Learning in Memristor Crossbar-Based Spiking Neural Networks Through Modulation of Weight-Dependent Spike-Timing-Dependent Plasticity

Nan Zheng, *Student Member, IEEE*, and Pinaki Mazumder, *Fellow, IEEE*

***Abstract*—In this paper, we propose a methodology to design learning systems based on a memristor crossbar structure. Learning is carried out with the help of a hardware-friendly spike-timing-dependent plasticity learning rule. Several simplifications and adaptations are made in order to apply the learning algorithm to memristor-based neural networks. The difficulties in conducting division and representing signed weights are circumvented using the proposed techniques. In addition, different conductance-changing behaviors are considered to demonstrate the effectiveness of the proposed algorithm and architecture when applied to practical memristor devices. Furthermore, various nonidealities existing in the neural network, including variations in CMOS neurons and memristor synapses and noises associated with updating the synaptic weights, are considered. We demonstrate that the proposed learning algorithm and hardware architecture are robust against most variations and noises. This robustness of learning is promising, as variations and stochastic behaviors of memristor devices are usually substantial. Memristor-based neural networks with the proposed learning algorithm are benchmarked with the MNIST handwritten digits recognition task. A recognition accuracy as high as 97.10% is demonstrated.**

***Index Terms*—Memristor, crossbar, supervised learning, machine learning, neuromorphic computing, spiking neural network, deep learning, hardware neural network, spike-timing-dependent plasticity, MNIST benchmark.**

I. Introduction

SINCE they were first demonstrated in 2008 [1], memristors have drawn many researchers' attentions. As the missing fourth circuit component, the existence of memristors was hypothesized four decades ago [2]. A memristor is essentially a resistor with memory. The resistance is determined by the amount of charges that pass through the device [3]. To date, various types of memristive devices have been fabricated and demonstrated such as the ferroelectric memristor [4], the memristor based on a single nanowire [5], and the TiO_2-based memristor [6], etc.

Numerous attempts have been made over the past decade to build neuromorphic hardware with memristors, as a memristor device satisfies almost all the requirements to be an artificial synapse. Just as their CMOS counterpart, memristor-based neural networks in the literature can be divided into the rate-based artificial neural networks (ANNs) [7]–[9] and the event-triggered spiking neural networks (SNNs) [10]–[13]. The conventional ANN was extensively studied in both the machine learning community and the circuit community, as its underlying mathematical foundation is well developed and they can be conveniently implemented in a conventional general-purpose processor. In recent years, however, more and more researchers in the circuit community started working on SNNs, motivated by several advantages that SNNs have.

Regardless of the benefits that SNNs can provide, the development of energy-efficient SNNs is seriously restricted by the lack of effective learning algorithms. The huge success achieved by the conventional ANN-based deep learning is largely attributed to the power of backpropagation learning algorithm [14]. Errors at output layers are propagated back to each synapse layer by layer with the help of the chain rule. Thanks to the well-established mathematical model of ANNs, the gradient associated with each synapse is readily available to be utilized in a gradient descent learning. Unfortunately, the conventional backpropagation method cannot be directly employed in an SNN due to the complicated dynamics of the spiking neurons.

One popular way to empower an SNN with intelligence is to first train an ANN using a conventional digital computer, and then the trained ANN is converted to an SNN with a mapping algorithm [15]–[18]. This way of learning has the advantage that well-developed learning techniques in ANNs can be employed to improve the performance. However, this mapping-based learning can only yield sub-optimal results, as the optimization is conducted on the ANN counterpart instead of the SNN itself. This is particularly true for a memristor-based SNN due to the difficulty of a precise mapping. For an SNN where memristor devices are employed as analog memory, the chip-in-the-loop configuration is more feasible as the variations in device parameters can be corrected through feedback.

Manuscript received December 25, 2017; revised February 19, 2018; accepted March 22, 2018. Date of publication March 30, 2018; date of current version May 8, 2018. This work was supported by the National Science Foundation under Grant 1710940. The review of this paper was arranged by Associate Editor M. Niemier. *(Corresponding author: Nan Zheng.)*

The authors are with the University of Michigan, Ann Arbor, MI 48105 USA (e-mail: zhengn@umich.edu; mazum@umich.edu).

Digital Object Identifier 10.1109/TNANO.2018.2821131

Another disadvantage of the mapping-based learning is that it is not able to provide an on-line learning capability, which is required by many applications. In order to empower SNNs with the on-line learning capability, many learning algorithms for SNNs have been proposed. For example, a supervised learning rule was proposed in [10] where a delta learning rule is employed. However, how to apply that rule in a multilayer network for a complicated task is not obvious, as the layer-by-layer learning in [10] requires the desired outputs of hidden layer units to be computed beforehand.

In addition to the learning algorithms that are derived mathematically, biologically plausible spike-timing-dependent plasticity (STDP) learning rule is proposed in recent years. STDP is a phenomenon observed in experiments on biological neural networks and it has been long hypothesized as the mechanism with which mammal can learn. It is demonstrated that the amount of changes in the synaptic strength of a synapse depends on the relative timings of presynaptic and postsynaptic spikes. When a postsynaptic spike occurs shortly after a presynaptic spike, the synapse undergoes a long-term potentiation, and the increase in synaptic weight decays exponentially as a function of the difference between the two spike timings. The reverse is true if a postsynaptic spike occurs before a presynaptic one. The synapse experiences a long-term depression in this case. Learning with STDP rules has been studied by many researchers [11], [12], [19], [20] in recent years. It is demonstrated in [12] and [19] that by smartly choosing the action potential of the firing neurons, the weight update in a memristor-based synapse approximately follows the shape of a biological STDP, providing a feasible way to implement STDP-based learning. Nevertheless, for most learning examples demonstrated in the literature, STDP is employed as a biologically plausible and empirically successful learning rule without its underlying principle being explained. The application of STDP-based learning is also mainly restricted to unsupervised learning. Despite of being effective, the usefulness of unsupervised learning is rather limited, as most applications of neural networks nowadays are based on supervised learning or reinforcement learning, which require the neural networks to be able to approximate arbitrary functions. Furthermore, how to conduct learning in a multilayer neural network with the conventional free-running STDP algorithm is not obvious. Nevertheless, most successes in various machine learning tasks are achieved by the deep neural networks [21]–[23]. In addition, it is unclear how closely one should follow the weight update curve in an STDP protocol in order to achieve a successful learning. It is well known that the nanoscale memristor devices based on resistive oxide or metallic nanowire generally suffer more from spatial variations compared to silicon based devices such as transistors [24]. The probabilistic filament formation process also results in high temporal variation, leading to the stochastic switching phenomenon observed in memristor devices [25]. Therefore, tightly controlling device behavior is extremely difficult and energy-consuming, especially considering that millions of memristor synapses are normally involved in a large-scale neural network.

To tackle abovementioned difficulties, we apply a recently developed hardware-friendly learning rule to memristor crossbar-based SNNs in this paper. It is shown in [26] and [27] that spike-timing (ST) information can serve as a good estimation for gradient components needed in a stochastic gradient descent (SGD) learning. This learning algorithm is particularly suitable for memristor-based neural networks, as the dynamics of the network is generally unknown due to process variations, which often poses difficulties for learning algorithms that require the complete information of the neural network dynamics. In order to deploy our learning rule in a memristor crossbar-based network, several adaptations are made. A division-free learning rule, which only keeps the sign information, is introduced to get rid of the energy-consuming division. In addition, a fixed-polarity network configuration is proposed to circumvent the problem that a memristor device can only represent one type of synapse (either excitatory or inhibitory) by its nature. Furthermore, one of the biggest issues that all the analog and mixed-signal neural networks face is the detrimental effect caused by the process variations. We examine how variations in neuron circuits and memristor synapses affect the system-level performance and provide corresponding solutions. The main objective of this paper is not to propose a specific implementation of memristor-based neural networks. Instead, we focus on investigating how to conduct effective on-line learning with memristor devices when realistic physical constraints are present. With this goal in mind, algorithms and architectures are developed to facilitate the learning in a memristor crossbar structure. All the simulation results presented in this paper are obtained from a simulator implemented in C++. In the simulator, behavior-level models are used to represent various circuit components as well as memristor devices.

In the following, we briefly review the learning algorithm introduced in [27] in Section II. In Section III, several adaptations to the learning algorithm are proposed so that it can be applied to a memristor crossbar-based neural network. In Section IV, effects of different variations in neurons and synapses are examined to show the robustness of the learning algorithm. Benchmark results obtained with the proposed learning algorithms and architectures are demonstrated in Section V. Section VI concludes this work.

II. Weight-Dependent STDP Learning

Fig. 1 illustrates the configuration of a memristor crossbar-based SNN. The ith neuron located at the lth layer is denoted as x_i^l, where $l = 0, 1, \ldots, o$. The number of neurons at layer l is denoted as N_l. Following the derivation in [27], the spike trains from a presynaptic x_i^l and a postsynaptic neuron x_j^{l+1} can be described by

$$x_i^l[n] = \sum_m \delta\left[n - n_{i,m}^l\right] \tag{1}$$

$$x_j^{l+1}[n] = \sum_m \delta\left[n - n_{j,m}^{l+1}\right] \tag{2}$$

where $\delta[n]$ is the unit sample sequence. We restrict us to a discrete-time implementation where spikes are expressed as pulses. This type of implementation is very popular in hardware SNNs. All the neurons are synchronous with a global clock,

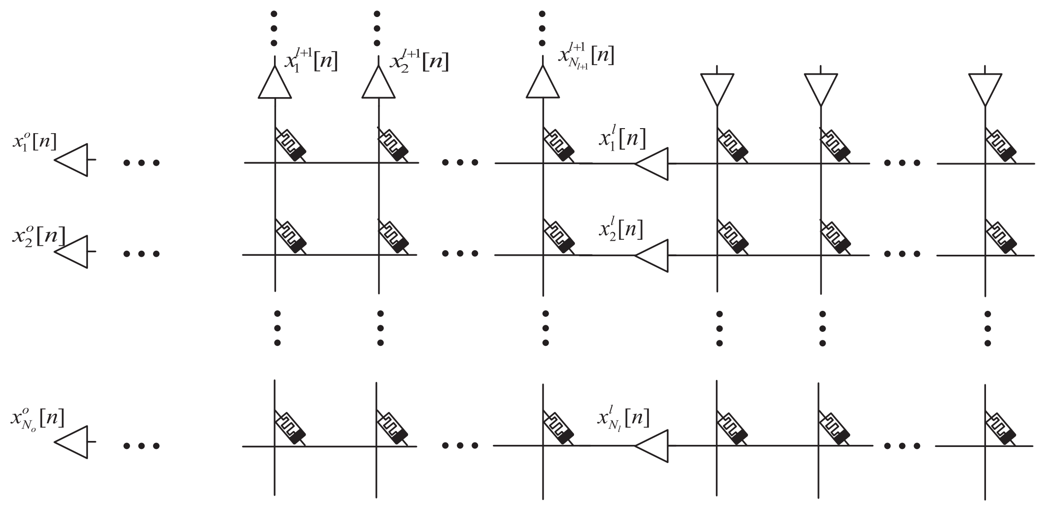

Fig. 1. Illustration of a multilayer memristor crossbar-based spiking neural network.

and the synchronization period is called a tick, following the convention used in [28].

The neuron model we use in this paper is a modified leaky integrate-and-fire (LIF) model [26], [27], [29], although many other neuron models can also be employed. The dynamics of the neuron model is

$$V_j^{l+1}[n] = \max\left(0,\ V_j^{l+1}[n-1] + \sum_i w_{ij}^l x_i^l[n-1] - L_j^{l+1} - x_j^{l+1}[n-1]\cdot th_j^{l+1}\right) \tag{3}$$

$$x_j^{l+1}[n] = \begin{cases} 0, & V_j^{l+1}[n] < th_j^{l+1} \\ 1, & V_j^{l+1}[n] \geq th_j^{l+1}\ \&\ x_j^{l+1}[n-1] = 0 \\ 1-R, & V_j^{l+1}[n] \geq th_j^{l+1}\ \&\ x_j^{l+1}[n-1] = 1 \end{cases} \tag{4}$$

where w_{ij}^l is the synaptic weight of the synapse connecting neuron x_i^l and x_j^{l+1}, V_j^{l+1} is the membrane voltage, L_j^{l+1} is the leakage, th_j^{l+1} is the threshold of the neuron, and R is a random variable obeying the Bernoulli distribution that is used to achieve a stochastic refractory period [27].

A variable $stdp_{ij}^l$ and its arithmetic average over a finite length of time can be defined as

$$stdp_{ij}^l[n] = x_i^l[n-T]\left(1 - x_i^l[n-T-1]\right) \times \left(\sum_{m=1}^{WIN_{STDP}} x_j^{l+1}[n+m-1] - \sum_{m=1}^{WIN_{STDP}} x_j^{l+1}[n-m]\right) \tag{5}$$

$$\overline{stdp_{ij}^l} = \sum_{n=T+1}^{D_L} stdp_{ij}^l[n]\,/\,(D_L - T) \tag{6}$$

where D_L is the learning duration, which serves as a design parameter and T is the system delay for evaluating spikes. In (5), WIN_{STDP} is used to specify the window size of summation. It is found in the simulations that a WIN_{STDP} of one is enough to feature an effective learning and a larger window does not result in significant improvements [27]. Therefore, its value is restricted to one in this paper.

It is shown in [27] that the gradient components needed in a SGD learning process can be estimated according to

$$\frac{\partial \mu_j^{l+1}}{\partial \mu_i^l} \approx \frac{E\left[\overline{stdp_{ij}^l}\right]}{\mu_i^l\left(1-\mu_i^l\right)} \approx \frac{\overline{stdp_{ij}^l}}{\overline{x_i^l}\left(1-\overline{x_i^l}\right)} \tag{7}$$

$$\frac{\partial \mu_j^{l+1}}{\partial w_{ij}^l} \approx \frac{E\left[\overline{stdp_{ij}^l}\right]}{w_{ij}^l\left(1-\mu_i^l\right)} \approx \frac{\overline{stdp_{ij}^l}}{w_{ij}^l\left(1-\overline{x_i^l}\right)} \tag{8}$$

where μ_i^l and μ_j^{l+1} represent the mean firing rates of neuron x_i^l and x_j^{l+1} and $\overline{x_i^l} = \sum_{n=T+1}^{D_L} x_i^l[n]/(D_L - T)$ represents the sample mean of the time sequence $x_i^l[n]$ during one learning iteration.

In a learning process, the errors at the output neurons need to be propagated back to each synapse in the network. This can be achieved through

$$\frac{\partial \mu_k^o}{\partial w_{ij}^l} = \frac{\partial \mu_k^o}{\partial \mu_j^{l+1}} \cdot \frac{\partial \mu_j^{l+1}}{\partial w_{ij}^l} \tag{9}$$

where

$$\frac{\partial \mu_k^o}{\partial \mu_j^{l+1}} = \sum_{i_o=k}^{k} \sum_{i_{o-1}=1}^{N_{o-2}} \cdots \sum_{i_{l+2}=1}^{N_{l+2}} \sum_{i_{l+1}=j}^{j} \prod_{p=l+1}^{o-1} \frac{\partial \mu_{i_{p+1}}^{p+1}}{\partial \mu_{i_p}^p} \tag{10}$$

Alternatively, a direct backpropagation is possible as shown in (11)

$$\frac{\partial \mu_k^o}{\partial \mu_j^{l+1}} = \frac{E\left[\overline{cstdp_{jk}^{l+1}}\right]}{\mu_j^{l+1}\left(1-\mu_j^{l+1}\right)} \tag{11}$$

where $cstdp_{jk}^{l+1}$ is the cross-layer STDP information with a definition similar to the one in (5) except delay across multiple layers are needed to be taken into considerations.

The intuition behind (7)–(11) is that each spike in an SNN serves as a small perturbation to the system. By observing how the outputs of the system change, we are able to estimate the gradient at each point. More importantly, when the spike timings for the neurons are uncorrelated, the gradients associate with each perturbation point can be estimated simultaneously. After obtaining the gradients, we can conduct a gradient descent learning with the objective to minimize the error function

$$E = \frac{1}{2}\sum_{k=1}^{N_o} (e_k^o)^2 \tag{12}$$

where $e_k^o = \overline{x_k^o} - t_k^o$ is the error at each output neuron. Here, t_k^o is the target mean firing rate of neuron x_k^o.

With the error function defined in (12), the synaptic weights can be updated according to (13) toward the gradient-descent direction in order to minimize the error function.

$$\Delta w_{ij}^l = -\alpha \cdot \sum_{k=1}^{N_o} \frac{\partial E}{\partial \mu_k^o} \cdot \frac{\partial \mu_k^o}{\partial w_{ij}^l} = -\alpha \cdot \sum_{k=1}^{N_o} e_k^o \cdot \frac{\partial \mu_k^o}{\partial w_{ij}^l} \tag{13}$$

where α is the learning rate.

III. Algorithm Adaptations

In this section, we make several adaptations to the learning algorithm presented in Section II so that it can be better applied to memristor-based neural networks. The configuration of a typical memristor crossbar-based multilayer feedforward neural network is illustrated in Fig. 1. The triangles in the figure represent neurons, and they are connected through memristor synapses as shown in the figure. When a neuron fires, a voltage is formed at the two terminals of corresponding memristors. The currents flowing on the synapses are integrated at the input of each neuron. A spike is emitted by a neuron if the accumulated charge on that neuron exceeds a pre-defined threshold. In (3), a current-based synapse is assumed. In circuit, this can be realized by forcing a virtual ground condition at the input of each neuron. Under this circumstance, the weight of a synapse, w_{ij}^l is proportional to the conductance of that synapse, G_{ij}^l. In other words, w_{ij}^l can be treated as a scaled version of G_{ij}^l. Therefore, we use these two quantities interchangeably in the rest of this paper for the ease of discussion.

Learning to classify the hand-written digits in the MNIST dataset is employed as an example. To accelerate the simulation process, a down-sampled dataset is used. The original 28×28 images are down-sampled to 16×16 images. Fig. 2 compares the first 16 data in these two sets of images. It can be observed that the down-sampled images preserve most important information included in the original dataset. The real values from the MNIST dataset is injected as an incremental membrane potential for each input neuron. Ten output neurons are used to represent corresponding ten digits. The objective of the learning is to infer the correct digit when an image is applied to the neural network. The output neuron corresponding to the shown

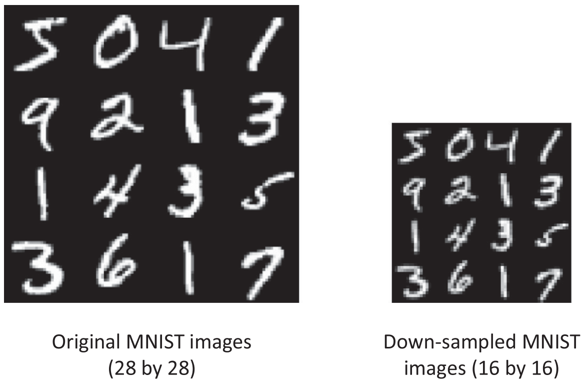
Original MNIST images (28 by 28)
Down-sampled MNIST images (16 by 16)

Fig. 2. Comparison of the original MNIST images and the down-sampled MNIST images.

digit should fire with a high density of ρ_H, whereas all other neurons fire with a lower density of ρ_L. To test the trained neural network, a digit is presented to the network, after an inference duration of D_I, the output neuron with the highest firing density is chosen as the winning neuron, and its corresponding digit is picked as the inferred result.

For all the results reported in Sections III and IV, learning is conducted on the down-sampled MNIST images and the first 500 images in the training set are used for learning. For testing, all 10000 images in the test set are used. For each result reported, 10 runs of learning are carried out. A run is a complete learning process including several learning iterations, where one iteration is a process of going through all 500 training images. The error bars in the figure correspond to the 95% confidence interval. The reason to use only the first 500 images in the training set for learning is to accelerate the process of evaluating the learning algorithm and various proposed techniques. Nevertheless, benchmark performances obtained with the full training set is reported in Section V for the purpose of comparing with the state-of-the-art result.

The first issue needs to be addressed is the computation of the division, as shown in (7) and (8). The term $(1 - \overline{x_i^l})$ can be neglected without introducing a significant error, considering the sparsity of spike trains outputted by each neuron. In a purely digital implementation, the divide-by-weight operation in (8) can be easily realized, as the weights are stored in the memory in a digital form. It is, however, not straightforward to conduct this division in a memristor crossbar implementation, as the weights are stored as analog values in the memristors. Power-consuming readout and quantization would be needed if a similar division operation in a digital implementation were employed. Fortunately, it was shown in [26] and [27] that division with only one-bit precision (i.e., with only the sign information) can still result in a successful learning. In other words, we can use the term $\overline{stdp_{ij}^l}/\text{sgn}(w_{ij}^l)$ to approximate the gradient in the learning process, where sgn$(\cdot)$ specifies the sign operation. To illustrate this, simulations are conducted. The learning results obtained with three different configurations are compared in Fig. 3. The first configuration is the baseline that employs (8) for weight updates. The second configuration uses the same learning rule except the terms $(1 - \overline{x_i^l})$ in (7) and (8)

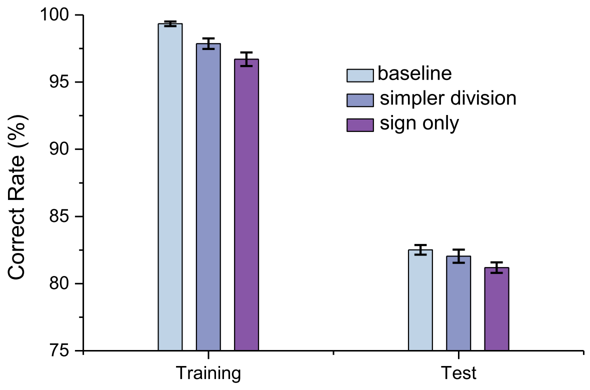

Fig. 3. Comparison of the learning performance achieved with and without exact division.

are ignored. The third configuration is the division-free configuration, where the term $\overline{stdp_{ij}^{l}}/\text{sgn}(w_{ij}^{l})$ is employed for a weight update. In other words, a STDP-like rule is applied to excitatory synapses, whereas an anti-STDP-like rule is applied to inhibitory synapses. Good learning results are achieved in all three cases. The results obtained with the baseline algorithm is slightly better than other simplified versions. Nevertheless, using the gradients estimated from the simplified rules can result in a comparable performance while remarkably reducing the computational complexity.

Even though the gradient estimation can be greatly simplified, the weight update still poses some difficulties in a memristor crossbar-based implementation. Different from a digital synapse, a memristor-based analog synapse can only represent a positive weight by its nature. A negative weight can be obtained through reversing the polarity at the current summing node. Therefore, a synapse that spans both positive and negative range can be implemented with two memristors. Through controlling the polarities of spikes, the effective synaptic weight is the difference between the resistances of the two memristors. However, when such a synapse is employed in the proposed algorithm, a mechanism of detecting the polarity of the effective weight is needed. How to conduct such a detection in an energy-efficient way is not obvious. To circumvent this difficulty, we propose a network configuration with fixed polarities. In the configuration, each synapse is either excitatory or inhibitory during the whole learning process. The polarities of the synapses are decided before the learning starts. Fig. 4 illustrates this concept by showing all the synapses connected to one presynaptic neuron. The excitatory and inhibitory synapses in the same column are complementary. Polarities of the synapses can be chosen randomly or with some strategies. The hardcoded configuration in Fig. 4(a) is conceptually simple yet not flexible in practice. A more programmable configuration is shown in Fig. 4(b), which is compatible with a standard densely connected crossbar structure. An excitatory synapse is achieved by programming the inhibitory synapse to G_{off}, the minimum conductance of the memristor devices. The "off" synapses of the network need to be refreshed periodically to ensure they stay at "off" state as the learning is going on. To verify the effectiveness of the proposed network configuration, simulations are

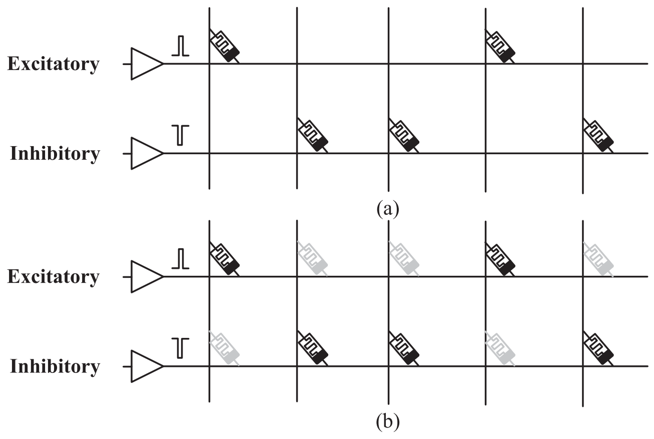

Fig. 4. Proposed fixed-polarity memristor crossbar-based neural network. (a) Polarities of the synaptic weights are hardcoded. (b) Polarities of the synaptic weights are run-time programmable.

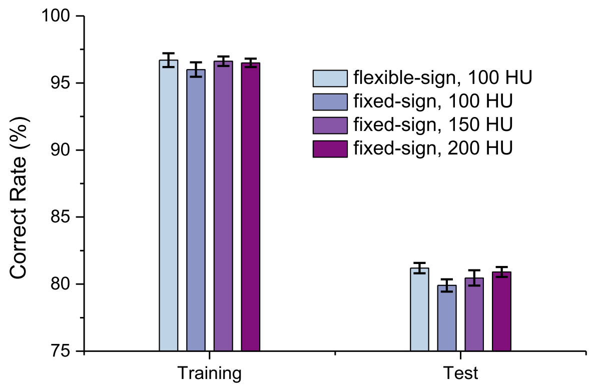

Fig. 5. Comparison of the classification correct rate for several network configurations. The flexible-sign configuration is the case where the synaptic weights can be programmed to be either excitatory or inhibitory, whereas the fixed-sign configuration corresponds to the configurations shown in Fig. 4.

carried out and the obtained results are compared with the baseline result in Fig. 5. In the figure, the network with fixed-polarity synapses has a slightly deteriorated performance compared to the flexible-polarity one when the same number of hidden-layer units are used. This is expected since restricting the polarity of each synapse reduces the entropy of the network. To give a fairer comparison, the learning results obtained by networks with more hidden-layer units are also compared. As the number of hidden-layer units increases, the performance is improved and the performance loss due to fixing the polarities of the synapses are compensated.

There are two ways of updating weights according to (13). One is to store the spike information temporarily in a memory and update the synaptic weights only once at the end of one learning iteration, as shown in (14).

$$\Delta w_{ij}^{l} = -\alpha \cdot e_{j}^{l+1} \cdot \frac{\overline{stdp_{ij}^{l}}}{\text{sgn}\left(w_{ij}^{l}\right)} \tag{14}$$

We call this way of updating weights a cumulative update. Another way is to update the synaptic weight whenever there is

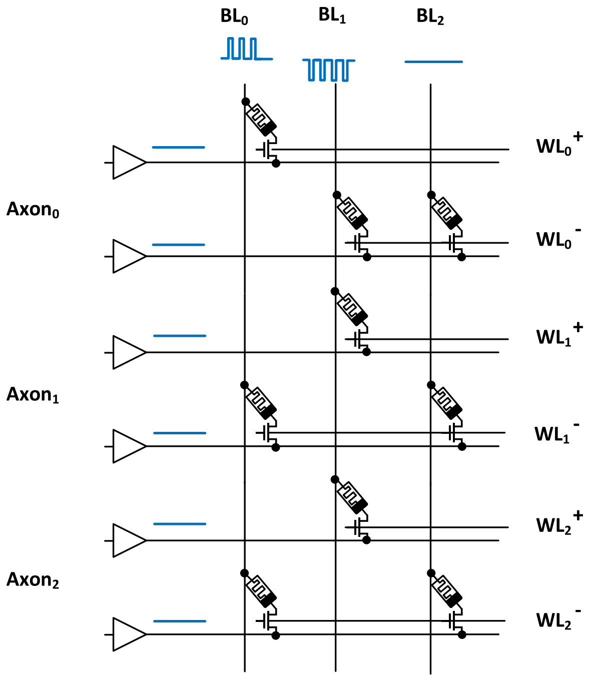

Fig. 6. An example of programming the memristor crossbar when the proposed learning algorithm is employed.

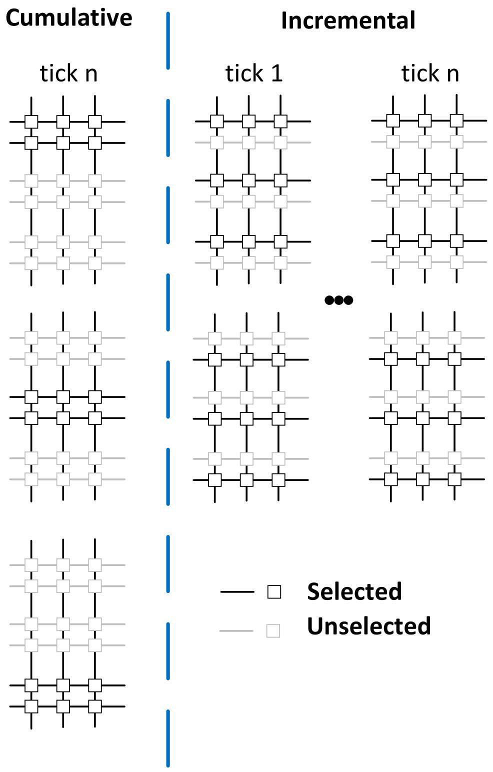

Fig. 7. Comparison of the updating schedules for the cumulative and incremental update style.

an STDP event, as shown in (15).

$$\Delta w_{ij}^{l} = \sum_{n} -\alpha \cdot \frac{e_j^{l+1}}{\operatorname{sgn}\left(w_{ij}^{l}\right)\left(D_L - T\right)} \cdot stdp_{ij}^{l}[n] \tag{15}$$

We call this type of updating an incremental update. The incremental update does not require memory to hold the spike-timing information as this information is implicitly accumulated in the memristor itself. It is expected that these two ways of updating weights lead to similar results, considering the weight changes during one iteration are small. Indeed, a good rule of thumb is to control the learning rate such that the weight changes in one iteration are less than one thousandth of the weights [30]. One possible way of updating weights according to (14) and (15) is illustrated in Fig. 6. A one-transistor-one-memristor configuration is assumed in the figure. For each axon, there are two word-lines (WLs) controlling the excitatory and inhibitory synapses, respectively. Whenever a WL is selected, the conductances of the memristors on that row can be altered through injecting currents through the bit-line (BL). Either pulse trains with a programmed pulse numbers or pulses with a modulated pulse width can be used for programming purposes. A pulse train is used for demonstration in Fig. 6. The programming schedule of the cumulative update is different from that of the incremental update. For the cumulative style, updating the synaptic weight is similar to writing the memory in a memristor-based memory. This is illustrated in Fig. 7. The conductances of the memristors are changed row by row. Since the two rows associated with the same axon is mutual-exclusive in the locations of the memristor. These two rows can be updated simultaneously, as shown in Fig. 7. In contrast, for the incremental update, all the excitatory synapses can be updated together, whereas all the inhibitory synapses can be updated together. This is because $stdp_{ij}^{l}[n]$ only takes three values: -1, 0, and 1. Therefore, all the synapses connected to the same postsynaptic neuron experience the same change in magnitude. The sign information implied by the term $\operatorname{sgn}(w_{ij}^{l})$ in (15), nevertheless, requires the positive and negative synapses to be programmed separately. In addition, only active axons need to be selected for an update, as the value of $stdp_{ij}^{l}[n]$ for inactive axons is 0. In Fig. 7, it is assumed that all the axons are active. In spite of the fact that the synapses associated with different axons can be updated simultaneously, an incremental update requires the update to occur at every tick, as illustrated in Fig. 7. It is worth noting that the method of updating weights presented in Figs. 6 and 7 is just one example of illustrating how the proposed learning algorithm and architecture function conceptually. There exist other ways of updating weights based on the actual need.

Fig. 8 compares the results obtained with these two different weight update styles. For the purposes of comparison, both the hardcoded and the programmable network configurations are employed. All four cases yield similar learning results. From a circuit designer's point of view, an incremental update has the advantage that it requires less amount of memories to store the ST information, whereas the cumulative update style is more efficient as the number of conductance update is greatly reduced. Another advantage of the cumulative update, as is demonstrated in Section IV, is that it is more robust against variations in memristors.

For a real memristor device, we might not always be able to control the increment and decrement in conductance linearly. In

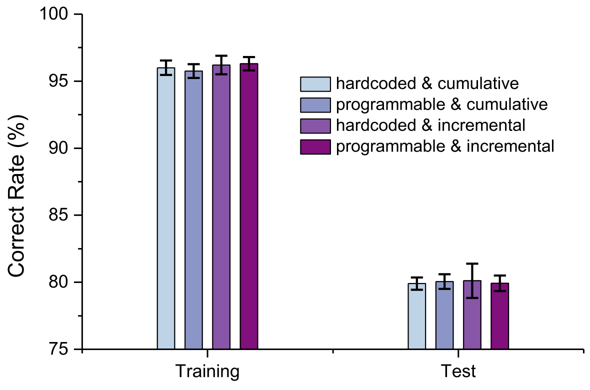

Fig. 8. Comparison of the learning performances obtained with the cumulative/incremental weight update. Both the hardcoded and programmable fixed-polarity network configurations are simulated.

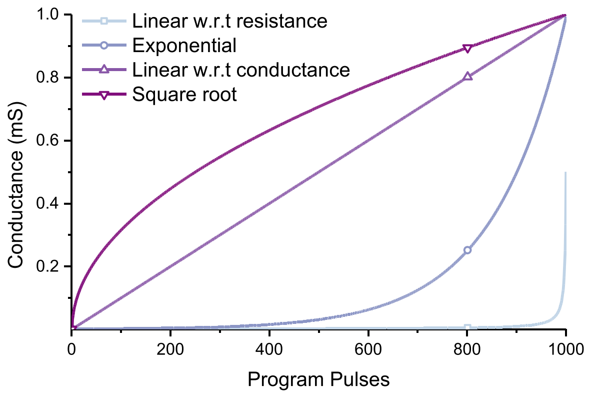

Fig. 9. Illustration of different conductance-changing characteristics.

other words, the weight change in a memristor may be a function of the weight itself. In general, the change of the conductance of a memristor device follows

$$\Delta G_{ij}^{l} = g\left(\Delta s_{ij}^{l}, G_{ij}^{l}\right) \tag{16}$$

where s_{ij}^{l} is a linearly (or approximately linearly) controllable state variable.

To more accurately capture the characteristics of a memristor device, we employ the VTEAM model in [31]. s_{ij}^{l} in (16) corresponds to the state variable in the VTEAM model. Linear control over Δs_{ij}^{l} can be achieved by modulating the pulse width or the number of pulses that is used to program memristors. Various conductance-changing characteristics are illustrated in Fig. 9 and the recognition accuracies achieved with these models after learning are compared in Fig. 10. In this set of simulations, only $g(\cdot)$ that are even functions of Δs_{ij}^{l} are considered. In other words, increments and decrements in weights are symmetrical. The more general asymmetrical case is considered in Section IV. A G_{off} of $10^{-6} S$ and G_{on} of $10^{-3} S$ are used, which correspond to the minimum and the maximum conductance that the memristors can achieve. The linear-R (defined as linear with respect to resistance) model shown in (17) and the exponential models shown in (18) are commonly used models in

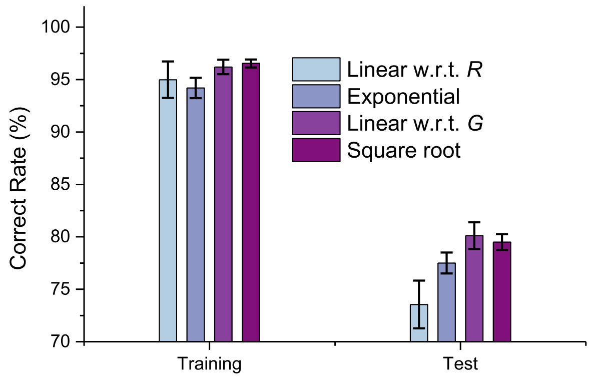

Fig. 10. Comparison of the learning performances achieved with different conductance-changing characteristics shown in Fig. 9.

VTEAM [31].

$$G_{ij}^{l} = \left(R_{\text{ON}} + \frac{R_{\text{OFF}} - R_{\text{ON}}}{s_{\text{off}} - s_{\text{on}}}\left(s_{ij}^{l} - s_{\text{on}}\right)\right)^{-1} \tag{17}$$

$$G_{ij}^{l} = \frac{\exp\left(-\frac{\lambda\left(s_{ij}^{l} - s_{\text{on}}\right)}{s_{\text{off}} - s_{\text{on}}}\right)}{R_{\text{ON}}} \tag{18}$$

The linear-G (defined as linear with respect to conductance) model is the one assumed in (14) and (15). Since both linear-R and exponential models are convex functions of the weight, we also compare the results obtained from a concave dependency on weight, a square-root model, in order to demonstrate the effectiveness of the proposed algorithm. In a square-root model, the relationship between the state variable s_{ij}^{l} and the weight is

$$G_{ij}^{l} = G_{\text{OFF}} + \left(G_{\text{ON}} - G_{\text{OFF}}\right)\sqrt{\frac{s_{ij}^{l} - s_{\text{on}}}{s_{\text{off}} - s_{\text{on}}}} \tag{19}$$

As shown in Fig. 10, impressive results can be achieved from all conductance-changing characteristics, even for the badly-behaved linear-R characteristic which may introduce many saddle points in the learning. This demonstrates the effectiveness and robustness of the proposed learning algorithm when applied to different memristor devices.

In principle, the conductance of a memristor device can be adjusted continuously through, for example, a pulse-width modulated voltage pulse. Nevertheless, the granularity of the conductance change, in reality, might be restricted by the device and the programming circuit. In training neural network models, a rule of thumb is to control the weight change to be less than one thousand of the weight itself. However, such a fine granularity might be hard to achieve with a memristor device. Fig. 11 investigates how the weight-programming granularity affects the learning. In the simulation, an exponential weight-changing characteristic is used for demonstration. The maximum allowed change in the state variable s, which is called $\Delta s_{\max}$, is swept from 1% to 5% in the simulation. The minimum programming granularity, which is called $\Delta s_{\min}$, is determined by both $\Delta s_{\max}$ and the number of bits (NOB) used to represent the programming voltage pulses. For example, a 4-bit precision indicates

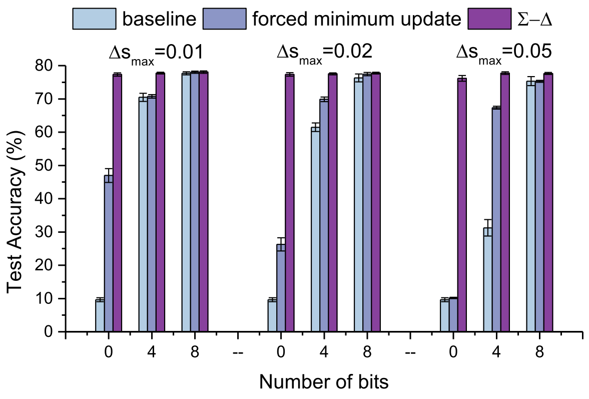

Fig. 11. Comparison of the test accuracies achieved with different programming granularities.

that the absolute change in conductance can only be a multiple of $2^{-4} \cdot \Delta s_{\max}$. As shown in the figure, learning with a baseline updating method experiences a degradation in performance when a $\Delta s_{\max}$ of 1% and a NOB of 4 is used, which corresponds to a programming granularity of 0.0625%. Such a degradation in the performance is mainly attributed to the fact that small changes are rounded to zero and are not applied to the weight updates. One method that can mitigate this problem is to force a minimum update on the weight. In other words, in the case where the weight update is less than the minimum allowed weight update, the minimum amount is applied. Such a method indeed helps improve the performance, as shown in Fig. 11. A more effective approach is to use a $\Sigma - \Delta$ modulation, which is a well-known technique to represent high-precision data with low-precision ones. The $\Sigma - \Delta$ modulator used here can be conveniently implemented with following equations

$$\Delta s'[n] = \left\lfloor \frac{\Delta s[n] + \Delta s_{\text{int}}[n]}{\Delta s_{\min}} \right\rceil \Delta s_{\min} \qquad (20)$$

$$\Delta s_{\text{int}}[n+1] = \Delta s_{\text{int}}[n] + \Delta s[n] - \Delta s'[n] \qquad (21)$$

where $\Delta s'[n]$ is the actual change in the state variable and it is rounded to a multiple of $\Delta s_{\min}$ in (20). $\Delta s_{\text{int}}[n]$ is the output of the integrator used to hold the residue generated in the $\Sigma - \Delta$ modulation.

With the help of the $\Sigma - \Delta$ modulation, the requirement on the programming granularity is greatly reduced as shown in Fig. 11. It is even possible to use only one programming levels to achieve a successful learning. A granularity of 5% is enough for the learning. Such a relaxed requirement on the programming granularity is mainly attributed to the fact that small changes are accumulated in the integrator before applying to the memristors.

IV. Non-idealities

One of the biggest problems with the analog implementations of neural networks is the performance degradation due to process variations. Conventionally, an off-line learning suffers from the mismatches between the assumed device parameters in the learning process and the actual device parameters in the inference phase. The proposed on-line learning, however, tackles this problem through an on-chip feedback, where the learning is conducted in situ with the actual devices. Nevertheless, there still exist some non-idealities in the real circuits that might possibly deteriorate the performance. In this section, we examine the effects of various non-idealities in neuron circuits and synapses, respectively.

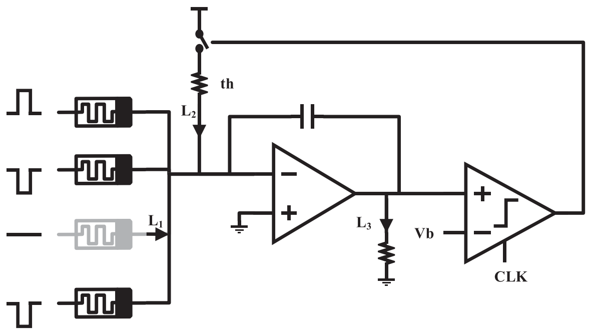

Fig. 12. An example of the analog neuron that can be employed in a memristor crossbar-based neural network.

A. Neuron Non-idealities

Fig. 12 illustrates one possible realization of the neurons in the proposed memristor crossbar-based neural network. An integrator sums up the currents from all firing neurons. A virtual ground is forced at the input of the neuron, as the proposed algorithm assumes a current-based synapse. At the output of the integrator, a clocked comparator is used to sample the membrane voltage. Once the voltage exceeds the threshold voltage, the comparator outputs a logic high, which serves as a voltage spike. Meanwhile, a current packet is delivered to the neuron through a one-bit digital-to-analog converter (DAC) in order to implement the dynamics shown in (3). The comparators are clocked by a global clock and the neurons are only allowed to spike synchronously with the clock. There are two important parameters in the neuron model as shown in (3) and (4): the threshold voltage and the leakage. These two parameters can be controlled well in a digital design. However, they are subject to variations in an analog neuron. As shown in Fig. 12, the variation of the threshold voltage mainly comes from the feedback DAC, whereas the variation of the leakage can be attributed to three terms. L_1 is the leakage from non-selected memristors. It can come from the sneak-path leakage or the subthreshold leakage of transistors, depending on the detailed implementation of the memristor array. L_2 is the leakage from the feedback DAC or any other branches that are connected to the input of the integrator. L_3 represents the leakage at the output of the integrator, including the kick-back noise from the comparator and the leakage that is intentionally put at the output of the integrator. L_3 can also model the leakage caused by the non-ideality of the integrator, namely the low-frequency error due to the finite gain of the operational amplifier [32].

Since the threshold and leakage terms do not show up in the proposed learning algorithm, it is expected that learning is somewhat robust against variations in these two parameters. Fig. 13 compares the results obtained with different levels of variations. In the simulation, the variations of the threshold voltage and

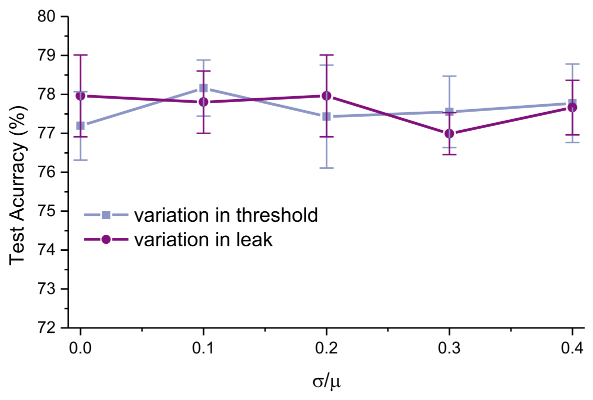

Fig. 13. Comparison of the test accuracies obtained with different levels of leakage and threshold variations. Learning is robust against variations in these two parameters.

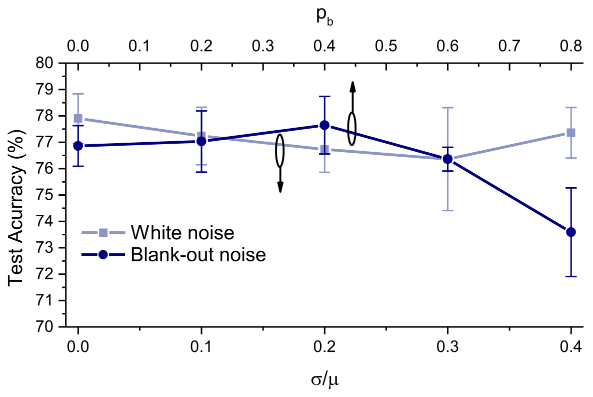

Fig. 14. Comparison of the test accuracies obtained with different levels of noises in changing the conductances of memristors. Two types of noises are considered: a normally distributed white noise with a standard deviation of σ and a blank-out noise with a blank-out probability of p_b.

the leakage for each neuron are treated as normally distributed random variables and they are added to the nominal threshold voltage and leakage. The standard deviations of these random variables are swept and the normalized value are shown in the x-axis. As shown in the figure, the learning is quite robust against the variations in both the threshold voltage and the leakage. This is good news to analog neurons, which typically have a good energy-efficiency and compact size yet suffer from process variations.

B. Synapse Non-idealities

It is well known that memristors are subject to significant variations. There are two types of variations associated with each memristor device: the temporal variation and the spatial variation. The temporal variation mainly comes from the stochastic formation of the conducting filament in a memristor device, whereas the spatial variation originates from the mismatch between devices when they are fabricated.

Let us examine the effect of the temporal variations first. It was observed that the conductance change in a memristor device is stochastic in nature, resulted from the probabilistic formation of the conducting filament [24], [25]. Fig. 14 compares the learning results obtained with two different types of noises. One is a white noise with a Gaussian distribution. With this noise, the actual weight update is scaled by a random variable with a mean of 1 and a standard deviation of σ. Another noise considered here is a multiplicative blank-out noise that prevents the weight update to be conducted, and the blank-out rate is p_b. As shown in the figure, the proposed algorithm is robust against noise in the weight update, making it well-suited for memristor-based neural network where significant variations exist. The learning rule can tolerate variations and noises as long as they are not severely biased. This robustness mainly comes from the stochastic nature of the learning algorithm.

Next let us discuss the spatial variation. Ideally, when we apply voltage pulses with the same amplitude and the same duration to two synapses with the same weight, we expect to see identical changes in the weights for these two synapses according to (16). In reality, however, there exist variations in each memristor synapse. As a result, different weight updates may occur even when the same write voltage pulse is applied. Fig. 15 compares the learning results obtained with different levels of variations in controlling s. In the simulation, we scale the weight update for each synapse by a normally distributed random variable with a mean of 1. Since we are interested in the spatial variation, these random variables are fixed throughout the learning in one run. When the variations in incrementing and decrementing of the weights are the same, or mathematically when the $g(\cdot)$ in (16) is an even function of Δs_{ij}^{l}, the curve with a label of "Sym" in the figure is produced. It is shown that the learning is robust against this type of variations. On the other hand, when the variations are not symmetrical, learning is deteriorated, especially for the case where the incremental update is employed. It is worth noting that the performance degradation caused by the imbalance in weight update is not unique to the proposed learning algorithm. Rather, it is a common threat to all SGD learning. Such a performance degradation is attributed to the bias term generated by the imbalance in the weight updates. More formally, assume the weight update is

$$\Delta w = \Delta w_0 + n_w \tag{22}$$

where Δw_0 is the desired weight update that helps move the cost function to its minimum and n_w is the noise term that comes from either the stochastic nature of a SGD learning process or from the STDP term.

For the convenience of analysis, let us assume n_w obeys a normal distribution with a probability density function of $\phi(n_w/\sigma_n)$, and $g(\cdot)$ has the following form

$$g(\Delta w, w) = \begin{cases} g_p \Delta w, & \Delta w > 0 \\ g_n \Delta w, & \Delta w < 0 \end{cases} \tag{23}$$

With above assumptions, it can be shown that the expectation of the weight update becomes

$$E(\Delta w) = \Delta w_0 + \frac{(g_p - g_n)\,\sigma_n}{\sqrt{2\pi}} \tag{24}$$

In a normal SGD, $E(\Delta w)$ should be equal to Δw_0, as the noise term n_w is unbiased, and the SGD is robust against a zero-mean noise. When the weight update in a memristor is

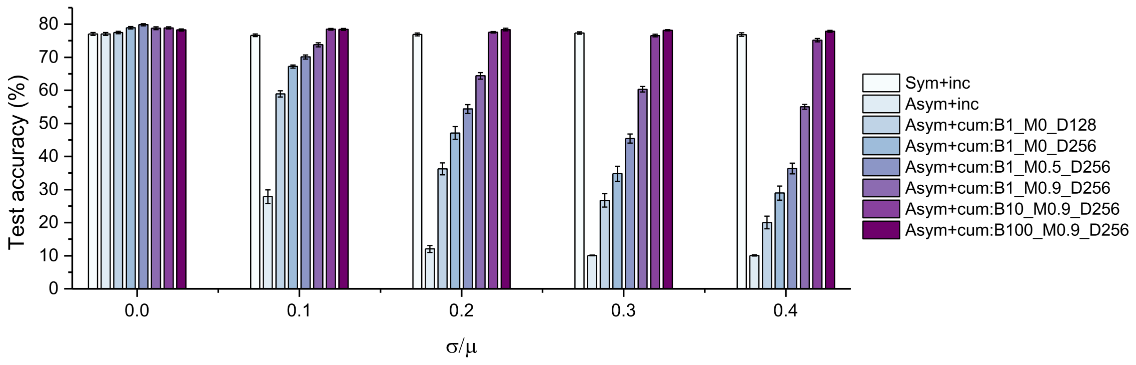

Fig. 15. Comparison of the test accuracies achieved with different non-idealities and learning hyperparameters. In the figure, "Sym" and "Asym" represent that the variations in incrementing and decrementing the conductance are symmetrical and asymmetrical, respectively. "inc" and "cum" represent the incremental and the cumulative update style, respectively. The numbers after letter "B", "M" and "D" represent the size of the mini-batch, the coefficient for the momentum, and the learning duration, respectively.

not symmetrical, however, a bias term is generated, as shown in (24). This bias term is detrimental to learning, as a constant error that yields a weight update that is equal to the bias term is needed to maintain a zero weight update in the steady state.

To reduce this bias term, one way is to make the updates of weights in memristors as symmetrical as possible. This may be done by properly engineering the device. Another way is to reduce σ_n. This can be achieved by using a cumulative update style, as noise is first averaged out before applying to the memristors. Fig. 15 compares the results achieved with the incremental and cumulative updates. The cumulative update is helpful in mitigating the performance degradation brought by the asymmetrical variations. Nevertheless, when the variation becomes more significant, more advanced techniques are needed. Three techniques are employed here to reduce the noise associated with the gradient-descent process. The first one is to use a longer learning duration. Increasing the learning duration is helpful in reducing the noise associated with the estimated gradients. As the proposed learning algorithm is based on stochastic approximation, a longer evaluation time can lead to a more accurate gradient [27]. The second technique is to use momentum [33] as

$$\Delta w\left(t\right) = \gamma \Delta w\left(t-1\right) - \alpha \cdot \frac{\partial E}{\partial w}\left(t\right) \tag{25}$$

The technique of momentum is widely used in gradient descent learning. Applying the momentum to a weight update is similar to applying a filter. It is expected that a larger momentum coefficient γ can lead to a better filtration of high-frequency noise, resulting in better performance when the asymmetrical variations exist. The third technique introduced is to use a mini-batch [33]. Momentum can help filter the updated weight in time domain, whereas the mini-batch technique helps filter the updated weight by averaging over multiple input samples. The results obtained with these three techniques are compared in Fig. 15. It is observed that all proposed techniques successfully improve the learning results. A longer learning duration, a larger momentum coefficient, and a bigger mini-batch size are helpful in achieving a better performance.

In addition to the random asymmetrical variations, a systematic asymmetrical weight-changing characteristic might exist in a real memristor device. Such an asymmetrical characteristic was observed in many different fabricated memristors. To study this effect, we adopt the memristor model used in [11] for simulations. The increment and decrement of the conductances are expressed as

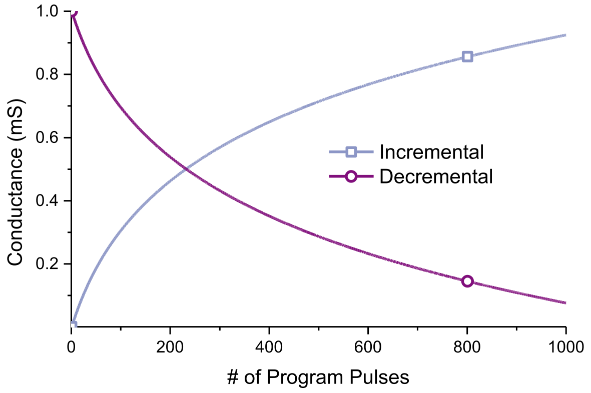

Fig. 16. Illustration of the weight-changing characteristics of the memristor model employed in [11].

$$\Delta G_p = a_p \exp\left(-b_p \frac{G - G_{\text{off}}}{G_{\text{on}} - G_{\text{off}}}\right) \tag{26}$$

$$\Delta G_n = a_n \exp\left(-b_n \frac{G_{\text{on}} - G}{G_{\text{on}} - G_{\text{off}}}\right) \tag{27}$$

The weight-changing characteristic of this model is illustrated in Fig. 16. As observed in the figure, the slope for incrementing and decrementing the conductance are different. Such an asymmetrical conductance-changing feature poses a similar problem in learning as the asymmetrical variations. The abovementioned three techniques can be exploited here to improve the performance in this case as well. The results are compared in Fig. 17. For comparison purposes, the learning results obtained from a symmetrical linear-G model is also simulated. It is observed in the figure that while the increased learning duration, momentum coefficient and mini-batch size do not change the results obtained from the symmetrical model, it brings remarkable improvements to the asymmetrical case. Such a result demonstrates

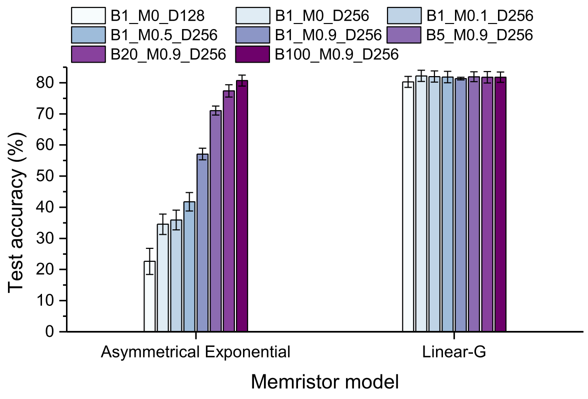

Fig. 17. Comparison of the test accuracies obtained with different hyperparameters. The numbers after letters "B", "M", and "D" represent the size of the mini-batch, the coefficient for the momentum, and the learning duration, respectively.

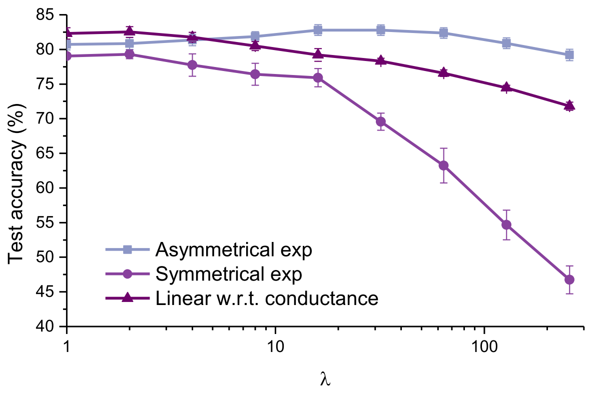

Fig. 18. The obtained test accuracy as λ increases.

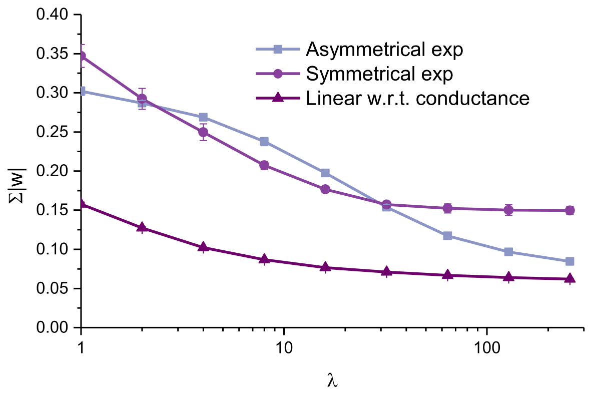

Fig. 19. The obtained sum of the absolute synaptic weights in the neural network when different λ is used.

that the proposed learning algorithm is effective for networks consist of various memristor models.

One possibility to further improve the performance when an asymmetrical conductance-changing characteristic is present is to encourage a small weight. The intuition behind this is that an asymmetrical characteristic such as the one shown in Fig. 16 tends to force the synaptic weights of the neural network to distribute toward the region where the slopes for increasing and decreasing the weight are approximately equal. This can be seen from (24) where the term $(g_p - g_n)\sigma_n/\sqrt{2\pi}$ serves as a regularization term. Recognizing that, one can intentionally strength the decrement in the weights. Such an arrangement is similar to the regularization used in training deep ANNs. Mathematically, this can be expressed as

$$\Delta w' = \begin{cases} \Delta w, & \Delta w \cdot w > 0 \\ \lambda \Delta w, & \Delta w \cdot w < 0 \end{cases} \tag{28}$$

where $\Delta w'$ is the actual weight update. The coefficient λ is a hyperparameter. A larger λ implies that small weights are more preferred. Similar to the regularization, the introduction of λ helps achieve smaller synaptic weights as well. This can be very useful in reducing the power consumption. A significant portion of power consumed in a memristor crossbar-based neural network is generated by the currents that flow through the memristors. This type of power consumption is proportional to $\sum |w_{ij}|$. Apparently, smaller synaptic weights are helpful in keeping this power consumption low.

Figs. 18 and 19 compare the test accuracy and the sum of the absolute values of all the synaptic weights in the neural network. It is observed that the test accuracy is improved for the asymmetrical exponential model when a λ that is larger than 1 is used. It is also noticed that increasing λ is helpful in reducing the absolute values of the synaptic weights. In addition, the performance of the learning is not affected noticeably until λ becomes larger than 10. Such an insensitivity to λ also demonstrates again the robustness of the proposed learning algorithm.

TABLE I
BENCHMARK PERFORMANCE ACHIEVED WITH DIFFERENT MEMRISTOR MODELS

Type of network	Memristor model	Network configuration	Test accuracy
Memristor crossbar	asymmetrical exponential	784-300-10	97.03%
		784-400-10	97.10%
	Linear-G	784-300-10	96.51%
		784-400-10	96.76%
Non-memristor baseline [27]	-	784-300-10	97.20%

V. BENCHMARKS

In this section, the proposed learning algorithm and various techniques are examined with the standard MNIST benchmark. The original 28 × 28 images are employed. The complete training set that contains 60000 hand-written digits are used for training and the test is conducted with the whole test set. Two different memristor models are evaluated: the linear-G model and the asymmetrical exponential model. The maximum and minimum achievable conductances are set as 1 mS and 1 μS, respectively. Only the results obtained from the cumulative update are reported as this update style can yield better performance, as shown in previous sections. The benchmark results are summarized in Table I. For all the results shown in the table that are obtained from memristor-based network, a mini-batch size of

100 and a momentum coefficient of 0.9 are used. For the asymmetrical exponential model, a λ of 32 is used. For the purpose of comparison, the results reported in [27], which is obtained from the original algorithm, is also compared in Table I.

100 iterations are conducted. During each iteration, all 60000 training images are fed to the neural network. Test accuracy is recorded for each iteration. The test accuracies for memristor-based network shown in Table I are obtained by averaging the accuracies obtained during the 91th iteration and the 100th iteration. Good recognition accuracies are achieved for both memristor models, demonstrating the efficacy of the learning algorithm.

VI. Conclusion

In this paper, we propose to use memristor crossbar to conduct learning tasks with the help of a weight-dependent STDP learning rule. The original learning algorithm is tailored to fit into the memristor crossbar context. It is demonstrated that only sign information associated with each synaptic weight is necessary to conduct an effective learning. To reduce the overhead posed by the sign detection, a fixed-polarity network configuration is proposed with two different programming styles. In order to demonstrate our proposed learning algorithm and hardware architecture can be applied to various memristor devices, we show examples of successful learning with different conductance-changing characteristics. It is found that even though certain conductance-changing behaviors outperform the others, the difference is marginal. In addition, we show that our proposed learning system is quite robust against different types of variations in device parameters. The only variation that may lead to a degraded performance is the asymmetrical variation in the conductance update. We propose multiple techniques to mitigate the performance degradation caused by this non-ideality. The proposed learning algorithm and techniques are verified with the help of the MNIST benchmark task. Recognition rates of 97.10% and 96.76% are achieved by two memristor crossbar-based SNNs with two different memristor models.

References

[1] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[2] L. Chua, "Memristor—The missing circuit element," *IEEE Trans. Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, Sep. 1971.

[3] O. Kavehei, A. Iqbal, Y. S. Kim, K. Eshraghian, S. F. Al-Sarawi, and D. Abbott, "The fourth element: Characteristics, modelling and electromagnetic theory of the memristor," *Proc. R. Soc. London A Math. Phys. Eng. Sci.*, vol. 466, no. 2120, pp. 2175–2202, 2010.

[4] A. Chanthbouala *et al.*, "A ferroelectric memristor," *Nature Mater.*, vol. 11, no. 10, pp. 860–864, 2012.

[5] S. L. Johnson, A. Sundararajan, D. P. Hunley, and D. R. Strachan, "Memristive switching of single-component metallic nanowires," *Nanotechnology*, vol. 21, no. 12, 2010, Art. no. 125204.

[6] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature Nanotechnol.*, vol. 3, no. 7, pp. 429–433, 2008.

[7] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 10, pp. 2408–2421, Oct. 2015.

[8] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor crossbar-based neuromorphic computing system: A case study," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 10, pp. 1864–1878, Oct. 2014.

[9] S. P. Adhikari, H. Kim, R. K. Budhathoki, C. Yang, and L. O. Chua, "A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 1, pp. 215–223, Jan. 2015.

[10] D. Chabi, Z. Wang, C. Bennett, J.-O. Klein, and W. Zhao, "Ultrahigh density memristor neural crossbar for on-chip supervised learning," *IEEE Trans. Nanotechnol.*, vol. 14, no. 6, pp. 954–962, Nov. 2015.

[11] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Trans. Nanotechnol.*, vol. 12, no. 3, pp. 288–295, May 2013.

[12] T. Serrano-Gotarredona, T. Prodromakis, and B. Linares-Barranco, "A proposal for hybrid memristor-CMOS spiking neuromorphic learning systems," *IEEE Circuits Syst. Mag.*, vol. 13, no. 2, pp. 74–88, Secondquarter 2013.

[13] J. A. Starzyk and Basawaraj, "Memristor crossbar architecture for synchronous neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 8, pp. 2390–2401, Aug. 2014.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.

[15] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proc. 2015 Int. Joint Conf. Neural Netw.*, 2015, pp. 1–8.

[16] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *Proc. 2011 IEEE Custom Integr. Circuits Conf.*, 2011, pp. 1–4.

[17] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Front. Neurosci.*, vol. 7, p. 178, 2013.

[18] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *Int. J. Comput. Vis.*, vol. 113, no. 1, pp. 54–66, 2015.

[19] B. Linares-Barranco, T. Serrano-Gotarredona, L. A. Camuñas-Mesa, J. A. Perez-Carrasco, C. Zamarreño-Ramos, and T. Masquelier, "On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex," *Front. Neurosci.*, vol. 5, 2011, Art. no. 26.

[20] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Front. Comput. Neurosci.*, vol. 9, p. 99, 2015.

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

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Syst.*, 2012, pp. 1097–1105.

[23] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, 2010.

[24] P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Trans. Nanotechnol.*, vol. 13, no. 2, pp. 283–293, Mar. 2014.

[25] S. H. Jo, K.-H. Kim, and W. Lu, "Programmable resistance switching in nanoscale two-terminal devices," *Nano Lett.*, vol. 9, no. 1, pp. 496–500, 2008.

[26] N. Zheng and P. Mazumder, "Hardware-friendly actor-critic reinforcement learning through modulation of spike-timing-dependent plasticity," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 299–311, Feb. 2017.

[27] N. Zheng and P. Mazumder, "Online supervised learning for hardware-based multilayer spiking neural networks through the modulation of weight-dependent spike-timing-dependent plasticity," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published.

[28] F. Akopyan *et al.*, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput. Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, 2015.

[29] A. S. Cassidy *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Proc. 2013 Int. Joint Conf. Neural Netw.*, 2013, pp. 1–10.

[30] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 599–619.

[31] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A general model for voltage-controlled memristors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 8, pp. 786–790, Aug. 2015.

[32] R. Stata, "Operational integrators," *Analog Dialogue*, vol. 1, pp. 1–9, 1967.

[33] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural Networks: Tricks of the Trade.*, New York, NY: Springer, 2012, pp. 599–619.

Nan Zheng (S'13) received the B.S. degree in information engineering from Shanghai Jiao Tong University, Shanghai, China, in 2011, and the M.S degree in electrical engineering from the University of Michigan, Ann Arbor, in 2014, where he is currently working toward the Ph.D. degree in electrical engineering.

In the summer of 2012, he had an internship at Qualcomm, CA, where he was involved on developing the antenna system for the next-generation communication network. His research interests include low-power circuit design, modeling and optimization with an emphasis on machine-learning applications.

Pinaki Mazumder (S'84–M'87–SM'95–F'99) received the Ph.D. degree from the University of Illinois at Urbana-Champaign, Urbana, in 1988.

He is currently a Professor with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. He was for six years with industrial R&D centers that included AT&T Bell Laboratories, where in 1985, he started the CONES Project—the first C modeling-based very large scale integration (VLSI) synthesis tool at India's premier electronics company, Bharat Electronics, Ltd., India, where he had developed several high-speed and high-voltage analog integrated circuits intended for consumer electronics products. He is the author or coauthor of more than 200 technical papers and four books on various aspects of VLSI research works. His current research interests include current problems in nanoscale CMOS VLSI design, computer-aided design tools, and circuit designs for emerging technologies including quantum MOS and resonant tunneling devices, semiconductor memory systems, and physical synthesis of VLSI chips.

Dr. Mazumder is a Fellow of the American Association for the Advancement of Science (2008). He was a recipient of the Digital's Incentives for Excellence Award, BF Goodrich National Collegiate Invention Award, and Defense Advanced Research Projects Agency Research Excellence Award.