Reliable and Robust RRAM-based Neuromorphic Computing

Grace Li Zhang[1], Bing Li[1], Ying Zhu[1], Shuhang Zhang[1], Tianchen Wang[2], Yiyu Shi[2], Tsung-Yi Ho[3], Hai (Helen) Li[4], Ulf Schlichtmann[1]
[1]Chair of Electronic Design Automation, Technical University of Munich (TUM), Germany
[2]Department of Computer Science and Engineering, University of Notre Dame, United States
[3]Department of Computer Science, National Tsing Hua University, Taiwan
[4]Department of Electrical and Computer Engineering, Duke University, United States

Abstract

RRAM-based crossbars are a promising hardware platform to accelerate computations in neural networks. Before such a crossbar can be used as an accelerator for neural networks, RRAM cells should be programmed to target resistances to represent weights in neural networks. However, this process degrades the valid range of the resistances of RRAM cells from the fresh state, called aging effect. Therefore, after a certain number of programming iterations, these RRAM cells cannot be programmed reliably anymore, affecting the classification accuracy of neural networks negatively. In addition, process variations during manufacturing and noise during programming of RRAM cells also lead to significant accuracy degradation. To solve the problems described above, in this paper, we introduce a software/hardware codesign framework to reduce the aging effect in RRAM crossbars. To counter process variations and noise, we first model them as random variables and then modify the computations in software training considering these variables. Simulation results show that the lifetime of RRAM crossbars can be extended by up to 11 times with the codesign framework and the mean value and the standard deviation of the inference accuracy under process variations and noise can be improved significantly.

ACM Reference Format:
Grace Li Zhang, Bing Li, Ying Zhu, Shuhang Zhang, Tianchen Wang, Yiyu Shi, Tsung-Yi Ho, Hai (Helen) Li, and Ulf Schlichtmann. 2020. Reliable and Robust RRAM-based Neuromorphic Computing. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3407579

1 Introduction

To accelerate the huge number of multiplications and additions which are required for deep neural networks, various hardware platforms with emerging devices, e.g, RRAM/memristor[1–4], Mach Zehnder Interferometer [5–7], spintronics device [8, 9] and Ferroelectric Field-Effect Transistor [10–13] have been proposed. One such promising platform is the RRAM-based crossbar. Such a crossbar has the advantages of high computing efficiency and low power consumption. Fig. 1 illustrates the structure of the RRAM-based crossbar. Its basic element, an RRAM cell, sits at the crossing point between a wordline and a bitline [3].

With the crossbar in Fig. 1, multiplications and additions in neural networks are implemented efficiently due to its analog nature [14]. Specifically, the multiplication of two values is implemented by applying a voltage on an RRAM cell. The resulting current is thus the multiplication of the voltage and the conductance of the RRAM cell. The sum operation is the natural result of the currents flowing into the same column. With this analog style of computation, the power consumption of such RRAM crossbars is much lower than the counterpart circuits using CMOS logic.

To apply RRAM crossbars to accelerate neural networks, software training can be conducted first to determine weights of a neural network. Afterwards, the resistance levels of RRAM cells are programmed to implement the corresponding network weights. To compensate the accuracy loss resulting from quantization, the resistances of RRAM cells are further tuned. Programming and tuning RRAM cells are implemented by applying a pulse of relatively high voltage on RRAM cells. This high voltage causes the valid resistance range of RRAM cells to become narrower than that of their fresh state, so that the number of available resistance levels decreases. This phenomenon is also called aging. After the RRAM crossbar experiences aging, a trained weight might not be mapped correctly since the programmed resistance might differ from the target resistance, leading to a degradation of the classification accuracy of neural networks.

In addition to the aging effect, the programming and tuning processes are challenging because process variations cause deviations of the electrical properties of RRAM cells from their nominal values. In addition, noise affects the programming process randomly. Therefore, the same programming voltage does not necessarily cause the same conductance changes of RRAM cells inside a crossbar and across crossbars. The method of individual programming for each RRAM cell is not practical since it requires a huge amount of programming-reading cycles and thus time-consuming when crossbars are applied for mass production.

To solve the problems described above, methods of countering aging, process variations and noises are introduced. Specifically, to alleviate the aging effect, the resistances of RRAM cells are pushed towards large values by skewing weights during software training to reduce currents flowing through RRAM cells. In addition, the aging status of RRAM crossbars is considered when mapping weights into resistances of RRAM cells. To counter process variations and noise, a statistical training method is introduced where process variations and noise are extracted and modeled as statistical random variables. These variables are then incorporated into weights

GLSVLSI '20, September 7–9, 2020, Virtual Event, China

ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3407579

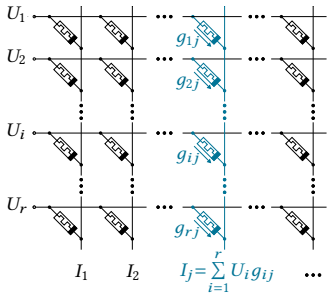

Figure 1: The structure of RRAM crossbar[15].

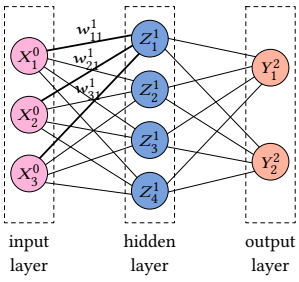

Figure 2: The structure of a 3-layer neural network [15].

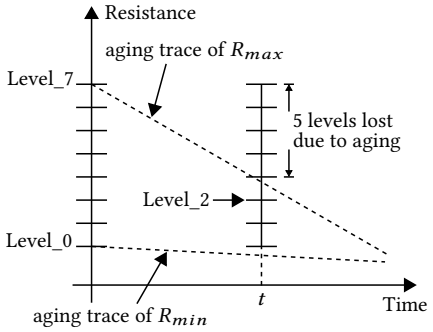

Figure 3: The valid resistance range of an RRAM cell decreases with programming time [16].

of neural networks. The computations, e.g, multiplication, addition, activation function and the cost function, are modified based on statistical weights. In this way, the modified cost function guides weight updates to maintain good accuracy of a neural network under process variations and noise.

The rest of this paper is organized as follows. In Section 2, we describe the background of RRAM-based neuromorphic computing. In Section 3, we describe the method to counter aging effects. In Section 4, the method to counter process variations and noise is described. Simulation results are shown in Section 5 and Conclusions are drawn in Section 6.

2 Background of RRAM-based Neuromorphic Computing

In this section, we describe the fundamentals of neural networks and the challenges of applying RRAM crossbars to accelerate neural networks.

2.1 Neural Networks

Fig. 2 illustrates the structure of a neural network with 3 layers: the input layer, the hidden layer and the output layer. In this neural network, nodes represent neurons and connections represent the relations between neurons in different layers. Assume the inputs at the first layer are X^0_1, X^0_2, X^0_3. The weights of the connections from X^0_1, X^0_2, X^0_3 to Z^1_1 are w^1_{11}, w^1_{21} and w^1_{31}. The input of the neuron at the top of the hidden layer can be evaluated as $I^1_1 = w^1_{11}X^0_1 + w^1_{21}X^0_2 + w^1_{31}X^0_3$. The output of this neuron is $Z^1_1 = \mathcal{A}(I^1_1)$, where $\mathcal{A}(\cdot)$ is the non-linear activation function, such as ReLU, softplus, tanh, or sigmoid functions.

To determine the weights in a neural network, the neural network should be trained in a way that data is fed to the input neurons and the outputs are compared with the expected values. Their difference is defined as the cost and is used to update weights to improve the classification accuracy. Specifically, the cost function is defined as

$$L = \sum_{i=1}^{M}(-\hat{Y}_i \log(Y_i) - (1-\hat{Y}_i)\log(1-Y_i)) + \sum_{i=1}^{N}\lambda \cdot \|\mathbf{W}_i\|^2 \quad (1)$$

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

where M is the number of neurons in the output layer, Y_i is the ith output generated by the neural network. $\hat{Y}_i$ is the expected output. N is the number of layers. $\mathbf{W}_i$ are the weights in the ith layer. λ is the penalty used in $L2$ regularization.

The weights in a neural network should be updated in a way to minimize the cost function in (1). To achieve this goal, the gradients of the cost function are used to adjust weights. After a certain number of training iterations, L converges gradually so that the weights can be determined. To apply RRAM crossbars to accelerate a neural network, a weight w_i in the neural network can be mapped linearly to the conductance g_i of the ith RRAM cell, expressed as

$$g_i = \frac{g_{max} - g_{min}}{w_{max} - w_{min}}(w_i - w_{min}) + g_{min} = \alpha w_i + \beta \quad (3)$$

where g_{max} and g_{min} are the maximum and minimum conductance values in the crossbars, respectively; w_{max} and w_{min} are the maximum and minimum weights in the neural network, respectively.

2.2 Challenges of Applying RRAM Crossbars to Accelerate Neural Networks

2.2.1 Aging of RRAM Crossbars After the target conductances of RRAM cells are determined with (3), RRAM cells should be programmed to reach the corresponding target conductances. In practice, it is difficult to program the conductances of RRAM cells to exact values. To simplify this process, the resistance range of RRAM cells is quantized into a given number of levels, e.g., 32[17], and RRAM cells are programmed to the resistance levels close to their target resistances. Due to this quantization, the inference accuracy with RRAM crossbars is lower than that after software training. To address this problem, RRAM cells are further tuned according to the results of applying training data on RRAM crossbars, called online tuning.

As described above, RRAM cells should be programmed iteratively to approximate their target resistances. In this process, a high voltage is applied on RRAM cells and generates currents flowing through them, leading to irrecoverable filament changes inside RRAM cells. Therefore, over time their valid range becomes narrower than that of their fresh state. This phenomenon is called aging [18–21]. It is shown in Fig. 3, where RRAM cells have 8 available resistance levels in their fresh state. The aging effect might lead to a mapping mismatch of weights to target resistances. For example, in Fig. 3, at time t, the resistance of an RRAM cell that should be programmed to Level_7 with a programming voltage can only reach Level_2. This mismatch degrades the classification accuracy significantly. To improve the accuracy, more programming iterations are required, which further aggravates the aging effect.

2.2.2 Process Variations and Weight Deviations Process variations are deviations of process parameters from their nominal specifications after manufacturing. These deviations cause variability in electrical properties of RRAM cells. Process variations can be classified into two categories, global variations and local variations. Global variations affect all RRAM cells on a crossbar in the same way. On the contrary, local variations affect each RRAM cell on a crossbar differently with correlations. Therefore, the conductance changes of RRAM cells vary diversely even if the same voltage is applied.

Assume the conductances of all RRAM cells are $\mathbf{G}$, process variations of all RRAM cells are $\mathbf{D}$ and random noise is $\mathbf{N}$. Since $\mathbf{D}$ are correlated among all RRAM cells, we first decompose the correlated $\mathbf{D}$ with principal component analysis (PCA). Afterwards, the real conductances considering process variations and noise can be written as follows

$$\mathbf{G} = \mathbf{G}_0 + \mathbf{f}(\mathbf{G}_0) \cdot \mathbf{V} \cdot \Sigma^{0.5} \cdot \mathbf{B} + \mathbf{h}(\mathbf{G_0})\mathbf{N} \tag{4}$$

where $\mathbf{G}_0$ denotes the mean values of conductances. $\mathbf{f}(\mathbf{G}_0)$ is the influence of process variations on conductances of RRAM cells. Σ are the eigenvalues of $\mathbf{D}$, $\mathbf{V}$ are the corresponding eigenvectors and $\mathbf{B}$ are independent random variables. $\mathbf{h}(\mathbf{G}_0)$ is the influence of noise on conductances of RRAM cells. $\mathbf{N}$ are independent random variables for RRAM cells.

To derive a weight under process variations and noise, we use the linear mapping relation between weight and conductance in (3). The canonical form of a weight [22] is expressed as

$$w_i = w_{i,0} + \sum_{k=1}^{N} w_{i,k} B_k + w_{i,n} N_i \tag{5}$$

where $w_{i,0}$ is the mean value of w_i. $w_{i,k}$ and $w_{i,n}$ are constant coefficients. B_k are independent random variables shared by all weights. N_i is the independent random variable for each weight.

3 Counter-aging Framework

In this section, a counter-aging framework with software/hardware codesign is introduced. At the software level, weights are pushed towards small values to reduce currents flowing through RRAM cells. At the hardware level, the aging status of RRAM crossbars is considered when mapping weights to resistance levels.

3.1 Weight Skewness in Software Training

As described in Section 2.2.1, aging effects result from currents flowing through RRAM cells during programming. To alleviate aging effects, currents through RRAM cells can be reduced by skewing the resistances of RRAM cells to large values. This concept is illustrated in Fig. 4, where weights are skewed towards small values. After being mapped to RRAM cells, these small weights lead to small conductances and thus large resistances. With this weight skewness, we can reduce currents flowing through RRAM cells, so that aging effects can be alleviated.

The weight skewness is viable during software training, since neural networks have much redundancy which allows to adjust weights to the desired direction. To realize the weight skewness during software training, the cost function should be modified to reflect the concentration of weights towards small values. This modification can be achieved by selecting a reference weight δ_i in the original weight range. This concept is shown in Fig. 5, where the original distribution of weights is illustrated as the solid curve,

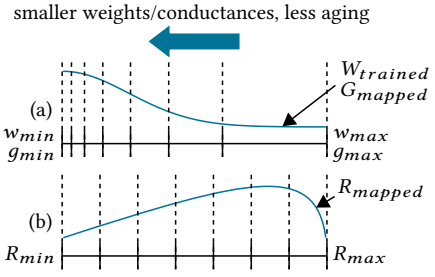

Figure 4: Skewed weight mapping. (a) Weights are skewed towards small values. (b) Resistance distribution corresponding to the skewed weight distribution [16].

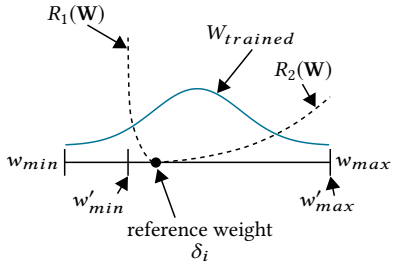

Figure 5: Modification of software training. $\mathbf{W}_{trained}$ is the original distribution of the weights. $R_1(\mathbf{W})$ and $R_2(\mathbf{W})$ are regularizations that added into the cost function to skew weights [16].

which is quasi-normal. To shape a skewed weight distribution, on the left side of δ_i, weights should be punished strongly to avoid that they fall into this range. For weights on the right side of δ_i, they should be punished according to the distance to the reference weight. For example, the larger distance to δ_i, the stronger weights are punished. The two dashed lines in Fig. 5 are the regularizations of the cost functions on both sides of δ_i.

To realize the weight punishment described above during software training, the term $R(\mathbf{W})$ in (2) is expanded into $R_1(\mathbf{W})$ and $R_2(\mathbf{W})$, expressed as

$$L = C(\mathbf{W}) + R_1(\mathbf{W}) + R_2(\mathbf{W}) \tag{6}$$

$$R_1(\mathbf{W}) = \sum_{i=1}^{N} \lambda_1 \cdot \|\mathbf{W}_i - \delta_i\|^2, \mathbf{W}_i < \delta_i \tag{7}$$

$$R_2(\mathbf{W}) = \sum_{i=1}^{N} \lambda_2 \cdot \|\mathbf{W}_i - \delta_i\|^2, \mathbf{W}_i \geq \delta_i \tag{8}$$

where δ_i is the reference weight. λ_1 and λ_2 are the penalty values for weights on the left side and right side of δ_i. λ_1 is larger than λ_2 to realize that weights on the left side are punished more strongly than those on the right side.

The cost function in (6) guides the weight update of a neural network to the desired direction while meeting the specified classification accuracy. When software training is finished, a skewed distribution of weights as in Fig. 4(a) is formed. The new weight range $[w'_{min}, w'_{max}]$ might be different from that in the original distribution.

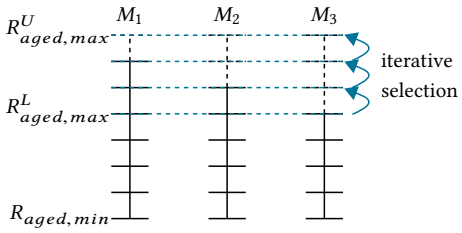

Figure 6: The selection of the upper bound of resistance in hardware mapping [16].

3.2 Hardware Mapping Considering Aging Status

The valid resistance range of RRAM cells becomes narrower than that of their fresh state after RRAM cells experience aging. If weights are mapped on aged RRAM cells, the inference accuracy of the neural network might be degraded significantly. To solve this problem, we consider the aging status of RRAM cells during hardware mapping. Specifically, the programming iterations of the RRAM cells at the centers of blocks with a certain size are traced. With this information, the upper and lower bounds of the resistance ranges are evaluated using the method presented in [16]. Since RRAM cells in a crossbar experience different numbers of programming iterations, their aging status is also different.

After the aging status of RRAM cells is extracted, we adopt an iterative selection method to determine the upper and lower bounds of their resistance range. Fig. 6 illustrates this selection process for three RRAM cells $M1$, $M2$ and $M3$. Initially, the upper bounds of their resistance ranges are the same. After aging, their upper bounds are reduced by 1, 2 and 3 levels, respectively. To implement hardware mapping, a common resistance range should be determined. To achieve this goal, all aged upper bounds of three RRAM cells between $R^L_{aged,max}$ and $R^U_{aged,max}$ in Fig. 6 are used to map weights iteratively. The upper bound that achieves the highest inference accuracy is considered as the new common resistance range. This new resistance range, however, does not necessarily cover the aged ranges of all RRAM cells. For example, if the range of M_2 is selected as the new common range, the mapping of weights to the common upper bound onto M_3 still fails, since this common upper bound is outside of its upper bound. To compensate for this inaccuracy, subsequent tuning of RRAM cells is conducted iteratively until a desired accuracy is achieved.

4 Statistical Training

As described in Section 2.2.2, weights considering process variations and noise can be expressed as a set of linear combinations of random variables. Therefore, they are statistical variables instead of deterministic variables. To incorporate such statistical variables into software training, the computations in neural networks such as multiplication, addition, activation functions and the cost function should be modified accordingly. The concept of statistical neural networks has been introduced in [23, 24]. However, inputs are modeled as statistical variables while weights still remain deterministic in their neural networks.

Statistical multiplication In a neural network such as shown in Fig. 2, the multiplication operation is conducted between the output result of a previous neuron and the corresponding weight. Since the output of a neuron is derived by previous computations, it is a statistical variable as shown in (5). This output result needs to be multiplied with the weight on the connection to the next neuron. Since the weight is now expressed in the canonical form in (5), this multiplication can be performed between two statistical variables. Assume the output result of a previous neuron and the corresponding weight are denoted as x_j and w_i. Their multiplication can be derived as follows

$$\begin{aligned} x_j \cdot w_i =& (x_{j,0} + \sum_{k=1}^{N} x_{j,k}B_k + x_{j,n}N_j)(w_{i,0} + \sum_{k=1}^{N} w_{i,k}B_k + w_{i,n}N_i) \\ =& x_{j,0}w_{i,0} + \sum_{k=1}^{N}(x_{j,0}w_{i,k} + x_{j,k}w_{i,0})B_k \\ &+ x_{j,0}w_{i,n}N_i + x_{j,n}w_{i,0}N_j \\ &+ \sum_{k=1}^{N}\sum_{l=1}^{N} x_{j,k}w_{i,l}B_kB_l + \sum_{k=1}^{N} x_{j,k}w_{i,n}B_kN_i \\ &+ \sum_{k=1}^{N} x_{j,n}w_{i,k}N_jB_k + x_{j,n}w_{i,n}N_jN_i \qquad (9) \\ \approx& x_{j,0}w_{i,0} + \sum_{k=1}^{N}(x_{j,0}w_{i,k} + x_{j,k}w_{i,0})B_k + \\ &+ \sqrt{(x_{j,0}w_{i,n})^2 + (x_{j,n}w_{i,0})^2}N_k. \qquad (10) \end{aligned}$$

The statistical multiplication result contains second order terms, making the data propagation across the neural network complex. To simplify the result, only the first order terms are maintained while the second order terms are removed. In addition, $x_{j,0}w_{i,n}N_i + x_{j,n}w_{i,0}N_j$ is approximated using $\sqrt{(x_{j,0}w_{i,n})^2 + (x_{j,n}w_{i,0})^2}N_k$ by maintaining their variances matched [22], where N_k is a new independent random variable. In this way, the multiplication result between two statistical variables can be expressed in the canonical from in (5). This result can be propagated through the neural network for further computations.

Statistical addition In a neural network, the addition operation should be performed at a neuron with the multiplication results of different input neurons. Assume that two multiplication results are x_i and x_j. Consequently, their sum can be derived as follows

$$x_i + x_j = (x_{i,0} + x_{j,0}) + \sum_{k=1}^{N}(x_{i,k} + x_{j,k})B_k + \sqrt{x_{i,n}^2 + x_{j,n}^2}N_k \qquad (11)$$

where N_k is a new independent random variable and its coefficient is evaluated by matching the original variance.

Softplus operation with statistical variables

At a neuron, the addition result of different previous neurons is the input of this neuron. Its output is determined by processing this addition result with an activation function. There are various types of activation functions. Inside a neural network, we use softplus $f(x) = \log(1 + e^x)$, also called SmoothReLU, as the activation function.

Assume the addition result of a neuron is $a_i = a_{i,0} + A_i$, where $A_i = \sum_{k=1}^{N} a_{i,k}B_k + a_{i,n}N_i$. To process this result with the softplus activation function, we first expand this function with Taylor series at the mean value $a_{i,0}$. During this expansion, the terms with an

order larger than 1 are removed to reduce the complexity. The addition result after passing through softplus can be expressed as follows

$$x_i = f(a_i) = \log(1 + e^{a_{i,0}}) + \frac{e^{a_{i,0}}}{1 + e^{a_{i,0}}} A_i + \sum_{l=2}^{\infty} \frac{f^{(l)}(a_{i,0})}{l!} A_i^l$$
$$\approx \log(1 + e^{a_{i,0}}) + \frac{e^{a_{i,0}}}{1 + e^{a_{i,0}}} A_i. \quad (12)$$

The linearization of the softplus function simplifies the non-linear computation, so that the information can be propagated to the next layer.

Sigmoid operation with statistical variables In the last layer of a neural network, usually the sigmoid function $f(x) = 1/(1 + e^{-x})$ is applied to perform the classification task. Similar to the Taylor series of the softplus function, the sigmoid function can be approximated as follows

$$x_i = f(a_i) = \frac{1}{1 + e^{-a_{i,0}}} + \frac{e^{-a_{i,0}}}{(1 + e^{-a_{i,0}})^2} A_i + \sum_{l=2}^{\infty} \frac{f^{(l)}(a_{i,0})}{l!} A_i^l$$
$$\approx \frac{1}{1 + e^{-a_{i,0}}} + \frac{e^{-a_{i,0}}}{(1 + e^{-a_{i,0}})^2} A_i. \quad (13)$$

Incorporation of statistical information into cost function During software training, the cost function of a neural network is minimized to guide the update of weights to improve the inference accuracy gradually. The cost function when process variations and noise are not considered is shown in (1). By minimizing the cost function, if the expected output value is equal to 1, $\hat{Y}_i = 1$, the corresponding output value generated by the neural network Y_i is pushed to approximate 1, since the value of the cost function is the smallest value 0 when Y_i equals 1. If Y_i differs much from 1, the L in (1) is large, which leads to the adjustment of weights to generate a smaller cost value. Similarly, the case when $\hat{Y}_i = 0$ also causes the cost function to push weights towards the correct direction.

Considering process variations and noise, the output of a neuron is a statistical variable instead of a deterministic value. Consequently, the comparison between the expected value, e.g., 1 or 0, and the real output value also becomes statistical. To reflect this comparison into the cost function, we try to shift the whole distribution of the output towards the expected value. To achieve this goal, the mean value μ_{Y_i} of the distribution is used to replace Y_i in (1). In addition, we also incorporate the probability $P(Y_i \leq 0.5)$ into the cost function to punish weights if its distribution Y_i is not close to the expected value $\hat{Y}_i = 1$. For example, if Y_i is far from 1, the probability $P(Y_i \leq 0.5)$ becomes large. Similarly, the probability $P(Y_i \geq 0.5)$ is incorporated into the cost function for the case where $\hat{Y}_i = 0$. Therefore, the cost function is modified as follows

$$L = \sum_{i=1}^{M} (- \hat{Y}_i P^p(Y_i \leq 0.5) \log(\mu_{Y_i})$$
$$- (1 - \hat{Y}_i) P^p(Y_i \geq 0.5) \log(1 - \mu_{Y_i})) \quad (14)$$

where p is a power used to amplify the relative influence of the different probabilities in the equation.

5 Simulation Results

5.1 Results of Counter-aging Framework

The effectiveness of the software/hardware codesign is verified with two different neural networks, LeNet-5 [25] and VGG-16 [26], onto Cifar10 and Cifar100 [27], respectively. The information of neural networks and datasets is shown in the first two columns of Table 1. To realize the weight skewness during software training, the reference weights δ_i shown in Fig. 5 for LeNet-5 and VGG-16 are $-\sigma_i$ and $-0.75\sigma_i$, respectively. The values of penalty λ_1 and λ_2 in (7) and (8) for LeNet-5 are 1 and 0.005, respectively. For VGG-16, the values of λ_1 and λ_2 are 0.001 and 0.005, respectively. These parameters are determined according to the results of both inference accuracy and the weight skewness with various combinations of parameters.

Table 1: Comparisons of Accuracy and Lifetime [16]

		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×

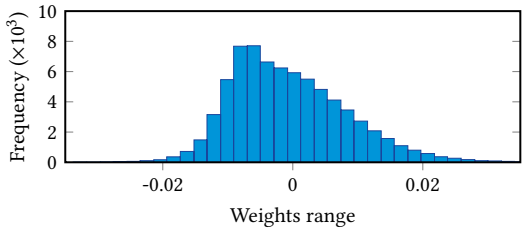

Figure 7: Skewed weight distribution of the third layer of VGG-16 [16].

In Table 1, the third and fourth columns show the accuracy of the traditional software training and the accuracy with weight skewness during software training. It is clear that the inference accuracy with weight skewness is slightly lower for LeNet-5 than that without weight skewness. For VGG-16, the inference accuracy with weight skewness is even higher than the accuracy without weight skewness. This accuracy comparison shows that neural networks have redundancy to adjust weights to the desired direction, while still guaranteeing a high inference accuracy. The software/hardware codesign method exploits the redundancy to improve the lifetime of the crossbars.

With weight skewness in software training, weights are pushed towards small values. The result of weight distribution for VGG-16 is shown in Fig. 7. It is clear that weights after software training with weight skewness are concentrated at small values.

To demonstrate the lifetime improvement with the counter-aging framework, we simulated the tuning process for a certain number of applications for three scenarios, T+T (traditional weight training and online tuning), ST+T (skewed weight training and online tuning) and ST+AT (skewed weight training with aging-aware mapping and online tuning). The comparison of lifetime improvement of the three scenarios is shown in the last three columns of Table 1. With weight skewness, the lifetime of LeNet-5 and VGG-16 have been improved by 6× and 7×. Combined with hardware mapping considering aging status, the lifetime can be improved by 8× and 11×.

5.2 Results of Statistical Training

To evaluate the effectiveness of the statistical training, we set the standard deviations of the distribution of process variations and noise to be 25% and 5% of the mean values [28, 29]. The global

Table 2: Accuracy with different training methods [15]

	DT			VT			ST		
NN	Acc_0	$\mu(Acc)$	$\sigma(Acc)$	Acc_0	$\mu(Acc)$	$\sigma(Acc)$	Acc_0	$\mu(Acc)$	$\sigma(Acc)$
FC1	0.91	0.87	0.10	0.91	0.89	0.05	0.91	0.90	0.03
FC2	0.97	0.32	0.33	0.92	0.38	0.32	0.92	0.92	0.01

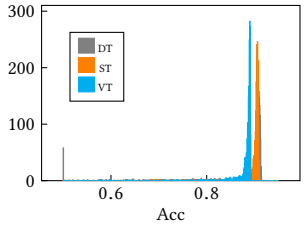

Figure 8: Accuracy distributions [15].

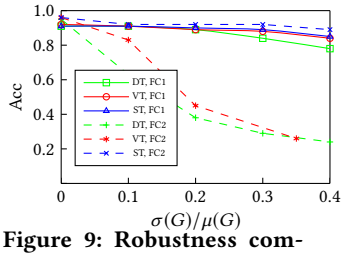

Figure 9: Robustness comparison [15].

variations were set to 60% of the total process variations. The covariance matrix of local variations in a crossbar was generated with the method in [30]. The power p in (14) was set to 2.

To verify the inference accuracy under process variations, 2000 chips were simulated with Monte Carlo simulations by sampling process variations and noise. These chips were tested with one-layer and two-layer fully-connected neural networks, denoted as FC1 and FC2, on MNIST. Table 2 shows the results for FC1 and FC2. In this table, DT represents the traditional training method when process variations and noise are not considered, VT is the training method in [31] and ST is the statistical training proposed in this paper. Acc_0 is the accuracy with ideal programming process where RRAM cells are programmed exactly to the target resistances. $\mu(Acc)$ and $\sigma(Acc)$ are the mean and standard deviation of the accuracy distribution collected from the simulated 2000 chips, respectively.

It is clear from Table 2 that the proposed ST method has a larger mean value and a smaller standard deviation compared with VT and DT. Therefore, it achieves a better accuracy distribution under process variations and noise. The accuracy distribution of simulated chips tested on $FC1$ is shown in Fig. 8, where more chips achieve a high accuracy with ST. To verify the effectiveness of ST under various process variations, different proportions of the standard deviation to the mean value were tested with FC1 and FC2. Fig. 9 illustrates the results, where ST still maintains a stable accuracy with the increase of process variations while the accuracy of DT and VT degrade significantly.

6 Conclusions

In this paper, we introduce a software/hardware codesign framework to reduce the aging effect in RRAM crossbars. At the software level, by pushing weights towards small values, the currents flowing through RRAM cells can be reduced to alleviate aging. At the hardware level, we consider aging status of RRAM cells in mapping weights into resistance levels. To counter process variations and noise, we introduce a statistical training method, where process variations and noise are modeled as random variables. Simulation results show that the lifetime of RRAM crossbars can be extended by up to 11 times and the mean value and the standard deviation of the inference accuracy under process variations and noise can be improved significantly.

References

[1] 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.

[2] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *Proc. Design Autom. Conf.* IEEE, 2017, pp. 1–6.

[3] 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.

[4] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects," in *IEEE International Conference on Artificial Intelligence Circuits and Systems*, 2020, pp. 11–15.

[5] S. S. Yichen Shen, Nicholas C. Harris, "Deep learning with coherent nanophotonic circuits," *naturephotonics*, vol. 11, pp. 441–446, 2017.

[6] Z. Zhao, D. Liu, M. Li, Z. Ying, L. Zhang, B. Xu, B. Yu, R. T. Chen, and D. Z. Pan, "Hardware-software co-design of slimmed optical neural networks," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2019, pp. 705–710.

[7] M. Y.-S. Fang, S. Manipatruni, C. Wierzynski, A. Khosrowshahi, and M. R. DeWeese, "Design of optical neural networks with component imprecisions," *Opt. Express*, vol. 27, pp. 14 009–14 029, 2019.

[8] J. Grollier, D. Querlioz, K.Y.Camsari, K.Everschor-Sitte, S. Fukami, and M. D. Stiles, "Neuromorphic spintronics," *IEEE Transactions on Electron Devices*, 2020.

[9] A. Sengupta, K. Yogendra, and K. Roy, "CMOS spintronic devices for ultra-low power neuromorphic computation," in *IEEE international symposium on Circuits and Systems (ISCAS)*, 2016, pp. 922–925.

[10] M. Li, X. Yin, X. S. Hu, and C. Zhuo, "Nonvolatile and energy-efficient feFET-based multiplier for energy-harvesting devices," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2019, pp. 526–567.

[11] X. Yin, C. Li, Q. Huang, L. Zhang, M. Niemier, X. S. Hu, C. Zhuo, and K. Ni, "FeCAM: A universal compact digital and analog content addressable memory using ferroelectric," *IEEE Transactions on Electron Devices*, vol. 67, no. 7, pp. 2785–2792, 2020.

[12] K. Ni, X. Yin, A. F. Laguna, S. Joshi, S. Dünkel, M. Trentzsch, J. Müller, S. Beyer, M. Niemier, X. S. Hu, and S. Datta, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, pp. 521–529, 2019.

[13] X. Yin, K. Ni, D. Reis, S. Datta, M. Niemier, and X. S. Hu, "An ultra-dense 2fefet tcam design based on a multi-domain feFET model," *IEEE Transactions on Circuits and Systems II: Express Briefs (TCAS II)*, 2018.

[14] 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.

[15] Y. Zhu, G. L. Zhang, T. Wang, B. Li, Y. Shi, T.-Y. Ho, and U. Schlichtmann, "Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise," in *Proc. Design, Autom., and Test Europe Conf.*, 2020.

[16] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *Proc. Design, Autom., and Test Europe Conf.*, 2020, pp. 1751–1756.

[17] 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.

[18] 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. 12.3.1–12.3.4.

[19] 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.

[20] 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.

[21] 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.

[22] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett, "First-order incremental block-based statistical timing analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2170–2180, 2006.

[23] T. Wang, J. Xiong, X. Xu, and Y. Shi, "SCNN: A general distribution based statistical convolutional neural network with application to video object detection," in *Proc. AAAI Conf. on Artificial Intelligence*, 2019.

[24] T. Wang, J. Xiong, X. Xu, M. Jiang, Y. Shi, H. Yuan, M. Huang, and J. Zhuang, "MSU-Net: Multiscale statistical U-net for real-time 3D cardiac MRI video segmentation," in *Proc. Medical Image Computing and Computer Assisted Interventions*, 2019.

[25] 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.

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

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

[28] E. Amat, A. Rubio *et al.*, "Memristive crossbar memory lifetime evaluation and reconfiguration strategies," *IEEE Trans. Emerg. Topics in Comput.*, vol. 6, no. 2, pp. 207–218, 2016.

[29] S. Choi, Y. Yang, and W. Lu, "Random telegraph noise and resistance switching analysis of oxide based resistive memory," *Nanoscale*, vol. 6, no. 1, pp. 400–404, 2014.

[30] J. Xiong, V. Zolotov, and L. He, "Robust extraction of spatial correlation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 4, pp. 619–631, 2007.

[31] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware training for memristor x-bar," in *Proc. Design Autom. Conf.*, 2015.