Training Itself: Mixed-signal Training Acceleration for Memristor-based Neural Network

Boxun Li[1], Yuzhi Wang[1], Yu Wang[1], Yiran Chen[2], Huazhong Yang[1]
[1]Dept. of E.E., TNList, Tsinghua University, Beijing, China
[2]Dept. of E.C.E., University of Pittsburgh, Pittsburgh, USA
[1] Email: yu-wang@mail.tsinghua.edu.cn

***Abstract*—The artificial neural network (ANN) is among the most widely used methods in data processing applications. The memristor-based neural network further demonstrates a power efficient hardware realization of ANN. Training phase is the critical operation of memristor-based neural network. However, the traditional training method for memristor-based neural network is time consuming and energy inefficient. Users have to first work out the parameters of memristors through digital computing systems and then tune the memristor to the corresponding state. In this work, we introduce a mixed-signal training acceleration framework, which realizes the self-training of memristor-based neural network. We first modify the original stochastic gradient descent algorithm by approximating calculations and designing an alternative computing method. We then propose a mixed-signal acceleration architecture for the modified training algorithm by equipping the original memristor-based neural network architecture with the copy crossbar technique, weight update units, sign calculation units and other assistant units. The experiment on the MNIST database demonstrates that the proposed mixed-signal acceleration is 3 orders of magnitude faster and 4 orders of magnitude more energy efficient than the CPU implementation counterpart at the cost of a slight decrease of the recognition accuracy ($< 5\%$).**

***Index Terms*—Neural Network; Training; Memristor;**

I. Introduction

Artificial neural networks have been widely employed in data processing applications, ranging from computer vision, speech recognition, pattern recognition to signal processing [1], [2]. On the top of that, more and more neuromorphic computing architectures have appeared and demonstrated a great potential for performance and energy efficiency gains [3].

In recent years, the innovation of memristor further explores the potential of neuromorphic computing systems. The memristor was first physical realized by HP Labs in 2008 and afterwards attracts significant research interest for its ultrahigh integration density, low power dissipation, and especially, the nonvolatile feature that the state could be tuned by the current passing through itself [4]. The similarity between the memristor and the biologic synapse makes the memristor a promising device to realize the neural network and mixed-signal neuromorphic systems [5]. For example, the memristor-based crossbar can efficiently perform the matrix-vector multiplication, which is the most common operation of ANN [6]. And the memristor-based neural network provides a promising solution to the low power on-chip approximate computing systems with power efficiency of more than 400 GFLOPS/W [7].

Training phase is the critical operation to obtain a neural network for a specific task. However, the process of training is usually time consuming and resource intensive [8]: it usually requires a large volume of memory and computing resources, while the time consumption can range from a few seconds to hundreds of hours, depending on the scale of the networks [9], [10]. Therefore, how to speed up the training process is one of the major concerns of the artificial neural network.

The training problem also exists on the memristor-based neural network: Although the memristor enables an efficient implementation of the operation phase of ANN, the training phase is still confined to the traditional CPU/GPU/FPGA systems because of the requirement of complicated calculation and caching a large amount of data. Users must first work out the parameters of the network through digital computing systems and then tune the memristor to the specific state [7]. However, the training process through digital systems is usually bulky and energy consuming [1], [10] and the tuning process of memristors is complicated, time consuming, hardware intensive and may even bring in unexpected errors [5], [11], [12]. Therefore, there urges an efficient acceleration for the memristor-based neural network.

Our object is to realize the self-training of memristor-based neural network through mixed-signal systems and achieve performance and energy gains. The analog unit of the mixed-signal training acceleration framework should be able to work out the training calculations efficiently and directly configure the memristor **without state tuning**. The digital unit only help control the state of training, instead of performing a large amount of complicate calculations. The following challenges must be overcome to realize this goal: 1) There're too many numerical calculations in the original training algorithm, which are hard for analog systems to realize. A modified algorithm is demanded to relieve the difficulty of analog numerical calculations. 2) Part of the results need to be back propagated when training, and therefore, there requires an efficient solution to the problem of caching analog signals. 3) The conversions between analog and digital in the mixed-signal system should be efficient and as few as possible in order to guarantee the performance and energy gains.

In this paper, we for the first time propose a mixed-signal training acceleration for memristor-based neural network and make it possible for memristor-based neural network to train itself. The contributions of this paper are:

1) We propose a mixed-signal training acceleration framework. The framework consists of a modified training algorithm and a mixed-signal training acceleration architecture, which can realize the self-training of memristor-based neural network. The experiment on the MNIST database demonstrates that our mixed-signal training acceleration framework is able to achieve a 3 orders of magnitude improvement of the training speed as well as a 4 orders of magnitude energy efficiency gains at the cost of a slight decrease of the recognition accuracy ($< 5\%$) compared with the CPU implementation counterpart.
2) We modify the original stochastic gradient descent algorithm by approximating the calculations of errors and decomposing the weight update calculations into sign calculations and numerical calculations. On top of that, we further replace the precious numerical calculations by an automatic adjustment of convergence rate. Such modifications reduce the difficulty of analog realization of the numerical calculations in the training process.
3) We propose *Copy Crossbar* technique and *Sign Calculation* unit to accomplish the analog numerical calculations and overcome the difficulty of caching analog signals. We configure the states of memristors through *Weight Update* units and vari-

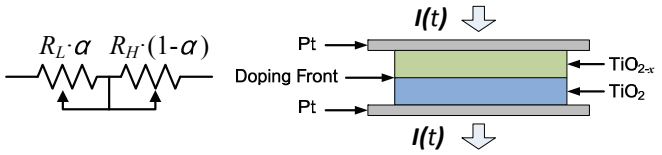

Fig. 1. Physical model of the HP memristor

able gain amplifiers (VGAs) directly without state tuning. In addition, we realize the automatic adjustment of convergence rate by controlling the factor gain of VGAs through digital unit.

The rest of this paper is organized as follows: Section 2 provides related background information. Section 3 introduces our modified training algorithm. The proposed mixed-signal training acceleration architecture is depicted in Section 4. Section 5 presents a case study of the MNIST database and Section 6 concludes this work.

II. Preliminaries

A. Memristor

Fig. 1 shows the physical model of the HP memristor [13]. There is a two-layer thin film of TiO_2: One layer consists of intact TiO_2 and the other layer consists of TiO_{2-x}, which lacks of a small amount of oxygen. The two layers have different electrical conductivity and the overall resistance is the sum of the two layers. There is a boundary (doping front) between the two layers. When a current is applied to the device, the boundary will move and make the resistance of memristor change. The memristor also has other attractive features, such as the 'pinched hysteresis loop'. In this paper, we mainly take advantage of the variable resistance states of the memristor.

B. Implementation of Memristor-based Neural Network

An N-layer network has $N-1$ weight matrixes. The calculation between each pair of neighbour layers can be conceptually expressed as a combination of weighted input variations (matrix-vector multiplication) with a sigmoid active function ($f(x) = \frac{1}{1+e^{-x}}$). The operation of weighted summation in the neural network can be realized through memristor-based matrix-vector multiplication in analog [7] and the sigmoid activation function can be accomplished by the circuit similar to that in [14].

Fig. 2(a) illustrates a memristor crossbar array and Fig. 2(b) shows a complete implementation of the memristor-based matrix-vector multiplication. Each memristor-based matrix-vector multiplication needs two crossbars (positive crossbar and negative crossbar) to represent the positive and negative weights of a network, since M can only be positive [7]. The practical output of the implementation in Fig. 2(b) can be expressed as:

$$V_{oj} = \sum_k w_{kj} \cdot V_{ik} \tag{1}$$

where:

$$w_{kj} = R \cdot (g_{kj(pos)} - g_{kj(neg)}), g_{kj} = \frac{1}{M_{kj}} \tag{2}$$

where V_{ik} and V_{oj} represent the input and output voltage of each row and column, respectively. k and j are the index numbers of input and output voltages. R is the resistor at the end of the array. M_{kj} and g_{kj} are the resistance and admittance of each memristor.

The architecture shown in Fig. 2(b) realizes the weighted summation operation of ANN. By adding a series of the sigmoid circuits to the output ports of the memristor-based matrix-vector multiplication, a double-layer memristor-based neural network is realized. Finally, a multilayer network can be accomplished by combining several double-layer memristor-based neural networks together.

One thing needs to be aware of is that the polarities of each pair of memristors need to be set to the opposite direction in order to

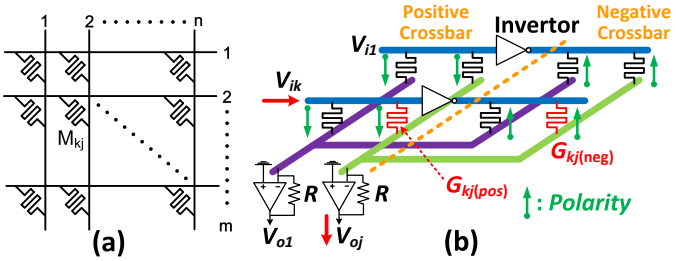

Fig. 2. (a) Memristor Crossbar; (b) Memristor Matrix-Vector Multiplication

make the deviations generated by the currents passing through the two memristors cancel each other, instead of accumulation [7]. This configuration will influence our circuit design in Section IV-C.

C. Stochastic Gradient Descent Algorithm

Stochastic gradient descent is one of the most common algorithms for neural network training [15]. The training process is realized by adjusting the weights in the network layer by layer. The update of each weight (w_{ji}) is:

$$w_{ji} \leftarrow w_{ji} + \eta \cdot \delta_j \cdot x_i \tag{3}$$

where η is the learning rate and δ_j is the error back propagated from the node j in the next neighbour layer. x_i is the input of the node i. For the node p in the output layer, δ_p is:

$$\delta_p = o_p \cdot (1 - o_p) \cdot (t_p - o_p) \tag{4}$$

where o_p and t_p are the actual and ideal output of the complete network, respectively. For the node h in the hidden layers:

$$\delta_h = o_h \cdot (1 - o_h) \cdot \sum_{k \in nextlayer} w_{kh}\delta_k \tag{5}$$

where o_h is the output of the node h in a hidden layer. w_{kh} is the weight between node h in this hidden layer and node k in the next neighbour layer. δ_k is the error of node k in the next neighbour layer.

The training algorithm is an iterative process and hard to be parallelized. For an N-layer network, where the amount of nodes in each layer is $N_1, N_2, ..., N_n$, the complexity of each iteration is $O(\sum_i^{n-1} N_i N_{i+1})$. In addition, it usually requires at least hundreds of thousands iterations to obtain an efficient network. Therefore, the time consumption of training process is usually quite large.

III. Modified Training Algorithm

There're two major modifications compared with the original algorithm: the approximation of the error calculations and accomplish the weight update calculating operations through sign calculations and automatically adjusting the convergence rate.

A. Error Approximation

It can be observed that there're too many operations of multiplication when calculating δ in Eq. (4) and Eq. (5) in the original algorithm. Such a large amount of multiplications is hard for analog circuits to realize.

We note that both $o_p \cdot (1 - o_p)$ and $o_h \cdot (1 - o_h)$ must be greater than zero because the output range of the sigmoid active function is $(0, 1)$. The polarities of δ only depend on $(t_p - o_p)$ in Eq. (4) and $\sum_k w_{kh}\delta_k$ in Eq. (5). Therefore, we decide to ignore $o_p \cdot (1 - o_p)$ and $o_h \cdot (1 - o_h)$ when calculating δ. In fact, we observe such an approximation of δ_p is efficient and has little impact on the training effect.

However, the neglect of $o_h \cdot (1 - o_h)$ may lead to a great error of training. The reason lies in the impact of o_h: The value of o_h is very likely to be close to 0 or 1, especially at the final state of the training process, and makes the value of δ_h close to zeros. Therefore, the complete neglect of $o_h \cdot (1 - o_h)$ will make the value of δ_h too

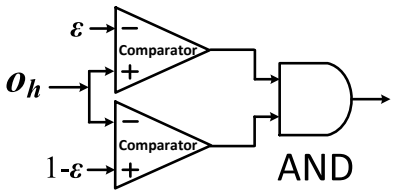

Fig. 3. Implementation of the Filter Operation

large and lead to a terrible training effect. To overcome this problem, we choose to filter o_h instead of approximating $o_h \cdot (1-o_h)$ directly to 1: When $o_h > 1-\epsilon$ or $o_h < \epsilon$, we will set $o_h \cdot (1-o_h)$ to 0, or we set the value of it to 1. Here ϵ is a small value (e.g, $\epsilon = 0.1$) to filter o_h.

In conclusion, the approximation of δ_p in Eq. (4) is:

$$\delta_p = t_p - o_p \tag{6}$$

And for δ_h in Eq. (5):

$$\delta_h = filter(o_h) \cdot \sum_{k \in nextlayer} w_{kh}\delta_k \tag{7}$$

The calculations of δ_p can be realized through an analog subtractor. And the implementation of the filter operation can be accomplished through the unit shown in Fig. 3. The calculation of $\sum_k w_{kh}\delta_k$ will require the copy crossbar technique to be described in Section IV-B.

B. Calculation Decomposition

The error approximation technique just reduces the complexity of the calculations of δ, while the calculations of the complete weight updates require another technique called *Calculation Decomposition*. To be specific, we first decompose the weight update calculations into sign calculations and numerical calculations.

The sign calculations are used to extract the directions of the weight updates. The calculation of the updates of the weights in Eq. (3) can be expressed as:

$$w_{ji} \leftarrow w_{ji} + \eta \cdot |\delta_j| \cdot sign(\delta_j) \cdot x_i \tag{8}$$

The advantage of the extraction of sign calculations is that the polarities of the weight updates can be achieved through zero-crossing detectors and analog comparators easily and quickly as shown in Fig. 4. And the computation between signs can be accomplished through digital logic efficiently. In addition, the digital data are able to get cached conveniently, which greatly reduces the difficulty of caching analog data.

As for the numerical calculations, in order to avoid the usage of the inefficient analog numerical computation, we accomplish the numerical calculation of $\eta \cdot |\delta_j| \cdot x_i$ in Eq . (8) by amplifying x_i signal with a gain factor equal to $\eta \cdot |\delta_j|$. However, the calculation of $\eta \cdot |\delta_j|$ is still hard to complete. Therefore, we choose to estimate the approximate value instead of performing the precise numerical calculations. Unfortunately, it is usually very difficult to make a good estimation of $\eta \cdot |\delta_j|$: A larger approximation will result in the error rebounding and a bad training effect, while a smaller approximation will make the speed of training too slow. In addition, the value of $\eta \cdot |\delta_j|$ also fluctuates frequently.

In order to make a better estimation of $\eta \cdot |\delta_j|$, we use the automatic convergence rate adjustment scheme: We set an initial convergence

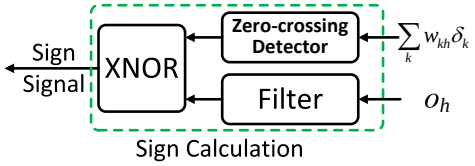

Fig. 4. Sign Calculation Unit

Algorithm 1: Modified Training Algorithm

Input: $Network$, $TrainingDataset$, η_{start}, η_{stop}, $DecayRate$, $MonitorPeriod$, $DetectNodes$
Output: $Network$

Initialize the weights in the $Network$ to small random values;
$\eta \leftarrow \eta_{start}$
$CurrentTrainState \leftarrow 0$
$LastTrainState \leftarrow Inf$
while $\eta > \eta_{stop}$ **do**
  for $times = 0 \rightarrow MonitorPeriod$ **do**
    Pick a sample from the $TrainingDataset$ and compute the $ActualOutput$ of the $Network$;
    $\gamma \leftarrow$ Generate a random value between 0 and 1
    $\delta_p \leftarrow t_p - o_p$
    $\delta_h \leftarrow filter(o_h) \cdot \sum_{k \in nextlayer} w_{kh}\delta_k$
    $w_{ji} \leftarrow w_{ji} + \gamma \cdot \eta \cdot sign(\delta_j) \cdot x_i$
    $CurrentTrainState \leftarrow CurrentTrainState + \sum_{p \in DetectNodes} abs(\delta_p)$
  end
  if $CurrentTrainState > LastTrainState$ **then**
    $\eta \leftarrow \eta / DecayRate$
  end
  $LastTrainState \leftarrow CurrentTrainState$
  $CurrentTrainState \leftarrow 0$
end

rate to represent the initial estimation of weight updates ($\eta \cdot |\delta_j|$). Then the training scheme will record the training states of several output nodes and periodically check the training results. Once the accumulation of the errors of the detected nodes rebound instead of decreasing after a period of monitoring, the convergence rate will decay automatically to guarantee the efficiency of the training process. In addition, if the convergence rate becomes lower than a certain small value, the scheme will assert that the training process is finished. We also multiply the convergence rate with a random value between 0 and 1 to imitate the fluctuation of the value. The complete modified updates of the weights are:

$$w_{ji} \leftarrow w_{ji} + \gamma \cdot ConvergenceRate_j \cdot sign(\delta_j) \cdot x_i \tag{9}$$

where $ConvergenceRate$ is equal to $\eta \cdot |\delta_j|$ and γ is a random value between 0 and 1.

Finally, the automatic adjustment of convergence rate is performed by the digital assistant unit (e.g., FPGA) which requires only a small amount of registers, adders and other digital logic.

C. The Complete Modified Training Algorithm

Algorithm 1 demonstrates the complete process of the proposed modified training algorithm, where $Network$ is the neural network to get trained. $TrainingDataset$ contains the labeled data for training. The content of each labeled training data is $< \vec{x}, \vec{t} >$, where $\vec{x}$ is the input data of each training sample and $\vec{t}$ is the ideal output of the network. η_{start} and η_{stop} are the initial and final convergence rate, respectively. $DecayRate$ represents the decay rate of the convergence rate. $MonitorPeriod$ is the period of monitoring the states of the training to adjust the convergence rate. $DetectNodes$ represents the selected nodes in the output layer to estimate the training states.

Compared with the original training algorithm, only the calculations of the actual network outputs remains the same. Both the calculations of δ_p and δ_h are approximated. The calculation of weight update is replaced by the product of a random value, convergence rate and the result of sign calculation. In addition, there's an automatic adjustment of the convergence rate (η) to replace the precious numerical calculations of $|\delta|$.

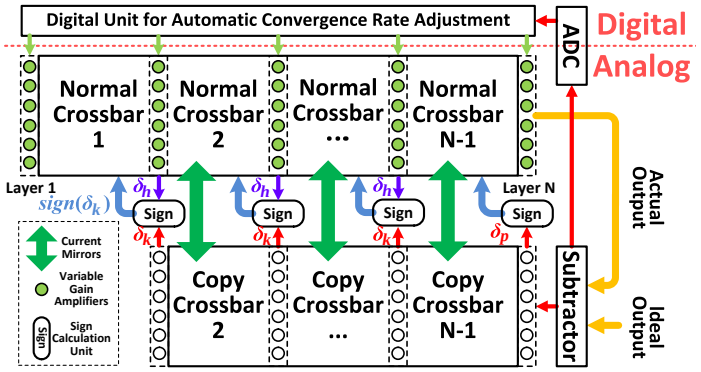

Fig. 5. Mixed-signal Training Acceleration Framework

IV. Mixed-signal Training Acceleration Architecture

Fig. 5 illustrates the overview of the proposed mixed-signal training acceleration framework for memristor-based neural network, which can realize the modified algorithm efficiently. For a training task of an N-layer neural network with $N-1$ weight matrixes, the mixed-signal training acceleration framework requires $N-1$ normal crossbar arrays and $N-2$ copy crossbar arrays. Each pair of normal crossbar and copy crossbar are connected by a series of mirror currents. A array of subtractors is used to work out the deviation (δ_p) between the actual and ideal output and δ_k will be calculated through the copy crossbar arrays. The details of the copy crossbar technique will be introduced later in Section IV-B. All the results of δ will be imported to the sign calculation units (mentioned in Section III-A and III-B). A array of VGAs is added to the input ports of each normal crossbar (mentioned in Section III-B). The outputs of the sign calculation units and VGAs will be imported to each normal crossbar to update the state of the memristor by the weight update unit equipped to each normal crossbar. The details of weight operation unit will be discussed later in Section IV-C. All of the above units are analog. At the same time, part of the results of δ_p will be converted to digital and then imported to the digital unit to control the gain factor of VGAs and perform the automatic adjustment of convergence rate as mentioned in Section III-B.

A. Operating Mechanism

Fig. 6 demonstrates the workflow of the mixed-signal training acceleration framework. There are two stages in each iteration in the training process: the forward computation state and the weight update state.

Before training, all the variables in the digital unit will be first initialized as the modified training algorithm. At the same time, all the crossbar arrays (both normal crossbar and copy crossbar) will be initialized to the same parameters. For example, all of the memristors in the network could be set to the same maximum state and then a series of pulses with the same random value will be set to each crossbar array.

Then the training process will begin. For an N-layer memristor-based neural network, there requires 1 phase for forward computation and $N-1$ phases for weight update in each training iteration:

- *Phase 1:* At the beginning of each iteration, the framework will first work at the forward computation state. The current mirrors are closed. The gain factor of all VGAs is set to 1. The framework will pick a sample $InputData$ from the $TrainingDataset$ and import it into the network. The normal crossbar arrays will work out the actual outputs of the network. At the same time, the ideal outputs of the network will be imported into the framework, too. A series of analog subtractors will compute the deviations (δ_p) of the actual and ideal outputs in parallel and import them directly to the copy crossbar to work out δ_k. The outputs of both the copy crossbar arrays and hidden layers will be imported into the sign calculation units to perform sign calculations. The sign calculation units will also convert the results of sign calculations to digital and cache them efficiently.
- *Phase* 2: After the results of sign calculations are cached, the framework will start to work at the weight update state. All the current mirrors will break over. As mentioned in Section II-C, the weights in the network get adjusted layer by layer, from the first normal crossbar array connected to the input layer (in *Phase 2*) to the last normal crossbar array connected to the output layer (in *Phase N*). In *Phase 2*, the *Normal Crossbar 1* between *Layer 1* and *Layer 2* will get updated. To be specific, the gain factor of the VGAs in *Layer 1* will be set to the corresponding convergence rate, and the VGAs in *Layer 2* will be closed to guarantee that only *Normal Crossbar 1* get updated. Then the weight update operation will be performed on the *Normal Crossbar 1*. The details of the weight update operation of each normal crossbar in each phase will be discussed later in Section IV-C.
- *Phase* $3 \sim N$: After *Phase 2* is completed, *Normal Crossbar 1* will get be updated. Then the rest normal crossbars will get updated through the same operations: For *Phase X* ($3 \leq X \leq N$), the factor of all the VGAs from *Layer 1* to *Layer X-2* will be set back to 1 to generate the current inputs (x_i in Eq. (9)) of *Layer X-1*. The gain factor of the VGAs in *Layer X-1* will be set to the corresponding convergence rate, and the VGAs in *Layer X* will be closed to guarantee that only *Normal Crossbar X-1* get updated. Then the weight update operation will be performed on *Normal Crossbar X-1*. At the same time, the copy crossbar will get the same update with the help of current mirrors.

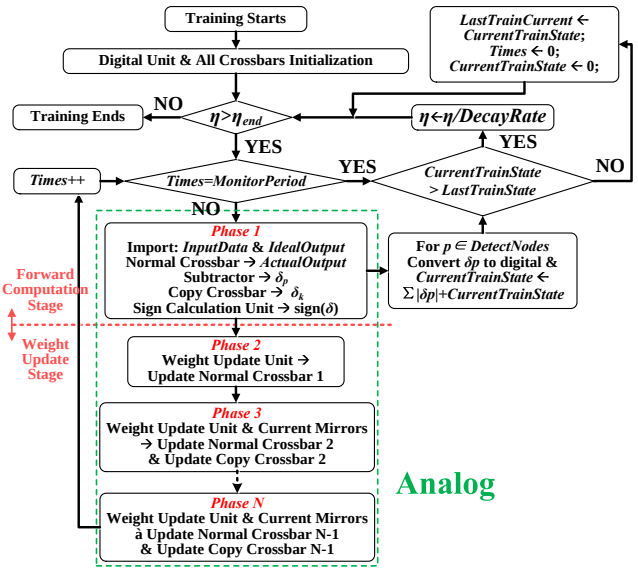

Fig. 6. Mixed-signal Training Flow

In addition, part of the results of deviations between the actual and ideal outputs of the network will be converted to digital and then imported to the digital system to monitor the training state of the network. The digital system will adjust the gain factor of VGAs according to the training state to realize automatic adjustment of the convergence rate.

B. Copy Crossbar Technique

Copy crossbar is a technique to duplicate the weight matrixes, which do not connect with the input layer. Therefore, an N-layer network will have $N-1$ weight matrixes will require $N-2$ copy crossbar arrays.

The copy crossbar is aimed at directly calculating the following part of Eq. (7)

$$\sum_{k \in nextlayer} w_{kh}\delta_k \tag{10}$$

instead of the scheme that first cache δ_k and then import it to the network in the reveres direction (from the output layer to the input layer).

In order to configure the copy crossbar, there will be a current mirror between each memristor in the normal crossbar and the corresponding memristor in the copy crossbar. Before training, all of the crossbar arrays (both normal crossbar and copy crossbar) will be initialized to the same parameters as mentioned in Section IV-A.

In each training iteration, the current mirrors will first be closed and let the copy crossbar complete the calculation of Eq. (10). Afterwards, the weight update operation will be executed to each normal crossbar. The current mirrors will break over and help realize the same update to the memristors in the corresponding copy crossbar by keep the current passing the pair of memristors at the same size.

In general, there will be a difference between each normal crossbar and the corresponding copy crossbar. However, we observe the difference rate is usually very small ($< 10\%$ in our simulations) and our latter experiment results will demonstrate that such a small difference rate only have a small effect on the training results.

C. Weight Update Unit

Fig. 7 demonstrates the weight update unit of our mixed-signal training framework. The unit is modified upon the memristor-based neural network described in Section II-B. The major modification is the addition of the training control unit between the output nodes of the crossbar arrays and the input nodes of sigmoid circuits. The Control Signal is used to switch between the forward computation state and the weight update state.

When Control Signal is ON, the network will work at the forward computation state. All the triodes at the end of the crossbar arrays will break over. And the current mirrors will close.All the crossbar arrays will carry out the multiplication between the weight matrix and the input voltages. And the complete network will generate the actual outputs.

When Control Signal is OFF, the network will work at the weight update state. In phase of the weight update stage, the conducting state of the triodes will depend on the Sign Signal. Sign Signal comes from the results of the sign calculations in Eq. (8). The hardware implementation of the sign calculation has been shown in Fig. 4.

When Sign Signal is ON, the memristor in the positive crossbar will be chosen. The memristor in the negative crossbar is insulated because of the invertor. A voltage pulse will be added to the input nodes of the crossbar array. According to the model in [13], when the current passing through the memristor is in the same direction of the memristor's polarity, the shift of the resistance of the memristor ($-\Delta_M$) caused by a pulse can be expressed as:

$$-\Delta_M = (R_H - R_L) \cdot \mu_v \cdot t_{pulse} \cdot V_{pulse}/h^2 \tag{11}$$

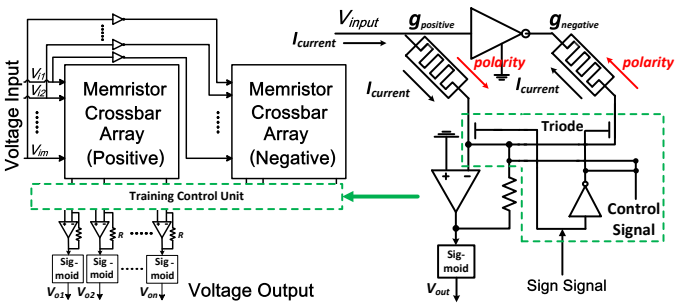

Fig. 7. Weight Update Unit

where R_L and R_H are the lowest and highest resistance of the memristor, respectively. μ_v is the equivalent mobility of dopants. t_{pulse} and V_{pulse} are the duration and value of each pulse. h is the thickness of the memristor.

As mentioned in Section III-B, the voltage pulse comes from the inputs of each layer amplified by VGAs, whose gain factor represents the convergence rate. Therefore, the shift of the resistance ($-\Delta_M$) is proportional to $\gamma \cdot ConvergenceRate_j \cdot x_i$. As the negative memristor is insulated, the total weight represented by this pair of memristors will shift as same as the positive memristor. Combining Eq. (2), the total update of the weight can be expressed as:

$$\begin{aligned}\Delta_w &= R \cdot \left(\frac{1}{M_p - \Delta_{M_p}} - \frac{1}{M_p}\right) = R \cdot \frac{\Delta_{M_p}}{M_p(M_p - \Delta_{M_p})} \\ &\approx R \cdot \frac{\Delta_{M_p}}{M_p^2} \propto \gamma \cdot ConvergenceRate_j \cdot x_i\end{aligned} \tag{12}$$

On the contrary, when Delta Signal is OFF, the positive memristor will be insulated and the state of the negative memristor will shift along with the input pulse because the polarities of memristors are the same as the current direction as mentioned in Section II-B. Therefore, the weight of the network will shift in the opposite direction of the pulse. In conclusion, the shift of each weight in the network will shift according to both Sign Signal and the input pulse, which realize the calculation of Eq. (9).

The weight update unit realizes an approximate configuration of the state of the memristor without memristor state tuning. In addition, we set each update of the weight to a very small value and a complete training task of the network can be realized through the accumulation of such small and approximate updates.

V. A Case Study: MNIST

In order to test the effects of the proposed mixed-signal training framework, we use the MNIST database as a case study. MNIST is a widely used database with more than 60,000 handwritten digits for optical character recognition [16], [17].

A. Experimental Setup

We choose 20,000 examples of handwritten digits of '0'∼'9' as the training set and 5,000 other examples for testing. The network used for pattern recognition is an 3-layer network with 784 input nodes (as the pixels of the input image is 28×28) with 300 hidden nodes. There're 10 output nodes in the network. The range of each input and output signal is $(0, 1)$. The amplitude of the output signal represents the similarity of the corresponding digit. We recognize the test image as the digit represented by the node with the biggest output.

The CPU implementation of the MNIST test is based on the Intel Math Kernel Library [18] and an Intel CPU core (i5-2320 @3.0GHz). The simulation of the mixed-signal training acceleration architecture is based on FPGA, MATLAB and SPICE, where the FPGA is used to implement and simulate the digital unit, the MATLAB is used to simulate the training process and the SPICE is used to simulate the power consumption.

The working frequency of the analog unit is 20MHz and each iteration of training costs 3 cycles. The lowest and highest resistance of the memristor (R_L and R_H) are set to 100Ω and $16k\Omega$, respectively. The equivalent mobility of dopants μ_v is set to $10^{-14}m^2 \cdot s^{-1} \cdot V^{-1}$ and the thickness (h) is set to $10nm$. The range of the input voltage V_{in} is $(0, 1)V$. As the analog unit works under 20MHz, t_{pulse} is equal to $50ns$. Taking the above value into Eq. (11), the shift of the resistance of the memristor (Δ_M) is equal to $0.0795V_{pulse}$. The ϵ in the Filter unit is set to 0.1. The $MonitorPeriod$ is set to 1,000. All the 10 output nodes ($DetectNodes$) are monitored through a 200MHz ADC. The resistance at the end the crossbar arrays (R) is

TABLE I
EXPERIMENT RESULTS OF THE MEMRISTOR IMPLEMENTATION OF TRAINING SCHEME

DecayRate	Noise Rate	Accuracy (%)	Iteration	Time (ms)	Energy (mJ)	Speed Up	Energy Saving	Accuracy Drop (%)
CPU		96.16	517,000	71067	6396030	-	-	-
1.2	0	94.84	140,000	21.0	146.89	3384	43544	1.37
	0.01	94.36	130,000	19.5	129.46	3644	49407	1.87
	0.05	94.40	134,000	20.1	132.97	3536	48102	1.83
	0.1	94.14	137,000	20.55	143.38	3458	44608	2.10
1.5	0	94.30	76,000	11.4	75.56	6234	84647	1.93
	0.01	93.96	77,000	11.55	81.43	6153	78543	2.29
	0.05	94.76	75,000	11.25	77.37	6317	82665	2.50
	0.1	93.34	66,000	9.9	67.31	7178	95021	2.93
1.8	0	93.02	50,000	7.5	49.53	9476	129128	3.27
	0.01	92.60	47,000	7.05	47.52	10080	134591	3.70
	0.05	92.70	46,000	6.9	47.04	10300	135984	3.60
	0.1	92.62	45,000	6.75	48.25	10528	132572	3.68

set to $1k\Omega$. The start and stop factor gains of the VGAs are set to 100 and 0.5.

B. Experiment Results

The simulation results of the power consumption of the analog unit is $\sim$1570.35mW. And the power of CPU and FPGA are $\sim$90W and $\sim$5W. The detailed time and energy consumption of the memristor implementation with $DecayRate$ (in Algorithm 1) and noise rate is given in Table I. The noise in the experiment consists of both the noise of the analog signals and device deviations, such as the deviations between the copy crossbar and corresponding normal crossbar. The noise rate represents the maximum noise ratio added to the system. The accuracy represents the correct rate of the recognition of 5,000 test examples. Iteration stands for the times of iterations when the training is finished. The accuracy drop means the relative rate of the decrease of the recognition accuracy.

It can be seen that the effect of training depends both on the $DecayRate$ and the noise rate. A lower $DecayRate$ with a smaller noise rate will help achieve a better training result but the time and energy consumptions will become greater. Therefore, users must balance between the cost and the effect of the training.

Finally, the experiment results demonstrate that the mixed-signal acceleration framework is able to achieves a 3 orders of magnitude improvement of the training speed as well as a 4 orders of magnitude energy efficiency gains. It can be also observed that there's a slight decrease of the accuracy of recognition ($< 5\%$). However, such a cost is deserved compared with the huge performance and energy gains.

VI. SUMMARY AND CONCLUSIONS

In this work, we propose a mixed-signal training acceleration framework for memristor-based neural network. We first introduce a modified training algorithm, which enables the feasibility of the mixed-signal realization of the modified algorithm. We then propose a mixed-signal acceleration architecture for the modified algorithm, which can accomplish the training task of memristor-based neural network efficiently. Finally, we use the MNIST database as a case study to test the performance of our mixed-signal training acceleration framework for memristor-based neural network. The experiment results show that our training acceleration framework is able to realize a 3 orders of magnitude speed-up as well as a 4 orders of magnitude energy efficiency gains compared with the CPU implementation counterpart.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (No. 61373026, No. 61261160501, No. 61028006), 973 project 2013CB329000, National Science and Technology Major Project (2013ZX03003013-003), youth talent development plan of Beijing (YETP0099) and NSF CAREER CNS-1253424.

REFERENCES

[1] Jeffrey Dean et al. Large scale distributed deep networks. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1232–1240, 2012.

[2] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*.

[3] Russell Beale and Tom Jackson. *Neural Computing-an introduction*. CRC Press, 2010.

[4] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[5] Beiye Liu, Miao Hu, Hai Li, Zhi-Hong Mao, Yiran Chen, Tingwen Huang, and Wei Zhang. Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine. In *Proceedings of the 50th Annual Design Automation Conference*, 2013.

[6] Miao Hu, Hai Li, Qing Wu, and Garrett S Rose. Hardware realization of bsb recall function using memristor crossbar arrays. In *Proceedings of the 49th Annual Design Automation Conference*, 2012.

[7] Li Boxun, Shan Yi, Hu Miao, Wang Yu, Chen Yiran, and Yang Huazhong. Memristor-based approximated computation. In *Proceedings of ISLPED 2013*.

[8] Rafael Menéndez de Llano and José Luis Bosque. Study of neural net training methods in parallel and distributed architectures. *Future Generation Computer Systems*, 26(2):267–275, 2010.

[9] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Prentice Hall New York, 2009.

[10] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Andrew Ng, and Quoc V Le. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.

[11] Sieu D Ha and Shriram Ramanathan. Adaptive oxide electronics: A review. *Journal of Applied Physics*, 110(7):071101–071101, 2011.

[12] Wei Yi, Frederick Perner, et al. Feedback write scheme for memristive switching devices. *Applied Physics A*, 102:973–982, 2011.

[13] Robinson E Pino, Hai Li, Yiran Chen, Miao Hu, and Beiye Liu. Statistical memristor modeling and case study in neuromorphic computing. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 585–590. IEEE, 2012.

[14] G. Khodabandehloo, M. Mirhassani, and M. Ahmadi. Analog implementation of a novel resistive-type sigmoidal neuron. *TVLSI*, 20(4):750 –754, april 2012.

[15] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

[16] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 1998.

[17] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[18] Intel. Intel math kernel library. http://software.intel.com/en-us/intel-mkl/.