Gradient Descent using Stochastic Circuits for Efficient Training of Learning Machines

Siting Liu, Honglan Jiang, *Student Member, IEEE,* Leibo Liu, *Senior Member, IEEE,*
Jie Han, *Senior Member, IEEE*

***Abstract*—Gradient descent (GD) is a widely used optimization algorithm in machine learning. In this paper, a novel stochastic computing GD circuit (SC-GDC) is proposed by encoding the gradient information in stochastic sequences. Inspired by the structure of a neuron, a stochastic integrator is used to optimize the weights in a learning machine by its "inhibitory" and "excitatory" inputs. Specifically, two AND (or XNOR) gates for the unipolar representation (or the bipolar representation) and one stochastic integrator are, respectively, used to implement the multiplications and accumulations in a GD algorithm. Thus, the SC-GDC is very area- and power-efficient. As per the formulation of the proposed SC-GDC, it provides unbiased estimate of the optimized weights in a learning algorithm. The proposed SC-GDC is then used to implement a least-mean-square algorithm and a softmax regression. With a similar accuracy, the proposed design achieves more than 30× improvement in throughput per area (TPA) and consumes less than 13% of the energy per training sample, compared with a fixed-point implementation. Moreover, a signed SC-GDC is proposed for training complex neural networks (NNs). It is shown that for a 784-128-128-10 fully-connected NN, the signed SC-GDC produces a similar training result with its fixed-point counterpart, while achieving more than 90% energy saving and 82% reduction in training time with more than 50× improvement in TPA.**

***Index Terms*—Adaptive filter (AF), gradient descent (GD), machine learning, neural networks (NNs), softmax regression (SR), stochastic computing (SC).**

I. Introduction

DEEP learning utilizes a computational model to automatically discover intricate structures from large raw data by following a general-purpose training procedure. By using a multiple-layer computational model, it has produced many promising results for various tasks including object recognition, natural language processing and autonomous driving [1]. However, a larger computational load is imposed on training a learning machine as a model becomes more complex; e.g., tens of millions of parameters or weights need to be optimized for image recognition in AlexNet [2]. To improve performance, graphics processing units (GPUs) with massively parallel computing resources have widely been used for machine learning.

To further improve performance, machine learning specific chips have been developed, such as the TrueNorth neuro-chip [3], the tensor processing unit (TPU) [4], and the Minerva [5]. However, most of these chips are designed for inference rather than training or optimizing the weights in a learning process. A recent study shows that only three days are required to train a network playing the Go game using four TPUs [6]. Nevertheless, it is still not energy-efficient for mobile or embedded applications.

To reduce the energy consumption in a learning system, quantization and binarization have been shown to be effective [7], [8]. However, both methodologies are mostly implemented in software and no dedicated hardware is available for training. In [7], binarized weights and activations are used to drastically reduce memory usage during the inference phase, whereas in the backpropagation, the binarization is not applicable, and real values are used to compute the optimal weights by using gradient descent (GD).

Stochastic computing (SC) is an alternative hardware-efficient computing paradigm. In SC, a random binary bit stream of "0"s and "1"s, or a stochastic sequence, is used to encode a real value by the occurrence frequency of 1's. Such a sequence is generated by a stochastic number generator (SNG) as shown in Fig. 1. In the unipolar representation, the value to be encoded, $x \in [0, 1]$, is compared with a random number (RN) that is uniformly distributed within the interval $[0, 1]$. The comparator generates a 1 if x is larger; otherwise it generates a 0. In the bipolar representation, a value $x \in [-1, 1]$ is first mapped to $[0, 1]$ by $(x+1)/2$, which is then converted to a stochastic sequence by using an SNG. Typically, only a small circuit is required to perform computations on the stochastic sequences. For example, a multiplier can be implemented by an AND gate for the unipolar representation, or an XNOR gate for the bipolar representation as shown in Fig. 2.

SC is also a biologically plausible computing paradigm [9]. The spike trains in the brain is similar to a stochastic sequence when considering a 1 in the sequence as a spike and a 0 as no spike. Inspired by this feature, SC has been applied to various machine learning algorithms, such as the spiking neural networks [10], multilayer perceptrons [11], [12], radial basis function neural networks [13], restricted

This article was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) 2018 and appears as part of the ESWEEK-TCAD special issue.

Siting Liu, Honglan Jiang and Jie Han were with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G1H9 Canada (e-mail: {siting2, honglan, jhan8}@ualberta.ca).

Leibo Liu was with Institute of Microelectronics, Tsinghua University, Beijing, 100084 P. R. China (e-mail: liulb@tsinghua.edu.cn).

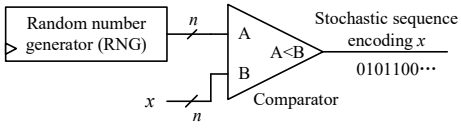


Fig. 1. A stochastic number generator (SNG).

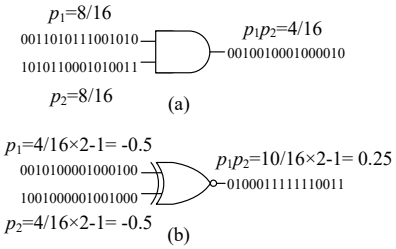


Fig. 2. (a) A unipolar and (b) bipolar stochastic multiplier.

Boltzmann machines [14], convolutional neural networks (CNNs) [15], [16] and deep neural networks (DNNs) [17]–[19]. Unfortunately, none of these SC designs implements the complex and time-consuming training process.

In this paper, a novel gradient descent circuit (SC-GDC) is proposed for efficient training of learning machines by using stochastic circuits. In the proposed design, the gradient information of a training sample is carried by stochastic bits. It is different from the conventional belief that the gradient value used during training cannot tolerate much inaccuracy [7].

By using the proposed SC-GDCs, the loss functions of a least-mean-square (LMS) adaptive filter (AF) and a softmax regression (SR) are minimized to obtain optimized weights. The simulation results show that SC-GDC-based LMS weight update unit achieves a higher accuracy with less than 0.1% of the computation time than a previous stochastic design. For handwritten-digit recognition, the proposed SR unit using an SC-GDC array produces a similar test accuracy with a software implementation using the same SR model. It takes only 42.6% of the computation time and less than 15% of the energy of a fixed-point design. A more complex 784-128-128-10 neural network (NN) is trained by a signed SC-GDC array. The signed SC-GDC achieves more than 90% energy saving and 82% reduction in time compared to its fixed-point implementation while preserving a similar test accuracy.

The variance bound for the proposed SC-GDC is given by an error analysis. Moreover, it is shown that sharing the random number generators (RNGs) for generating some of the input stochastic sequences reduces the variance of the computed result.

II. Background

A. Gradient descent

As a basic optimization algorithm, GD has widely been used in machine learning to optimize the weights of a model by minimizing the loss function. Let $L(\mathbf{w})$ be a multivariate differentiable loss function and the vector $\mathbf{w}$ be the weights in a learning machine, GD computes the local minimum of the loss function by an iterative optimization [20]:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \nabla L(\mathbf{w}_i), (i = 0, 1, 2, \dots), \tag{1}$$

where $\mathbf{w}_i$ is the optimized weight vector at the ith step; η is a constant or variable step size, or learning rate, which determines how fast the model learns; $\nabla L(\mathbf{w}_i)$ is the gradient of the loss function at $\mathbf{w} = \mathbf{w}_i$. $\nabla L(\mathbf{w})$ is given by

$$\nabla L(\mathbf{w}) = \left[\frac{\partial L(\mathbf{w})}{\partial w_1}, \frac{\partial L(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial L(\mathbf{w})}{\partial w_j}, \dots \right], \tag{2}$$

where w_j is the jth element in vector $\mathbf{w}$. If the step size is constant, the optimization result at the kth step can be obtained by accumulating (1) from $i = 0$ to $k - 1$,

$$\mathbf{w}_k = \mathbf{w}_0 - \eta \sum_{i=0}^{k-1} \nabla L(\mathbf{w}_i), \tag{3}$$

where the vector, $\mathbf{w}_0$, is usually initialized randomly [20].

B. Stochastic integrator

Stochastic integrators are sequential SC elements that accumulate the difference between two stochastic sequences [21]. As shown in Fig. 3, a stochastic integrator consists of an n-bit counter, an RNG and a comparator. The counter updates its value by [22]

$$C_{i+1} = \begin{cases} C_i + 1 & \text{if } a_i = 1 \text{ and } b_i = 0 \\ C_i & \text{if } a_i = b_i \\ C_i - 1 & \text{if } a_i = 0 \text{ and } b_i = 1, \end{cases} \tag{4}$$

where a_i and b_i are the ith bits from the stochastic sequences A and B respectively, i.e., they are the values of A and B at the ith clock cycle since each bit is generated per clock cycle. C_i and C_{i+1} are the n-bit binary numbers stored in the counter at the ith and $(i + 1)$th clock cycles. Equivalently, we have

$$C_{i+1} = C_i + a_i - b_i. \tag{5}$$

As in an SNG, the output stochastic sequence is generated by comparing the n-bit binary number with an n-bit RN generated by the RNG. So, at the ith clock cycle, the probability generating a 1 equals to $2^{-n}C_i$, i.e., the value encoded by the n-bit binary number is $P_i = 2^{-n}C_i$ in the unipolar representation. Normalizing (5) by 2^{-n} leads to

$$P_{i+1} = P_i + 2^{-n}(a_i - b_i). \tag{6}$$

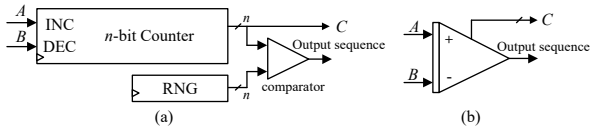


Fig. 3. A stochastic integrator: (a) the circuit block diagram; (b) a symbol.

Assume the initial value is P_0, then by an iterative accumulation of (6) from $i = 0$ to $k-1$, the value encoded by the counter at the kth clock cycle is obtained as

$$P_k = P_0 + 2^{-n} \sum_{i=0}^{k-1} (a_i - b_i). \tag{7}$$

Taking the expectation of (7) gives us

$$\mathbb{E}[P_k] = P_0 + 2^{-n} \sum_{i=0}^{k-1} (\mathbb{E}[a_i] - \mathbb{E}[b_i]). \tag{8}$$

Comparing (8) and (3), the stochastic integrator provides an unbiased estimate of a weight optimized by the GD algorithm, i.e. $\mathbb{E}[P_k] = \mathbf{w}_k$, under the conditions that: 1) the stochastic integrator is initialized with $\mathbf{w}_0$; 2) the expectation of the difference of the stochastic sequences equals to the negative of gradient, i.e., $\mathbb{E}[a_i] - \mathbb{E}[b_i] = -\nabla L(\mathbf{w}_i)$; and 3) the step size equals to 2^{-n} [23].

III. Proposed SC-GDC Design

A. SC-GDC circuit design

Fig. 4 shows the proposed unipolar SC-GDC for optimizing a weight, $w_{i,j}$, i.e., the jth element in the vector $\mathbf{w}$ at time step i. If there are N elements in $\mathbf{w}$, N SC-GDCs are required to optimize the N weights. In Fig. 4, $F(\mathbf{w}_i, \mathbf{x}_i)$ is the inferred value given by the model, and t_i is the target or desired output value for input $\mathbf{x}_i$, which is used to supervise the training of a model. $\partial F(\mathbf{w}_i, \mathbf{x}_i)/\partial w_{i,j}$ is the input signal for a linear model. For an NN model, these signals can be obtained by a back-propagation. The SNGs are used to stochastically binarize the inputs of SC-GDC, and the stochastic multiplier and integrator are used to efficiently compute (1). The SC-GDC works in an online manner, which means that it uses $\{F(\mathbf{w}_i, \mathbf{x}_i), \partial F(\mathbf{w}_i, \mathbf{x}_i)/\partial w_{i,j}, t_i\}$ of training sample $\mathbf{x}_i$ sequentially to update the weight. When the bipolar representation is used, the AND gates are replaced by XNOR gates for multiplication.

The circuit design of a stochastic integrator is shown in Fig. 5. One n-bit adder is used to compute (5) by taking advantage of SC. Since a_i can only be 0 or 1, it is used as the carry input of the adder. n copies of b_i are used to perform "$-b_i$" in (5). Specifically, $(b_i b_i \dots b_i)_2$ represents -1 in 2's complement when b_i is 1, otherwise it represents 0. The RNs are generated by a linear feedback shift register (LFSR) that works as an RNG. The LFSR and the comparator can be removed if the output sequence is not used for further stochastic computation.

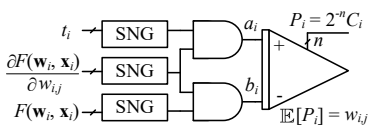


Fig. 4. Proposed unipolar SC-GDC.

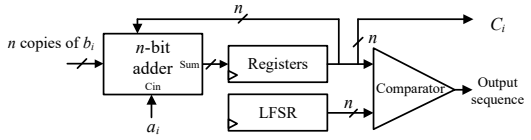


Fig. 5. Circuit design of a stochastic integrator.

B. Formulation of SC-GDC

To train a learning machine by using GD, the loss function is defined first. The quadratic error between the inferred value and the target value is a commonly used loss function, which is also known as the LMS error, given by

$$L(\mathbf{w}_i) = 0.5[t_i - F(\mathbf{w}_i, \mathbf{x}_i)]^2. \tag{9}$$

The gradient is the partial derivative of the loss function as per (2). For $w_{i,j}$, it is the partial derivative of the loss function with respect to $w_{i,j}$, i.e.,

$$\frac{\partial L(\mathbf{w}_i)}{\partial w_{i,j}} = -\frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i + \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i). \tag{10}$$

Equation (3) is then transformed to

$$\begin{aligned} w_{k,j} = w_{0,j} - \eta \sum_{i=0}^{k-1} [& -\frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i \\ & + \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i)]. \end{aligned} \tag{11}$$

As shown in Fig. 4, two AND gates are used to implement the multiplications in (10), and we have

$$\begin{aligned} \mathbb{E}[a_i] &= \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i, \\ \mathbb{E}[b_i] &= \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i), \end{aligned} \tag{12}$$

where a_i and b_i are the two output bits of the AND gates. They are used as the inputs for the stochastic integrator to implement the accumulation in (11). As per (8) and (12), the expectation of the value encoded by the counter at the kth clock cycle is given by

$$\begin{aligned} \mathbb{E}[P_k] = P_0 + \frac{1}{2^n} \sum_{i=0}^{k-1} [& \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} t_i \\ & - \frac{\partial F(\mathbf{w}_i, \mathbf{x}_i)}{\partial w_{i,j}} F(\mathbf{w}_i, \mathbf{x}_i)]. \end{aligned} \tag{13}$$

As per (11) and (13), if the counter is initialized with the value of $w_{0,j}$, the proposed SC-GDC provides an unbiased estimate to the weight to be optimized with a step size of 2^{-n}, i.e.,

$$\mathbb{E}[P_k] = w_{k,j}, \text{ for } \eta = 2^{-n} \text{ and } P_0 = w_{0,j}. \tag{14}$$

Therefore, the SC-GDC is used to perform GD-based online learning, and the weights are stochastically optimized by the SC-GDC. Each bit from the stochastic sequence accounts for 2^{-n} in the unbiased estimate of the optimized

weight and the randomness of each bit can be canceled out during the accumulation. As a result, this method leads to a high accuracy without using a long stochastic sequence encoding each gradient.

IV. Error Analysis

A. Single-step variance for SC-GDC

The variance of the estimated weights produced by the SC-GDC is obtained by analyzing the probability mass function (PMF) of the result by one-step optimization, i.e., the value of w_{i+1} updated from w_i. For the proposed unipolar design in Fig. 4, let $Y = F(\mathbf{w}_i, \mathbf{x}_i)$, $X = \partial F(\mathbf{w}_i, \mathbf{x}_i)/\partial w_{i,j}$, and $T = t_i$ for simplicity; further let y_s, x_s and t_s be the stochastic bits encoding these values, generated by the three SNGs. Thus, $\mathbb{E}[y_s] = Y$, $\mathbb{E}[x_s] = X$ and $\mathbb{E}[t_s] = T$. y_s and x_s are independently generated to ensure the correctness of the multiplications, so are t_s and x_s. However, y_s and t_s are not necessarily independent. If y_s and t_s are independently generated, the PMF of w_{i+1} is listed in Table I. As per (6), only when t_s =1, x_s =1 and y_s =0, is w_i increased by 2^{-n}. When t_s =0, x_s=1 and y_s =1, w_i is decreased by 2^{-n}; otherwise, w_i does not change.

The variance of a random variable x is given by

$$\text{Var}[x] = \mathbb{E}[(x - \mathbb{E}[x])^2]. \tag{15}$$

As per Table I and (15), the variance of w_{i+1} is computed as

$$\begin{aligned}\text{Var}_{\text{ind}}[w_{i+1}] =& 2^{-2n}[XY(1-XY) + TX(1-TX) \\ &+ 2TYX(1-X)].\end{aligned} \tag{16}$$

However, when the same RNG is used to generate y_s and t_s, they are not statistically independent. The variance of w_{i+1} can be computed similarly by using its PMF. Due to the space limitation, the variance of w_{i+1} is directly given here, by

$$\text{Var}_{\text{share}}[w_{i+1}] = 2^{-2n}(|T-Y|X)(1-|T-Y|X). \tag{17}$$

The variance is reduced by $2^{-2n+1}\min\{T,Y\}(1 - \max\{T,Y\})$ compared to the case when y_s and t_s are independently generated. Therefore, it reduces the variance, thus improving the accuracy when using the same RNG to generate stochastic bits y_s and t_s.

TABLE I
Probability distribution of w_{i+1} when y_s, x_s and t_s are independently generated.

w_{i+1}	Probability
$w_i - 2^{-n}$	$(1-T)XY$
w_i	$1 - TX - XY + 2TXY$
$w_i + 2^{-n}$	$(1-Y)TX$

B. Multiple-step variance bound

If every stochastic bit is independently generated at each step, i.e., they are temporally independent, the multi-step variance is the summation of the single-step variances [24]. When the RNG is shared, the maximum single-step variance is 2^{-2n-2} only when $|T-Y|X = 0.5$. Thus, the variance bound of the estimated optimized weights after k steps is given by

$$\text{Var}_{\text{bound}}[w_k] = 2^{-2n-2}k. \tag{18}$$

As per (18), the variance bound exponentially decreases with the bit width n. When the bipolar representation is used, the variance can be derived similarly, and the bound is given by $2^{-2n}k$.

V. Applications

A. System identification using least-mean-square (LMS) adaptive filters (AFs)

To assess the efficiency of the proposed SC-GDC, it is used in an LMS AF for system identification. The block diagram of an AF is shown in Fig. 6. An AF system consists of a linear filter and an optimization module that adjusts the weights of the linear filter. It can be considered as a simple learning machine with one neuron and a linear activation function. An AF has been implemented in approximate arithmetic circuits as a cerebellar model to control eye movement [25].

In an AF system, the output of the linear filter at the ith step, y_i, is given by

$$y_i = F(\mathbf{w}_i, \mathbf{x}_i) = \mathbf{w}_i\mathbf{x}_i = \sum_{j=0}^{M-1} w_{i,j}x_{i-M+j+1}, \tag{19}$$

where M is the length of the filter; $\mathbf{w}_i$ is a vector of M weights, $\mathbf{w}_i = [w_{i,0}, w_{i,1}, ..., w_{i,j}, ..., w_{i,M-1}]$; and $\mathbf{x}_i$ is the input vector at the ith step, $\mathbf{x}_i = [x_{i-M+1}, ..., x_{i-M+j}, ..., x_i]^{\mathbf{T}}$. The desired signal, t_i, guides the estimation of the weights in the target system [26].

In the LMS algorithm, the loss function is the quadratic error between t_i and y_i, which is given by (9) with $F(\mathbf{w}_i, \mathbf{x}_i) = y_i = \mathbf{w}_i\mathbf{x}_i$. As per (19), $\partial F(\mathbf{w}_i, \mathbf{x}_i)/\partial w_{i,j} = x_{i-M+j+1}$. Thus, the weight update unit is constructed from the SC-GDCs as shown in Fig. 7. M SC-GDCs are used to minimize the loss function and hence, to estimate the weights of the target system.

Since $\mathbf{x}_i$, $\mathbf{w}_i$ and y_i take values within $[-1, 1]$, the bipolar representation is used. Thus, the stochastic multipliers are

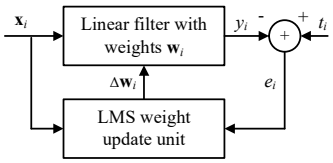


Fig. 6. An adaptive filter (AF).

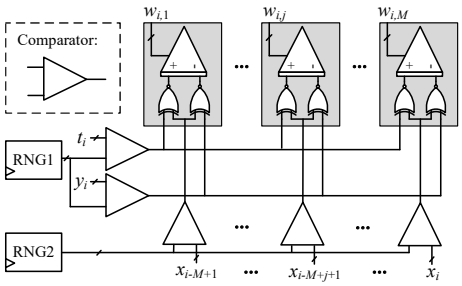


Fig. 7. LMS weight update unit using SC-GDCs.

implemented by XNOR gates. The stochastic sequences encoding y_i, t_i and $\mathbf{x}_i$ are generated by the RNGs and comparators. As discussed in Section IV-A, to reduce the variance as well as the hardware cost, the RNG is shared to generate stochastic sequences for y_i and t_i. As the M SC-GDCs are independent of each other, the RNG generating sequences for the vector $\mathbf{x}_i$ is also shared. For the same reason, the stochastic sequences encoding y_i and t_i are, respectively, shared among different SC-GDCs.

B. Handwritten-digit recognition using softmax regression

A softmax layer is usually the output layer in NNs for multi-class classification, which can be trained by using the GD algorithm. A softmax layer itself can be considered as a learning machine, which can perform classification of a relatively simple dataset. Fig. 8 shows an SR model, where $w_{m,j}$ denotes the weight of the connection between the jth input and the mth neuron. The output of the neurons $\mathbf{a} = [a_1, \ldots, a_M]^T$ is given by

$$\mathbf{a} = \mathbf{w}\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,J} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,J} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_J \end{bmatrix}. \quad (20)$$

The probability of input $\mathbf{x}$ belonging to class m, $P(\mathbf{x} \in m|\mathbf{w})$, is then estimated by the softmax function [27],

$$y_m = P(\mathbf{x} \in m|\mathbf{w}) = \frac{e^{a_m}}{\sum_{k=1}^{M} e^{a_k}}. \quad (21)$$

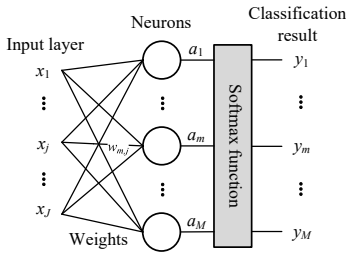


Fig. 8. A softmax regression (SR) model.

The loss function of an SR model is evaluated by cross entropy, given by [27]

$$L_{ce}(\mathbf{w}) = \sum_{m=1}^{M} -t_m \log y_m, \quad (22)$$

where m is the class label, and t_m is the actual classification result in a one-hot code. t_m is 1 when the input data belongs to class m; otherwise, it is 0. The partial derivative of the cross entropy with respect to $w_{m,j}$ is obtained by

$$\frac{\partial L_{ce}(\mathbf{w})}{\partial w_{m,j}} = \frac{\partial L_{ce}(\mathbf{w})}{\partial y_m} \frac{\partial y_m}{\partial a_m} \frac{\partial a_m}{\partial w_{m,j}} = -(t_m - y_m)x_j, \quad (23)$$

which is similar to the gradient of the quadratic error loss function. Thus, a GD-based SR unit can be implemented by using an SC-GDC array with y_m, x_j and t_m as its inputs. Since $x_j, t_m, y_m \in [0, 1]$, the unipolar SC-GDCs are used. The SR unit is shown in Fig. 9 for training the MNIST handwritten-digit dataset.

In Fig. 9, the input samples are 28×28 gray-scale images, which are flattened into vectors of 784 values. There are 10 classes of digits, so 784×10 weights are to be optimized by at least 784×10 SC-GDCs if the weights are updated in parallel. The RNGs and comparators are shared to the maximal extend to reduce the hardware cost and improve the accuracy as in the LMS weight update unit. Thus, two independent RNGs are used.

At clock i, one of the training samples, image i, is loaded to the SR model. Corresponding $\{t_{m,i}, y_{m,i}, x_{j,i}\}$ are computed and connected to the SC-GDC array. Meanwhile, in the SC-GDC array, $\{t_{m,i}, y_{m,i}, x_{j,i}\}$ are stochastically binarized and the stochastic bits encoding the gradient information are used to update the weights stored in the stochastic integrators. Since only one clock cycle is used to train one image and to update the weights, the performance of the SC-GDC array is much higher than conventional stochastic circuits where a long sequence is usually used to ensure the accuracy.

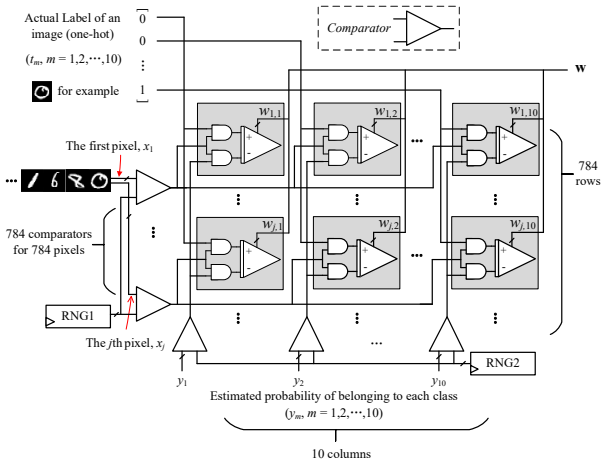


Fig. 9. An SC-GDC array for the training of the SR model.

However, if a counter needs to count beyond the maximum/minimum value it can reach, an overflow occurs. In this design, an overflow is avoided by adding extra bits and using the 2's complement representation. For example, for an 8-bit counter, it can encode a negative number or a number larger than 1 by adding 2 bits to its most significant bit, e.g., "01 0000 0001" encodes $257/2^8$ and "11 1111 1111" encodes $-1/2^8$ in the extended counter. If the weights are still out of the representation range of the extended counter in some other applications, the counter can always be further extended.

VI. Experiments and Results

A. Accuracy evaluation

1) System identification using an LMS AF: The proposed LMS weight update unit is used to perform system identification for a high pass FIR filter (target system) with 103 weights, so 103 SC-GDCs are required. The frequency response of the AF after training is shown in Fig. 10. It indicates that the results produced by the SC-GDCs are very close to the target system. After 2^{20} steps of training using a 15-bit counter (for a step size of 2^{-15}), the root mean squared error (RMSE) between the optimized and actual weights is around 6.45×10^{-4}, and the maximum absolute error is 2.90×10^{-3} for 100 runs. According to the 3-σ rule and the theoretical bound of variance derived in Section IV, the maximum error, in this case, is under the 3-σ bound, i.e., $2.90 \times 10^{-3} \ll 3 \times \sqrt{2^{20}/2^{2\times 15}}$.

2) Handwritten-digit recognition using SR: The proposed SR unit in Fig. 9 is used to recognize the handwritten digits in the MNIST dataset. In this paper, 60,000 images are used for training by using the SC-GDCs, and 10,000 images are used for evaluating the optimized weights without cross-validation. In the SC-GDCs, 8-bit counters are used, and the weights are initialized with random values.

An epoch of training is completed when the model is exposed to every training sample exactly once. The recognition accuracy and the cross entropy are shown against the number of training epochs in Fig. 11. The accuracy and cross entropy are reported every 10,000 steps or training samples.

Fig. 11(a) shows that the optimized weights produce a recognition accuracy around 92% for both the training and test data, which is similar to a software implementation using the same SR model [27]. The cross entropy converges rapidly at the first 10,000 samples and it becomes stable after about 4 epochs of training.

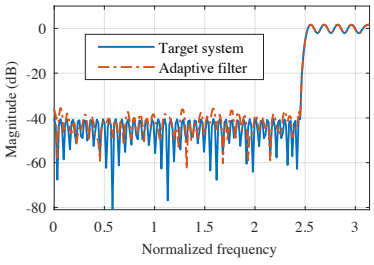
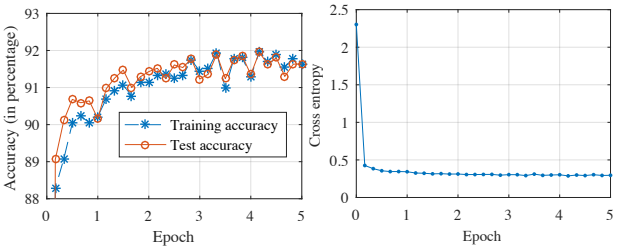


Fig. 10. System identification results.



(a) (b)

Fig. 11. (a) Recognition accuracy and (b) cross entropy using the SC-GDCs.

B. Hardware evaluation

The hardware efficiency of the proposed SC-GDC is evaluated in terms of speed, throughput and energy consumption. The proposed designs are implemented in VHDL and synthesized in Synopsys Design Compiler with a 28nm STM process. As per the formulation of the SC-GDC, one training sample is loaded to the circuit per clock cycle, and it does not require a long sequence for computing one result as in a conventional SC circuit. Therefore, the proposed design is more efficient than the conventional SC design.

The GD algorithm can be considered as walking from point A (initial weights) to point B (optimal weights) in a high-dimensional space. A larger step size leads to a smaller number of steps to reach point B, thus lower latency for the gradient descent circuit. Therefore, to optimize the energy efficiency and speed, a larger step size is preferable. However, if the step size is too large, the optimal point could be missed. It then may incur instability. Therefore, for a fair comparison, the maximum step size in a power of 2 that does not incur instability is used for each application by an exhaustive search, so that their energy efficiency and speed are optimized.

1) LMS weight update unit: The LMS weight update unit is compared with an existing SC design [28] and a fixed-point implementation for the same task. The fixed-point GD circuit is shown in Fig. 12, consisting of a subtractor, a multiplier, a shifter, an adder and registers. The shifter is used as a multiplier for multiplying the step size, 2^{-k}, where k is a positive integer.

The accuracy of the fixed-point circuit and the proposed design is matched by observing the convergence of the misalignment. The misalignment is defined as the normalized mean squared error between the optimized weight $\hat{\mathbf{w}}$ and the actual value $\mathbf{w}$ of the target system,

$$\text{Misalignment} = \frac{\mathbb{E}[(\mathbf{w} - \hat{\mathbf{w}})^2]}{\mathbb{E}[\mathbf{w}^2]}. \tag{24}$$

The convergence curves in misalignment are shown in Fig. 13.

For the stochastic design, the step size is 2^{-11} by using 11-bit counters in the SC-GDCs. Fig. 13(a) shows that the misalignment for the stochastic design converges to -23dB after about 64,000 steps of training. For the fixed-point circuit, a 16-bit design is used with a step size of 2^{-6}, and the misalignment converges to around -23dB after about 1100 steps. One hundred simulations are performed to measure the average minimum steps required for the misalignment to decrease to -23dB. The average minimum steps and the critical path delay are then used to estimate the minimum computation time ("Min. time" in Table II). Also, the energy per operation (EPO) is used to evaluate the energy consumption for training one sample, and the total energy is used to measure the energy cost of the circuits for the whole training process. The throughput per area (TPA) is used to evaluate the hardware efficiency by computing maximum number of samples that can be trained by the circuit per unit time and per unit area.

The results in Table II show that the proposed stochastic design consumes only about 0.047% of the total energy and 0.1% of the computation time of the stochastic design in [28] with a higher accuracy. Compared to the fixed-point implementation, the proposed design achieves 87.4% energy saving for each training sample and 35.3× TPA. However, the large number of steps leads to a large total energy cost for the proposed design.

2) SR unit: For the SR unit, the design in [29] realizes only the inference phase of an SR. To the best of our knowledge, no previous stochastic design is available for training an SR model, thus the proposed design is only compared with a fixed-point implementation using the GD circuit in Fig. 12. The weights are considered as converged when the cross entropy is below 0.3. The proposed design

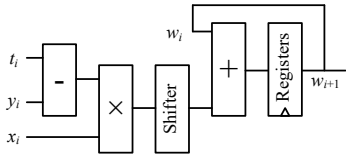


Fig. 12. Fixed-point LMS weight update circuit.

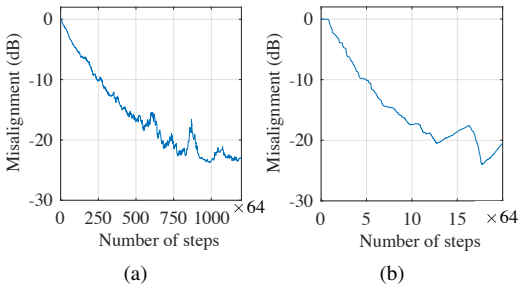


Fig. 13. Convergence curves in misalignment of the (a) SC-GDC-based, and (b) fixed-point LMS weight update units.

TABLE II
HARDWARE EVALUATION OF THE LMS WEIGHT UPDATE UNITS.

Metrics	SC-GDCs	[28]	Fixed-point	Ratio
Step size	2^{-11}	2^{-10}	2^{-6}	-
Steps	58504	16384	912	-
Min. time (ns)	6.0×10^4	6.5×10^7	3.6×10^3	17:18056:1
EPO (fJ)	1.2×10^4	9.1×10^7	9.5×10^4	1:7583:8
Total energy (fJ)	7.0×10^8	1.5×10^{12}	8.8×10^7	8:17045:1
TPA (Sa./$\mu s/\mu m^2$)	6.7×10^{-5}	1.0×10^{-8}	1.9×10^{-6}	6700:1:190
Misalign. (dB)	-23	-6	-23	-

takes about 4 epochs to converge when the step size is 2^{-8}, whereas 2 epochs are required for the fixed-point design with a step size of 2^{-7}. One epoch takes 60,000 clock cycles for both the stochastic and fixed-point circuits. The training samples are randomly shuffled at each epoch. As shown in Table III, the hardware evaluation results indicate that the proposed design costs 42.55% computation time of its fixed-point counterpart without any accuracy loss ("Aver. test Accu." in the table). Meanwhile, 85.6% total energy saving and 73.9× TPA are achieved by using the SC-GDCs. The small difference in accuracy could be caused by the random initialization and different order of the image samples.

C. Discussion

The hardware evaluation results indicate that for weight estimation tasks such as system identification, the proposed stochastic design using SC-GDCs obtains an adequate accuracy, although it takes a larger number of steps than its fixed-point counterpart. However, for applications such as image recognition that can tolerate more errors, the number of epochs required for the proposed stochastic design and conventional fixed-point design are on the same level, which indicates a high-performance and energy-efficient stochastic design. However, the accuracy of the image recognition of the MNIST dataset is quite low (around 92%) compared to the state-of-the-art result (around 99%). *This is due to the inherent simplicity of the SR model rather than the training method or the SC-GDC.* Next, a complex NN model that produces higher recognition accuracy is used to test the performance of the SC-GDCs, where hundreds of thousands of weights are to be optimized.

TABLE III
HARDWARE EVALUATION OF THE SR UNITS.

Metrics	SC-GDCs	Fixed-point	Ratio
Step size	2^{-8}	2^{-7}	-
Epochs	4	2	-
Min. time (ns)	2.0×10^5	4.7×10^5	1:2.35
EPO (fJ)	5.5×10^5	7.5×10^6	1:14
Total energy (fJ)	1.3×10^{11}	9.0×10^{11}	1:7
TPA (images/s/μm^2)	1.7×10^3	2.3×10^1	74:1
Aver. test Accu.	91.75%	91.69%	-

VII. Signed SC-GDC units training complex neural networks

A. Background for back-propagation

An NN consists of a set of neurons and the connections between them; the neurons are typically organized layer by layer. Fig. 14 shows an NN with one input layer, two hidden layers and one output layer. The output signals of an NN are generated based on the input signals and the weights of the connections. For an image recognition task, the input signals are the pixels of an image, and the output signals are the classification results. To produce a correct classification for an image, a GD algorithm can be used to train an NN by adjusting the weights of the connections. However, for the hidden layers in an NN, the target value and the loss function cannot be computed directly. Typically, it requires both forward- and backward-propagation (FP and BP) algorithms to obtain the gradients.

In FP, each neuron computes the weighted sum of the outputs from the previous layer (or the training data from the input layer). An activation function is then used to decide whether the neuron is activated based on the weighted sum result as shown in Fig. 14(b). The output layer is usually a softmax layer for a multi-class classification task. During FP, the weights remain unaltered.

In BP, the error signals, $\{e_j\}$, are first obtained as the differences between the outputs and the class-labels of the training data,

$$e_j = t_j - y_j, \tag{25}$$

where y_j is the jth output of the NN and t_j is the target output, i.e., the actual class-label in one-hot code. t_j is 1 when the input data belongs to class j; otherwise, it is 0. Then, the local gradient, $\delta_j^{(l)}$ for neuron j in layer l is computed using the error signals and the weights by

$$\delta_j^{(l)} = \begin{cases} e_j & \text{for neuron } j \text{ in output layer} \\ f'(v_j^{(l)}) \sum\limits_m \delta_m^{(l+1)} w_{m,j}^{(l+1)} & \text{for neuron } j \text{ in hidden layer } l, \end{cases} \tag{26}$$

where $v_j^{(l)}$ is the weighted sum of neuron j in layer l,

$$v_j^{(l)} = \sum_h w_{j,h}^{(l)} y_h^{(l-1)}. \tag{27}$$

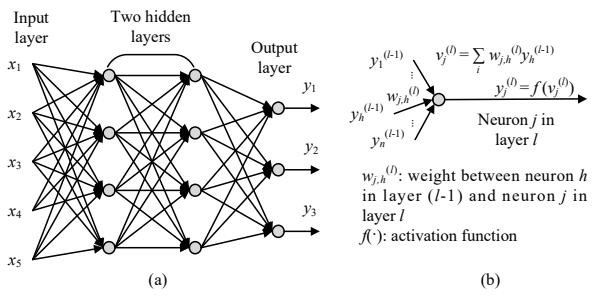


Fig. 14. (a) A multilayer neural network (NN). (b) The function of a neuron during forward propagation (FP).

$w_{j,h}^{(l)}$ denotes the weight of the connection between neuron h in layer $l-1$ and neuron j in layer l. When neuron j is in the softmax output layer, its local gradient equals to the error signal, as shown in the first equation in (26). The signal flow of the local gradient during BP in an NN is shown in Fig. 15.

Then, the gradient with respect to each weight is given by

$$\nabla w_{j,h}^{(l)} = -\delta_j^{(l)} y_h^{(l-1)}. \tag{28}$$

Note that when $l = 1$, $y_h^{(l-1)}$ is the training data from the input layer.

Finally, the GD algorithm can be used to optimize the weights with the gradient function of $\nabla w_{j,h}^{(l)}$.

B. Design of signed SC-GDCs

In an NN, data are usually normalized to have a mean of 0 and the weights are initialized with small RNs near 0 to improve the training efficiency [20]. It results in a lot of near-0 intermediate results during the computation. Meanwhile, the bipolar SC suffers the most from accuracy loss when representing near-0 numbers due to its large variance [30]. The reason is as follow. Let the probability of 1's in a stochastic sequence be p. The variance of the sequence is given by $p(1-p)/L$, where L is the sequence length. When $p = 0.5$, the variance reaches its maximum value, which indicates a possibly large error. For the bipolar representation, $x = 2p - 1$ ($x \in [-1, 1]$). So the stochastic sequence encodes x in the least accurate manner when $x = 0$ (or $p = 0.5$). It means that using the bipolar representation will dramatically increase the variance of the results, thus leading to increased error. However, it is necessary to be able to encode negative numbers by using stochastic sequences in this application. Thus, instead of using the bipolar representation, a sign bit is added to the unipolar representation to encode a negative number, which leads to the sign-magnitude representation [31]. The variance $p(1-p)/L$ approaches 0 when p (or x) approaches 0 when using the unipolar representation. In this way, the variance is very small for the encoded values near 0, so the

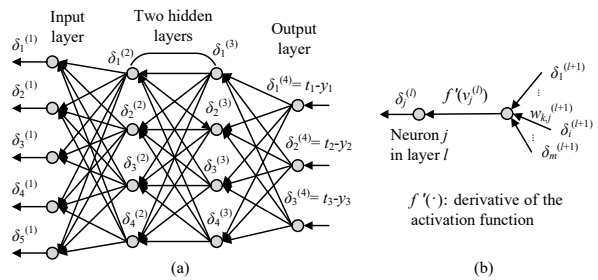


Fig. 15. (a) The signal flow of local gradients during the backward propagation (BP). (b) The local gradient is given by the product of the derivative of the activation function and the weighted sum of local gradient from the next layer.

computed results are more accurate than the ones using the bipolar representation.

The stochastic circuits are adjusted to work with the sign-magnitude representation. The signed SNG and multiplier are shown in Figs. 16(a) and (b) respectively. In the signed SNG, the n-bit input x is in 2's complement. Thus, the sign bit for x is its most significant bit, $x[n-1]$, denoted as X_{sign}. The absolute value of a negative number, $|x|$, is approximated by flipping all bits using inverters. The RNG and the comparator are then used to generate the magnitude bit, X. In the signed multiplier, the XOR gate is used to compute the sign bit and the AND gate serves as a unipolar stochastic multiplier. The symbol of a signed stochastic integrator using the sign-magnitude representation is shown in Fig. 16(c). The counter in the signed stochastic integrator updates its value according to Table IV. In this way, the signed stochastic integrator implements the same function as an ordinary stochastic integrator for (5), where a_i and b_i can take either 0, -1 or +1.

The signed stochastic integrator takes "differential" signals to update its value, i.e., one signal to increase the value and another to decrease the value. Thus, the local gradient signal in (26) has to be rewritten as a differential pair to work with the stochastic integrator. By applying the distributive law of multiplication, (25) and (26) are combined and rewritten as

$$\delta_{j,+}^{(l)} = \begin{cases} t_j & \text{for neuron } j \textit{ in output layer} \\ f'(v_j^{(l)}) \sum_k \delta_{k,+}^{(l+1)} w_{k,j}^{(l+1)} & \text{for neuron } j \textit{ in hidden layer } l, \end{cases}$$

$$\delta_{j,-}^{(l)} = \begin{cases} y_j & \text{for neuron } j \text{ in output layer} \\ f'(v_j^{(l)}) \sum_k \delta_{k,-}^{(l+1)} w_{k,j}^{(l+1)} & \text{for neuron } j \text{ in hidden layer } l, \end{cases} \tag{29}$$

and $\delta_j^{(l)} = \delta_{j,+}^{(l)} - \delta_{j,-}^{(l)}$. Thus (28) can be rewritten as

$$\nabla w_{j,h}^{(l)} = -(\delta_{j,+}^{(l)} - \delta_{j,-}^{(l)}) y_h^{(l-1)}. \tag{30}$$

As per the formulation of the SC-GDC, the signed SC-GDC is proposed to calculate the gradients and to update the weights as shown in Fig. 17. In a signed SC-GDC, two signed stochastic multipliers and a signed stochastic integrator are used. Also, the magnitude stochastic bits of $\delta_{j,+}^{(l)}$ and $\delta_{j,-}^{(l)}$, i.e., the unipolar stochastic sequences encoding $|\delta_{j,+}^{(l)}|$ and $|\delta_{j,-}^{(l)}|$ are generated by the same RNG to reduce hardware cost and variance of the results. It is assumed that $\delta_{j,+}^{(l)}$ and $\delta_{j,-}^{(l)}$, i.e., the local gradients are available to the signed SC-GDC by a BP. It means that (29) is computed by other methods (such as a systolic array) other than SC, because it results in a significant accuracy loss if the entire BP algorithm is computed in SC. However, the updating of the weights is purely implemented by using the signed SC-GDCs. The overflow of the counters is handled in a similar manner as for the SR units.

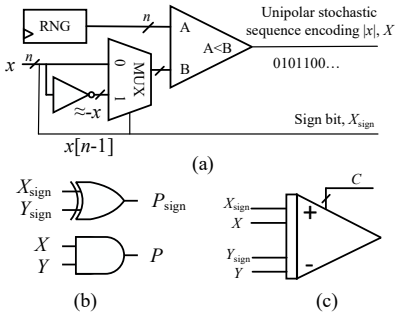


Fig. 16. (a) A signed stochastic number generator (SNG). (b) A signed stochastic multiplier. (c) A symbol of signed stochastic integrator.

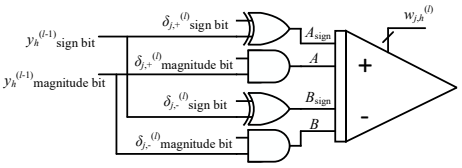


Fig. 17. Proposed signed SC-GDC. The magnitude bit is a unipolar stochastic bit used to encode the absolute value of the number.

C. Handwritten-digit recognition

The MNIST handwritten-digit dataset is used to test the effectiveness of the proposed signed SC-GDCs. The input data are pre-processed to have a mean value of 0. A 784-128-128-10 fully connected NN is used and hyperbolic tangent (tanh) function is selected as the activation function for the hidden layers, which is given by

$$y = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{31}$$

and its derivative function is

$$\tanh'(x) = 1 - y^2. \tag{32}$$

"Fully connected" means that each neuron in layer l has connections with every neuron in layer $l-1$. *Typically, a fully connected NN has more connections and more weights to be trained than a CNN. So, training a fully connected NN is a more challenging task for the proposed circuitry than training a CNN.* Therefore, a fully connected NN is selected instead of a CNN to evaluate the proposed design, though CNN has a better test accuracy in most cases.

TABLE IV
THE LOGIC OF SIGNED STOCHASTIC INTEGRATOR.

C_{i+1}	A_{sign}	A	B_{sign}	B	C_{i+1}	A_{sign}	A	B_{sign}	B
C_i	0	0	0	0	C_i	1	0	0	0
C_i-1	0	0	0	1	C_i-1	1	0	0	1
C_i	0	0	1	0	C_i	1	0	1	0
C_i+1	0	0	1	1	C_i+1	1	0	1	1
C_i+1	0	1	0	0	C_i-1	1	1	0	0
C_i	0	1	0	1	C_i-2	1	1	0	1
C_i+1	0	1	1	0	C_i-1	1	1	1	0
C_i+2	0	1	1	1	C_i	1	1	1	1

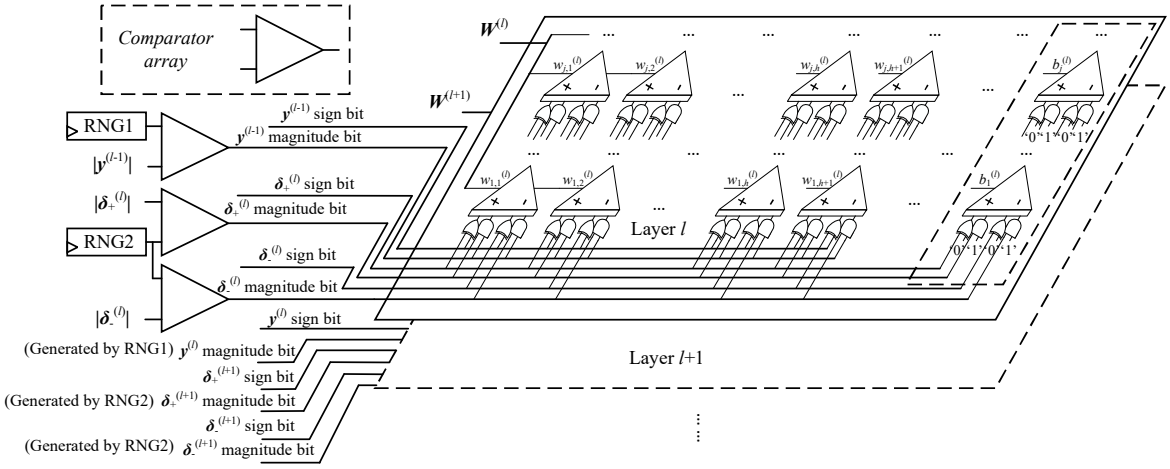


Fig. 18. Proposed signed SC-GDC array for the training of a multilayer NN.

Similar to the circuit for the training of the SR model in Fig. 9, an array of signed SC-GDCs are used to train the weights in all layers. In total, $784\times128+128\times128+128\times10=118016$ weights and $128+128+10=266$ biases are trained by 118282 signed SC-GDCs. Only the "naive" GD algorithm in (1) is considered to train the NN model. Other optimization techniques and variants of the GD algorithm, such as cross-validation, weight regularization, momentum terms and adaptive optimization, are not considered. The array of signed SC-GDCs works in the same manner as in the SR training unit. One clock cycle is required to train one sample. Also, the RNG can be shared among the inputs and two RNGs are sufficient to generate the stochastic sequences: one for $\{y_h^{(l-1)}\}$; one for $\{\delta_{j,+}^{(l)}\}$ and $\{\delta_{j,-}^{(l)}\}$. The proposed circuit for the training of a multilayer NN is shown in Fig. 18. The signed SC-GDCs are organized layer-wise to show their connections and signals instead of the actual mapping of the circuit. The rightmost column of the signed SC-GDCs in each layer is used to train the biases, which can be considered as "weights" with an input of constant 1. Hence, the input signals, $\mathbf{y}^{(l-1)}$, for these SC-GDCs encode a 1.

D. Experiments and results

The average test accuracy and cross entropy against epoch are shown in Fig. 19 over 10 trials to reduce the effect of random initialization. It illustrates that the stochastic circuit using the bipolar representation has a slow convergence process of the cross-entropy, thus leading to a low accuracy. Unless the model is trained by a smaller step size, the test accuracy produced by the bipolar stochastic circuits remains at around 95% after 20 epochs. Compared to the fixed-point implementation, the signed SC-GDCs produce a slightly lower accuracy with the same step size of 2^{-10}; however, it has a similar convergence speed. A double precision floating-point implementation is also compared as a reference; it shows a similar accuracy with the fixed-point design.

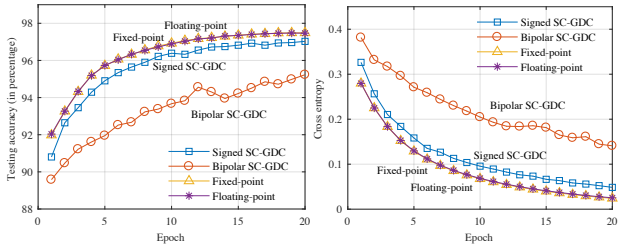
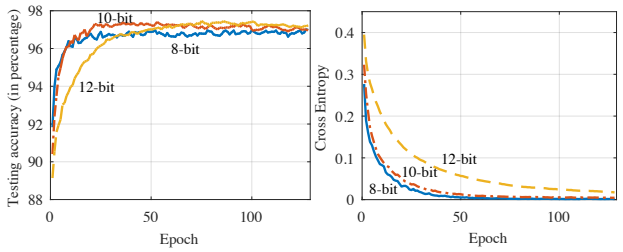


Fig. 19. (a) Test accuracy and (b) cross entropy produced by the signed SC-GDCs, the bipolar SC-GDCs and the fixed-point implementation.

Fig. 20 shows the classification accuracy and cross entropy for different widths of SC-GDCs, for which 8-, 10- and 12-bit SC-GDCs with the same initial values for weights are considered. The 8-bit design converges the fastest during the first few epochs. However, due to the large step size ($1/2^8$), it loses its advantage in accuracy after a few epochs to the 10-bit design, and converges to a slightly lower value. On the other hand, the 12-bit design can produce a finer estimate of the optimal weights. However, it takes a longer time to converge, which would incur a higher energy consumption.

The hardware cost of the circuits are measured and estimated as shown in Table V by using the same method as for the AF and SR applications. The fixed-point implementation used for comparison is shown in Fig. 12, where the additions and accumulations are implemented by fixed-point circuits. Table V shows that the signed SC-GDC-based design consumes 10.0% of the energy and 25.5% the computation time of the fixed-point implementation, while providing more than 50 times TPA, with a similar test accuracy.



(a) (b)

Fig. 20. (a) Test accuracy and (b) cross entropy produced by the signed SC-GDCs with different widths.

TABLE V
HARDWARE EVALUATION OF THE SIGNED SC-GDC ARRAY TRAINING A 784-128-128-10 NEURAL NETWORK.

Metrics	Signed SC-GDCs	Fixed-point	Ratio
Step size	2^{-10}	2^{-10}	-
Epochs	20	20	-
Min. time (ns)	1.2×10^6	4.7×10^6	1:4
EPO (fJ)	1.1×10^7	1.1×10^8	1:10
TPA (image/s/μm^2)	8.3×10^1	1.5	55:1
Aver. test Accu.	97.03%	97.47%	-

E. Related works and discussion

In [12], extended stochastic logic is used to implement both FP and BP of a multilayer perceptron. By using a binary search, a reconfigurable stochastic computational activation unit and an LFSR sharing scheme, the design achieves lower area and energy consumption compared to the binarized neural network (BNN), and the floating- and fixed-point implementations. With a similar network size and structure, a similar accuracy is obtained in this paper compared to [12]. However, a long sequence is required for [12] to achieve a high accuracy, which incurs a long latency. Specifically, 256 clock cycles with $16\times$ parallelization are required to handle one image with a maximum operation frequency of 112.4MHz, while it takes only 1 clock cycle in this work to accomplish the gradient accumulation of one training sample with a maximum operation frequency of 1.03GHz. Also, the use of extended stochastic logic in [12] requires extra stochastic divider and computation time to convert a stochastic sequence back to a binary number. However, a converter is not required in this design since the value stored in the SC-GDCs is already in 2's complement format.

In [7], the weights and activations are binarized to +1 or -1 to reduce the power consumption and hardware resources. However, the binarization is only applicable to the FP of an NN. In fact, by using real-valued variables and gradient during the training process, a larger workload is imposed on training binarized weights and activations. Compared to [7], this paper focuses on the efficient training of an NN by stochastic binarization of the gradient instead of the weights and activations. Also, dedicate hardware using stochastic circuits are proposed to perform the GD algorithm. It makes the SC-GDC applicable to most optimization tasks that can be solved by a GD algorithm, such as system identification that clearly cannot be solved by binarizing the weights.

Recently, a TernGrad method is proposed to reduce the communication cost for synchronizing gradients and parameters in distributed training [8]. In TernGrad, the gradient is compressed to only three levels, $\{-1, 0, +1\}$, stochastically. Compared to [8], this paper focuses on enhancing the computation efficiency instead of reducing the communication cost. Therefore, the activations and local gradients used for training the NN are stochastically binarized, and the stochastic bits are used in the computation of gradients by stochastic circuits instead of fixed- or floating-point multipliers and adders. Due to the simplicity of the proposed stochastic circuits, significant energy saving and hardware efficiency are achieved compared to conventional arithmetic circuits. Generated by the stochastic circuits, the stochastic bits encoding the gradients in an SC-GDC are similar to the ternary gradients, and can be directly used to reduce the communication cost.

The proposed design can also be adapted to deal with more complex learning tasks. For example, to implement a variant of GD algorithm using a dynamically adjustable step size, an additional stochastic sequence can be used to encode the step size. Then, one more stochastic multiplier can be used for the SC-GDC to multiply the additional stochastic sequence. To implement batch learning, multiple stochastic sequences encoding the gradient information can be used as the inputs of the SC-GDCs. It resembles a spiking neuron, in which the probability of an action potential occurring in a postsynaptic neuron is determined by multiple excitatory and inhibitory synapses in the presynaptic neurons.

VIII. CONCLUSION

In this paper, a novel SC-GDC for online learning is proposed by using stochastic circuits to implement the GD algorithm. By encoding the gradient information for each training sample as stochastic bit, the SC-GDC provides an unbiased estimate for the optimized weights in a learning algorithm. The proposed SC-GDC units are then utilized in system identification and handwritten-digit recognition using SR model. Compared to a conventional SC system identification design, the proposed design provides 6.7×10^3 times TPA improvement, $1000\times$ speed-up and 99.9% energy reduction. For the SR model, the proposed SC-GDC consumes 42.6% of the computation time and less than 15% of the energy with $73.9\times$ of the TPA of a fixed-point design, while providing a similar accuracy.

Moreover, a signed SC-GDC is proposed to improve the accuracy of the bipolar SC-GDC; it is then used to implement the training of a complex NN. For a 784-128-128-10 fully connected NN, the use of the sign-magnitude representation significantly improves the accuracy of SC, thus leading to a faster convergence compared to the use of the bipolar representation. Compared to its fixed-point counterpart, a similar accuracy is obtained while

90% energy saving per training sample, 74% reduction in training time and more than 50× improvement in TPA are achieved.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NIPS*, 2012.

[3] F. Akopyan and *et al.*, "TrueNorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Trans. on Computer-Aided Design (CAD) of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[4] N. P. Jouppi and *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. of ISCA*, 2017.

[5] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. of ISCA*, 2016.

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[8] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "TernGrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. of NIPS*, 2017.

[9] 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*, 2017.

[10] S. C. Smithson, K. Boga, A. Ardakani, B. H. Meyer, and W. J. Gross, "Stochastic computing can improve upon digital spiking neural networks," in *Proc. of SiPS*, 2016.

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

[12] 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, Sept 2018.

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

[14] B. Li, M. H. Najafi, and D. J. Lilja, "An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams," in *Proc. of ASAP*, 2015.

[15] 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 *Proc. of DATE*, 2017.

[16] 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 *Proc. of ASPLOS*, 2017.

[17] 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*, vol. 25, no. 10, pp. 2688–2699, Oct 2017.

[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 *Proc. of DAC*, 2016.

[19] Y. Xie, S. Liao, B. Yuan, Y. Wang, and Z. Wang, "Fully-parallel area-efficient deep neural network design using stochastic computing," *IEEE Trans. on Circuits and Systems II: Express Briefs*, 2017.

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

[21] B. R. Gaines, *Stochastic Computing Systems*. Boston, MA: Springer US, 1969, pp. 37–172.

[22] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel, "IIR filters using stochastic arithmetic," in *Proc. of DATE*, 2014.

[23] S. Liu and J. Han, "Hardware ODE solvers using stochastic circuits," in *Proc. of DAC*, 2017.

[24] W. Qian, C. Wang, P. Li, D. J. Lilja, K. Bazargan, and M. D. Riedel, "An efficient implementation of numerical integration using logical computation on stochastic bit streams," in *Proc. of ICCAD*, 2012.

[25] H. Jiang, L. Liu, and J. Han, "An efficient hardware design for cerebellar models using approximate circuits: special session paper," in *Proc. of CODES*, 2017.

[26] B. Farhang-Boroujeny, *Adaptive filters: theory and applications*. John Wiley & Sons, 2013.

[27] "MNIST for ML beginners," https://www.tensorflow.org/get_started/mnist/beginners, 2017, accessed: 2017-10-25.

[28] H. Jiang, C. Shen, P. Jonker, F. Lombardi, and J. Han, "Adaptive filter design using stochastic circuits," in *Proc. of ISVLSI*, 2016.

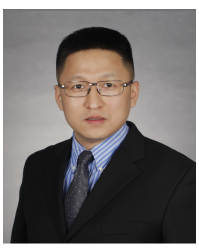
[29] Z. Yuan, J. Li, Z. Li, C. Ding, A. Ren, B. Yuan, Q. Qiu, J. Draper, and Y. Wang, "Softmax regression design for stochastic computing based deep convolutional neural networks," in *Proc. of GLSVLSI*, 2017.

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

[31] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *Proc. of DAC*. ACM, 2017, p. 29.

Siting Liu received the B.Eng. and M.Eng. degrees in electrical engineering and automation from Harbin Institute of Technology, Harbin, Heilongjiang, China, in 2012 and 2014, respectively. He is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His current research interest includes stochastic computing and its application in .

Honglan Jiang (S'14) received the B.Sc. and Master degrees in instrument science and technology from Harbin Institute of Technology, Harbin, Heilongjiang, China, in 2011 and 2013, respectively. Since September 2013, she has been a Ph.D. candidate student in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. Her current research interests are approximate computing and stochastic computing.



Leibo Liu (M'10-SM'17) received the B.S. degree in electronic engineering and the Ph.D. degree with the Institute of Microelectronics, both from Tsinghua University, Beijing, China, in 1999 and 2004, respectively. He is currently a Full Professor with the Institute of Microelectronics, Tsinghua University. His current research interests include reconfigurable computing, mobile computing, and very large-scale integration digital signal processing.

Jie Han (S'02-M'05-SM'16) received the B.Sc. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree from Delft University of Technology, The Netherlands, in 2004.

He is currently an associate professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada. His research interests include approximate computing, stochastic computation, reliability and fault tolerance, nanoelectronic circuits and systems, novel computational models for nanoscale and biological applications.

Dr. Han and coauthors received the Best Paper Award at the International Symposium on Nanoscale Architectures (NanoArch 2015) and Best Paper Nominations at the 25th Great Lakes Symposium on VLSI (GLSVLSI 2015), NanoArch 2016 and the 19th International Symposium on Quality Electronic Design (ISQED 2018). He was nominated for the 2006 Christiaan Huygens Prize of Science by the Royal Dutch Academy of Science. His work was recognized by *Science*, for developing a theory of fault-tolerant nanocircuits (2005). He is currently an associate editor for IEEE Transactions on Emerging Topics in Computing (TETC) and IEEE Transactions on Nanotechnology. He served as a General Chair for GLSVLSI 2017 and the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2013), and a Technical Program Committee Chair for GLSVLSI 2016 and DFT 2012.