEEE and ACM.

Vincent T. Lee is a Research Scientist at Facebook Reality Labs Research. He works on software/hardware codesign for machine perception. He also works at the intersection of hardware and other fields, such as computer vision, compilers, privacy, and security. Lee has a PhD in computer science and engineering from the University of Washington, Seattle, WA, USA.

Weikang Qian is an Associate Professor with the University of Michigan-Shanghai Jiao Tong University Joint Institute at Shanghai Jiao Tong University. His main research interest includes electronic design automation. Qian has a PhD in electrical engineering from the University of Minnesota, Minneapolis, MN, USA. He is a member of IEEE.

■ Direct questions and comments about this article to Ilia Polian, Institute of Computer Architecture and Computer Engineering, University of Stuttgart, 70569 Stuttgart, Germany; ilia.polian@informatuk.uni-stuttgart.de.