Aging-aware Lifetime Enhancement for Memristor-based Neuromorphic Computing

Shuhang Zhang[1,2], Grace Li Zhang[1], Bing Li[1], Hai (Helen) Li[2,3], Ulf Schlichtmann[1]
[1]Chair of Electronic Design Automation, [2]Institute for Advanced Study, Technical University of Munich (TUM), Munich, Germany
[3]Department of Electrical and Computer Engineering, Duke University, Durham, NC, United States
Email: {shuhang.zhang, grace-li.zhang, b.li, ulf.schlichtmann}@tum.de, hai.li@duke.edu

***Abstract*—Memristor-based crossbars have been applied successfully to accelerate vector-matrix computations in deep neural networks. During the training process of neural networks, the conductances of the memristors in the crossbars must be updated repetitively. However, memristors can only be programmed reliably for a given number of times. Afterwards, the working ranges of the memristors deviate from the fresh state. As a result, the weights of the corresponding neural networks cannot be implemented correctly and the classification accuracy drops significantly. This phenomenon is called aging, and it limits the lifetime of memristor-based crossbars. In this paper, we propose a co-optimization framework combining software training and hardware mapping to reduce the aging effect. Experimental results demonstrate that the proposed framework can extend the lifetime of such crossbars up to 11 times, while the expected accuracy of classification is maintained.**

I. Introduction

Neural networks have been applied in various fields and achieved remarkable performances. As the structure of neural networks is becoming more complex, a huge number of vector-matrix multiplications are required. However, traditional CMOS-based architectures are inefficient in implementing these computations. Although general-purpose and specific hardware platforms, e.g., GPU [1] and FPGA [2], have been adopted to implement neural networks, power consumption is still a limiting factor for such platforms [3]. To meet the increasing computing demand in complex neural networks, memristor-based crossbar has been introduced as a new circuit architecture to implement vector-matrix multiplications. By taking advantage of its analog nature, this architecture can provide a remarkable computing efficiency with a significantly lower power consumption.

One of the major properties of memristors is that their conductances can be programmed. With this feature, computing-intensive vector-matrix multiplication operations can be implemented efficiently, with the crossbar architecture shown in Fig. 1 [4]. The voltage vector $\mathbf{V}^I = [V_1^I, V_2^I, ...V_m^I]$ applied to the memristors in the jth column generates a current $I_j^O = \sum_{i=1}^{m} V_i^I g_{ij}$, where g_{ij} is the conductance of the memristor in the ith row and jth column. Furthermore, the current at the bottom of the jth column is converted to voltage. Considered as a whole, the relation between the input voltage vector $\mathbf{V}^I$ and the output voltage vector $\mathbf{V}^O$ can be expressed as $\mathbf{V}^O = \mathbf{V}^I \cdot \mathbf{G} \cdot \mathbf{R}$, where $\mathbf{G}$ is the conductance matrix and $\mathbf{R}$ is the diagonal resistance matrix. Since the conductances $\mathbf{G}$ of the memristors can be programmed, this crossbar can thus be used to accelerate the vector-matrix computations in neural networks.

Memristor crossbars have successfully been implemented in hardware to accelerate neural networks [5]. The methods to integrate them into computing systems can be categorized into two groups. The first approach is online training [6], [7]. In this approach, the conductances of the memristors are firstly programmed and then tuned in further iterations according to the accuracy of classification. The second approach of using memristor-based crossbars combines software training and online tuning. At first, the training process is performed at software level and the trained weights are mapped to the conductances of memristors. Thereafter, the conductances are further fine tuned to improve the classification accuracy. This method can achieve an expected accuracy more rapidly because the initial mapped conductances are already close to their target values.

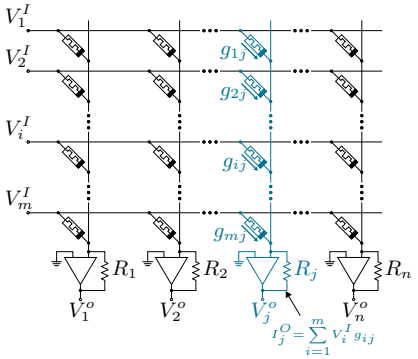

Fig. 1. Memristor crossbar architecture.

No matter which approach above is used to implement neural networks with memristor-based crossbars, repetitive tuning is involved. During tuning the conductance of a memristor, also called programming, a pulse of relatively high voltage is applied to the memristor. This high voltage may cause change of filament inside the memristor, leading to a degradation of the valid range of its conductance and thus the number of usable conductance levels, an effect called aging. If a trained weight is mapped by assuming the fresh state of the memristor after aging, the target conductance might fall outside of the valid range of the memristor and the programmed conductance can thus deviate significantly from the target conductance. Consequently, the memristor needs to be programmed more times in the following tuning process, leading to even more degradation of the conductance range.

The aging problem of memristors is different from the drifting effect as described in [8], where the conductance of the memristor drifts away from its programmed value after being read repetitively during execution of classification. This drifting can be recovered by reprogramming the conductance,

978-3-9819263-2-3/DATE19/©2019 EDAA

while the aging effect described above is irreversible [9], [10].

To reduce the aging effect, several methods have been proposed. In [9], programming voltages in triangular and sinusoidal forms are used. Consequently, the applied voltage may cause less aging because the average of the applied voltage is lower than the constant DC voltage. In [11], a resistor is connected to a memristor in series to suppress irregular voltage drop on the memristor. Furthermore, the swapping method in [12] uses rows of memristors that are slightly aged to replace the rows that are heavily aged to extend the lifetime of the whole crossbar.

The previous methods above, however, deal with the aging effect with a gross granularity. These techniques incur either extra cost or a higher complexity of the system. In this paper, we propose a new framework to counter aging in software training and hardware mapping without extra hardware cost. The contributions of this paper include:

- The mechanism of aging of memristors is analyzed and a corresponding skewed–weight training is proposed. By pushing the conductances of memristors towards small values, the current flowing through memristors can be reduced to alleviate the aging effect.
- When mapping the weights acquired from the training process to memristors, the current aging status of the memristors is taken into account. This status is traced using representative memristors and the conductances of memristors are adjusted further with respect to the aging status of these memristors. Consequently, the number of tuning iterations after mapping can be reduced, leading to less aging and thus a longer lifetime.
- The proposed software training and hardware mapping are integrated into an aging-aware framework and can improve the lifetime of memristor-based crossbars up to $11\times$ compared with those without considering aging, while no extra hardware cost is incurred.

The rest of this paper is organized as follows. In Section II, we explain the background of memristor-based neuromorphic computing. The motivation of the proposed method is presented in Section III. In Section IV, we describe the ideas to counter aging in software training and hardware mapping. Experimental results are shown in Section V and conclusions are drawn in Section VI.

II. Background of Memristor-based Neuromorphic Computing

To implement neural networks using memristor-based crossbars, several steps are involved: software training, hardware mapping, online tuning. Afterwards, the crossbars can be used to perform tasks such as image classification and speech recognition.

A. Neural Networks and Software Training

The basic structure of neural networks is shown Fig. 2, which consists of an input layer, a hidden layer, and an output layer. The nodes represent neurons and connections represent the relations between neurons in different layers. In a neural network, the output of a neuron can be expressed as a function of previous neurons, e.g., $z_1 = A(w_{11}\cdot x_1 + w_{21}\cdot x_2 + w_{31}\cdot x_3)$, where w_{11}, w_{21} and w_{31} are the weights of the connections from x_1, x_2, x_3 to z_1, respectively. $A(\cdot)$ is the activation function such as ReLu, Tanh, and sigmoid functions, which are used in each layer to introduce non-linearity in the neural network.

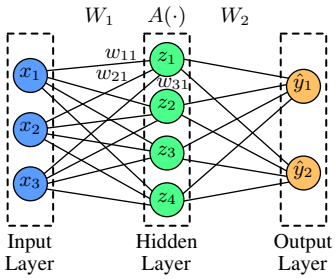

Fig. 2. Basic structure of neural networks.

Before a neural network can be used to process applications, its weights should be determined. For this purpose, training data are fed into the neural network. The output of the network is compared with the expected output in the training data and the difference between them is defined as the cost, which is used as the feedback to the neural network to adjust the weights. In practice, $L2$ regularization is also often appended to prevent overfitting. Consequently, the cost function can be expressed as follows

$$Cost = \sum_{i=1}^{N_{output}} (-y_{(i)} log(\hat{y}_{(i)}) - (1 - y_{(i)}) log(1 - \hat{y}_{(i)})) + \sum_{i=1}^{N_{layers}} \lambda \cdot \|\mathbf{W}_i\|^2 \quad (1)$$

$$= C(\mathbf{W}) + R(\mathbf{W}) \quad (2)$$

where $\hat{y}_{(i)}$ is the ith output of the neural network, $y_{(i)}$ is the ith expected output from the training data, $\mathbf{W}_i$ represents the weights in the ith layer, and λ is the penalty used in $L2$ regularization. The first term of the cost function (1) denotes the cross entropy. The second term in (1) implements the $L2$ regularization.

According to the result of the cost function, a back propagation algorithm is then applied to update the weights in the neural network to reduce the cost for a higher classification accuracy. For example, the weights can be updated as

$$\mathbf{W}_i = \mathbf{W}_i - \mathcal{LR} \cdot \frac{\partial Cost}{\partial \mathbf{W}_i} \quad (3)$$

where $\mathcal{LR}$ denotes the learning rate.

B. Hardware Mapping

After training, the neural network needs to be implemented. As shown in Fig. 2, the input of a neuron is determined by the linear combination of those neurons in the previous layer. When all the neurons in a layer are considered together, a vector-matrix multiplication needs to be performed for each layer. To accelerate this computation, the memristor-based crossbar shown in Fig. 1 can be used as described in Section I.

In a memristor-based crossbar, the weights of the matrix are implemented in the programmable conductances of the memristors. Assume that the maximum and the minimum weights are w_{max} and w_{min}, respectively. Also assume that the maximum and the minimum conductances of memristors are g_{max} and g_{min}, respectively. The mapping of the weight w_{ij} from the ith neuron in a layer to the jth neuron in the next

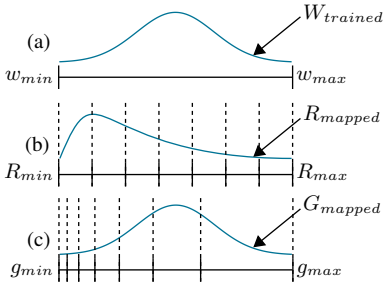

Fig. 3. Hardware mapping and quantization. (a) Weights after software training. (b) Resistance distribution after mapping with quantization. (c) Conductance distribution after mapping with quantization.

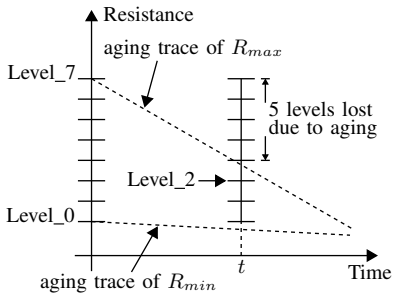

Fig. 4. Aging scenario of a memristor with the upper and lower bounds of the resistance range decreasing with programming time.

layer onto the conductance of the corresponding memristor can be expressed as

$$g_{ij} = (\frac{g_{max} - g_{min}}{w_{max} - w_{min}} \cdot (w_{ij} - w_{min}) + g_{min}). \quad (4)$$

In (4), a common conductance range $g_{max} - g_{min}$ is used during the mapping process because the currents should be summed linearly in each column.

In practice, it is difficult to program the conductances of memristors to exact values and thus the resistances are usually programmed instead [13], [14]. After software training, the weights usually satisfy a quasi-normal distribution as shown in Fig. 3(a). However, due to the inverse relationship between conductance and resistance, the distribution of resistances of memristors has a skewed shape, as shown in Fig. 3(b). To simplify the programming circuitry, the ranges of the resistances are further discretized into a given number of levels, shown as the dashed lines in Fig. 3(b). For example, 32 levels are used in [14] and 64 levels in [15]. During mapping, the target resistance of a memristor is then approximated by the closest level in the discrete range. This process is called quantization. The quantized levels in the resistance range also lead to quantized levels in the conductance range, which are, however, not evenly distributed anymore due to the inverse relation between resistance and conductance, shown as the dashed lines in Fig. 3(c).

C. Online Tuning

Due to quantization, the accuracy of classification directly using a neural network after weight mapping is lower than that of software training. To improve this accuracy, the conductances of the memristors are tuned further according to the results of applying training data, a process similar to software training. However, in hardware level, it is difficult to calculate the derivatives accurately to update the conductances using (3). Therefore, a simplified tuning calculation is applied by considering only the signs of the derivatives. Based on these signs, the polarities of constant amplitudes used to program the conductances can be determined [16], as

$$\mathbf{V}_i \propto sign(-\frac{\partial Cost}{\partial \mathbf{W}_i}) \quad (5)$$

where $\mathbf{V}_i$ represents the voltage levels of the tuning pulses for the memristors in the ith layer. This online tuning process is executed repetitively in the neural network until the target classification accuracy is achieved.

III. Aging of Memristor Crossbars and Motivation of Proposed Counter-aging Methods

As described in section II, the memristor-based computing framework requires a large number of programming iterations during online tuning. To program a memristor, a high voltage needs to be applied. This voltage generates a current flowing through the memristor and causes the filament inside it to change irrecoverably due to repetitive electrical charging/discharging or changes in temperature. Consequently, the valid resistance range of the memristor degrades, an effect called aging in the memristor, as observed in [9], [10], [17], [18].

Assume that the upper bound and the lower bound of the resistance of a memristor are written as R_{max} and R_{min}, respectively. Generally, the new bounds of the resistance range of a memristor after aging can be expressed as

$$R_{aged,max} = R_{fresh,max} - f(T,t) \quad (6)$$

$$R_{aged,min} = R_{fresh,min} - g(T,t) \quad (7)$$

where T represents temperature and t the accumulated time in which the memristor is programmed. $f(\cdot)$ and $g(\cdot)$ are aging functions. Both aging functions are Arrhenius-based [17], [18] and the parameters in the functions can be extracted from measurement data. Among various aging failure types in memristors with different materials and parameters, a common aging scenario is illustrated in Fig. 4, where both upper and lower bounds of the resistance range decrease as the accumulated programming time t increases.

After a memristor is programmed many times, the target resistances may fall outside of its aged resistance range. For example, a programming attempt to set the resistance of a memristor to Level_7 at time t in Fig. 4 can only end up with the resistance set to Level_2. This mismatch of weight mapping and target resistance necessitates more programming iterations during online tuning to improve the classification accuracy, which unfortunately leads to an even larger aging effect due to more programming iterations. To solve these problems, counter-aging methods are proposed in the following section.

IV. Proposed Counter-aging Framework

To counter the aging effect in programming memristor-based crossbars, we propose to combine software training and hardware mapping to adjust the weights in the neural network and thus the conductances of memristors in the crossbars. The former technique pushes weights from training towards small values to reduce the currents flowing through

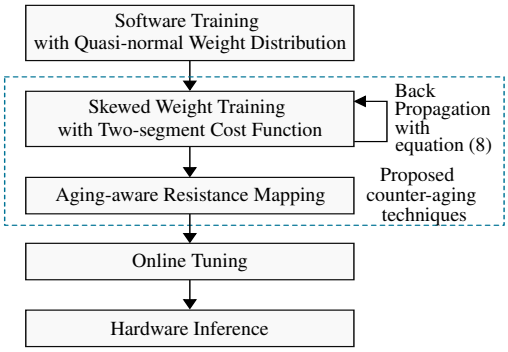

Fig. 5. Proposed counter-aging work flow.

memristors, while the latter maps the weights to quantized levels considering the aging status of the crossbars to reduce online tuning iterations further. The proposed work flow is illustrated in Fig. 5.

A. Skewed Weights in Software Training

As described previously, memristors experience aging when being programmed due to the generated currents flowing through them. To reduce these currents, we propose to map the resistances of the memristors to large values, as shown in Fig. 6. This can be achieved by concentrating the weights to small values during software training. After being mapped to the crossbar, these small weights lead to small conductances and thus large resistances programmed into the memristors to reduce the currents.

Another benefit of the skewed weight distribution is that the conductances can be approximated more accurately during quantization. Since the levels of resistances are spread evenly in the resistance range, the corresponding discrete levels of conductances are concentrated to the region of small values due to the inverse relation between resistance and conductance, as shown in Fig. 3(c). With the original quasi-normal weight distribution, only a small portion of conductances are located in the lower end of the conductance range, although there are more levels available in this region to approximate them. On the contrary, if the proposed weight concentration is applied, more weights are pushed towards small values, as shown in Fig. 6(a), where the denser levels on the left side of the range keep more information of the weights after quantization. Consequently, the accuracy after weight mapping onto memristor crossbars tends to be higher than that achieved by original quasi-normal weight distribution, leading to fewer iterations during online tuning.

The technique of weight concentration to small values is feasible in software training, because neural networks have much flexibility to achieve the required classification accuracy with different sets of weights. To implement the desired skewed distribution, the cost function during training is modified to incorporate the concentration of weights into back propagation.

Comparing the expected weight distribution in Fig. 6(a) and the original weight distribution in Fig. 3(a), we can observe that we need to select a reference weight δ_i in the original range of the weights. On the left side of this point, weights should be penalized strongly so that they tend not to fall into this range. On the right side of this point, weights should

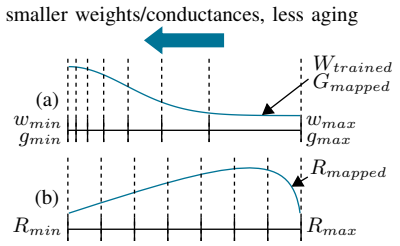

Fig. 6. Skewed weight mapping and quantization. (a) Wights are pushed towards small values, resulting a skewed distribution in contrast to Fig. 3b. (b) Resistance distribution corresponding to the skewed weight distribution.

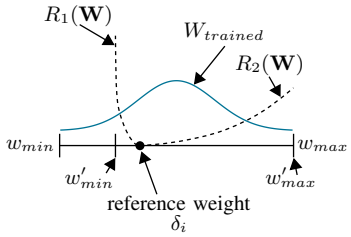

Fig. 7. Regularizations in skewed weight training. $\mathbf{W}_{trained}$ is the distribution of the original weights. $R_1(\mathbf{W})$ and $R_2(\mathbf{W})$ are the new terms for the cost function to generate skewed weights. With these regularizations, a skewed weight distribution as in Fig. 6(a) can be generated.

also be concentrated in a way that the farther they are from this reference weight, the stronger they are penalized. The concept of this cost function for skewed weight distribution is illustrated in Fig. 7, where the solid curve is the quasi-normal distribution of the originally trained weights, and the two dashed lines are the cost functions on the left side and on the right side of the selected reference weight.

To implement the two-segment concentration of the weights in software training, we expand the L2 regularization term $R(\mathbf{W})$ in (2) into two terms $R_1(\mathbf{W})$ and $R_2(\mathbf{W})$, shown as follows

$$Cost = C(\mathbf{W}) + R_1(\mathbf{W}) + R_2(\mathbf{W}) \tag{8}$$

$$R_1(\mathbf{W}) = \sum_{i=1}^{N_{layers}} \lambda_1 \cdot \|\mathbf{W}_i - \delta_i\|^2, \mathbf{W}_i < \delta_i \tag{9}$$

$$R_2(\mathbf{W}) = \sum_{i=1}^{N_{layers}} \lambda_2 \cdot \|\mathbf{W}_i - \delta_i\|^2, \mathbf{W}_i \geq \delta_i \tag{10}$$

where δ_i is the reference weight around which the skewed weights are concentrated. λ_1 sets the penalty of the first regularization for weights falling on the left side of δ_i and λ_2 sets the penalty of the second regularization for the weights on the right of δ_i. Since the weights on the left of the δ_i should be penalized heavily, λ_1 is larger than λ_2. $\mathbf{W}_i$ are the weights in the ith layer in the neural network, and are updated in iterations during training to meet the specified classification accuracy.

The two terms $R_1(\mathbf{W})$ and $R_2(\mathbf{W})$ in the cost function impose more penalty when a weight is far from the reference weight δ_i, although the accuracy of the classification is still improved gradually as in the normal training process. When training is finished, a quasi-normal weight distribution in Fig. 7 is transformed into a skewed weight distribution as in Fig. 6(a), which also indicates the new weight range $[w'_{min}, w'_{max}]$ by its largest and smallest values, respectively. The shape of the

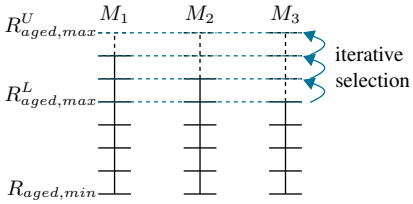

Fig. 8. Range selection in hardware mapping.

skewed weight distribution is highly affected by the constants δ_i, λ_1 and λ_2. The setting of these parameters is provided in section V.

B. Aging-aware Mapping

Aging of the memristors is a dynamic process and depends on the history of programming. Without considering aging, the weights after software training are usually mapped to the resistance ranges of memristors in the fresh state. With aging, the upper bounds and the lower bounds of their resistance ranges and the numbers of quantized levels in the ranges may change. For example, the number of resistance levels in Fig. 4 decreases from eight in the fresh state to three after being programmed for the accumulated time t. If the weight is supposed to be mapped to Level_7 after aging, the memristor can only be programmed to Level_2. After programming, if this aged memristor is used directly for classification, the accuracy may degrade significantly. If otherwise online tuning is applied, many programming iterations need to be applied because many weights need to be re-trained to achieve the expected accuracy. This re-training, unfortunately, requires more programming iterations and increases aging further.

To alleviate the mapping problem after aging, we take the aged resistance ranges of memristors into account during hardware mapping. In the proposed method, we trace the programming iterations of every one out nine memristors, namely, the memristor at the center of every 3×3 block, and estimate the aged upper and lower bounds of the resistance ranges using (6) and (7). Afterwards, the mapping of the weights onto memristors can be performed using (4). Since the memristors in a crossbar experience different programming iterations, their aged resistance ranges also differ. However, the currents flowing through the memristors in a column in the crossbar should be summed linearly, so that the resistance ranges of these memristors should be the same. In the proposed method, this common range is determined by an iterative selection process.

Assume that the aged resistance ranges of three memristors $M1$, $M2$ and $M3$ are shown as in Fig. 8. After aging, the original lower bounds are usually still located in the aged range as shown in Fig. 4. The upper bounds of resistance ranges, however, have degraded by one, two and three levels, respectively. To determine which common resistance range should be used for hardware mapping, we iterate through all the aged upper bounds of the traced memristors between $R^L_{aged,max}$ and $R^U_{aged,max}$ as shown in Fig. 8 and map the weights. Thereafter, the upper bound that produces the highest classification accuracy is selected as the new common resistance range. The selected range may still not cover the aged ranges of all the memristors. For example, if the range of M_2 is selected, a weight mapped to the upper bound of the common resistance range onto M_3 may still fall outside of the common range. This inaccuracy is compensated by online tuning following hardware mapping, whose iterations are still reduced owing to the estimated common range to achieve the required classification accuracy.

TABLE I
ACCURACY AND LIFETIME COMPARISON BETWEEN TRAINING METHODS

		Accuracy Comparison		Lifetime Comparison		
NN	Dataset	w/o Skewed	Skewed	T+T	ST+T	ST+AT
LeNet-5	Cifar10	75.44%	73.30%	1×	6×	8×
VGG-16	Cifar100	71.50%	71.76%	1×	7×	11×

TABLE II
PARAMETERS IN SKEWED TRAINING

	δ_i	λ_1	λ_2
LeNet-5	$-\sigma_i$	1	0.001
VGG-16	$-0.75\sigma_i$	0.005	0.005

V. EXPERIMENTAL RESULTS

To evaluate the effectiveness of the proposed counter-aging framework, two different neural networks, LeNet-5 [19] and VGG-16 [20] were applied onto two different image datasets, Cifar10 and Cifar100 [21], respectively. The information of the neural networks and the datasets are shown in the first two columns of Table I. LeNet-5 has 5 layers, with 2 convolutional layers and 3 fully-connected layers. VGG-16 has 13 convolutional layers and 3 fully-connected layers. Cifar10 consists of 60000 images, which are grouped into 10 different classes. Cifar100 also has 60000 images, which are grouped into 100 classes instead. The neural networks and the proposed algorithm were implemented using Tensorflow [22] and tested with an Intel 3.6 GHz CPU and an NVIDA GeForce GTX 1080Ti graphics card.

For skewed weight training, the constants of the reference weight δ_i shown in Fig. 7 and the penalty values λ_1 and λ_2 in (9) and (10) are shown in Table II. By setting various combinations during software training, the parameters are selected to maintain both the classification accuracy and the expected skewed weight distribution. In Fig. 7 the mean value of the quasi-normal distribution is close to zero so that the reference weights were set to the standard deviation σ_i multiplied by a constant value. λ_1 was much larger than λ_2 to implement the skewed penalty in LeNet-5. However, VGG-16 has a large number of layers, and the accuracy tends to be more sensitive to the parameters, so that λ_1 was set to the same value as λ_2 to prevent accuracy degradation during software training. With this setting, a proper skewed weight distribution can still be achieved. Fig. 9 shows the skewed weight distribution of the third layer of VGG-16 as an example. The weight distributions of other layers have similar tendencies with the setting of λ_1 and λ_2 above .

In Table I the two columns on accuracy comparison show the classification accuracy without and with skewed weight software training. The classification accuracy is slightly lower with LeNet-5, while VGG-16 has a higher accuracy even with the skewed weight distribution. This comparison demonstrates that neural networks actually have flexibility in weight selection to produce results of the same quality. The proposed framework essentially takes advantage of this feature to optimize the lifetime of the crossbars.

As shown in Fig. 9, most weights of VGG-16 are con-

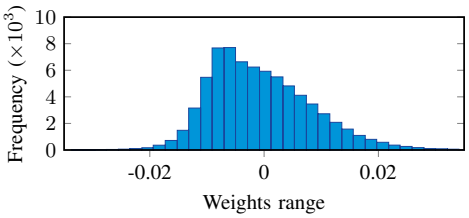

Fig. 9. Skewed weight distribution of the third layer of VGG-16.

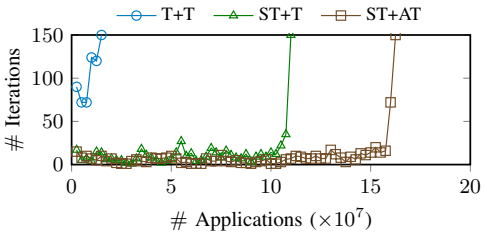

Fig. 10. Comparison of online tuning for VGG-16.

centrated towards small values in software training. After these weights are determined, iterations of online tuning are applied to achieve the specified classification accuracy. If the number of online tuning iterations becomes too large, the tuning cannot converge and thus the crossbars fail. For demonstration, the trends of iterations of online tuning of and VGG-16 with respect to the number of applications are shown in Fig. 10. As the number of applications reaches certain thresholds, the numbers of online tuning iterations increase suddenly, indicating that the memristor crossbars are failing. From this comparison, it can be observed that the proposed ST+AT framework can significantly improve the number of applications that can be processed successfully.

The last three columns of Table I show the lifetime comparison of the test cases. For this comparison, we simulated the online tuning process for a certain number of applications and demonstrate the lifetime results with three scenarios including, traditional weight training and online tuning (T+T), skewed weight training and online tuning (ST+T), skewed weight training with aging-aware mapping and online tuning (ST+AT) We set the target accuracy of the two neural networks to ensure a given number of applications, 4×10^7 in the experiments, can be implemented with fewer than 150 tuning iterations. The proportional relation of the numbers of applications being processed successfully in the cases T+T, ST+T and ST+AT are shown in the last three columns of Table II. With skewed weight distribution, these numbers and thus the lifetime of LeNet-5 and VGG-16 have been improved by $6\times$ and $7\times$. Together with aging-aware mapping, the lifetime can be improved by $8\times$ and $11\times$.

In reality, neural networks may contain layers of different types such as convolutional layers and fully-connected layers. To demonstrate the aging effects in these layers, the average upper bounds of resistances of all memristors in the convolutional layers and fully connected layers of LeNet-5 and VGG-16 are shown in Fig. 11. According to this comparison, the aging effect in convolutional layers is stronger than fully-connected layers, because convolutional layers are used to extract the features of input data and are thus programmed more often. Therefore, such layers have a higher priority to apply counter-aging measures.

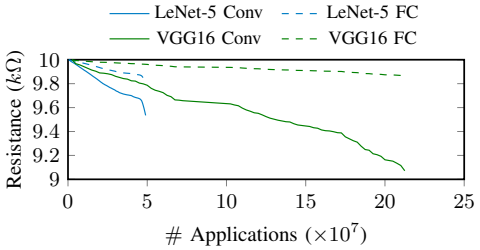

Fig. 11. Aging of convolutional layers and fully-connected layers.

VI. Conclusions

In this paper, we have proposed a framework combining software training and hardware tuning to counter the aging effect of memristor-based neural networks. By pushing the weights to the region of small values, the currents flowing through memristors can be reduced to alleviate aging. The proposed aging-aware hardware mapping sets the valid resistance ranges of memristors properly to reduce the number of online tuning iterations and thus the aging effect further. Experimental results confirm that the proposed framework can extend the lifetime of memristor-based crossbars by up to 11 times without extra hardware cost.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] C. Farabet, Y. LeCun, K. Kavukcuoglu, E. Culurciello, B. Martini, P. Akselrod, and S. Talay, "Large-scale FPGA-based convolutional networks," *Scaling up Machine Learning: Parallel and Distributed Approaches*, pp. 399–419, 2011.

[3] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: avoiding the power wall with low-leakage, stt-mram based computing," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, 2010, pp. 371–382.

[4] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, B. Liu, Y. Chen, H. Li, Q. Wu, and H. Jiang, "A spiking neuromorphic design with resistive crossbar," in *Proc. Design Autom. Conf.*, 2015, pp. 1–6.

[5] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[6] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, "Training itself: Mixed-signal training acceleration for memristor-based neural network," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2014, pp. 361–366.

[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, 2015.

[8] B. Yan, J. J. Yang, Q. Wu, Y. Chen, and H. H. Li, "A closed-loop design to enhance weight stability of memristor based neural network chips," in *Proc. Int. Conf. Comput.-Aided Des.*, 2017, pp. 541–548.

[9] B. Chen, Y. Lu, B. Gao, Y. Fu, F. Zhang, P. Huang, Y. Chen, L. Liu, X. Liu, J. Kang *et al.*, "Physical mechanisms of endurance degradation in TMO-RRAM," in *Proc. Int. Electron Dev. Meeting*, 2011, pp. 12–3.

[10] Y. Y. Chen, B. Govoreanu, L. Goux, R. Degraeve, A. Fantini, G. S. Kar, D. J. Wouters, G. Groeseneken, J. A. Kittl, M. Jurczak *et al.*, "Balancing SET/RESET Pulse for $> 10^{10}$ Endurance in HfO_2/Hf 1T1R Bipolar RRAM," *IEEE Trans. Electron Devices*, vol. 59, no. 12, pp. 3243–3249, 2012.

[11] K. M. Kim, J. J. Yang, J. P. Strachan, E. M. Grafals, N. Ge, N. D. Melendez, Z. Li, and R. S. Williams, "Voltage divider effect for the improvement of variability and endurance of TaO_x memristor," *Scientific reports*, vol. 6, p. 20085, 2016.

[12] Y. Cai, Y. Lin, L. Xia, X. Chen, S. Han, Y. Wang, and H. Yang, "Long live TIME: improving lifetime for training-in-memory engines by structured gradient sparsification," in *Proc. Design Autom. Conf.*, 2018, pp. 1–6.

[13] M.Tarkov, "Mapping weight matrix of a neural network's layer onto memristor crossbar," *Optical Memory and Neural Networks*, vol. 24, no. 2, pp. 109–115, 2015.

[14] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. Design Autom. Conf.*, 2016, pp. 1–6.

[15] J. Park, M. Kwak, K. Moon, J. Woo, D. Lee, and H. Hwang, "TiO_x-based RRAM synapse with 64-levels of conductance and symmetric conductance change by adopting a hybrid pulse scheme for neuromorphic computing," *IEEE Electron Device Lett.*, vol. 37, no. 12, pp. 1559–1562, 2016.

[16] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose, "BSB training scheme implementation on memristor-based circuit," in *Computational Intelligence for Security and Defense Applications*, 2013, pp. 80–87.

[17] R. Degraeve, A. Fantini, P. Roussel, L. Goux, A. Costantino, C. Chen, S. Clima, B. Govoreanu, D. Linten, A. Thean *et al.*, "Quantitative endurance failure model for filamentary RRAM," in *Symp. VLSI Technol.*, 2015, pp. T188–T189.

[18] S. Balatti, S. Ambrogio, Z. Wang, S. Sills, A. Calderoni, N. Ramaswamy, and D. Ielmini, "Voltage-controlled cycling endurance of HfO_x-based resistive-switching memory," *IEEE Trans. Electron Devices*, vol. 62, no. 10, pp. 3365–3372, 2015.

[19] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *Proc. Int. Conf. Artif. Neural Netw.*, vol. 60, 1995, pp. 53–60.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[21] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[22] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *Operating Systems Design and Implementation*, vol. 16, 2016, pp. 265–283.