An Energy-Efficient Online-Learning Stochastic Computational Deep Belief Network

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

***Abstract*—Deep neural networks (DNNs) are effective machine learning models to solve a large class of recognition problems, including the classification of nonlinearly separable patterns. The training of DNNs is, however, particularly difficult due to the large size and high energy consumption of the networks. Recently, stochastic computation (SC) has been considered to implement DNNs to reduce the hardware cost. However, it requires a large number of random number generators (RNGs) and long stochastic sequences that lower the energy efficiency of the network. To overcome these limitations, we propose the design of an energy-efficient deep belief network (DBN) with online learning capacity based on stochastic computation. In the SC-DBN, a reconfigurable structure is utilized to implement the fast greedy learning algorithm and an adaptive moment estimation (ADAM) circuit is designed to improve the speed of the training process. An approximate SC activation unit (A-SCAU) is further designed to implement different types of activation functions in the neurons. The A-SCAU is immune to signal correlations, so the RNGs can be shared among all neurons in the same layer with no accuracy loss. The area and energy of the proposed design are less than 5.5% and 3.7% (or 29.3% and 33.3%) of a pipelined 32-bit floating-point (or an 8-bit fixed-point) implementation. The proposed SC-DBN design achieves a higher classification accuracy compared to the fixed-point implementation. The accuracy is in a range of 0.12% to 0.37% lower than the floating-point design with a significantly lower (or slightly higher) energy consumption than the pipelined (or non-pipelined) circuit for both online learning and inference processes.**

***Index Terms*—stochastic computing, deep belief network, rectifier linear unit, cognitive computing.**

I. Introduction

As a type of deep neural networks (DNNs), a deep belief network (DBN) substantially improves the performance of conventional artificial neural networks such as a multilayer perceptron [1]. A DBN can perform unsupervised learning and solve nonlinearly separable pattern recognition problems such as the classification of objects [2], speech [3] and handwritten characters [4]. In the training process of DBNs, the fast greedy learning algorithm is used to attain a faster computation than the commonly-used gradient descent algorithm and a higher network depth can be achieved in DBNs than in conventional multilayer perceptrons. The DBN can also process unlabeled samples in a dataset. However, the size of a DBN and the number of parameters increase rapidly with the complexity of a problem. A DBN requires a large memory for the weights due to its low weight sharing rate. Therefore, it results in a lower performance for image classification than deep convolutional neural networks (DCNNs) [5]. Recently, the depth of DBNs has been exceeded by long short-term memory recurrent neural networks (LSTM RNNs) which show significant advantages in time-related problems, such as speech recognition and prediction [6]. Nevertheless, a DBN is useful due to its unsupervised learning ability. The relatively easy-to-implement structure also makes it suitable as a platform to evaluate the performance of new design techniques such as approximate computing and stochastic computing (SC). However, an implementation of large DBNs requires a large hardware and a high energy consumption. Hence, it is difficult to implement a machine learning algorithm using a DBN on a resource-limited system such as a mobile device or an embedded system. It has become imperative to develop efficient hardware design for implementing a DBN at a small circuit area and low power consumption.

The recent resurgence of SC provides such an opportunity [7] [8]: an SC circuit reduces the hardware footprint of many fundamental arithmetic circuits, such as adders, subtractors [9] [10] and multipliers [11] [12]. The hyperbolic tangent ($tanh$) and exponential functions can be implemented by linear finite state machines (LFSMs) [13]. Recently, SC designs have been utilized to implement radial basis function neural networks [14], a multilayer perceptron [15], a convolutional neural network [16], a DBN [17] and other types of DNNs [18] [19]. In these designs, the neural networks are pre-trained to perform the nonlinear classification in hardware. As a result, these networks are not applicable to problems that require real-time or online learning.

In spite of the simple SC circuits, stochastic number generators (SNGs), consisting of random number generators (RNGs) and comparators, incur a large area and high power consumption [18] [19], thus reducing the energy efficiency of an SC design. Moreover, because different types of activation functions are needed for various requirements in the training process, the performance of SC-based DNNs is limited as it is difficult to reconfigure the activation function without re-implementing the design.

In this paper, a stochastic computational DBN (SC-DBN) is proposed to overcome the above limitations. An approximate SC activation unit (A-SCAU) is proposed to implement different types of activation functions such as the sigmoid,

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada

Y. Liu, and J. Han are with Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 1H9 Canada (email: {yidong1, jhan8}@ualberta.ca).

Y. Wang is with Electrical Engineering and Computer Science Department, Syracuse University, Syracuse, NY 13244, USA (email: ywang393@syr.edu)

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

the rectifier linear and the pure line functions. In the SC-DBN, the use of RNGs is shared among all neurons in the same layer. Therefore, the circuit area and energy consumption are significantly reduced. The Modified National Institute of Standards and Technology (MNIST) dataset is used for the evaluation of the proposed design. Some preliminary results have been published in [20]. As a significant extension, this paper presents an improved SC-DBN design with online learning capacity. It makes the following novel contributions:

- A reconfigurable structure of the SC-DBN is proposed to implement the fast greedy learning algorithm. The layer weights are adaptively updated according to the learning samples, thus it is capable of performing real-time online learning.
- The adaptive moment estimation (ADAM) algorithm is implemented in SC circuits. The energy consumption and latency of the training process are reduced by 74.8% and 65.2% compared to the SC-DBN without the ADAM circuit.
- For both pre-trained and online learning implementations, the SC-DBN achieves a smaller area, lower power and energy consumption with a similar accuracy and computation speed compared to conventional pipelined floating- and fixed-point implementations.

The remainder of this paper is organized as follows. Section II introduces the background for the learning algorithms used in a DBN and stochastic logic. Section III presents the proposed design. Section IV shows the application and simulation results. Section V concludes the paper.

II. Review

A. The structure of DBNs

A DBN consists of one input layer, multiple hidden layers and one output layer (Fig. 1). One of the most widely-used learning algorithms for a DBN is the fast greedy learning algorithm [1]. In this algorithm, the training process is divided into unsupervised and supervised phases. During the unsupervised phase, each pair of layers in the network forms an encoder-decoder pair. The layers are trained as restricted Boltzmann machines (RBMs) [21]. The neurons in the encoder encode the input data, whereas the neurons in the decoder decode the computed results. By comparing the decoded result with the original input, the RBM adjusts the layer weights for each training process.

In the encoding process, assume that the input data are given by a row vector $\boldsymbol{X}$ with D dimensions and the encoder in the current layer consists of E neurons; x_j is then the j^{th} dimensional value in $\boldsymbol{X}$, and $\boldsymbol{W}$ is the matrix of layer weights with w_{ij} denoting the weight for x_j and the i^{th} neuron ($i = 1, 2, ..., E$). Assume the output of the encoder is a row vector $\boldsymbol{Y_E}$ with E dimensions, the computed result of the i^{th} neuron in the encoder is given by

$$y_i^e = \varphi(\sum_{j=1}^{D} x_j \cdot w_{ij}),\ i = 1, 2, ..., E, \tag{1}$$

where $\varphi(\cdot)$ is the activation function [1].

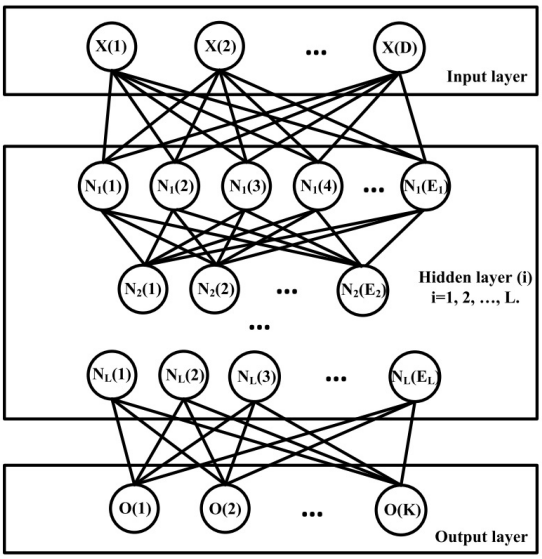

Fig. 1. A DBN consisting of one input layer with D neurons, L hidden layers with each layer consisting of E_i neurons ($i = 1, 2, ..., L$) and one output layer with K neurons.

The encoder computes the positive part of the difference in the updated layer weight, as

$$\boldsymbol{\delta_P} = \boldsymbol{X^T Y_E}, \tag{2}$$

where $\boldsymbol{X^T}$ is the transpose of $\boldsymbol{X}$.

The decoder is used to convert the output of the encoder $\boldsymbol{Y_E}$ back to a D-dimensional signal, so it consists of D neurons. The output of the decoder $\boldsymbol{Y_D}$ is computed from $\boldsymbol{Y_E}$ and the layer weight $\boldsymbol{W^T}$, the computed result of the i^{th} neuron in the decoder is given by

$$y_i^d = \varphi(\sum_{j=1}^{E} y_j^e \cdot w_{ij}^T), \tag{3}$$

where $i = 1, 2, ..., D$ is the index to each neuron in the decoder and $j = 1, 2, ..., E$ is the index to $\boldsymbol{Y_E}$. Note that the layer weights $\boldsymbol{W^T}$ are from the transpose of $\boldsymbol{W}$. The decoded result $\boldsymbol{Y_D}$ is sent back to the encoder to generate an encoded signal $\boldsymbol{Y_{E_2}}$. The computed result of the k^{th} signal in $\boldsymbol{Y_{E_2}}$ is given by

$$y_k^{e_2} = \varphi(\sum_{j=1}^{D} y_j^d \cdot w_{kj}), \tag{4}$$

where $k = 1, 2, ..., E$. The decoder computes the negative part of the difference in the updated layer weight as

$$\boldsymbol{\delta_N} = \boldsymbol{Y_D^T Y_{E_2}}, \tag{5}$$

where $\boldsymbol{Y_D^T}$ is the transpose of $\boldsymbol{Y_D}$. At the completion of this process, the layer weights at epoch t are updated on the basis of the positive and negative parts of the difference, i.e.

$$\boldsymbol{W(t)} = \mu \boldsymbol{W(t-1)} + \varepsilon(\boldsymbol{\delta_P} - \boldsymbol{\delta_N}), \tag{6}$$

where μ and ε are the learning rates, $\mu, \varepsilon \in (0, 1)$ [1]. This process is known as the one-time Gibbs sampling. The Gibbs sampling is repeated until either the maximum allowed number

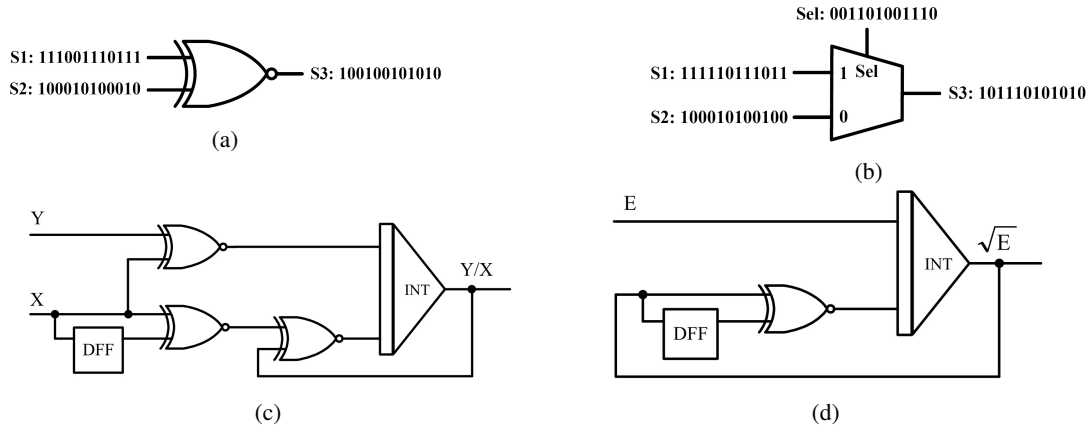

Fig. 2. (a) A bipolar stochastic multiplier: $P(S1) = 1/2, P(S2) = -1/3$, and $P(S3) = P(S1) \cdot P(S2) = -1/6$. (b) A stochastic adder: $P(S1) = 2/3, P(S2) = -1/3$, and $P(S3) = 0.5 \times (P(S1) + P(S2)) = 1/6$. (c) A bipolar stochastic divider [9], with INT denoting an integrator. (d) An SC square root circuit [7].

of samplings is reached, or the value of $\boldsymbol{\delta_P - \delta_N}$ is lower than a pre-determined threshold [1].

After the unsupervised phase, the supervised phase is implemented to adjust the layer weights based on the backward propagation algorithm [22].

For inference, assume that the number of neurons in layer $l-1$ and l are M and E, w_{ij}^l denotes the weight between neuron j in layer $l-1$ and neuron i in layer l. The output signal of neuron i in layer l at epoch t, $y_i^l(t)$, is given by

$$y_i^l(t) = \varphi(\sum_{j=1}^{M} y_j^{l-1}(t) \cdot w_{ij}^l),\ i = 1, 2, ..., E, \tag{7}$$

B. Activation function

In a DBN, different activation functions can be utilized for various requirements in the training process. One of the most widely used activation function is the sigmoid function [22], defined as

$$\varphi(x) = \frac{1}{1 + exp(-x)} = \frac{1}{2}(tanh\frac{x}{2} + 1). \tag{8}$$

Recently, deep sparse rectifier neural networks (DSRNNs) have been proposed to improve the performance of conventional DBNs [23]. In a DSRNN, the activation function is given by the rectifier linear function

$$\varphi(x) = min(1, max(0, x)). \tag{9}$$

A rectifier linear unit (ReLU) allows a network to eliminate the random fluctuation generated during the Gibbs sampling process [23]. This unit requires a simple circuit by avoiding the implementation of a complex activation function.

Another widely used activation function is the pure line function [22], defined as

$$\varphi(x) = min(1, max(-1, x)). \tag{10}$$

C. Adaptive moment estimation (ADAM)

In a DBN, the backward propagation algorithm is performed in multiple epochs. In each epoch, the network is trained on the training dataset. The backward propagation requires multiple epochs for convergence, resulting in high latency and energy consumption.

Recent researches have shown that the stochastic optimization methods, (including the adaptive subgradient method (AdaGrad) [24] and adaptive moment estimation (ADAM) [25]) can significantly reduce the number of epochs in the training process by adjusting the learning rates, thereby improving the energy efficiency of the neural networks.

Considering the computational complexity and overall performance, ADAM is considered as an improved stochastic optimization method. In ADAM, assume α is a pre-determined step size, $\beta_1 \in [0, 1)$ and $\beta_2 \in [0, 1)$ are the exponential decay rates and $f(\theta)$ is the loss function with parameter θ and g_t is the gradient. Let the moment vector m_t, v_t and the computation time step t initialized to 0. For each time step, the parameter θ is updated by:

$$\begin{cases} t = t + 1, \\ g_t = \bigtriangledown_\theta f_t(\theta_{t-1}), \\ m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t, \\ v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2, \\ \hat{m}_t = m_t / (1 - \beta_1^t), \\ \hat{v}_t = v_t / (1 - \beta_2^t), \\ \theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon). \end{cases} \tag{11}$$

The typical parameter values are given by $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ as recommended in [25].

D. Stochastic logic elements

In SC, assume that there are a 1's in a random binary bit stream with a length of b bits; the bit stream encodes the value of a/b within $[0, 1]$ in the unipolar representation or $(2a - b)/b$ within $[-1, 1]$ in the bipolar representation [7] [8]. Some fundamental computational elements can be implemented by simple circuits. For example, a bipolar multiplier

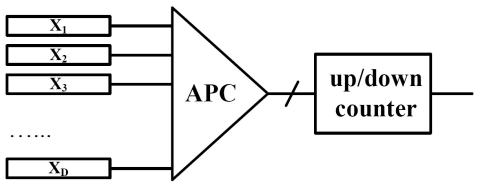

Fig. 3. An SC implementation of the $Btanh$ function, consisting of an accumulative parallel counter (APC) and an up/down counter [18]. The input sequence is D dimensional and X_j is the $j^{th}(j = 1, 2, ..., D)$ input bit stream.

is implemented by an XNOR gate (Fig. 2 (a)) and an adder is implemented by a multiplexer with the select signal encoding a probability of 0.5 (Fig. 2 (b)) [9]. Fig. 2 (c) and (d) show the designs of a bipolar stochastic divider and a square root circuit. The designs are based on integrators (denoted by INT), which can be implemented by an up/down counter and an SNG [7]. Compared to conventional binary designs, the area and power consumption of these simple stochastic circuits are significantly smaller.

A linear finite state machine (LFSM) is another useful stochastic circuit that can be used to implement the $tanh$ function in neural networks [9]. As per (8), the sigmoid function can be implemented by stochastic multipliers, adders and the $tanh$ function. A bounded random walking based $tanh$ ($Btanh$) function can be implemented by an accumulative parallel counter (APC) and an up/down counter (Fig. 3) [18]. This circuit achieves a high accuracy at a low latency due to the high level of parallelization in computation. Unfortunately, the computed result of this circuit is severely affected by correlations between the input sequences. Therefore, a straightforward use of shared RNGs for generating the input sequences would significantly decrease the accuracy of the computation.

III. Design of the SC-DBN

A. Overall structure

An SC-DBN structure is proposed to implement the learning and inference processes. The number of neurons in each layer and the values of weights are both reconfigurable in the proposed structure.

The proposed SC-DBN structure consists of six components: an encoder-decoder pair, a layer weight updater, output converters, input converters, weight buffers and data buffers (Fig. 4). The layer weights and internal data are stored in buffers as binary values. The encoder-decoder pair and the layer weight updater are based on SC designs. Every time a training process begins, the binary values are converted into stochastic sequences by the output converters. Then, the encoder-decoder pair reconfigures the structure of the current layer and performs the fast greedy learning algorithm in SC circuits. Following the training process, the layer weights are updated by the layer weight updater. As per (6), the layer weight updater is implemented by SC adders, subtractors and multipliers. The updated layer weights are then converted into binary values by the input converters and stored in the weight buffers. At each epoch, the encoder processes all samples and stores the intermediate results in the data buffers. For inference, the data buffers store the output signals of the neurons in each layer. The SC-DBN computes the output signals layer-by-layer based on the stored data.

In the Gibbs sampling process, the computation in the next encoder-decoder pair pauses until the computation in the current pair is completed; therefore, only one RBM is active and all the other RBMs are inactive during the training process. It is highly inefficient to implement this process layer-by-layer in hardware; so, the encoder-decoder pairs in the reconfigurable SC-DBN structure are reused to implement each layer in the SC-DBN. Therefore, it must be able to implement the largest layer in the network. That is, the number of neurons in the encoder-decoder pair is the same as the number of neurons in the largest layer of the network. For example, 784 neurons are needed in the encoder-decoder pair for the DBN with the configuration of 784-400-200-10; it would require 1394 neurons in a conventional structure.

B. Encoder-decoder design

The designs of the encoder and decoder are shown in Fig. 5. An encoder includes five components: two SNG arrays, two SC multiplier array and an A-SCAU array (Fig. 5 (a)). The SNG arrays convert the binary input signals and the layer weights to stochastic sequences.

(1) is implemented by an SC multiplier array and an A-SCAU array. Another multiplier array is used to compute the positive part of the difference in the updated layer weights as per (2). The MUX is used to select the input signals to the SC multiplier array between the input data $\boldsymbol{X}$ and the output signal $\boldsymbol{Y_D}$ of the decoder.

As per (3), (4) and (5), the structure of the decoder (Fig. 5 (b)) is similar to that of the encoder. The transpose of the layer weight matrix is computed by the decoder.

After the positive and negative parts of the difference are obtained, the layer weights are updated by the layer weight updater as per (6).

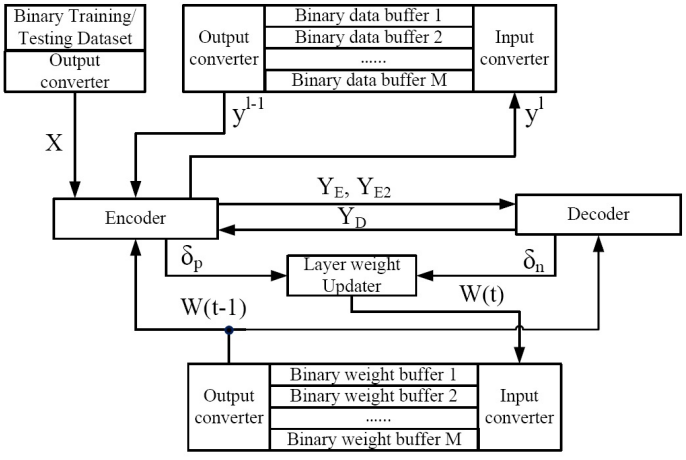

Fig. 4. Design of the reconfigurable SC-DBN structure, with signals following the same definitions in (2) to (7).

C. Design of the reconfigurable A-SCAU

The A-SCAU consists of an accumulative parallel counter (APC), a linear approximation unit (LAU), an RNG and a comparator (Fig. 6). The D-dimensional input sequences of the A-SCAU ($K_i,\ i = 1, 2, ...D$) are generated by the SC multiplier arrays in Fig. 5. The A-SCAU first computes the sum of the values encoded in the sequence K_i in the bipolar representation (k_i), following

$$x = \sum_{i=1}^{D} k_i,\ i = 1, 2, ..., D. \tag{12}$$

x is given by the output of the APC and serves as the input to the LAU. The LAU then computes different activation functions such as (8), (9) and (10). An activation function implemented by the LAU has the generalized form of

$$\psi(x) = min(1, max(p, \frac{1}{r}x + s)), \tag{13}$$

where p, r and s are parameters that can be configured to implement different functions.

For the sigmoid function (8), for example, the output range is $[0, +1]$. If $x = 0$, $\psi(x) = \varphi(x) = 1/2$. Therefore, p and s are set to 0 and $1/2$, respectively, and a search is conducted to find the optimal value of r. Fig. 7 shows the mean squared error (MSE) between the computed results by the sigmoid function and (13) when r varies in $[+2, +10]$ with a step size of 0.01. As can be seen, $r = 5.27$ leads to the minimum MSE, 6.16×10^{-4}, between $\psi(x)$ and $\varphi(x)$. The value of r is set to 4 to simplify the hardware implementation. Hence, (8) is approximated by

$$\psi(x) = min(1, max(0, \frac{1}{4}x + \frac{1}{2})). \tag{14}$$

As a result, the sigmoid function is approximated by using the configuration $p = 0, r = 4$ and $s = 1/2$ in the LAU.

The ReLU function (9) can be directly implemented by the LAU with the configuration $p = 0, r = 1$ and $s = 0$. The pure line function (10) is implemented by the configuration $p = -1$, $r = 1$ and $s = 0$.

Note that the LAU implements an approximate model of the sigmoid function but accurate models of the rectifier linear and pure line functions. Fig. 8 shows the simulation results of the A-SCAU with different configurations of the LAU. The range of the signal x in (12) is set to $[-10, +10]$. With a sequence length of 4096 bits, the MSEs are 1.1×10^{-3}, 6.1×10^{-4} and 8.9×10^{-4}; the maximum errors are 0.076, 0.051 and 0.087 for the sigmoid, ReLU and pure line functions. Table I shows the MSEs of the A-SCAU with different sequence lengths and activation functions.

TABLE I. MSEs of the A-SCAU ($\times 10^{-3}$)

sequence length (bits)	512	1024	2048	4096
sigmoid	7.41	2.81	2.08	**1.10**
ReLU	3.33	1.69	1.12	**0.61**
pure line	4.58	1.41	1.32	**0.89**

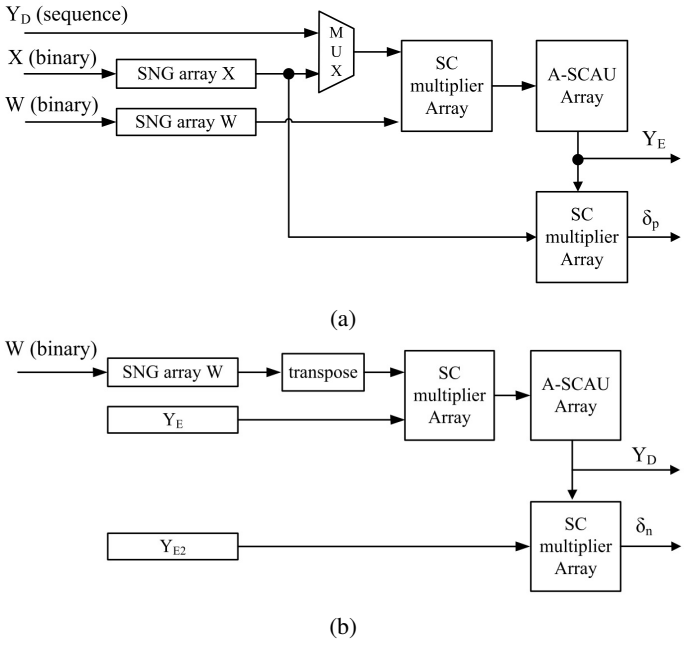

Fig. 5. The system diagram of (a) an encoder; and (b) a decoder. The signal definitions are the same as for (1) to (6).

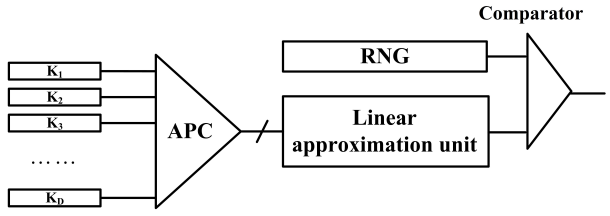

Fig. 6. Design of the A-SCAU, including an APC, an LAU, an RNG and a comparator.

D. Immune-to-correlation feature

As the core component in the A-SCAU, the LAU is implemented using a binary circuit (Fig. 6). As a result, the accuracy of the LAU is not affected by the correlations in the stochastic sequences.

In the A-SCAU, each input is implemented by a parallelization of q levels. For D-dimensional input sequences K_i ($i = 1, 2, ..., D$), the APC converts every qD-bit input

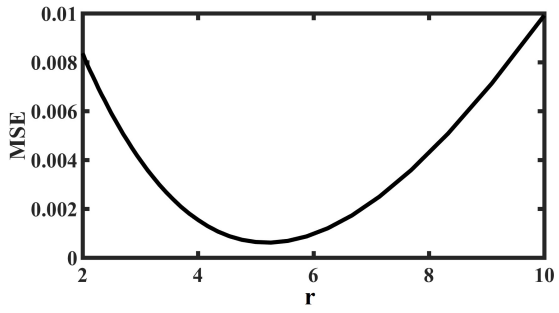

Fig. 7. Search result of optimal approximation parameters for the sigmoid function. The MSE is between (8) and (13).

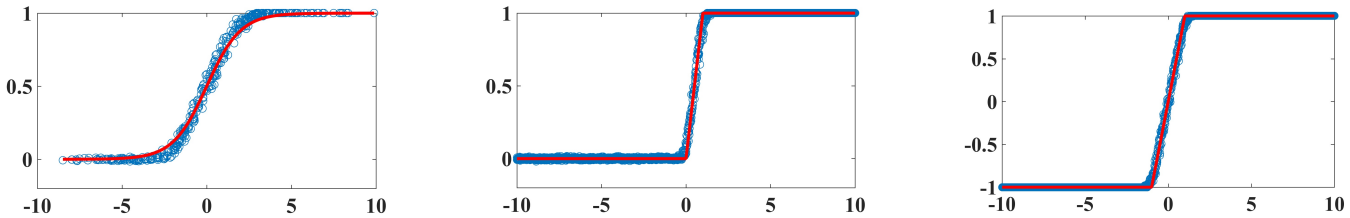

Fig. 8. The simulation results of the A-SCAU for (a) the sigmoid function, (b) the ReLU function and (c) the pure line function.

combination into a binary vector of m bits as inputs to the LAU.

For the n-bit stochastic sequences, the APC outputs n m-bit binary integers in series. Then, the LAU accumulates n cycles of the output from the APC and updates the output. Let the j^{th} binary integer generated by the APC be c_j and let k_i and $k_i{}'$ be the i^{th} values encoded in the sequence K_i in the bipolar and unipolar representations (Fig. 6). Following the definition in (12), x can be approximated by the output of the APC, as

$$x = \sum_{i=1}^{D} k_i = \sum_{i=1}^{D} (2k_i{}' - 1) \approx 2\sum_{j=1}^{n} \frac{c_j}{n} - D. \quad (15)$$

The range of $\psi(x)$ is $[0, 1]$ for the sigmoid and ReLU functions, and $[-1, 1]$ for the pure line function, encoded in the bipolar representation. Because the SNG in the A-SCAU requires unsigned integers to generate stochastic sequences, the LAU needs to produce an integer output for the value of $\psi(x)$ interpreted as the unipolar representation. This value is given by

$$\psi'(x) = \frac{1}{2}(\psi(x) + 1). \quad (16)$$

For an m-bit LAU, the integer output, $\Psi(x)$, is given by

$$\Psi(x) = (2^m - 1) \times \psi'(x) = (2^m - 1) \times (\frac{\psi(x) + 1}{2}). \quad (17)$$

A stochastic sequence is then generated for $\Psi(x)$ by the RNG and comparator as the output of the A-SCAU.

Applying (13), (15) to (17), the output of the LAU is given by

$$\Psi(x) = (2^m - 1)\times \\ min(1, max(\frac{p+1}{2}, \frac{2\sum_{j=1}^{n} c_j + nr(s+1) - nD}{2nr})). \quad (18)$$

Define an internal signal T as

$$T = 2\sum_{j=1}^{n} c_j + nr(s+1) - nD, \quad (19)$$

(18) can be approximated by

$$\Psi(x) = (2^m - 1) \times min(1, max(\frac{p+1}{2}, \frac{T}{2nr})) \\ = min(2^m - 1, max(\frac{2^m - 1}{2} \cdot (p+1), \frac{2^m - 1}{2nr} \cdot T) \quad (20) \\ \approx min(2^m - 1, max(2^{m-1} \cdot (p+1), \frac{2^{m-1}}{nr} \cdot T)).$$

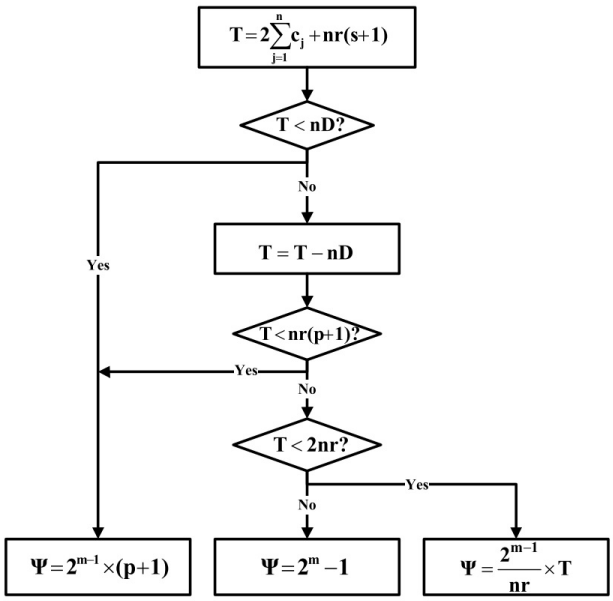

Fig. 9. An algorithmic flowchart for the LAU. T: a temporary variable used to store the intermediate result in the computation. The definitions of other signals are the same as for (18) and (20).

From (20), it can be seen that the output of the LAU can be one of the three 3 different values: $2^m - 1$, $2^{m-1} \cdot (p+1)$ and $2^{m-1} \cdot T/nr$. The circuit can be implemented by accumulators, subtractors, multipliers and dividers. An algorithmic flowchart is shown in Fig. 9 and a circuit design is shown in Fig. 10. In the SC implementation, the n is set to 16 and the parameter r is set to a value in a power of 2 in both the SC and binary implementations, so the multipliers and dividers are implemented by using shift registers. As the internal signals encode unsigned integers, an additional comparator is used to prevent the overflow in subtractions as well as to determine the final output. Note that T in (19) is implemented by an accumulator, an adder, a subtractor and shift registers, as shown in Fig. 10.

As per (18), the output of the LAU is only determined by the number of 1's computed by the APC in the input sequences, regardless of the bit correlations. Therefore, the computation accuracy of the A-SCAU is not affected by the correlations due to the sharing of RNGs in the circuit.

This immune-to-correlation feature makes it possible to dramatically reduce the number of RNGs in the circuit. Fig.

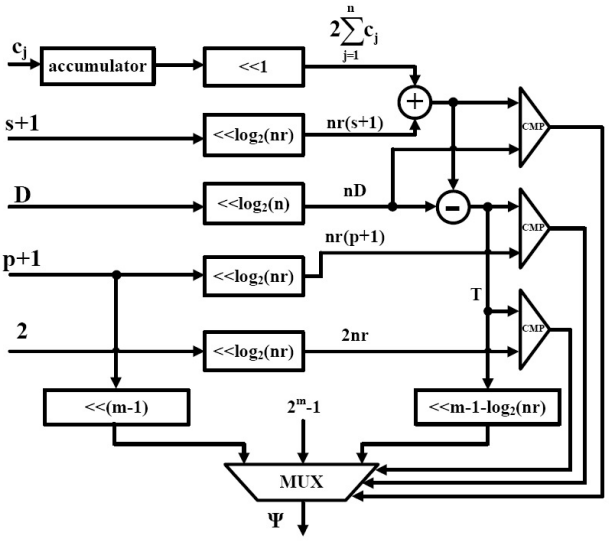

Fig. 10. Circuit design of the LAU. CMP: comparator. <<: shift register. The width of the output signal is set to m. The definitions of the signals are the same as for (18).

11 shows the test circuit for comparing the proposed A-SCAU with the $Btanh$ based sigmoid design. In the simulation, sequences for D-dimensional input signals are generated by shared RNGs but different comparators. The parallelization is set to $16\times$ and sequence length is set to 256 bits, so in total $256 \times 16 = 4096$ bits for each input. The simulation results of the A-SCAU and the $Btanh$ based circuit are shown in Fig. 12. As can be seen, the $Btanh$ circuit does not produce correct results, whereas the A-SCAU achieves a good accuracy.

E. RNG sharing

The SNG array in Fig. 5 is utilized for a parallel operation to reduce the computation latency. The design also reduces the area and energy cost of the encoder-decoder pair by sharing the RNGs. In the SNG array, each signal in a D-dimensional input is converted into q parallel stochastic sequences to reduce latency, see Fig. 13. As the A-SCAU is immune to the correlations among input stochastic sequences, the RNGs are shared among parallel A-SCAU components

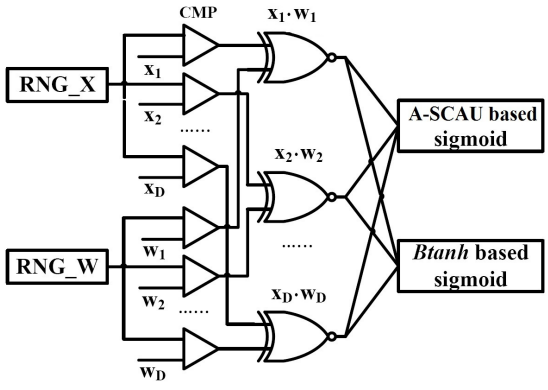

Fig. 11. Test circuit for the A-SCAU and the $Btanh$ based sigmoid functions.

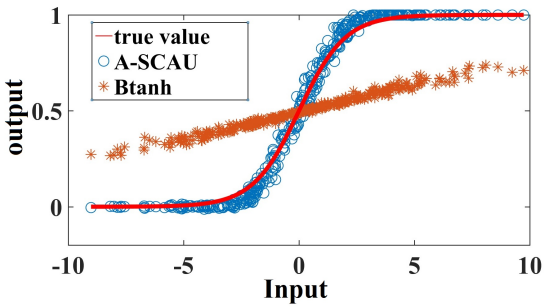

Fig. 12. Simulation results of the A-SCAU and the $Btanh$ based sigmoid circuit. Both use shared RNGs.

without loss of computation accuracy. The RNGs can not only be shared among the signals in a single neuron, but also among all neurons in the same layer. Therefore, the number of RNGs is changed to $1/D$ of those required in a conventional design with the same level of parallelization but no sharing structure. The RNGs are implemented using different initial seeds and feedback polynomials to avoid generating correlated sequences, thus reducing the autocorrelation in the output sequences.

Consider a 2-layer network with 784 neurons in the input layer and 100 neurons in the output layer; the dimension of the input signals, the output signals and the layer weights are 784, 100, and 78400. Without considering the parallelization, 79284 RNGs are required in a conventional design with no sharing structure. In the proposed design, however, because the RNGs can be shared among neurons, it only requires 3 RNGs to generate the input, output and layer weight sequences, resulting in significant savings in area and energy consumption.

F. Design of ADAM circuits

As per (11), the ADAM algorithm can be implemented by SC circuits, including adders/subtractors, multipliers/dividers, square root and power function circuits. The ADAM circuits are shown in Fig. 14.

The circuit in Fig. 14 (a) updates the moment vector m_t. The updater for m_t is implemented by SC adders, subtractors and multipliers. Note that the output is $0.25m_t$ because two SC adders are connected in series. Therefore, the scaling factor 0.25 is eliminated prior to the computation of $\hat{m}_t$ in (11). The updater for v_t has a similar structure as for m_t, with the computation of g_t^2 implemented by an XNOR gate and a D-flipflop (dashed in Fig. 14 (a)).

Fig. 14 (b) shows the power function circuit to compute β^t. Assume the sequence length is set to k bits and the value encoded in the input sequence is set to β in the bipolar representation for each computation step t. The k-bit shift register is initialized to be all ones when $t = 0$. During computation, each bit of the XNOR gate is stored in the LSB of the register and the register is left-shifted. At the end of the computation in step t, the k-bit output sequence is stored in the shift register and then is used to multiply the input sequence encoding β in step $t + 1$. It can be seen that for each t, the value encoded in the output sequence follows $f(\beta) = \beta^t$ in the bipolar representation. The result is then used to update the

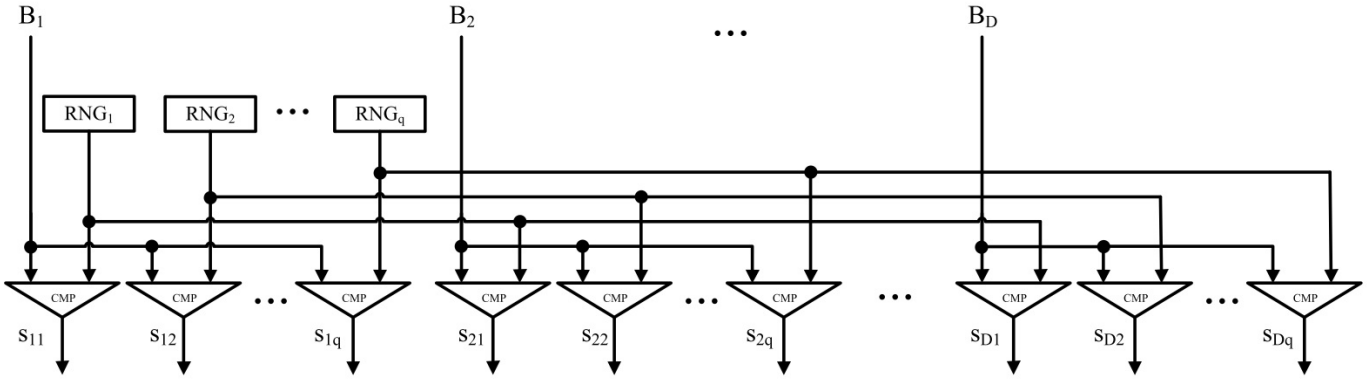

Fig. 13. An SNG array in the encoder-decoder pair with an input dimension D and parallelization level q. CMP: comparator. B_i $(i = 1, 2, ..., D)$ is the i^{th} binary value of the input signal. s_{ij} is the j^{th} parallel stochastic output sequence of B_i $(i = 1, 2, ..., D;\ j = 1, 2, ..., q)$.

value of $\hat{m_t}$ and $\hat{v_t}$ by following (11), using an SC subtractor and a divider. For each computation with a sequence length of 4096 bits and 16× parallelization, an array of 16 power function circuits is implemented with k set to 256 bits.

Fig. 14 (c) shows the circuit computing the value of $\sqrt{\hat{v}_t}+\varepsilon$.

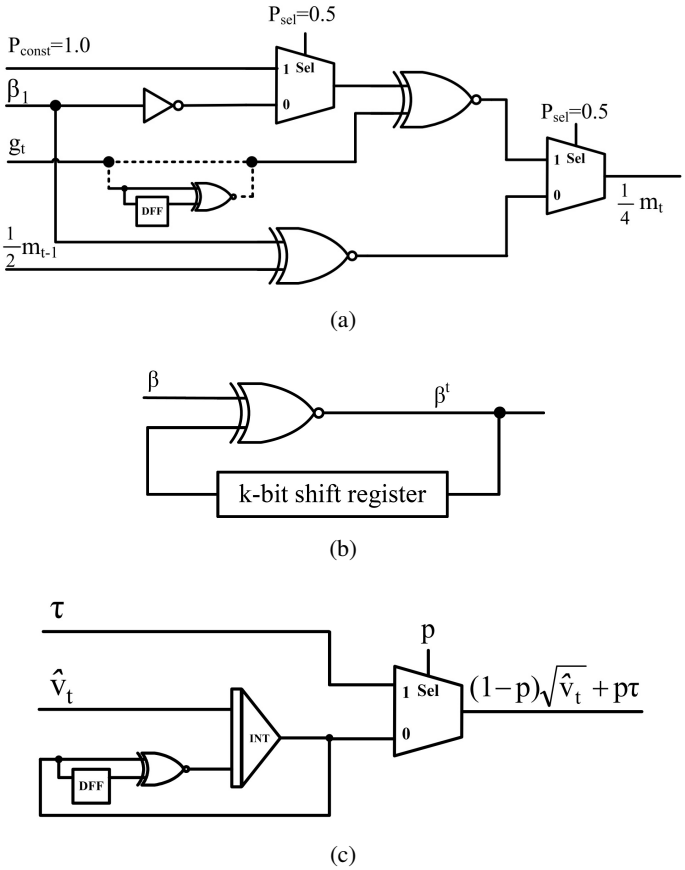

Fig. 14. Design of the ADAM circuits. (a) Moment vector updater, (b) Power function circuit for computing β_1^t and β_2^t. (c) The circuit to compute $p\tau + (1-p)\sqrt{\hat{v}_t}, \tau = \varepsilon/p$. All signals are encoded in stochastic sequences in the bipolar representation, following the same definitions as in (11).

It consists of an SC square root circuit and an adder. The binary search algorithm [7] [26] is utilized in the SC square root circuit to reduce the computation latency. Assume that the values encoded in the input sequences are τ and $\hat{v}_t$ in the bipolar representation, the probability of the select signal in the MUX is p and the value encoded in the output signal in the bipolar representation follows

$$h(\hat{v}_t,\ \tau) = (1-p)\sqrt{\hat{v}_t} + p\tau, \quad p, \tau > 0. \tag{21}$$

When p is set to a small value close to 0, there exists

$$\lim_{p \to 0} h(\hat{v}_t,\ \tau) = \sqrt{\hat{v}_t} + p\tau, \quad \tau > 0. \tag{22}$$

The value of $p\tau$ has the same role as ε in (11), leading to $\varepsilon = p\tau$. For an N-bit sequence, $\tau_{min} = 2/N$ for the bipolar representation and the resolution of the select signal is $p_{min} = 1/N$. As a result,

$$\varepsilon_{min} = \frac{2}{N^2}, \tag{23}$$

which determines the minimum value of ε that can be implemented by using N-bit sequences. For $N = 4096$ bits, $\varepsilon_{min} = 1.19 \times 10^{-7}$ is used in the design. With a shorter sequence length, the value of ε_{min} is increased and the performance of ADAM is reduced.

The MSEs of the ADAM circuits are listed in Table II. In the simulation of the power function, the value of β is set to 0.9 and the value of t is set to 31, which are the same as used in the SC-DBNs for the MNIST dataset. The power function circuit has the highest MSE among the components, and the change in the sequence length has no significant effect on the accuracy. The MSEs of the square root and divider circuits are low because of the binary search algorithm which improves both the computation accuracy and speed.

IV. Evaluation

A. Accuracy

The SC-DBN is evaluated on the MNIST dataset [27] using (8) as the activation function. The samples are grayscale images with 28×28 pixels of 10 different handwritten characters labeled as '0' to '9'. The structure of the network is

TABLE II. MSEs of the ADAM circuits ($\times 10^{-4}$)

sequence length (bits)	256	512	1024	2048	4096
moment vector updater	0.18	0.20	0.35	0.06	0.06
power function	8.74	8.51	8.26	8.35	7.90
square root	38.0	11.1	4.1	1.7	1.3
binary search divider [26]	11.7	7.0	5.1	5.2	4.8

optimized by the pruning algorithm [28], consisting of one input layer with 784 neurons, two hidden layers with 100 and 200 neurons, and one output layer with 10 neurons. An 8-bit fixed-point and 32-bit floating-point (FP) implementation with the same configuration are also evaluated on the dataset.

The SC-DBN is implemented with both pre-trained weights and online learning. For the pre-trained networks, Table III shows the classification error rates of the different implementations for inference. It can be seen that for the SC-DBN, the classification accuracy improves rapidly when the sequence length is under 256 bits, increasing from 89.9% (by 32 bits) to 98.9% (by 128 bits). Using 64-bit sequences ($\times 16$ parallelization), the proposed design achieves a higher accuracy than the results in the literature [17] [18] [19] [29]. Note that most of the designs in the literature require a larger latency than the proposed design (from 1024 bits to 4096 bits) except for the integral stochastic implementation [19] and the hybrid stochastic-binary network [30]. The network in [30] is based on SC CNN, so different from the other networks in Table III. Moreover, with a sequence no less than 128-bit, the SC-DBN achieves a higher classification accuracy than an 8-bit fixed-point implementation, which is only 0.12% to 0.37% lower than a 32-bit FP implementation.

TABLE III. Pre-trained Networks Accuracy Comparison

Network	sequence length (bit)	accuracy (%)
SC-DBN ($16\times$ parallelization)	32	89.90
	64	97.78
	128	**98.90**
	256	**99.15**
8-bit fixed point	–	98.10
32-bit floating-point	–	99.27
Integral stochastic NN [19]	64	97.73
Hybrid SC-binary NN [30]	128	99.01
SC DNN [18]	1024	97.59
FPGA-RBM [29]	1024	94.28
FPGA-DBN [17]	4096	94.10

For the SC-DBN with online learning, the number of learning epochs is initially set to 200. The sequence length varies from 64 to 256 bits for learning and from 32 to 256 bits for inference, both with $16\times$ parallelization. The classification accuracy of the different implementations is shown in Fig. 15.

The classification accuracy rapidly improves by increasing the sequence length for learning. For example, with a 32-bit sequence for inference, the classification accuracy is improved from 51.50% to 78.60% when the sequence length increases from 128 to 256 bits in the training process. Similarly, with a 256-bit sequence for inference, the accuracy is improved from 83.46% to 98.55%. With 256-bit sequences for both learning and inference, the SC-DBN achieves a higher accuracy than

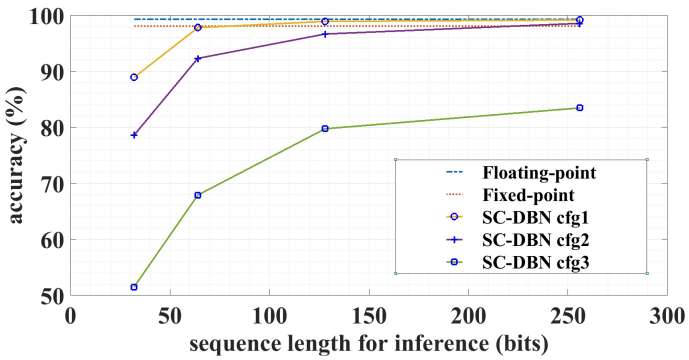

Fig. 15. Classification accuracy of different implementations of the DBN. SC-DBN cfg1: the pre-trained SC-DBN; cfg2: SC-DBN with 256-bit sequences for learning; cfg3: SC-DBN with 128-bit sequences for learning.

the 8-bit fixed point implementation (98.10%), and it is only 0.60% and 0.72% lower than the pre-trained SC-DBN and the FP implementation results. This suggests that a 256-bit sequence for learning is sufficient for this application. With this configuration, the online learning SC-DBN achieves an accuracy similar to the pre-trained implementations. However, with a 64-bit sequence in training, the computation of the SC-DBN fails and the accuracy for inference is around 10.00% (not shown in Fig. 15).

B. Hardware efficiency for pre-trained implementations

ASIC implementations of the DBNs are assessed in area, power and energy consumption using VHDL synthesized by the Synopsys Design Compiler with ST's 28 nm technology library. The sequence length of the SC-DBN is set to 128 and 256 bits with $16\times$ parallelization. The conventional FP design is implemented with and without pipelining. In the non-pipelined implementation, it requires 1 clock cycle for the computation in each layer, resulting in 4 cycles to process each sample. In the pipelined FP DBN, 6 clock cycles are required for an adder, 4 cycles for a multiplier and 24 cycles for an activation function. These numbers are 4, 4 and 10 for the pipelined fixed-point design.

In Table IV, the simulation results indicate that the SC-DBN requires the smallest area and lowest power among the different implementations. With 256-bit sequences, the SC circuit takes 5.3%, 4.5%, 3.3% and 73.6% of the area, power, energy consumption and latency (per sample) of the pipelined 32-bit FP implementation. These figures of merit are 26.9%, 27.8%, 29.9% and 107.3% when compared to the 8-bit fixed-point implementation. With 128-bit sequences, the latency and energy cost of the SC-DBN is approximately reduced by 50%, while incurring a loss of accuracy by only 0.25%. The proposed circuit takes 6.5% and 6.1% of the area and power of the non-pipelined 32-bit FP implementation, with a $1.3\times$ energy consumption and $21\times$ latency. The high latency is a general challenge for SC designs [8] [31].

Note that although the number of cycles to process a single sample is significantly increased by pipelining (from 4 to 412), the total computation latency is decreased because of the lower

TABLE IV. Hardware Efficiency (Inference)

	SC-DBN	8-bit fixed-point circuit	32-bit floating-point circuit (pipelined, non-pipelined)
Area (μm^2)	23345	86875	(437767, 357548)
Power (mW)	1.12	4.01	(24.86, 18.32)
Frequency (MHz)	134.7	167.3	(159.7, 90.2)
Cycle (/sample)	128/256	296	(412, 4)
Latency (μs/sample)	0.94/1.90	1.77	(2.580, 0.044)
Energy (nJ/sample)	1.05/2.12	7.10	(64.14, 0.81)

throughput. With a dataset of 10000 samples in the MNIST, the total computation latency of the pipelined FP implementation is approximately 14.7% of that of the non-pipelined design.

To compare the proposed SC-DBN with other types of SC NN designs, we considered an SC-DCNN [32] and performed simulation using the 28 nm technology library. With 256-bit sequences, the classification accuracy is 98.26% on the MNIST dataset and the energy consumption is 281 nJ/sample, much higher than that of the SC-DBN. The main reason for the higher energy consumption in the SC-DCNN is the inherently higher computation complexity of the DCNN required to achieve a high accuracy. However, the neurons in the fully connected layers of the SC-DBN have more inputs than the neurons in the convolutional layers of the SC-DCNN, so the inaccuracy in the SC computation can be better mitigated inside the neurons, resulting in a better performance in the SC-DBN.

C. Hardware efficiency for online learning

The area and energy consumption of the SC-DBN for online learning are reported in Table V. All the binary implementations are based on pipelined circuits for their higher efficiency in total computation time. The proposed encoder-decoder pair is reused in both the training and inference processes. The back-propagation and learning control unit are implemented to perform the backward propagation in the training process. Note that due to the complex timing control of SC circuits and the conversion between stochastic sequences and binary integers, the learning control unit of the SC-DBN is twice as large as that of the fixed- and floating-point implementations.

The SC-DBN with online learning achieves the lowest area, which is only 29.3% and 5.5% of the fixed- and floating-point implementations. The energy consumption of the online learning is significantly increased from that for inference. In the training process with 200 epochs, the SC-DBN takes 4.37 μJ to process each sample. However, the SC-DBN still achieves the lowest energy consumption among different implementations, which is 33.3% and 3.7% of the fixed- and floating-point implementations. The latency of the SC-DBN is 1.52 ms, approximately 110% and 80.9% of that of the fixed- and floating-point implementation. Similar as for the pre-trained implementations, the proposed design shows no significant disadvantage in performance compared to conventional binary designs.

TABLE V. Hardware Efficiency (Online Learning)

	SC-DBN	8-bit fixed-point circuit	32-bit floating-point circuit
Back-propagation circuit area (μm^2)	33150	116413	656651
Learning control unit area (μm^2)	3525	1785	1829
Total area (μm^2)	60019	205072	1096247
Latency per epoch (μs)	7.60	6.92	9.43
Total latency (200 epochs) (ms)	1.52	1.38	1.88
Energy per sample (μJ)	4.37	13.11	117.40

D. SC-DBN with the ADAM circuit

The ADAM improves the convergence speed of the back-ward propagation during the training process [25], thus decreasing the energy consumption and latency. With the ADAM circuit, the number of epochs in the training process can be reduced from 200 to 31 without losing inference accuracy on the MNIST dataset. However, the SC implementation of ADAM significantly increases the area and power consumption of the backward propagation circuit and requires extra computation cycles to update the learning rates. The total area and energy consumption of the SC-DBN with the ADAM circuit is shown in Table VI.

Compared to Table V, the area of the ADAM is comparable to that of the back-propagation circuit in the SC-DBN. Therefore, the total area of the SC-DBN is increased by 45.5%, from 60019 μm^2 to 87200 μm^2. The latency in each epoch is increased by 8.53 μs due to the ADAM circuit. However, the number of learning epochs is reduced by 84.5% (from 200 to 31), so the energy consumption of processing each sample is reduced by 74.8%, i.e. from 4.37 μJ to 1.10 μJ per sample. The total latency of processing each sample is also reduced by 65.2%, from 1.52 ms to 529.1 μs. Although the latency in a single epoch and the area are increased, the SC-DBN with the ADAM circuit achieves significant advantages in overall energy consumption and computation speed.

TABLE VI. Hardware Efficiency of the SC-DBN with the ADAM circuit

ADAM area (μm^2)	27181
Total area (μm^2)	87200
Total latency (31 epochs) (μs)	529.1
Energy per sample (μJ)	1.10

V. Conclusion

In this paper, an SC-DBN is proposed to reduce the area and energy consumption of DNNs. A reconfigurable structure is proposed to implement the fast greedy learning algorithm and enable the sharing of hardware by reusing the encoder-decoder pair. An ADAM circuit is utilized to improve the energy efficiency by significantly reducing the number of

epochs in the training process. An A-SCAU is reconfigurable to implement different activation functions; it also leverages the shared use of RNGs among neurons in the same layer, so significantly smaller area and lower energy consumption are obtained for the proposed design. For both pre-trained (inference) and online learning (training), the classification accuracy of the SC-DBN is higher than a fixed-point design and slightly lower than a floating-point design. Compared to the conventional binary implementations, the proposed design requires significantly smaller area and lower power. The energy consumption of the SC-DBN is significantly lower than that of the pipelined 32-bit FP design and slightly higher than the non-pipelined design.

REFERENCES

[1] 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.

[2] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in Neural Information Processing Systems*, pp. 2553–2561, 2013.

[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[4] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *IEEE Conf. on. Computer Vision and Pattern Recognition (CVPR)*, pp. 3642–3649, 2012.

[5] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

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

[7] B. R. Gaines *et al.*, "Stochastic computing systems," *Advances in information systems science*, vol. 2, no. 2, pp. 37–172, 1969.

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

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

[10] B. D. Brown and H. C. Card, "Stochastic neural computation. II. soft competitive learning," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 906–920, 2001.

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

[12] 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*, pp. 1–8, 2015.

[13] 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*, pp. 757–762, 2012.

[14] 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*, pp. 880–883, 2015.

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

[16] 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*, pp. 678–681, 2016.

[17] 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*, pp. 1–5, 2015.

[18] 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*, p. 124, 2016.

[19] 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.

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

[21] Y. W. Teh and G. E. Hinton, "Rate-coded restricted Boltzmann machines for face recognition," in *Advances in neural information processing systems*, pp. 908–914, 2001.

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

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[24] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Transactions on Computers*, 2018.

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

[28] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

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

[30] 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 *DATE*, pp. 13–18, 2017.

[31] 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.

[32] 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," in *ASPLOS*, pp. 405–418, 2017.