A Survey of Stochastic Computing Neural Networks for Machine Learning Applications

Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, *Fellow, IEEE,* and Jie Han, *Senior Member, IEEE*

***Abstract*—Neural networks (NNs) are effective machine learning models that require significant hardware and energy consumption in their computing process. To implement NNs, stochastic computing (SC) has been proposed to achieve a trade-off between hardware efficiency and computing performance. In an SC NN, hardware requirements and power consumption are significantly reduced by moderately sacrificing the inference accuracy and computation speed. With recent developments in SC techniques, however, the performance of SC NNs has substantially been improved, making it comparable with conventional binary designs yet by utilizing less hardware. In this article, we begin with the design of a basic SC neuron and then survey different types of SC NNs, including multilayer perceptrons, deep belief networks, convolutional NNs, and recurrent NNs. Recent progress in SC designs that further improve the hardware efficiency and performance of NNs is subsequently discussed. The generality and versatility of SC NNs are illustrated for both the training and inference processes. Finally, the advantages and challenges of SC NNs are discussed with respect to binary counterparts.**

***Index Terms*—Stochastic computing, neural network.**

I. Introduction

Neural networks (NNs) have widely been used in many artificial intelligence and machine learning applications, such as feature extraction [1], classification [2], and system control [3]. Their nonlinear characteristics, flexible configuration ability and self-adaptability make them convenient for machine learning applications [4]. Originally, NNs were inspired by simulating some functions of the human brain, and they modeled the way in which the brain performs a particular task or functions of interests [5]. An NN is usually implemented as a parallel distributed processor consisting of simple processing units as neurons. It resembles the human brain, so acquiring knowledge through a training process and storing the knowledge in layer weights that are associated with the interneuron connections.

There have been multiple types of NNs based on different structures and learning algorithms. A multilayer perceptron (MLP) is a type of artificial neural neworks (ANNs) in which neurons are interconnected by several layers [6]. MLP supports gradient descent-based supervised learning such as the backward propagation (BP) algorithm [6]. It can be used for the classification of nonlinear separable patterns [7] [8]. A deep neural network (DNN) is generally considered to consist of more than two nonlinear layers between the input and output layers [9]. As an example, a deep belief network (DBN) dramatically improves the performance of an MLP. By using a fast greedy learning algorithm [10], a DBN can perform unsupervised learning and solve problems such as object recognition [11], speech recognition [12] and the recognition of handwritten characters. Compared with other NNs, convolutional neural networks (CNNs) achieve a better performance in image-classification applications and significantly reduce the memory required for storing layer weights by utilizing weight sharing and pooling operations [13]. Recurrent neural networks (RNNs) are widely used for solving time-related problems, such as speech recognition [14], [15]. The long short-term memory (LSTM) structure has been introduced to improve the accuracy of RNNs and has become one of the most widely-used RNN structures [16].

Compared to software implementations, hardware implementations of NNs offer the advantages of an inherently high degree of parallelism and fast processing speed. Unfortunately, complex hardware is required because NNs may involve thousands of neurons in a single layer, so resulting in millions of parameters that need to be adjusted to achieve high classification accuracy [2]. Since a large NN can easily overfit the dataset, several techniques have been developed to solve the overfitting problem, including the use of weight noise [17], Dropout [18], DropConnect [19], binarized neural networks (BNNs) [20], and quantized NNs (QNNs) [21], [22]. These techniques add noise to the activation function or layer weights. Using these methods, large networks generally achieve higher accuracy compared to small networks. Recently, NNs have been implemented in FPGAs [23], graphics processing units [2], and embedded systems [24]; however, the power and energy consumption of these implementations remain high.

In contrast to conventional binary circuits, a stochastic computing (SC) circuit requires a small hardware cost with a low power consumption and high fault tolerance to computational

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Projects RES0018685 and RES0025211 (Y. Liu, S. Liu and J. Han), the National Science Foundation (NSF) grant No. CCF-1919117 (Y. Wang), and the NSF grants CCF-1953961 and 1812467 (F. Lombardi).

Yidong Liu was with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada. Current address: the China Aerospace Science and Industry Corporation Limited (CASIC), Beijing, China (e-mail: liuyidong@embed-tec.com).

Siting Liu was with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada. Current address: Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 0G4, Canada (e-mail: siting.liu@mcgill.ca).

Jie Han is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada (e-mail: jhan8@ualberta.ca).

Y. Wang and F. Lombardi are with Department of Electrical and Computer Engineering, Northeastern University, Boston 02115, MA USA (e-mail: yanz.wang@northeastern.edu, lombardi@ece.neu.edu).

and soft errors [25]. SC reduces the size of many fundamental arithmetic circuits, such as adders, subtractors [26], [27] and multipliers [28], [29]. The sigmoid function, for example, the hyperbolic tangent ($tanh$) and exponential functions can all be implemented by linear finite state machines (FSMs) [30]. These designs make it possible to implement SC NNs at a significantly lower hardware cost by moderately sacrificing computation accuracy. In addition, SC uses stochastic sequences to encode real values. Therefore, it introduces stochasticity and, thus, noise into the SC NNs. Noise can potentially be utilized to solve the overfitting problem to improve accuracy in inference [31].

Due to the long sequence length and a large number of stochastic number generators (SNGs) required in the circuit, however, it is a challenge for an SC NN to achieve a lower computational latency and energy consumption compared with conventional designs. To overcome this challenge, several improved SC encoding methods have been proposed to reduce the sequence length [32], [33], [34], thereby improving the performance and energy efficiency. Some designs focus on the improvement and reuse of random number generators (RNGs), so leading to better hardware and energy efficiency [35], [36]. These new techniques make SC NNs competitive in both hardware efficiency and computation performance compared with conventional binary implementations.

In this article, we survey the recent developments of SC NNs, including radial basis function NNs [37], MLPs [35], [38], [39], [40], [41], CNNs [42], DBNs [36], [43] and RNNs [34], [44], and their applications to machine learning. It is shown that SC techniques can be adapted for various applications hence they are versatile for implementing NNs. In addition to inference, SC circuits have been used to implement the BP algorithm for training and, at the same time, attaining high hardware efficiency and flexible network configuration [36], [40], [41], [45]. These developments show the potential of SC techniques for machine learning applications.

This article is organized as follows. First, we review the basic components in an SC neuron. Then, we discuss the specific components required in different types of SC NNs. Recent progress in various SC techniques is introduced, including those extending the computational range, reducing the sequence length, and efficient encoding. Performance of SC NNs is compared with those of BNNs and QNNs. Finally, a conclusion is drawn on current developments.

II. Backgrounds

NNs consist of neurons as structural constituents. It is estimated that there are approximately ten billion neurons in a human brain [6]. Fig. 1 illustrates the typical structure of a neuron. In the neuron, the cell body receives input signals from the dendrites connected to the synaptic terminals controlled by the other neurons. The signals are converted and encoded as a series of sharp voltage pulses known as spikes and then passed to other neurons through the axon. The human brain can be developed to adapt to the surrounding environment by two mechanisms: the creation of new synaptic connections between neurons and the modification of existing synapses, both of which lead to changes in the structure and the parameters of the NN.

Based on the biological structure, the neuron in an NN is designed as an information processing unit and is utilized as the fundamental unit of the network. The function of the neuron is shown in Fig. 2. The input signals of the neuron are denoted by $\{x_i\}$, where $i = 1, 2, ..., m$, and the parameters of the neuron (given as a layer weight) are $\{w_i\}$, $i = 1, 2, ..., m$. The output of the neuron is generated by

$$y = \phi(\sum_{i=1}^{m} w_i x_i), \tag{1}$$

where ϕ is the activation function. In SC NNs, there are different types of widely used activation functions, including the $tanh$ function, sigmoid function, and rectifier function (ReLU). Assume the input of the activation function is v, the activation functions are given by

$$\phi(v) = \begin{cases} \tanh(2v), & tanh \\ \frac{1}{1+\exp(-v)}, & sigmoid \\ \max(0,\ v). & ReLU \end{cases} \tag{2}$$

Each of the functions can be used to solve nonlinear classification problems [26]. The ReLU is the dominant type of activation function in the state-of-the-art NNs. Compared with the $tanh$ and the sigmoid function, the ReLU function allows a network to obtain sparse representations by eliminating random fluctuations.

A simplistic NN, for example, an MLP, includes one input layer, at least one hidden layer, and one output layer (see Fig. 3). Each layer consists of multiple neurons as the basic computation units. One of the most widely used learning algorithms is the BP algorithm. This algorithm proceeds in two phases: forward propagation phase and backward propagation phase [6]. The forward propagation phase generates output signals based on the current inputs and layer weights. In the BP phase, error signals are first obtained from the output signals; then the layer weights are updated using the error signals.

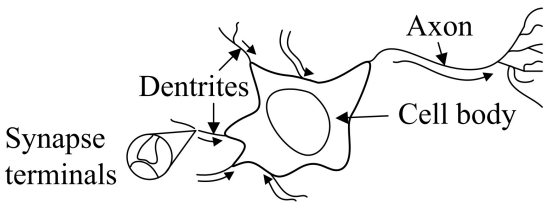

Fig. 1: Typical structure of a neuron in the human brain [46].

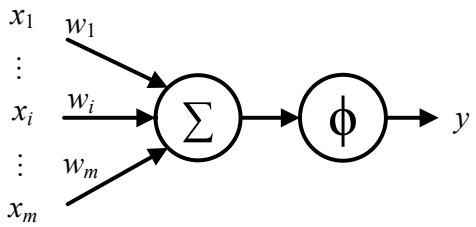

Fig. 2: Function of a neuron in ANNs [6].

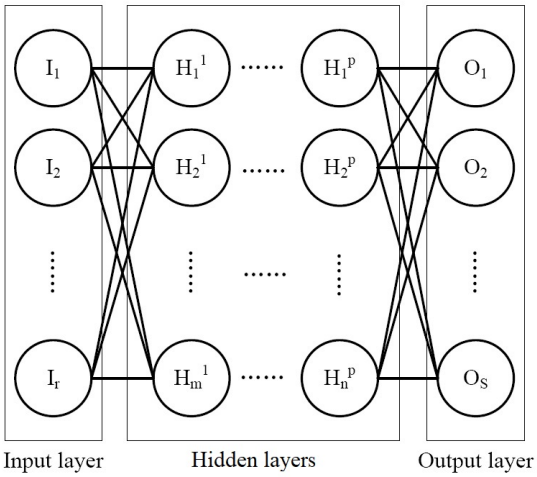

Fig. 3: Structure of an MLP. I_i is the i^{th} neuron in the input layer, H_j^p is the j^{th} neuron in the p^{th} hidden layer and O_k is the k^{th} neuron in the output layer.

The BP algorithm is executed in multiple stages called epochs. In each epoch, the NN is trained on the training dataset. Let $y_j^l(n)$ be the output signal of neuron j in layer l at epoch n, and $w_{ji}^l(n)$ be the layer weight between the neuron j at layer l and the neuron i in the previous layer $l-1$. In the forward propagation phase, the output signal of the neuron is computed following (1) and (2). In the BP phase, the layer weight $w_{ji}^l(n)$ is updated as

$$w_{ji}^l(n+1) = w_{ji}^l(n) + \eta \Delta w_{ji}^l(n), \tag{3}$$

where

$$\Delta w_{ji}^l(n) = \delta_j^l(n) \cdot y_i^{l-1}(n), \tag{4}$$

η is the learning rate, and $\delta_j^l(n)$ is the local gradient, defined as

$$\delta_j^l(n) = \begin{cases} (t_j(n) - o_j^l)\phi'(v_j^l(n)), & output\,layer; \\ \phi'(v_j^l(n)) \sum_{i=1}^{p} \delta_i^{l+1} w_{ij}^{l+1}(n), & hidden\,layer. \end{cases} \tag{5}$$

In (5), $t_j(n)$ is the j^{th} variable in the class label of a training sample. For the neurons in the output layer, the local gradient is determined by the error signal generated by $t_j(n) - o_j^l$. For the neurons in the hidden layers, the local gradient is determined by the sum of $\delta_i^{l+1} w_{ij}^{l+1}$. In both cases, $\phi'(v_j^l(n))$ is the derivative of the activation function with respect to $v_j^l(n)$.

The forward and backward propagation processes are repeated until the maximum allowed number of epochs is reached, or an early stopping criterion is met [6].

III. SC Neurons: Overall Structure, Arithmetic Operators and Activation Functions

A. Overall Structure

In SC, the presence of p 1's in a random binary bit stream with length q encodes the value p/q in the closed interval $[0, 1]$ in the unipolar representation, or the value $(2p-q)/q$ in the closed interval $[-1, +1]$ in the bipolar representation. SC significantly reduces the complexity of an arithmetic circuit

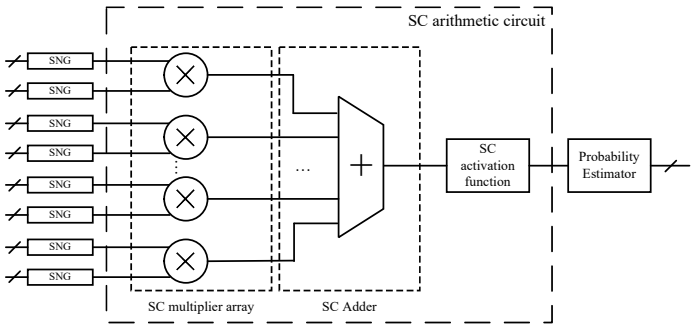

Fig. 4: Typical structure of an SC neuron.

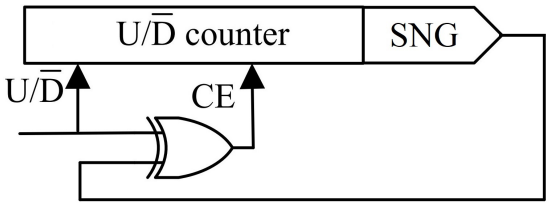

Fig. 5: ADDIE-based PE design, consisting of an up-down counter, an XNOR gate, and an SNG [26].

by processing the bit streams. It has been widely used in a variety of applications, such as low-density parity check (LDPC) decoding [47], image processing [48], [49], digital filters design [50], [51] [52], and circuit reliability evaluation [53], [54]. The reader is referred to [28], [55], and [56] for a general introduction to SC.

As the fundamental unit in NNs, most SC neurons share a similar structure (see Fig. 4). It consists of an array of SNGs, an SC arithmetic circuit, and a probability estimator (PE). An SNG [consisting of an RNG and a comparator (CMP)] is used to convert a binary input into a stochastic sequence. The SC arithmetic circuit implements the function of the neuron. As per (1) and (2), the SC neuron can be implemented by multipliers, adders and activation circuits. These arithmetic circuits are required in different types of NNs for inference [26] and can be implemented by different SC designs, as introduced in the remainder of this section. The PE is utilized to convert a stochastic sequence back into a binary value. It can be implemented by an adaptive digital element (ADDIE)-based circuit (see Fig. 5) [26]. This design compares the probabilities encoded in the input sequence and the generated sequence from the SNG; then the value in the up-down counter is increased when the probability encoded in the input sequence is larger and vice versa, until the same probability is obtained. After convergence, the value in the up-down counter remains unchanged and is considered as an estimate for the probability encoded in the input sequence.

B. Multipliers

The multiplier is one of the fundamental arithmetic circuits in a neuron. The SC multiplier is implemented by an AND gate for the unipolar representation or an XNOR gate for the bipolar representation, as shown in Fig. 6 [26]. The SC multiplier

significantly reduces the area by increasing the computation time compared with conventional binary multipliers.

C. Adders

The original SC adder is implemented by a 2-input multiplexer (MUX) with the probability of the select signal set to 0.5, as shown in Fig. 7 (a) [25], [26]. The probability of the output signal is not affected by any correlation between the input signals. Thus, the RNGs can be shared among the input signals to reduce the hardware cost and energy consumption. However, the select signal must be uncorrelated with the input signals, so additional RNGs are still required. An SC adder tree can be implemented for multiple input signals; the output of the SC adder is scaled by 0.5, that is

$$p_3 = \frac{p_1 + p_2}{2}, \tag{6}$$

where p_1, p_2, and p_3 are the values encoded in the input and output sequences [see Fig. 7 (a)]. The computation reduces the resolution by half, so the output signals need to be re-normalized before the next step of processing, therefore increasing the hardware of the adder tree.

An accumulate parallel counter (APC) based SC adder is introduced to improve performance and circumvent the scaling problem [see Fig. 7 (b)] [57]. In the APC-based SC adder, the 1's in the D-dimensional input sequences (S_i, where $i =$ 1, 2, ..., D) are simply added up. The probability of the output signal is determined only by the sum of the probabilities of the input signals; thus, the RNGs can be shared among the input signals with no loss in the computation accuracy. Because no SNGs are required to generate the select signals, the hardware required by this design is lower than that of the original SC adders for the same number of input signals. The processing speed of the APC-based adder is higher than the original SC design because computation is performed in parallel without the use of an adder tree. Moreover, the output of the APC-based adder is in binary. The area of the design can further be reduced by replacing the APC with an approximate parallel counter (AxPC) (see Fig. 8). Compared with the APC, the full adders (FAs) in the first layer are replaced by pairs of OR and AND gates to reduce hardware.

A T flip-flop (TFF) based SC adder is introduced in [58] [see Fig. 7 (c)]. Assume that the values encoded in the input and output sequences in the unipolar representation are p_x, p_y, and p_z. One can show that

$$p_z = \frac{p_x + p_y}{2}. \tag{7}$$

Because the sequence generated by the TFF is uncorrelated with the input sequences, the output of the circuit is not influenced by the auto-correlation in the input signals. Compared with the conventional SC adder, this adder requires no additional stochastic sequences for the select signal to the MUX, thus reducing the area of the design.

D. Activation Circuits

The activation functions of (2) are typically implemented using two different methods: one is based on FSMs and the other using SC polynomial arithmetic circuits.

FSM-based computational elements implement multiple activation functions with different state transition settings [26]. The design includes a saturating counter with the state of the counter controlled in a closed-loop. The state transitions of the $tanh$ and exponentiation circuits are shown in Fig. 9.

A $Btanh$ circuit is proposed in [35] to improve the computation speed of the SC activation functions. The design of the $Btanh$ circuit is shown in Fig. 10; instead of updating the state of every single bit in the input sequence, the $Btanh$ circuit adds up multiple bits from the input signals with an APC and changes the state of the up-/down-counter-based FSM in multiple steps within each computation cycle [35]. The algorithm significantly improves the computation speed and

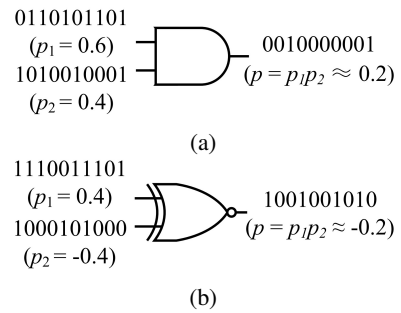

Fig. 6: (a) SC multiplier for the unipolar representation. (b) SC multiplier for the bipolar representation [26].

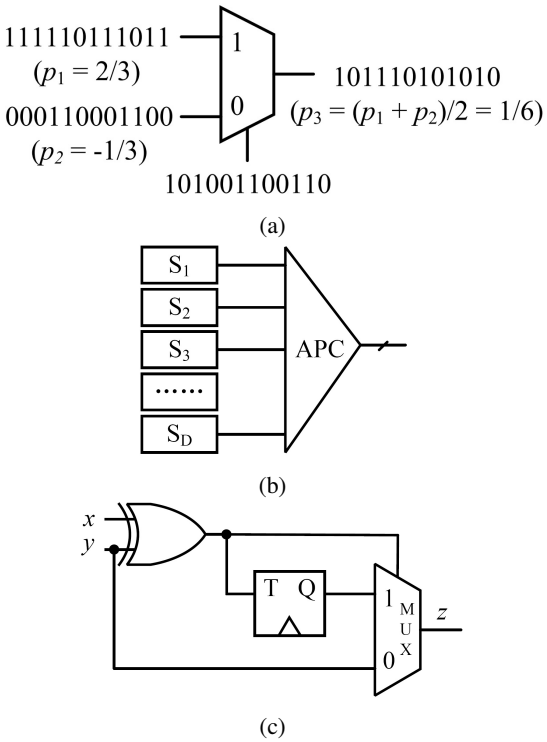

Fig. 7: (a) Original SC adder [26]. (b) APC based SC adder [57]. (c) TFF-based SC adder with $p_z = (p_x + p_y)/2$ [58].

obtains accurate results even when the sum of the probabilities of input signals goes beyond $[-1, +1]$.

The $Btanh$ circuit is further improved in [60]. The design is shown in Fig. 11. The up/down counter is replaced by a linear approximation unit (LAU). Based on a binary design, the function of the LAU has the generalized form of

$$\psi(x) = \min(1, \max(p, \frac{1}{r}x + s)), \tag{8}$$

Multiple types of activation functions are implemented by configuring the value of p, r and s in the LAU. For example, the sigmoid function is implemented with the configuration

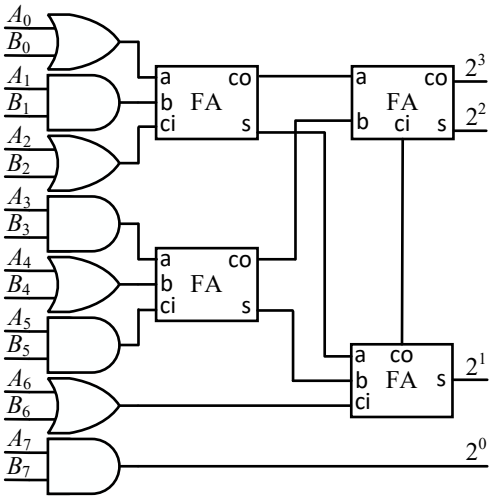

Fig. 8: Design of an AxPC with 8-bit input signals. In the full adders (FAs), a and b are the input bits, ci is the carry-in bit, s is the sum and co is the carry-out bit [59].

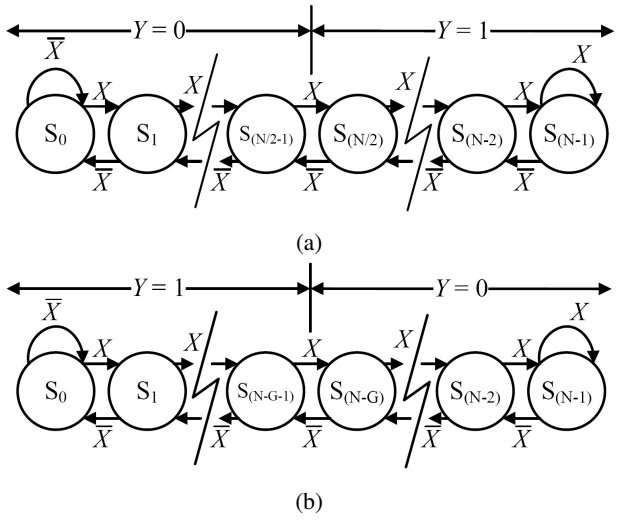

Fig. 9: State transition diagram of (a) a $tanh$ circuit and (b) an exponentiation circuit [26]. X indicates that the input bit is "1" and $\bar{X}$ indicates that the input bit is "0". Y is the output bit. Assume that the probabilities that are encoded in the input and output sequences are x and y, and that the state number is N. One can show that the circuits implement the functions $y = tanh(\frac{N}{2}x)$ and $y = e^{-2Gx}$, respectively, when $N >> G$.

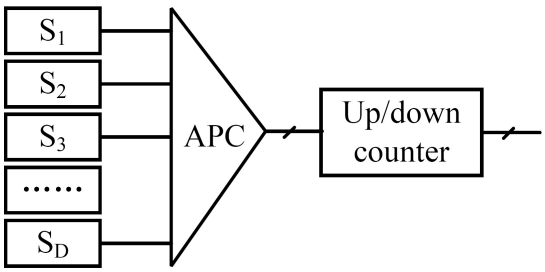

Fig. 10: Design of the $Btanh$ circuit [35]. S_i, $i = 1, 2, ..., D$ is the input sequence, with a dimension of D.

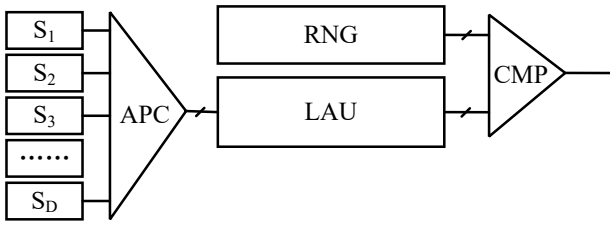

Fig. 11: Design of the reconfigurable activation circuit [60], consisting of an APC, an LAU, an RNG, and a comparator (CMP).

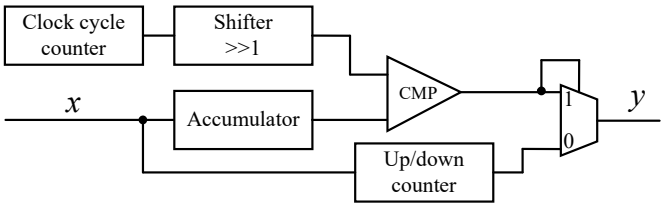

Fig. 12: Design of the SC ReLU circuit [61]. CMP represents a comparator. x and y are the values encoded in the input and output sequence in the bipolar representation.

of $\{p = 0, r = 4, s = 1/2\}$, and ReLU function with $\{p = 0, r = 1, s = 0\}$.

An SC ReLU circuit is proposed in [61] (see Fig. 12). The input to the circuit is accumulated and compared with half of the passed clock cycles. The CMP output is used as the input and the control signal of the multiplexer at the same time. If the accumulation result is smaller than the reference number, the CMP outputs a '1' and is selected by the multiplexer as the output. Otherwise, the output is determined by the $Btanh$ circuit implemented by an up-down counter. The circuit ensures that the value encoded in the output sequence is no less than 0.5 in the unipolar representation or 0 in the bipolar representation. Therefore, assuming that the values encoded in the input sequence and the output sequence are x and y in the bipolar representation, the function of the circuit is then given by

$$y = \max(0,\ \tanh(2x)). \tag{9}$$

SC polynomial arithmetic circuits are also commonly used to implement activation functions. The nonlinear activation functions are first expanded by using a Taylor series or Bernstein polynomials [62]. A finite number of terms are then

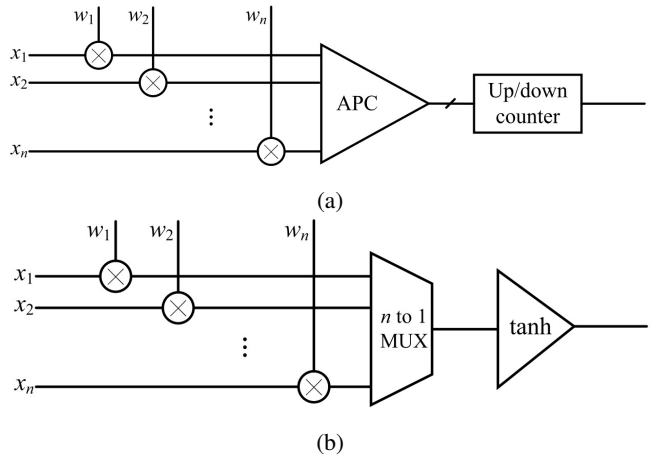

Fig. 13: (a) An APC-based neuron. (b) A MUX-based neuron [59]. The dimension of the input sequences is n. The multipliers are implemented in SC.

computed by SC polynomial circuits [63] or exponentiation circuits [see Fig. 9 (b)].

Compared with the FSM method, an SC polynomial activation circuit is easier to be reconfigured but with a larger area. In a recent implementation [64], the activation function is approximated by linear functions. The AxPC is utilized in SC adders to reduce the area and power consumption.

IV. SC NNs: Forward and Backward Propagation Circuits

A. Forward Propagation Circuits

There are two types of SC neurons for the forward propagation: the APC- and MUX-based neurons [59]. The structure of an APC-based neuron is shown in Fig. 13 (a). The structure is similar to the $Btanh$ circuit. Simulation results have shown that the computation error of the APC-based neuron slowly decreases, while the area, power, and energy of the circuit linearly increases as the input size grows [65]. The MUX-based neuron is introduced in [59] [see Fig. 13 (b)]. This design achieves a lower area (60%, or a reduction by 40%) than the APC-based neuron with the same number of input signals. However, the accuracy degradation is more significant as the size of the input signal increases. As found in [59], the computation error of the MUX-based neuron is significantly higher than for the APC-based neuron. Therefore, longer bit streams are required in a MUX-based neuron for accuracy compensation, thus resulting in a higher energy consumption compared to its APC-based counterpart.

B. Backward Propagation Circuits

The BP components are required for the training process. The arithmetic SC circuits for the BP in MLPs are introduced in [27] and [66]; it is shown that the BP circuit can be implemented using subtractors and multipliers.

In [66], the BP components implement the BP phase in four steps: the computation of the error signals in the output layer [see Fig. 14 (a)]; the error signals in the hidden layer [see Fig. 14 (b)]; the local gradient [see Fig. 14 (c)] and the updated layer weights [see Fig. 14 (d)]. For the nth epoch, first, the error signal in the output layer is generated by $t_j(n)-o_j^l$, as in (5). The error signal in the hidden layer is then computed by $\sum_{i=1}^{p}\delta_i^{l+1}w_{ij}^{l+1}(n)$ and the gradient is generated following (5). Finally, the layer weights are updated following (3) and (4). The unipolar representation is considered in the implementation, so it requires two output signals of the layer weight updater to indicate if the value of the layer weight has increased or decreased. In addition, it requires four stochastic sequences to encode each gradient signal in the computation.

The SC BP circuits are proposed in [40] to simplify the computation process and expand the computation range. The computation is based on the extended stochastic logic (ESL) and the values encoded in the sequences are in the bipolar representation. The ESL utilizes two stochastic sequences to represent a value and increase the computation range of SC. The principle of ESL is introduced in the following sections. The BP circuit for a hidden layer neuron is shown in Fig. 15. When the $tanh$ function is set as the activation function, we have

$$\begin{aligned}\tanh'(2\cdot v) &= 2(1-tanh^2(2\cdot v)),\\ &= 2(1+tanh(2\cdot v))(1-tanh(2\cdot v)),\end{aligned} \tag{10}$$

where the value of $\tanh(2\cdot v)$ is computed by the forward propagation components. The differential function of $\phi'(v_j)$ can be implemented by ESL adders, subtractors, and multipliers. The layer weight adjust calculator updates the layer weights as in (3) and (4). Compared to the design of [66], each gradient signal is represented by two stochastic sequences in ESL. Therefore, computation is simplified with the expansion of its range. However, it still requires a long sequence length to achieve an acceptable accuracy for inference in the SC BP design. For example, the sequence length is set to 4096 bits to achieve a 97.95% accuracy for inference in [40].

The implementation of the SC BP circuits is further discussed in [41] to reduce the sequence length in training. Training of a NN can be considered to be an optimization problem for the weights in a NN, and the gradient descent is a simple but efficient method for the optimization by an iterative addition of the gradients. In [41], the gradients of the training samples are computed and accumulated by an SC gradient descent circuit (GDC). Due to the randomness cancelation effect in the accumulation process, the sequence length used in the design can be aggressively reduced to achieve both high performance and energy efficiency. One bit is used in training to encode the gradient information of one training sample, so each bit can have a different probability to be "1". The stochastic bits encoding the gradients are accumulated by stochastic integrators [41]. The randomness of the stochastic bits is eliminated during accumulation; also due to the error-tolerance in an MLP, the SC-based training circuit achieves a training accuracy similar to its fixed-point (FxP) and floating-point (FP) counterparts. Note that only weight updating is implemented by SC circuits in the training process (not the entire BP).

The SC NNs with BP circuits achieve a higher hardware efficiency in the training process with the potential for im-

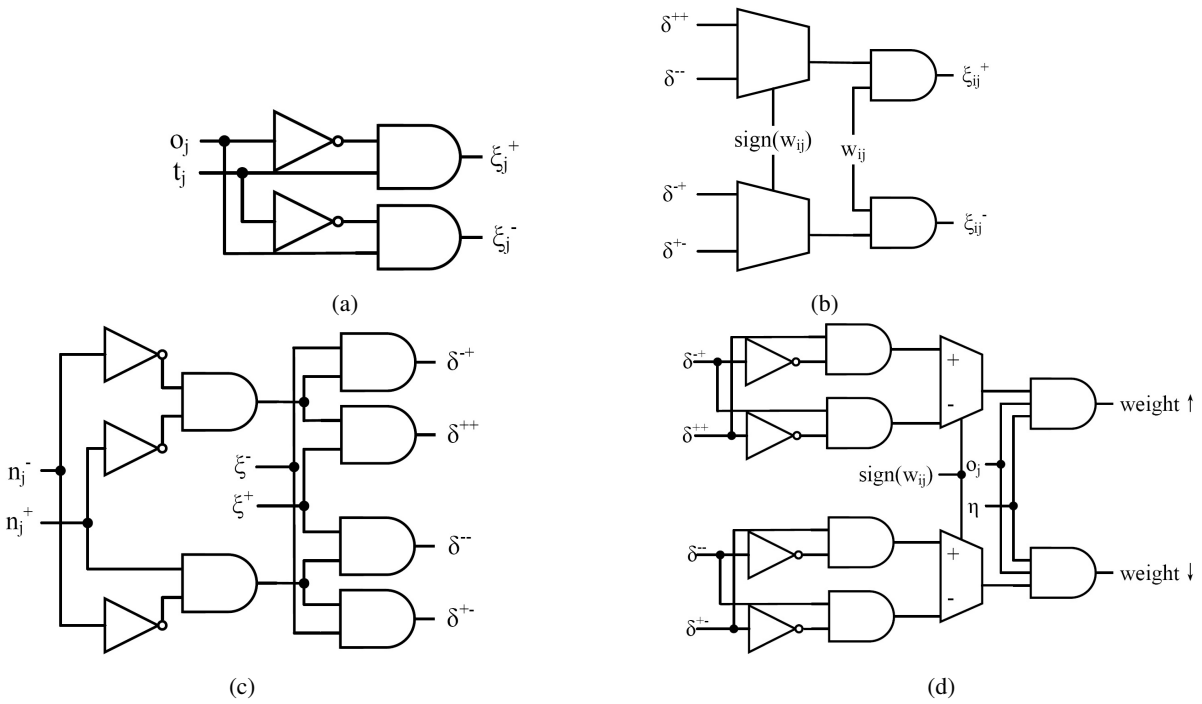

Fig. 14: The computation circuits for the BP in MLPs [66]. (a) Error signal generator, (b) Circuit computing the error in the BP, (c) Gradient generator, and (d) Layer weight updater. o is the output signal generated by the forward propagation. t is the target label. ξ is the error signal. η is the intermediate signal generated by the layers in the forward propagation. δ is the gradient. w is the layer weight and $weight$ is the difference between the values of the previous and updated layer weights. i and j are the indexes of the two neurons connected through the layer weight.

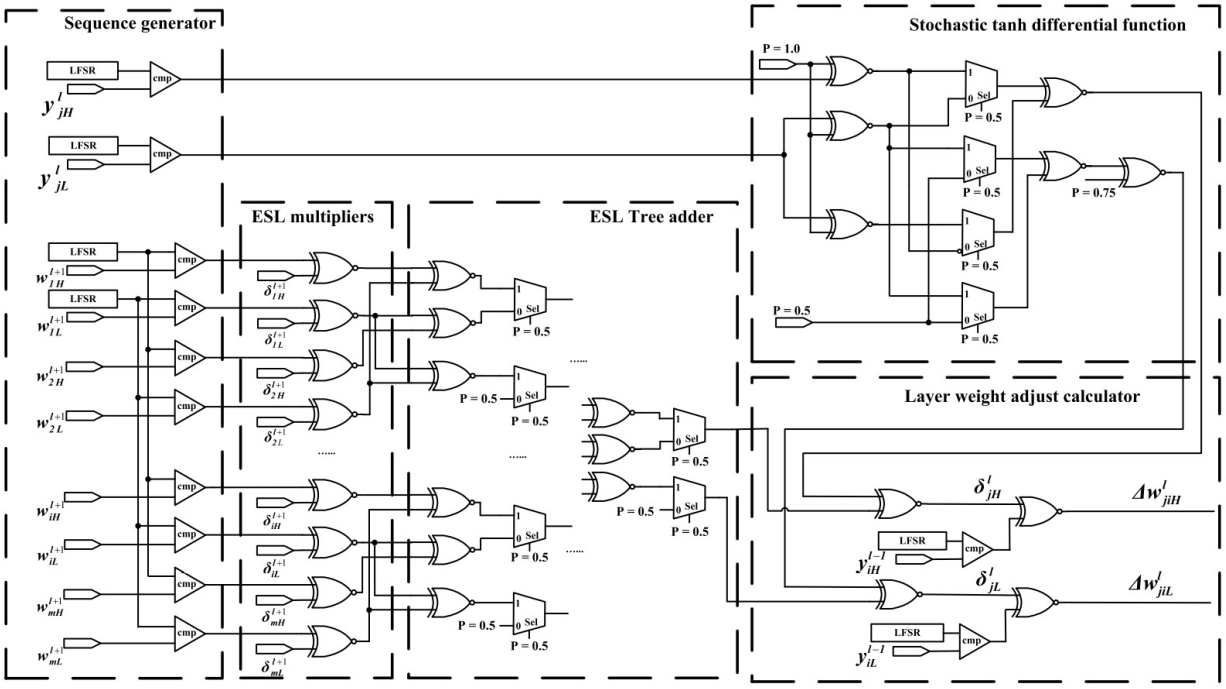

Fig. 15: The BP circuit for a hidden layer neuron in [40]. The signals follow the same definitions in (3)-(5) using ESL.

plementing an online learning algorithm and supporting the fine tuning of networks, thereby dynamically improving the inference accuracy and flexibility of the system. Moreover, SC NNs can achieve a higher noise tolerance in the computation process, compared to conventional binary implementations [44].

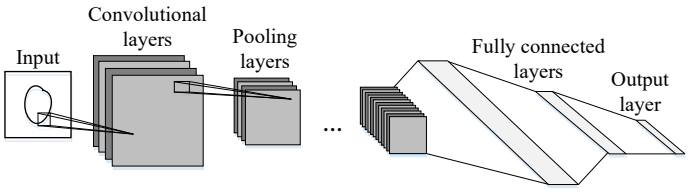

Fig. 16: Structure of a CNN, consisting of convolutional layers, pooling layers, fully connected layers, and an output layer [13].

C. Computation Circuits in SC CNNs

A CNN consists of four types of layers: convolutional (CONV) layers, pooling layers, fully connected layers and a softmax based output layer (see Fig. 16).

CONV layers consist of multiple sessions of *feature maps*, within which all the neurons share the same set of weights. The computation in the CONV layer is inner-product based [13]. Therefore, it can be implemented using SC multipliers and adders. Similarly, the fully connected layers can be implemented using SC forward propagation circuits.

The pooling layers are implemented for sub-sampling to reduce the complexity of the computation. The most common strategies are average pooling and max pooling.

In average pooling, the neuron computes the average probability of the input sequences. It can be implemented by conventional SC adders because as per (6), the value encoded in the output sequence is the average of the values encoded in the two input sequences. Thus, the average pooling circuit can be implemented by an SC adder tree with each adder implemented by a MUX with the select signal encoding a value of 0.5 in the unipolar representation [see Fig. 17 (a)] [59].

In the max pooling operation, the neuron computes the maximum probability of the input sequences. The design of a max pooling neuron is introduced in [65] [see Fig. 17 (b)]; it assumes that the bit-stream encoding the globally highest probability has also the highest probability for a specific segment of the stream because all 1's are randomly distributed in the stochastic bit streams. Thus, instead of computing the number of 1's in the entire bit-streams, the circuit counts every time the 1's in a segment of the streams and makes the comparison, thereby reducing the computation time. A $tanh$-based max pooling circuit is proposed in [65] [see Fig. 17 (c)]. The $tanh$ circuit is implemented by an FSM with an enable (En) input. Assume that the input bits of v_1 and v_2 are different, the state increases with v_1 being "1" and decreases with v_1 being "0". When the probability of v_1 is higher than v_2, the output of the $tanh$ circuit is more likely to be "1" so v_1 is selected as the v_{max}; otherwise v_2 is selected.

V. Advanced SC Techniques for NNs

New techniques have been proposed with the development of SC NNs, including improvements for expanding the computation range, reducing sequence length, and efficient encoding.

A. Computational Range Expansion

1) ESL: The computational range of the conventional SC is limited in $[0,\ 1]$ in the unipolar representation and $[-1,\ +1]$ in the bipolar representation; this feature restricts the use of the SC circuits to specific NN applications. ESL is one of the methods to overcome this drawback [67]. In ESL, a real number is encoded as the ratio of two stochastic sequences using the bipolar representation. Assume that the two sequences encode the values of p_h and p_l in the bipolar representation. Then a real number x is approximately given by the following quotient [67]:

$$x = \frac{p_h}{p_l}. \tag{11}$$

By doing so, the computational range of SC is expanded to $(-2^{N-1},\ 2^{N-1})$ for a binary representation in N bits. Based on the definition of ESL, a multiplier in the bipolar representation can be implemented by two XNOR gates [see Fig. 18 (a)]. To implement the ESL adder, assume that the operands (s_1, s_2) are represented by sequences $\{S_{1h},\ S_{1l}\}$

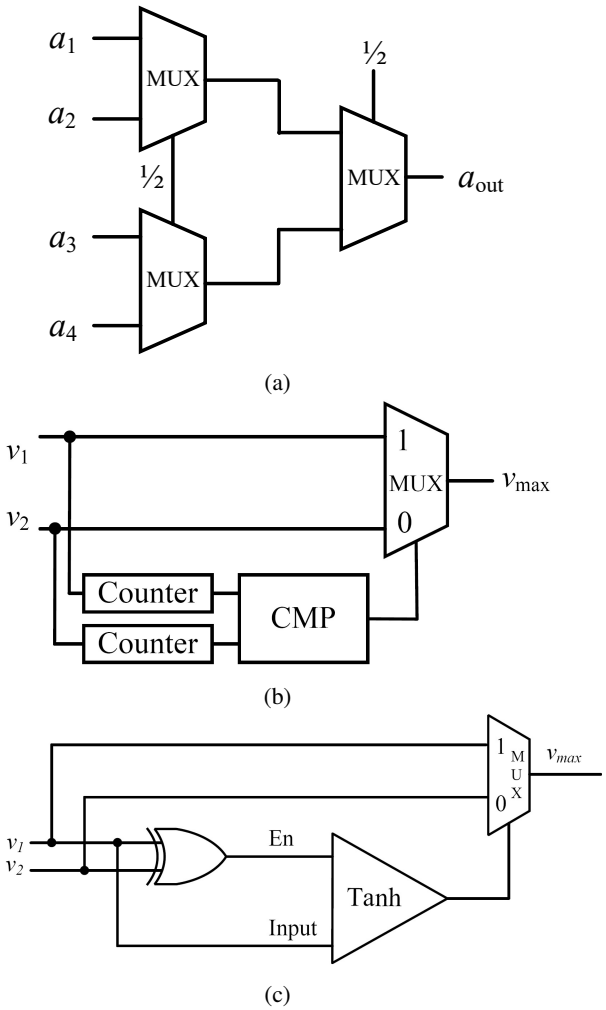

Fig. 17: Design of (a) An average pooling circuit, implementing $a_{out} = \frac{1}{4}\sum_{i=1}^{4} a_i$ [59], (b) a counter-based max pooling circuit [65], and (c) a $tanh$-based max pooling circuit, both implementing $v_{max} = max(v_1,\ v_2)$ [65].

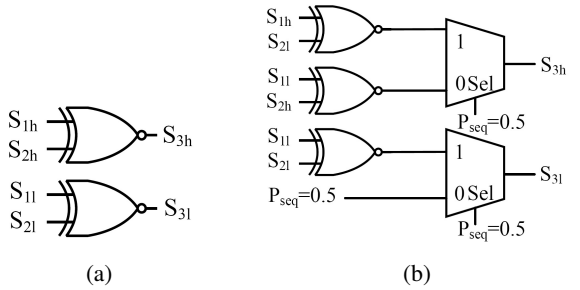

Fig. 18: (a) ESL multiplier, and (b) ESL adder [67].

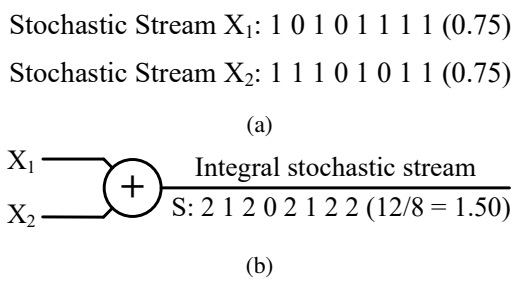

Fig. 20: (a) 0.75 in the unipolar representation and (b) An integral stochastic representation of 1.5.

and $\{S_{2h},\ S_{2l}\}$, and the values encoded in the sequences are $\{s_{1h},\ s_{1l}\}$ and $\{s_{2h},\ s_{2l}\}$ in the bipolar representation, respectively. We then have

$$s_1 + s_2 = \frac{s_{1h}}{s_{1l}} + \frac{s_{2h}}{s_{2l}} = \frac{s_{1h} \cdot s_{2l} + s_{1l} \cdot s_{2h}}{s_{1l} \cdot s_{2l} + 0}. \quad (12)$$

Therefore, the ESL adder in the bipolar representation can be implemented by three XNOR gates and two MUXes [see Fig. 18 (b)]. An SC divider is utilized to convert the ESL sequences into conventional SC sequences. The implementation of an ADDIE-based SC divider is introduced in [26] (see Fig. 19). The design follows a similar algorithm as used in the ADDIE-based PE.

ESL increases the complexity and overhead of the SC circuit; however, the random fluctuations in the divisor [i.e. p_l in (11)] significantly reduce the accuracy of the value of x, so it requires a relatively long sequence length to achieve an acceptable computational accuracy [40].

2) Integral SC: The integral SC is another method to extend the computational range [39]. In the integral SC, the real value is represented as the sum of the values encoded by multiple binary stochastic sequences when it is beyond the range of $[-1, +1]$. Fig. 20 shows an example of representing the value of 1.5 in the integral SC in the unipolar representation. Compared with the ESL with the fluctuations in the divisor, the integral SC requires a shorter sequence length, thus achieving higher computational performance. However, it incurs a larger area due to the more complex arithmetic circuits. The FPGA implementation of the integral SC NN achieves a nearly negligible inference accuracy loss, with 21.3% less energy consumption and 33.9% less area compared to an FP design.

A simplified integral SC has been introduced in [68]. Its main feature is to generate an integral sequence $S^* = s/n$, where s is the target value and n is a positive integer, and then set $S = S^* \times n$ when regenerating a sequence for computation. Because the value encoded in the integral sequence is reduced by n times, then this method also reduces the area of the circuit.

A new integral SC (NISC) has been proposed in [69]; the FSM-based $Btanh$ activation function is improved by combining multiple states into one to reduce the area and power dissipation. The NISC-based NN requires a $6\% - 64\%$ smaller area for different parameters in the $tanh$ function. The computation accuracy of the activation function is similar to a conventional SC design when the input is not larger than 0.6. Otherwise, it suffers a significantly higher accuracy loss.

B. Computation Performance Improvement

A PE is used to convert a stochastic sequence to a binary value. The conventional PE (see Fig. 5) requires long sequences for computation, so significantly increasing the latency and energy consumption of the SC NNs. A binary search based PE is introduced in [40] to overcome this limitation (see Fig. 21).

This design follows the same algorithm as used in conventional PEs; however, a binary search algorithm is utilized to reduce computational complexity. Triple modular redundancy (TMR) is utilized to reduce fluctuation errors, so the required sequence length is further reduced. This design requires 2.5%

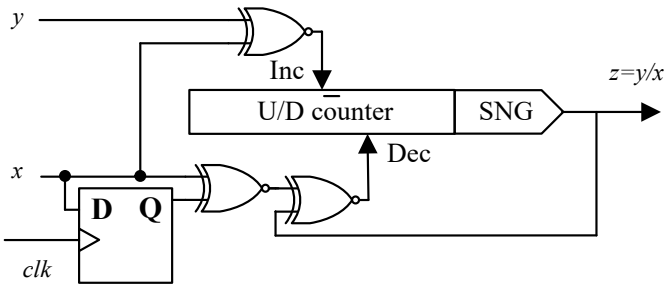

Fig. 19: Design of the ADDIE-based SC divider, consisting of an up-down counter, three XNOR gates, a D-flip-flop and an SNG [26]. x, y, and z are the values encoded in the sequences in the bipolar representation.

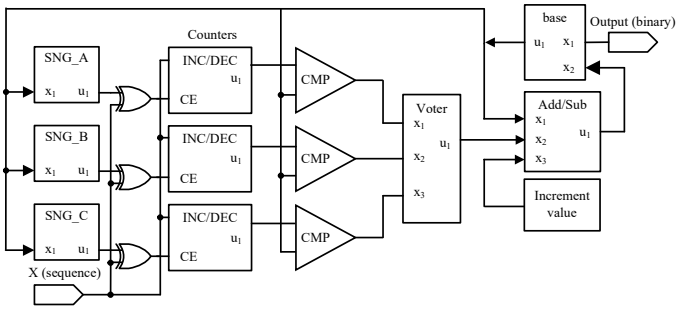

Fig. 21: Design of the TMR binary search based PE [40]. CMP represents comparators.

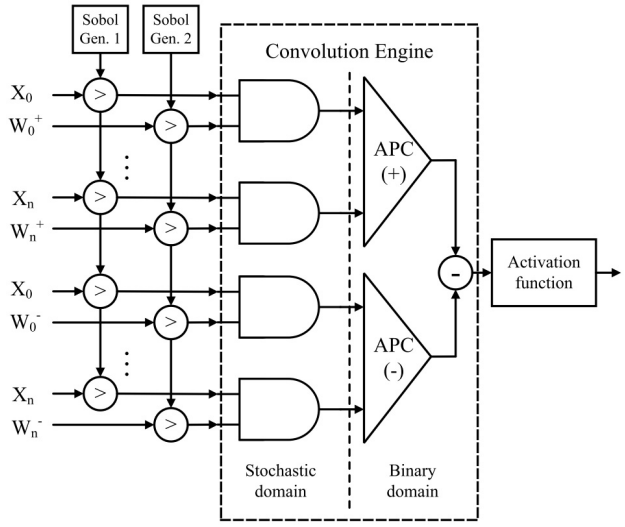

Fig. 22: A hybrid neuron in the convolutional layer, using Sobol sequences for multiplication [32]. The '>' symbols represent comparators.

of the sequence length of a conventional PE, so it significantly improves the performance. This technique is also used to improve the performance of the SC divider used in ESL circuits. The Simulation shows that the computational accuracy and performance of the circuit are substantially improved.

To maintain a high accuracy and reduce the sequence length, two types of low-discrepancy (LD) sequences, Halton and Sobol sequences, have been used to generate the stochastic bit streams [70], [71]. The use of LD sequences in SC results in a more accurate computation with a reduced sequence length compared to the use of conventional LFSR-generated pseudo-random sequences. Also, a stochastic multiplier using Sobol sequences takes roughly half of the sequence length required by Halton sequences to achieve a similar accuracy [72]. In [32], the Sobol sequence is used for the computation of the convolutional layers of a CNN, and especially, the multiplications. In the neuron design in Fig. 22, two Sobol sequence generators are used to produce random numbers for stochastic sequence generation. For a better accuracy, multiplications are implemented using the unipolar stochastic circuit (i.e., the AND gates) instead of the bipolar circuit. The products are then divided into the positives (+) and negatives (-) and set as the input signals for the APCs. Accumulations are implemented using binary circuits for the positive and negative products, followed by a binary activation function circuit. This design has been tested for handwritten digit recognition, based on the LeNet-5 topology [73]. It has been shown that at a reduced sequence length, a similar or better classification accuracy is obtained using this hybrid design with a higher energy efficiency compared with conventional SC implementations.

An improved SC encoding method has been proposed in [74], as shown in Fig. 23. Assume that a value x in $[0, 1]$ is encoded in 4-bit FxP representation with each bit being x_i, where $i = 0, 1, 2, 3$. In this method, the weights (or probabilities) of x_i are set to $\{1/16, 1/8, 1/4, 1/2\}$ by a

Fig. 23: An SNG based on FSM-MUX circuits [74].

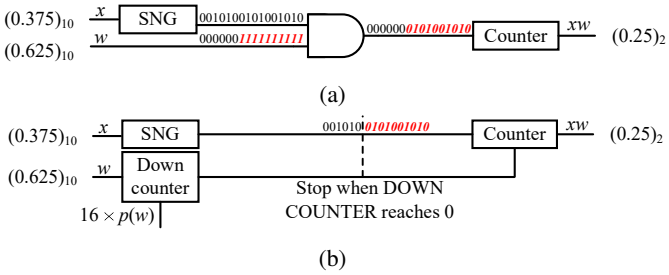

Fig. 24: The principle of the NSC multiplication. (a) Reordering the bits for the sequence w, (b) NSC multiplication, using a down counter to cut off bits in sequence x [74]. Assume that the sequence length is 16 bits, and the down counter is initialized to an integer for $16 \times p(w)$. $p(w)$ is the probability of w and the width of $p(w)$ is 4-bit in the binary representation.

16-state FSM to generate a sequence encoding x in the unipolar representation. Compared with the pseudo-random sequences generated by conventional SNGs, the probability of this sequence is more accurate, because it is determined by the weights of the bits in the binary representation.

The binary interfaced stochastic computing (BISC) is proposed to improve the performance of the SC by combining the conventional binary memory architecture with stochastic computing arithmetic circuits [33]. For example, the so-called new SC (NSC) multiplier uses counters to reduce the number of bits required in stochastic sequences for BISC [74]. The principle of the design is shown in Fig. 24. Note that in Fig. 24 (a), compared with conventional SC sequences, the sequence of w is reordered so that the 1's are placed continuously at the beginning of the bit stream (from the right to the left in Fig. 24). However, the result of the multiplication is unchanged when the stochastic bit streams are statistically uncorrelated after the reordering. It can be seen that the 0's in the stochastic sequence w and the corresponding bits in the sequence x make no contribution in the final outcome. Therefore, these bits can be skipped by using a down counter, as shown in Fig. 24 (b). The NSC multiplier significantly reduces the sequence length with increased area compared with conventional SC designs.

The randomness of a stochastic sequence is further reduced by the stochastic quantized (SQ) encoding in [75]. This method also uses continuous 1's in a sequence. Fig. 25 gives an example of SC multiplier using the SQ encoding. The value encoded in the SQ sequence P_i, where i = 1, 2, 3, 4, in the unipolar representation is quantized to one value in $\{1/4, 1/2, 3/4, 1\}$ using 2-bit quantization. Assume the value encoded in the sequence X is x, then the values encoded in the output sequences are exactly $\{1/4x, 1/2x, 3/4x, x\}$. The

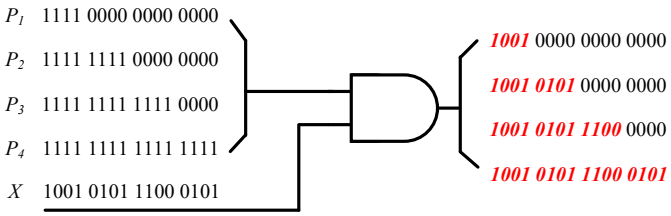

Fig. 25: Multiplication in SQ encoding. The sequences P_i (i = 1, 2, 3, 4.) encodes the values of $\{1/4, 1/2, 3/4, 1\}$ in the unipolar representation. X is a Bernoulli sequence.

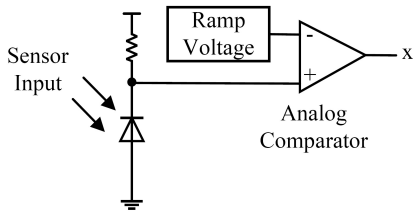

Fig. 26: Design of an analog-to-stochastic converter [58].

SQ sequences can be generated without using conventional SNGs. Therefore, this method achieves higher computation accuracy and higher hardware efficiency, compared to conventional SC designs. However, it is required that X is a Bernoulli sequence; otherwise, the computation accuracy can be significantly decreased. Nevertheless, the output sequences are no longer Bernoulli sequences, which may influence the computation in the following stages.

In [58], a sequence generator is proposed based on analog circuits (see Fig. 26). This design utilizes an analog-to-stochastic converter [76] to replace the SNG used in conventional SC circuits. A ramp-voltage signal is compared with an analog signal from sensors to generate stochastic sequences. The potential of a hybrid design consisting of SC, binary, and analog circuits is therefore proposed. Performance of this SC-CNN will be discussed in Section VI.

VI. Accuracy and Hardware Efficiency of SC NNs

Recently, SC designs have been proposed to implement multiple types of NNs, including MLPs, DBNs, CNNs, and RNNs. MLPs typically utilize a rather simplistic structure of neural networks for supervised learning [6]. Compared with MLPs, DBNs perform unsupervised learning, so they can be trained on unlabeled datasets with a higher complexity in hardware implementation [10]. CNNs are usually large-scale networks and can achieve a high inference accuracy in pattern and object recognition, whereas RNNs are effective in processing temporal datasets, so they are widely used in time-related applications such as voice or speech recognition.

A. Accuracy

Accuracies for inference in different SC NNs are reported in Table I for the Modified National Institute of Standards and Technology (MNIST) dataset [73] with details of the implementations. Note that if multiple configurations of a network are available in the technical literature, the structure and the sequence length are selected, such that they achieve the highest inference accuracy for the MNIST dataset. The missing information is represented by '–' in the table.

Most of the SC NNs incur less than 1% degradation in inference accuracy due to the computation accuracy loss in SC, compared with 32-bit FP implementations. The SC CNNs utilize the most complex network structure and achieve the highest inference accuracy ($\geqslant 98\%$). The SC-DBNs and SC-MLPs produce similar inference accuracy, between 94% and 99%, with a similar size of networks. Most of the SC NNs require no less than a 256-bit sequence length to achieve an acceptable inference accuracy. However, the Sobol CNN [32], the integral stochastic NN [39], and sign-magnitude SC (SM-SC) CNN [34] require significantly shorter sequence lengths: 8 bits, 16 bits, and 32 bits, respectively, thus evidencing the advantage of Sobol sequences and the use of improved encoding in SC NNs. Most of the NNs in Table I do not include the SC training components, except for the designs of [36], [40], [41], and [45].

Several SC NNs have been utilized to process more complex datasets (CIFAR-10 and ImageNet) using different structures (AlexNet, GoogleNet, and VGG), as shown in Table II. These results suggest that SC NNs are promising and potentially useful for complex machine learning applications.

B. Hardware Efficiency

Next, different SC implementations are considered with respect to their hardware efficiency. For the different evaluation methods used in the reference papers, the hardware efficiency in terms of area and energy consumption is discussed as follows and summarized in Table I.

1) SC-MLPs: MLPs are among the earliest applications of SC NNs. In [27], based on SC arithmetic circuits, an SC-MLP is implemented to solve the problem of magnetic ink character recognition (MICR) [77]. Computation is performed in the bipolar representation so the circuits are simplified compared to the designs in Fig. 14 [66]. The stochastic implementation achieves similar accuracy compared to a deterministic FP system with a higher hardware efficiency.

An ESL based SC-MLP is introduced in [67]. This SC-MLP endures higher noise levels compared to the binary design, hence showing a noise tolerance capacity in SC designs.

In the SC NN of [35], the $Btanh$ circuit is utilized to implement the activation function. The elimination of the near-zero layer weights is used to reduce the computation time. In addition, an energy-efficient RNG [78] is utilized to reduce the hardware cost of RNGs. Compared with a 9-bit FxP design, the SC NN with SNGs increases energy consumption by $3.0\times$. However, the SC NN without SNGs decreases the energy by 70.0%, compared with the same FxP design. So the SNGs account for a significant part of the energy consumption of the SC circuits. Compared with an FxP design, the SC-DNN without considering SNGs is $4.61\times$ faster while the NN including SNGs is $1.53\times$ slower.

In [40], an SC-MLP with both forward propagation and BP is implemented to solve the optical character recognition (OCR) problem using the MNIST and Street View House

Numbers datasets [79]. The ESL circuits are implemented to improve the computation accuracy in the inference and training processes. Thus, this design achieves only a 1% loss in inference accuracy with a significantly lower area and energy consumption, compared to an 8-bit FxP design.

In [41], the RNGs in the SC-GDC array can be shared so that only two RNGs are used for the circuit implementing the gradient descent algorithm. Due to the simple SC circuit and the short sequence lengths (1 bit per gradient), the signed SC-GDC array consumes about 10% of the energy and 25% of the time of the 16-bit FxP circuit and achieves about $55\times$ of the throughput per area of the FxP circuit.

2) SC-DBNs: DBNs are based on the fast greedy learning algorithm [10]. The design of the SC-DBNs incorporates several improvements compared to SC-MLPs, including the implementation of the reconfigurable network structure based on the restricted Boltzmann machine (RBM) in SC.

As an example, conventional SC adder/multipliers and an FSM-based $tanh$ circuit are utilized to implement an RBM based DNN in [80]. Based on this design, an FPGA implementation classifies a standard handwritten input image about $700\times$ faster than a software-based MATLAB implementation.

A reconfigurable structure is introduced to implement both the training and inference of DBNs [36]. An SC-based adaptive moment estimation (ADAM) circuits are utilized to adjust the learning rate of the network in the training process, such that the number of training epochs is significantly reduced. The SC-ADAM circuit thus reduces the energy consumption in the training process. This design requires 5.3%, 4.5%, 3.3% and 73.6% of the area, power, energy consumption and latency (per sample) of a pipelined 32-bit FP implementation with negligible accuracy loss on the MNIST dataset.

3) SC-CNNs: Recently, SC-CNNs have been proposed using different types of neurons in the convolutional layers [59], including the APC- and MUX-based neurons. By comparing the performances of these two implementations (based on the LeNet-5 CNN), it is shown that the SC CNN using MUX-based neurons requires a smaller hardware footprint but a longer sequence length, thus higher energy consumption to achieve a similar inference accuracy, compared to the APC-based counterpart. Therefore, the MUX-based design shows advantages in area-constraint embedded systems, while the APC-based design is more suitable for energy-constraint designs.

The design is further optimized in [61]. The SC ReLU circuit (see Fig. 12) is utilized to implement the activation function (9). The hardware efficiency of APC-based neurons is improved by replacing APCs with AxPCs (Fig. 8). For the LeNet-5 network, this design achieves 99.07% in inference accuracy, a 0.1% degradation, but with $4.1\times$, $6.5\times$ and $5.5\times$ improvements in throughput, area efficiency and energy efficiency compared to a previous design [65]. For the AlexNet [2] implementation, the SC-CNN achieves a top-5 accuracy of 80.48% on the ImageNet dataset [81] with significant improvement in throughput, area, and energy, compared to other existing NN platforms [82] [83] [84].

An SC LeNet-5 network is implemented and tested on the MNIST dataset in [32]. The Sobol sequence is utilized to improve the computation speed of SC multipliers. This design achieves $19\times - 30\times$ area reduction and energy saving, compared to FxP designs in the convolutional layers. The design achieves higher inference accuracy with shorter computational cycles ($3.1\% - 12.5\%$) compared with6 the use of conventional stochastic sequences. It shows that low-discrepancy sequences, specifically, the Sobol sequences, can be used in SC NN applications to achieve significant improvement in performance without reducing the computation accuracy.

In another SC-CNN design [45], an XNOR-based inner product is utilized to implement the convolutional layer. The multipliers are implemented by XNOR gates and the adders are implemented by MUXes. The SC average pooling circuits [see Fig. 17 (a)] is utilized in the pooling layers; the activation function is set to $tanh$ and is implemented by FSM circuits in CONV layers and fully-connected layers. In [85], the SNGs are shared, so the energy efficiency is improved by $5.3\times - 9.2\times$.

An SC-CNN is implemented based on the improved SC encoding method (see Fig. 23) and NSC multipliers (see Fig 24) in [74]. The down counters are shared among different multipliers to achieve higher hardware efficiency without accuracy degradation. With a similar structure to the binary design introduced in [86], the proposed SC-CNN achieves $29 - 44\%$ reduction in area-delay-product (ADP) and higher energy efficiency (by $23 - 29\%$ for CIFAR-10 and 10% for MNIST) compared with an FxP design. In [87], the complexity of the network structure is further increased to implement the AlexNet and GoogLeNet. The sequence lengths (or precisions) are set differently depending on applications to achieve high performances in ADPs. Overall, the SC-CNN achieves $34\% - 46\%$ reduction in ADP, or $52\% - 85\%$ increase in operations-per-area with less than 1% accuracy loss, compared to FP designs.

The stochastic-binary neural network (SB-NN) proposed in [58] is based on the LeNet-5 topology. The TFF-based adder [see Fig. 7 (c)] is introduced to improve computation accuracy as well as ignoring the auto-correlation in the input sequences. The analog input data is converted into stochastic sequences by the analog-to-stochastic converter (Fig. 26). The first CONV layer is implemented by SC while other layers are implemented by binary designs, forming an analog-SC-binary hybrid structure. In the simulations, the binary part of the NN is retrained to compensate for the precision losses caused by the SC circuits. As a result, the NN achieves a high inference accuracy, with 1.04% and 0.94% misclassification rates for 4-bit and 8-bit precisions. The area of this design is similar to the binary design at 8-bit precision but is $2\times$ larger than the binary design at 4-bit precision. However, the energy consumption of this SC NN is 81% and 10.2% of that of the 8-bit and 4-bit binary designs due to the lower power consumption.

4) SC-RNNs: Recently, an SM-SC RNN is implemented in [34]. For the MNIST dataset, this network achieves the same inference accuracy using 32-bit sequences (with no parallelization) as the conventional SC design with 1024-bit sequences, thus achieving $32\times$ improvement in the aspect of computation speed.

A detailed implementation of an SC LSTM-RNN is intro-

duced in [44]. Based on a hybrid-structured memory cell, this network achieves slightly lower inference accuracy (93.8%) than an FP design (94.9%) with a signal-to-noise ratio (SNR) of 20 dB. However, for an SNR of 5 dB, this design achieves 3.2% higher accuracy than the FP design, showing a higher noise tolerance. The synthesis results indicate the SC LSTM-RNN requires significantly lower area ($1.6\% - 2.3\%$) and energy consumption ($6.5\% - 10.9\%$) with $2\times$ computation latency, compared to the FP design.

VII. Comparison with BNNs/QNNs

Albeit based on different operational principles, SC NNs share the same design objectives as some other types of NNs proposed to improve the hardware and energy efficiency. For example, BNNs [20] and QNNs [21] have been developed for a better balance between accuracy and energy consumption. In a BNN, the layer weights and the intermediate computation results are converted from a real value to $+1$ or -1. Hence, the multiplications can be eliminated by utilizing XNOR operations. Moreover, the outputs of neurons are binarized and multiplied with the binarized layer weights during the forward propagation. These signals are also used to compute the local gradient, whereas the original real-valued parameters are used for updating the layer weights [20].

Reference [90] has shown that an SC NN and a BNN can be functionally interchanged without affecting the computational accuracy. Moreover, the energy consumptions of SC NNs and BNNs similarly increase (by the same order) with the network size.

In [40], an SC-MLP and a BNN are implemented by the same network structure and compared with respect to accuracy, area and energy consumption on the MNIST and SVHN datasets. The SC-MLP achieves slightly higher inference accuracy. In the BNN, batch normalization is performed and the layer weights are updated and stored at a full length (i.e., 8 bits) at the end of the BP. The SC-MLP requires a smaller area ($80.7\% - 87.1\%$) and lower energy dissipation (by approximately 20%) compared to the BNN.

Both SC NNs and BNNs achieve low hardware and power consumption because of the simpler arithmetic circuits compared to FP NNs. SC NNs can share the use of hardware and adopt several sequence length reduction methods to achieve low energy consumption. A high degree of parallelism is often required for a low latency in SC designs. SC implementations are more noise-tolerant than BNNs and FP NNs, whereas BNNs are better optimized in the literature. Table III briefly summarizes the performance comparison between SC NNs, BNNs, and FP NNs.

Binarization, or in general, quantization, can be integrated with SC for a better hardware utilization. In the SC quantized NN (SC-QNN) [75], the layer weights are quantized into 2-bit to 4-bit representations and encoded by stochastic quantized (SQ) bit-streams (see Fig. 25). The SC-QNN reaches a similar inference accuracy with $69\times$, $119\times$, and $10\times$ smaller area, power, and energy, respectively, compared with binary implementations. These results show that BNNs and QNNs can be viewed as highly optimized SC NNs with 1-bit sequences or sequences encoding quantized probabilities in the computation process. These implementations expand the usage of SC techniques in hardware for energy efficient NN designs.

VIII. Conclusion

The recent development of various types of SC NNs has been reviewed. Compared with FP and FxP implementations, SC NNs offer considerable advantages in area and energy consumption with comparable accuracy and higher noise tolerance. With a high degree of parallelism, an SC design can achieve a similar performance as a conventional binary design. These advantages make SC NNs a potentially competitive candidate in resource-limited applications.

In spite of the recent advances, there are still challenges for SC NNs to be implemented in the industry. It would be imperative to establish general design methodologies and testing standards to clearly evaluate the reliability, performance and hardware efficiency of various SC NNs. Challenges also exist in overcoming the stochasticity, further improving the classification accuracy and at the same time, maintaining a high energy efficiency. Nevertheless, SC designs have been utilized to implement large networks such as the AlexNet and GoogLeNet. Hence, SC provides an alternative, scalable solution to NN implementations with a potential for efficent machine learning.

References

[1] J. Mao and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 296–317, 1995.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[3] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 4, pp. 589–600, 1998.

[4] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

[5] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[6] S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA, 2009, vol. 3.

[7] Valle, Maurizio, Caviglia, D. D, and G. M. Bisio, "An experimental analog VLSI neural network with on-chip back-propagation learning," *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 3, pp. 231–245, 1996.

[8] H. Hikawa, "A digital hardware pulse-mode neuron with piecewise linear activation function," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1028–1037, 2003.

[9] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[10] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[11] V. Nair and G. E. Hinton, "3D object recognition with deep belief nets," in *Advances in Neural Information Processing Systems*, 2009, pp. 1339–1347.

[12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[13] P. Y. Simard, D. Steinkraus, J. C. Platt *et al.*, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

TABLE I: Comparison of SC Networks in Inference Accuracy and Hardware Efficiency

Network	Reference	Structure Seq. length (bit)	Inference accuracy for MNIST (%)		Hardware efficiency (% compared to FxP)		Training in SC included (Y/N?)
			SC	32-bit FP	Area	Energy	
MLP	SC Btanh NN [35]	784-100-200-10 1024	97.59	97.77	90 (w/ RNG) 50 (w/o RNG)	300 (w/ RNG) 30 (w/o RNG)	N
	SC-MLP [40]	784-200-100-10 4096	97.95	99.27	40.7	38.0	Y
	SC-GDC [41]	784-128-128-10 1 (per gradient)	97.03	97.47	7.3[1]	10	Y
DBN	FPGA-DBN [43]	– 4096	94.1	94.2	–	–	
	SC-RBM [80]	784-100-200-10 4096	97.86	98.0	–	–	N
	SC-DBN [36]	784-100-200-10 4096	99.15	99.27	29.3	33.0	Y
	Integral SC NN [39]	784-300-600-10 16	97.73	97.7	66.1	78.7	N
	FPGA-RBM [88]	784-100-200-10 1024	94.28	–	–	–	N
CNN	Budget-Driven DCNN [59]	LeNet-5 256	98.00	–	–	–	N
	HEIF [61]	LeNet-5 –	99.07	99.17	–	–	N
	Sobol CNN [32]	LeNet-5 8	99.20	99.19	5.3	3.6	N
	SC learning system [45]	LeNet-5 32768	98.49	98.46	–	–	Y
	SC CNN [85]	LeNet-5 –	99.19	99.23	–	25 nJ (LFSR) 8 nJ (MTJ-SNG)	N
	NSC CNN [74]	– –	>99	>99	50	29	N
	DPS CNN [87]	– –	98.26	99.04	55[2]	–	N
	SM-SC CNN [34]	– 32	98.9	98.9	–	–	N
	SB-NN [58]	LeNet-5 256	99.06	99.11 (8-bit)	100	81	N
RNN	SM-SC RNN [34]	– 1024	99	99	111.2 (w/ SNG) 18.4 (w/o SNG)	–	N

[1] Only weight updating is implemented by SC circuits in the training process.
[2] This number is estimated from the 55% ADP in [87] by assuming that the SC and FxP implementations achieve the same performance (i.e., delay).

TABLE II: Inference Accuracy for SC NN Applications with Higher Complexities

Design Dataset	Network structure	Inference accuracy (%)	Hardware efficiency (vs. FxP)
HEIF [61] ImageNet	AlexNet	80.48 (top-5)	36.5% in area 0.6% in energy (vs. [83])
SC CNN [85] CIFAR-10	Customized CNN[1]	83.57	8-25 nJ
DPS CNN [87] ImageNet	AlexNet	79.99 (top-5)	55% ADP
	GoogleNet	88.44 (top-5)	
	VGG	82.47 (top-5)	
SkippyNN [89] ImageNet	AlexNet	80 (top-5)	1.2x speedup 37% in energy
	VGG	90 (top-5)	
SC LSTM-RNN [44] TIMIT	Customized RNN[2]	71.9	28% in area 83% in energy

[1] The structure of the customized CNN is 2Conv-1Max-2Conv-1Max-2FC.
[2] The structure of the customized RNN is 12-(250, 250)-(250, 250)-48.

TABLE III: Performance Comparison of NNs

	SC NNs	BNNs	FP NNs
Hardware cost	low	low	high
Power consumption	low	low	high
Energy consumption	low in most cases	low	high
Latency	high	low	high
Noise tolerance	high	low	low

[14] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the ISCA*, 2014.

[15] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 6645–6649.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems*, 2011, pp. 2348–2356.

[18] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[19] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013, pp. 1058–1066.

[20] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *NIPS*, 2016, pp. 4107–4115.

[21] ——, "Quantized neural networks: Training neural networks with low

precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[22] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5918–5926.

[23] A. R. Omondi and J. C. Rajapakse, *FPGA implementations of neural networks*. Springer, 2006, vol. 365.

[24] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[25] B.R.Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Springer, 1969, pp. 37–172.

[26] B. D. Brown and H. C. Card, "Stochastic neural computation. I. computational elements," *IEEE TC*, vol. 50, no. 9, pp. 891–905, 2001.

[27] ——, "Stochastic neural computation. II. soft competitive learning," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 906–920, 2001.

[28] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *DAC*, 2015, p. 59.

[29] A. Alaghi and J. P. Hayes, "Dimension reduction in statistical simulation of digital circuits," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, 2015, pp. 1–8.

[30] P. Li, W. Qian, M. D. Riedel, K. Bazargan, and D. J. Lilja, "The synthesis of linear finite state machine-based stochastic computational elements," in *IEEE ASP-DAC*, 2012, pp. 757–762.

[31] G. B. Orr and K.-R. Müller, *Neural networks: tricks of the trade*. Springer, 2003.

[32] S. R. Faraji, M. H. Najafi, B. Li, K. Bazargan, and D. J. Lilja, "Energy-efficient convolutional neural networks with deterministic bit-stream processing," in *Design, Automation, and Test in Europe (DATE)*, 2019.

[33] H. Sim, D. Nguyen, J. Lee, and K. Choi, "Scalable stochastic-computing accelerator for convolutional neural networks," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 696–701.

[34] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, "Sign-magnitude SC: getting 10X accuracy for free in stochastic computing for deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[35] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *DAC*, 2016, p. 124.

[36] Y. Liu, Y. Wang, F. Lombardi, and J. Han, "An energy-efficient online-learning stochastic computational deep belief network," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 454–465, 2018.

[37] Y. Ji, F. Ran, C. Ma, and D. J. Lilja, "A hardware implementation of a radial basis function neural network using stochastic logic," in *DATE*, 2015, pp. 880–883.

[38] J. L. Rosselló, V. Canals, and A. Morro, "Probabilistic-based neural network implementation," in *IEEE IJCNN*, 2012, pp. 1–7.

[39] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[40] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Transactions on Computers*, vol. 67, no. 9, pp. 1273–1286, 2018.

[41] S. Liu, H. Jiang, L. Liu, and J. Han, "Gradient descent using stochastic circuits for efficient training of learning machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2530–2541, 2018.

[42] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *IEEE ICCD*, 2016, pp. 678–681.

[43] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou, "FPGA implementation of a deep belief network architecture for character recognition using stochastic computation," in *IEEE CISS*, 2015, pp. 1–5.

[44] Y. Liu, L. Liu, F. Lombardi, and J. Han, "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, accepted, 2019.

[45] A. Ren, Z. Li, Y. Wang, Q. Qiu, and B. Yuan, "Designing reconfigurable large-scale deep learning systems using stochastic computing," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–7.

[46] G. J. Tortora and B. H. Derrickson, *Principles of anatomy and physiology*. John Wiley & Sons, 2008.

[47] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Conference Record of the Thirty-Ninth Asilomar Conference onSignals, Systems and Computers, 2005.* IEEE, 2005, pp. 713–717.

[48] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, 2011, pp. 154–161.

[49] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, "Stochastic circuit design and performance evaluation of vector quantization for different error measures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3169–3183, 2016.

[50] Y.-N. Chang and K. K. Parhi, "Architectures for digital filters using stochastic computing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 2697–2701.

[51] K. K. Parhi and Y. Liu, "Architectures for IIR digital filters using stochastic computing," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2014, pp. 373–376.

[52] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, "Design, evaluation and fault-tolerance analysis of stochastic FIR filters," *Microelectronics Reliability*, vol. 57, pp. 111–127, 2016.

[53] J. Han, H. Chen, J. Liang, P. Zhu, Z. Yang, and F. Lombardi, "A stochastic computational approach for accurate and efficient reliability evaluation," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1336–1350, 2012.

[54] P. Zhu, J. Han, L. Liu, and F. Lombardi, "A stochastic approach for the analysis of dynamic fault trees with spare gates under probabilistic common cause failures," *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 878–892, 2015.

[55] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM TECS*, vol. 12, no. 2s, p. 92, 2013.

[56] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515–1531, 2017.

[57] P.-S. Ting and J. P. Hayes, "Stochastic logic realization of matrix operations," in *2014 17th Euromicro Conference on Digital System Design*. IEEE, 2014, pp. 356–364.

[58] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 13–18.

[59] J. Li, A. Ren, Z. Li, C. Ding, B. Yuan, Q. Qiu, and Y. Wang, "Towards acceleration of deep convolutional neural networks using stochastic computing," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 115–120.

[60] Y. Liu, Y. Wang, F. Lombardi, and J. Han, "An energy-efficient stochastic computational deep belief network," in *DATE, 2018*. IEEE, 2018, pp. 1175–1178.

[61] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu *et al.*, "HEIF: Highly efficient stochastic computing based inference framework for deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[62] R. T. Farouki and V. Rajan, "On the numerical condition of polynomials in Bernstein form," *Computer Aided Geometric Design*, vol. 4, no. 3, pp. 191–216, 1987.

[63] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Proceedings of the 45th annual Design Automation Conference*. ACM, 2008, pp. 648–653.

[64] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, "Neural network classifiers using a hardware-based approximate activation function with a hybrid stochastic multiplier," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 1, p. 12, 2019.

[65] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 405–418, 2017.

[66] J. A. Dickson, R. D. McLeod, and H. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," in *IEEE International Conference on Neural Networks*. IEEE, 1993, pp. 711–716.

[67] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 551–564, 2016.

[68] R. Hu, B. Tian, S. Yin, and S. Wei, "Optimization of softmax layer in deep neural network using integral stochastic computation," *Journal of Low Power Electronics*, vol. 14, no. 4, pp. 475–480, 2018.

[69] S.-I. Chu, C.-E. Hsieh, and Y.-J. Huang, "Design of FSM-based function with reduced number of states in integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[70] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 76.

[71] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel Sobol sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1326–1339, 2018.

[72] ——, "Energy efficient stochastic computing with sobol sequences," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 650–653.

[73] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[74] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[75] B. Li, M. H. Najafi, B. Yuan, and D. J. Lilja, "Quantized neural networks with new stochastic multipliers," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2018, pp. 376–382.

[76] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester, "Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering," in *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*. IEEE, 2014, pp. 1–4.

[77] S. Mori, H. Nishida, and H. Yamada, *Optical character recognition*. John Wiley & Sons, Inc., 1999.

[78] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 256–261.

[79] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.

[80] B. Li, M. H. Najafi, and D. J. Lilja, "Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 36–41.

[81] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on Computer Vision and Pattern Recognition*. Ieee, 2009, pp. 248–255.

[82] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[83] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 609–622.

[84] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[85] J. Yu, K. Kim, J. Lee, and K. Choi, "Accurate and efficient stochastic computing hardware for convolutional neural networks," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 105–112.

[86] A. Rahman, J. Lee, and K. Choi, "Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1393–1398.

[87] H. Sim, S. Kenzhegulov, and J. Lee, "Dps: Dynamic precision scaling for stochastic computing-based deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 13.

[88] B. Li, M. H. Najafi, and D. J. Lilja, "An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2015, pp. 68–69.

[89] R. Hojabr, K. Givaki, S. Tayaranian, P. Esfahanian, A. Khonsari, D. Rahmati, and M. H. Najafi, "Skippynn: An embedded stochastic-computing accelerator for convolutional neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 132.

[90] Y. Wang, Z. Zhan, J. Li, J. Tang, B. Yuan, L. Zhao, W. Wen, S. Wang, and X. Lin, "On the universal approximation property and equivalence of stochastic computing-based neural networks and binary neural networks," *arXiv preprint arXiv:1803.05391*, 2018.