Statistical Training for Neuromorphic Computing using Memristor-based Crossbars Considering Process Variations and Noise

[1]Ying Zhu, [1]Grace Li Zhang, [2]Tianchen Wang, [1]Bing Li, [2]Yiyu Shi, [3]Tsung-Yi Ho and [1]Ulf Schlichtmann
[1]Technical University of Munich, [2]University of Notre Dame, [3]National Tsing Hua University
{ying.zhu, grace-li.zhang, b.li, ulf.schlichtmann}@tum.de, {twang9, yshi4}@nd.edu, tyho@cs.nthu.edu.tw

***Abstract*—Memristor-based crossbars are an attractive platform to accelerate neuromorphic computing. However, process variations during manufacturing and noise in memristors cause significant accuracy loss if not addressed. In this paper, we propose to model process variations and noise as correlated random variables and incorporate them into the cost function during training. Consequently, the weights after this statistical training become more robust and together with global variation compensation provide a stable inference accuracy. Simulation results demonstrate that the mean value and the standard deviation of the inference accuracy can be improved significantly, by even up to 54% and 31%, respectively, in a two-layer fully connected neural network.**

I. Introduction

Memristor-based crossbars are a promising hardware platform to accelerate computation operations in deep neural networks due to their high energy efficiency. The structure of the memristor-based crossbar [1] is shown in Fig. 1, where memristors sit between the horizontal wordlines and the vertical bitlines at the crossing points.

In the vector-matrix multiplication executed by a crossbar, the matrix is represented by the conductance values of the memristors. In neuromorphic computing, these conductance values correspond to the weights of a neural network after training, and should be programmed into the memristors before the crossbar is used for computation acceleration. In reality, this programming is challenging, because process variations make memristors after manufacturing differ from each other and noise affects the programming accuracy as well. Consequently, the same programming voltage may lead to different conductance changes of the memristors inside a crossbar and across crossbars. A straightforward way to overcome this problem is to program memristors individually with many reading-programming cycles. But this method is too time-consuming for crossbars in large-volume industrial production.

To alleviate the effect of weight deviation in memristors, [2] trains the weights using Monte Carlo simulation to minimize the expected value of the cost function. [3] minimizes the corner cases of the statistical cost function and adjusts the mapping between the weights and conductance values of the memristors. In addition, [4] applies iterative training and remapping to reduce the weight variance for a given crossbar. Furthermore, [5] exploits skewed weight distribution to reduce wearing of memristors. These methods, however, either are timing-consuming or require heavy on-chip tests and redundant memristors to counter large variations. More recently, the concept of statistical neural networks has been proposed in [6], [7], but the canonical forms there are used to model the correlation in the inputs and the weights remain constant. In this paper, we propose a training method for memristor-based crossbars with statistical weights, which result in a fundamentally different problem.

In this paper, we introduce a statistical training method to model process variations and noise as correlated random variables and incorporate them into weights of neural networks during training. The cost function during training is modified to represent the probability of the correct output values, so that the resulting weights can maintain a good inference accuracy after being mapped onto memristors under process variations and noise. In addition, global variation is compensated by scaling the target programming values according to the average of variations of memristors in a column. With the techniques above, the inference accuracy of crossbars after manufacturing can be well maintained with a narrow distribution despite process variations and noise.

The rest of this paper is organized as follows. We first introduce the conventional training method of neural networks and process variations in Section II. Weight variations under process variations and noise are described in Section III. The proposed statistical training and global variation compensation are then explained in Section IV and Section V, respectively. Simulation results are presented in Section VI and conclusions are drawn in Section VII.

II. Background

In this section, we describe the general structure of neural networks and process variations.

A. Neural Networks

The basic structure of a neural network is shown in Fig. 2, which consists of an input layer, a hidden layer and an output layer, where the nodes represent neurons. In the neural network, the output of a neuron can be expressed as a function of the input neurons. For example, in Fig. 2, $Z_1^1{=}\mathcal{A}(w_{11}^1X_1^0{+}w_{21}^1X_2^0{+}w_{31}^1X_3^0)$, where w_{11}^1, w_{21}^1 and w_{31}^1 are the weights of the connections from X_1^0, X_2^0, X_3^0 to Z_1^1, respectively. $w_{11}^1X_1^0{+}w_{21}^1X_2^0{+}w_{31}^1X_3^0$ can be calculated efficiently as the output of a column in a crossbar such as shown in Fig. 1. $\mathcal{A}(\cdot)$ is the activation function used to introduce non-linearity to the neural network, such as ReLU, softplus, tanh, and sigmoid

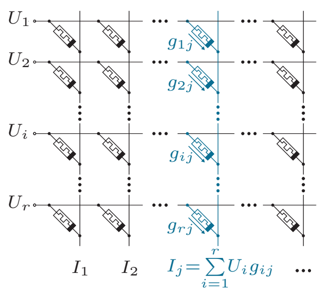

Fig. 1. Memristor crossbar architecture.

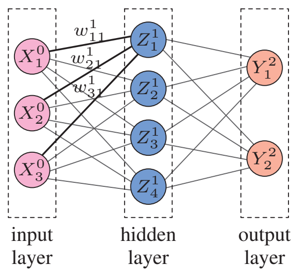

Fig. 2. Basic structure of neural networks.

functions.

During training, training data are applied to the input neurons of the neural network. The data at the outputs are compared with the expected values. The difference is used to construct a cost function to adjust the weights to improve the computational quality, i.e., inference accuracy. The cost function can be expressed as

$$L=\sum_{i=1}^{M}(-\hat{Y}_i\log(Y_i)-(1-\hat{Y}_i)\log(1-Y_i)) \qquad (1)$$

where M is the number of neurons in the output layer, Y_i is the ith output of the neural network, and $\hat{Y}_i$ is the expected value of the output.

To minimize the cost function L, the weights in the neural network, denoted as w_i, are adjusted according to the gradient of L to the weights, namely, the gradient descent method [8], as

$$w_i \leftarrow w_i-\gamma\cdot\frac{\partial L}{\partial w_i} \qquad (2)$$

where γ is the learning rate and $\frac{\partial L}{\partial w_i}$ is the gradient of L with respect to w_i, which can be calculated by back-propagation [9].

After a sufficient number of iterations, the weights can be determined as L converges. Assume that the ith weight w_i in the neural network is mapped to the conductance g_i of the ith memristor. This mapping can be described as

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

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, respectively.

B. Process Variations

Process variations are inherent in the manufacturing process [10]. After manufacturing, the variations would cause the physical and electrical properties of memristors to differ from each other. Consequently, the conductance values of memristors after programming deviate from their nominal values according to training. Since process variations are statistical, this conductance deviation is also statistical.

Process variations consist of global variation and local variations. The former is shared by all memristors and the latter are specific to individual memristors. In the manufacturing process, local variations are correlated. In addition, the shared global variation also increases the correlation between the memristors. The overall correlation between the memristors can be expressed using a covariance matrix $\mathbf{R}$. This matrix can be decomposed using principle component analysis (PCA) [11] to facilitate the computation during training, as

$$\mathbf{R}=\mathbf{V}\cdot\mathbf{\Sigma}\cdot\mathbf{V}^T \qquad (4)$$

where $\mathbf{\Sigma}=diag(\lambda_1,\lambda_2,...,\lambda_N)$ contains the eigenvalues of $\mathbf{R}$, and $\mathbf{V}=[V_1,V_2,...,V_N]$ contains the corresponding eigenvectors which are orthogonal to each other.

Assume that the variations of all the memristors are written together as $\mathbf{D}$. After the decomposition in (4), $\mathbf{D}$ can be expressed as

$$\mathbf{D}=\mathbf{V}\cdot\mathbf{\Sigma}^{0.5}\cdot\mathbf{B} \qquad (5)$$

where $\mathbf{B}=[B_1,B_2,...,B_N]$ are independent random variables. The representation in (5) expresses the variations on memristors as linear combinations of independent random variables. To reduce computation complexity, the independent random variables corresponding to small eigenvalues can also be discarded without affecting the modeling accuracy significantly.

III. Weight Variations and the Canonical Form

Due to process variations, the actual conductance values programmed into memristors vary from their nominal values. Assume the conductance values of all the memristors are expressed as $\mathbf{G}$. The actual conductance values considering process variations can be expressed as

$$\mathbf{G}=\mathbf{G}_0+\mathbf{F}(\mathbf{G}_0,\mathbf{D})\approx\mathbf{G}_0+\mathbf{f}(\mathbf{G}_0)\cdot\mathbf{D} \qquad (6)$$

where $\mathbf{G}_0$ represents the nominal conductance values. $\mathbf{F}(\mathbf{G}_0,\mathbf{D})$ is a function representing the relation between process variations and the change of conductance, which can be approximated by $\mathbf{f}(\mathbf{G}_0)\cdot\mathbf{D}$ due to the relatively small value of the process variations compared with the nominal values. $\mathbf{f}(\mathbf{G}_0)$ can be characterized from device measurement directly. Together with (5), (6) can be transformed into

$$\mathbf{G}=\mathbf{G}_0+\mathbf{f}(\mathbf{G}_0)\cdot\mathbf{V}\cdot\mathbf{\Sigma}^{0.5}\cdot\mathbf{B}. \qquad (7)$$

Besides process variations, programming noise also causes random drifting of the conductance values [12]. To incorporate this effect, (7) can be modified as

$$\mathbf{G}=\mathbf{G}_0+\mathbf{f}(\mathbf{G}_0)\cdot\mathbf{V}\cdot\mathbf{\Sigma}^{0.5}\cdot\mathbf{B}+\mathbf{h}(\mathbf{G_0})\mathbf{N} \qquad (8)$$

where $\mathbf{h}(\mathbf{G}_0)$ represents the impact of noise on the conductance values of memristors and $\mathbf{N}$ contains independent random variables specific to certain memristors.

According to (8) and the relation between the weights and the conductance values in (3), the weights can be expressed as linear combinations of $\mathbf{B}$ and $\mathbf{N}$, since $\mathbf{G_0}$ and all the coefficients of $\mathbf{B}$ and $\mathbf{N}$ are known. Therefore, a weight under process variations and noise can be expressed in the canonical form [13] as

$$w_i=w_{i,0}+\sum_{k=1}^{N}w_{i,k}B_k+w_{i,n}N_i \qquad (9)$$

where $w_{i,0}$ is the nominal value of the weight, $w_{i,k}$ and $w_{i,n}$ are constant coefficients. B_k are random variables independent from each other but shared by all the weights. N_i is the pure random variable individual to each weight.

IV. Statistical Training

Since the weights have been expressed as linear combinations of random variables to incorporate process variations and noise, the training of neural networks should also be adapted, in which the multiplication, addition, activation functions as well as the cost function should be modified accordingly.

Multiplication At a neuron in Fig. 2, multiplication of the input value from a previous neuron and the corresponding weight needs to be conducted. The input value is the result of previous computation operations, so that it is already in the canonical form (9). Consequently, the multiplication should be performed between two expressions in the canonical form. Assume the input is denoted as a_j and the weight to multiply is denoted as w_i, The multiplication can be performed as

$$\begin{aligned} a_j{\cdot}w_i=&(a_{j,0}+\sum_{k=1}^{N}a_{j,k}B_k+a_{j,n}N_j)(w_{i,0}+\sum_{k=1}^{N}w_{i,k}B_k+w_{i,n}N_i)\\ =&a_{j,0}w_{i,0}+\sum_{k=1}^{N}(a_{j,0}w_{i,k}+a_{j,k}w_{i,0})B_k\\ &+a_{j,0}w_{i,n}N_i+a_{j,n}w_{i,0}N_j\\ &+\sum_{k=1}^{N}\sum_{l=1}^{N}a_{j,k}w_{i,l}B_kB_l+\sum_{k=1}^{N}a_{j,k}w_{i,n}B_kN_i\\ &+\sum_{k=1}^{N}a_{j,n}w_{i,k}N_jB_k+a_{j,n}w_{i,n}N_jN_i \qquad (10)\\ \approx&a_{j,0}w_{i,0}+\sum_{k=1}^{N}(a_{j,0}w_{i,k}+a_{j,k}w_{i,0})B_k+\\ &+\sqrt{(a_{j,0}w_{i,n})^2+(a_{j,n}w_{i,0})^2}N_k. \qquad (11)\end{aligned}$$

The multiplication above produces terms of the second order and complicates the further propagation of the data across the neural network. To reduce computational complexity, we only keep the first-order terms and approximate $a_{j,0}w_{i,n}N_i+a_{j,n}w_{i,0}N_j$ using $\sqrt{(a_{j,0}w_{i,n})^2+(a_{j,n}w_{i,0})^2}N_k$ by matching their variances [13], where N_k is a new independent random variable. Consequently, the result of a multiplication can also be represented in a canonical form and propagated further through the neural network using the same implementation of the computation operations.

Addition At a neuron, the results of multiplication operations from different input neurons should be added together. Assume that the results of two multiplications are a_i and a_j, their sum can be computed as

$$a_i+a_j=(a_{i,0}+a_{j,0})+\sum_{k=1}^{N}(a_{i,k}+a_{j,k})B_k+\sqrt{a_{i,n}^2+a_{j,n}^2}N_k \qquad (12)$$

where N_k is a new independent random whose coefficient is determined by matching variance with $a_{i,n}N_i+a_{j,n}N_j$.

Softplus operation At a neuron, the result of multiplication and addition needs to be processed by an activation function. Inside a neural network, the active function used at neurons is often ReLU $y=max(x,0)$. However, this function is nondifferentiable at 0, a condition that is needed to expand this function into Taylor series to allow the propagation of the canonical form (9) through the neural network. Therefore, we choose the softplus $f(x)=\log(1+e^x)$, also called SmoothReLU, as the activation function.

Assume the input to the softplus function is in the canonical form (9) and denoted also as $z_i=z_{i,0}+Z_i$, where $Z_i=\sum_{k=1}^{N}z_{i,k}B_k+z_{i,n}N_i$. The softplus function can be expanded using Taylor series at the mean value $z_{i,0}$ and approximated by discarding terms with an order higher than 1, as

$$\begin{aligned} a_i=f(z_i)=&\log(1+e^{z_{i,0}})+\frac{e^{z_{i,0}}}{1+e^{z_{i,0}}}Z_i+\sum_{l=2}^{\infty}\frac{f^{(l)}(z_{i,0})}{l!}Z_i^l\\ \approx&\log(1+e^{z_{i,0}})+\frac{e^{z_{i,0}}}{1+e^{z_{i,0}}}Z_i. \qquad (13)\end{aligned}$$

Note that this linearization of the softplus function adapts itself according to the mean value of the input data to allow useful information to be propagated to the next layer.

Sigmoid operation At the outputs of the neural network, sigmoid function $f(x)=1/(1+e^{-x})$ may be applied to generate classification results. Similar to softplus operation, this function can be expanded into Taylor series and approximated as

$$\begin{aligned} a_i=f(z_i)=&\frac{1}{1+e^{-z_{i,0}}}+\frac{e^{-z_{i,0}}}{(1+e^{-z_{i,0}})^2}Z_i+\sum_{l=2}^{\infty}\frac{f^{(l)}(z_{i,0})}{l!}Z_i^l\\ \approx&\frac{1}{1+e^{-z_{i,0}}}+\frac{e^{-z_{i,0}}}{(1+e^{-z_{i,0}})^2}Z_i. \qquad (14)\end{aligned}$$

Modification of cost function As shown in (1), the cost function is used to guide the adjustment of the weights. If the expected value of an output is equal to 1, i.e., $\hat{Y}_i=1$, the actual value of this output Y_i is expected to be close to 1. If Y_i deviates much from 1, L in (1) quickly becomes large, so that the weights that contribute to this deviation are punished, as indicated by (2). This mechanism works similarly in the case when the output $\hat{Y}_i$ is equal to 0.

When process variations and noise are considered, the actual value at an output is represented in the canonical form (9), so that the comparison between the actual value and the expected value, which is either 1 or 0 for classification, becomes statistical. To signify that the whole distribution of the output should be shifted toward the expected value, we use the mean value μ_{Y_i} of the distribution to replace Y_i in the original cost function (1). In addition, we punish an output if its distribution is not close to the expected value. If the expected value $\hat{Y}_i$ is 1 but Y_i incorrectly drifts to 0, the probability $P(Y_i{\leq}0.5)$ becomes large. Therefore, we also include this probability into the cost function. Similarly, the case that the expected value equal to 0 is also adapted. Consequently, the cost function (1) is transformed as

$$\begin{aligned} L=&\sum_{i=1}^{M}(-\hat{Y}_iP^p(Y_i{\leq}0.5)\log(\mu_{Y_i})\\ &-(1-\hat{Y}_i)P^p(Y_i{\geq}0.5)\log(1-\mu_{Y_i})) \qquad (15)\end{aligned}$$

where p is a power used to magnify the influence of the probability.

V. Compensation for global variation

As explained in Section II, global variation is shared by all the memristors, so that this variation contributes a large part of the correlation between the process variations. Consequently, the variations in the conductance values of the memristors also exhibit a tendency to vary into the same direction. To capture

TABLE I
Accuracy of SNN with different training methods

	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

this tendency, memristors in a column in the crossbar as shown in Fig. 2 can be tested by applying a test voltage U_t to all the wordlines. The current measured at the output of the jth column can be written as

$$I'_j = (g'_{1j} + g'_{2j} + ... + g'_{rj}) \cdot U_t \tag{16}$$

where $g'_{1j}, g'_{2j}, ..., g'_{rj}$ are the actual conductance values of the memristors in the jth column. Due to process variations and noise during programming, these conductance values deviate from the values determined by weights after training. However, since the random variations experienced by the memristors can cancel each other out when the conductance values are added together, this sum can be used to evaluate the overall effect of global variation.

Since the current with the ideal conductances for these memristors can be expressed as

$$I_j = (g_{1j} + g_{2j} + ... + g_{rj}) \cdot U_t \tag{17}$$

the current deviation caused by global variation on the memristors in this column can be expressed as a ratio $R = I'_j / I_j$. When programming memristors, we then adjust the programming voltage to offset the conductance by $1/R$ to compensate global variation in advance to improve the inference accuracy.

VI. Simulation Results

The proposed framework was implemented using Tensorflow [14] and tested with an Intel 3.6 GHz CPU and an Nvidia GeForce GTX1080Ti graphics card. The standard deviations of the distribution of process variations and noise were set to 25% and 5% of the nominal values [10], [12]. The covariance matrix was generated using the method in [15] and the global variation was set to 60% of the total process variations.

We simulated 2000 chips using Monte Carlo simulation and tested them with one-layer and two-layer fully-connected neural networks. The results are shown in Table I, where DT is the conventional training method with deterministic weights without considering process variations and noise, VT is the vortex training method [3] and ST is the method proposed in this paper. In this experiment, the power p in (15) was set to 2. NN stands for neural network, FC1 and FC2 are one-layer and two-layer fully-connected neural networks, respectively. Acc_0 is the accuracy of the ideal case when memristors are programmed to conductance values equal to the target values mapped from weights, $\mu(Acc)$ and $\sigma(Acc)$ are the mean and standard deviation of the inference accuracy of the simulated 2000 chips, respectively.

As shown in Table I, the proposed ST method has a larger mean value and a smaller standard deviation in the inference accuracy compared with VT and DT, indicating that the simulated chips can achieve a good accuracy with much fewer failing outliers. The distribution of the inference accuracy of the simulated chips tested with $FC1$ is shown in Fig. 3, where

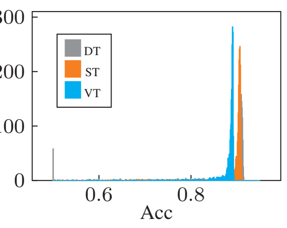

Fig. 3. Accuracy distributions.

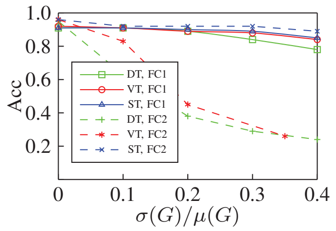

Fig. 4. Robustness comparison.

the results of ST have more chips concentrated toward high accuracy.

To verify the robustness of the proposed method, we tested different ratios of the standard deviation to the mean value of process variations, denoted as σ_G/μ_G, using FC1 and FC2. The results are illustrated in Fig. 4, where the y-axis shows the mean value of the inference accuracy of the simulated chips. According to this comparison, the proposed ST method maintains a stable accuracy as process variations increase, while the other methods suffer a significant accuracy loss.

VII. Conclusions

In this paper, a statistical training method has been proposed for neural networks. By modeling process variations and noise as random variables and applying global variation compensation, the inference accuracy can be improved significantly, even as process variations become large. Future work includes exploring statistical training on more complex multi-layer neural networks and convolutional neural networks.

Acknowledgment

The project is supported in part by the Deutsche Forschungsgemeinschaft through TUM International Graduate School of Science and Engineering, and in part by the National Science Foundation under grant CCF-1919167.

References

[1] 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.

[2] M. Conti, S. Orcioni, and C. Turchetti, "Training neural networks to be insensitive to weight random variations," *Neural networks*, vol. 13, no. 1, pp. 125–132, 2000.

[3] 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.

[4] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Proc. Design, Autom., and Test Europe Conf.*, 2017.

[5] S. Zhang, G. L. Zhang, B. Li, H. Li, and U. Schlichtmann, "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *Proc. Design, Autom., and Test Europe Conf.*, 2019, pp. 1751–1756.

[6] 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.

[7] 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.

[8] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

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

[10] 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.

[11] G. A. Seber, *Multivariate observations*. John Wiley & Sons, 2009, vol. 252.

[12] 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.

[13] 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.

[14] 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.

[15] 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.