Rescuing Memristor-based Neuromorphic Design with High Defects

Chenchen Liu, [†]Miao Hu, [†]John Paul Strachan and [§]Hai (Helen) Li

Department of Electrical and Computer Engineering, University of Pittsburgh

[†]Hewlett Packard Laboratories, [§]Department of Electrical and Computer Engineering, Duke University
CHL192@pitt.edu, [†]{miao.hu, john-paul.strachan}@hpe.com, [§]hai.li@duke.edu

ABSTRACT

Memristor-based synaptic network has been widely investigated and applied to neuromorphic computing systems for the fast computation and low design cost. As memristors continue to mature and achieve higher density, bit failures within crossbar arrays can become a critical issue. These can degrade the computation accuracy significantly. In this work, we propose a defect rescuing design to restore the computation accuracy. In our proposed design, significant weights in a specified network are first identified and re-training and remapping algorithms are described. For a two layer neural network with 92.64% classification accuracy on MNIST digit recognition, our evaluation based on real device testing shows that our design can recover almost its full performance when 20% random defects are present.

1. INTRODUCTION

In deep learning networks, the matrix-vector (and matrix-matrix) multiplications are basic operations that determine the overall computation speed, accuracy, and power consumption [1]. Accelerating the execution of matrix-vector multiplication emerges as an important task and extensive studies have been carried out. Revolutionary paradigms on general-purpose platforms, *e.g.*, GPU [2] and CPU [3], and domain-specific hardware like FPGA [4] have been developed. However, the computation efficiency improvement is hindered by the traditional von Neumann architecture, resulting in high hardware cost and energy consumption [5].

The recent rebirth of neuromorphic computing inspires a new solution of implementing neural networks in specialized VLSI designs to overcome the above difficulty. An example is the *TrueNorth* chip released by IBM: a digital spiking neuromorphic system on CMOS technology that adopts SRAM for storing synaptic weights and has demonstrated ultra-low power consumption [6, 7]. Emerging technologies such as spin devices and memristor also create new opportunities to develop neuromorphic systems with high scalability and efficiency [8, 9, 10, 11, 12]. Among these technologies, memristor has been considered as one of the most attractive choices

because the crossbar structure in Figure 1 can be used for matrix-vector computation naturally and efficiently [9, 10].

Predicted by Professor Leon O. Chua in 1971 [13], the memristor was first experimentally identified by researchers in 2008 in a two-terminal metal-oxide device [14]. Recently, memristor-based platforms for matrix-vector multiplication have been developed and demonstrated [11, 12, 15]. The design utilizes the high-density crossbar array illustrated in Figure 1. At the crossing of any horizontal wordline (WL) and vertical bitline (BL) sits a memristor device. When conducting the matrix-vector computation, the $N \times M$ matrix is represented by the memristor conductance of the crossbar array $G_{N \times M}$. The input vector V_I is denoted as a set of analog voltages supplied to the WLs simultaneously. The currents at the BLs are sensed by a trans-impedance amplifier (TIA) and then translated to a set of analog voltages V_O . As such, $V_O = V_I^T \cdot G_{N \times M} \cdot R_s$, where R_s is the sensing resistor in the feedback loop of the TIA.

Memristor-based matrix-vector multiplication platform provides fast computation, high accuracy and low design cost [12]. However, as the development of memristor technology is still maturing, device defects and fabrication yield may be a significant concern. Specifically, the single-bit failure (SBF) denotes a device that freezes in a high conductance state (“stuck-on”) or a low conductance state (“stuck-off”). Although neural networks usually can tolerate a certain number of imperfect synaptic weights, high SBF rate degrades the computation accuracy significantly. For example, we tested a feed-forward neural network for MNIST database: as the SBF rate increases to 20%, the average recognition accuracy rapidly dropped from 92.64% to 39.4%, which is far below an acceptable range. Redundancy schemes have been widely adopted in memory designss [16]. But it is not efficient for the memristor-based analog computations with high precision requirement.

In this work, we propose a defect rescuing methodology that leverages the application-specific features to improve

Figure 1: A memristor-based matrix-vector multiplication platform.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '17, June 18–22, 2017, Austin, TX, USA

© 2017 ACM. ISBN 978-1-4503-4927-7/17/06...\$15.00

DOI: <http://dx.doi.org/10.1145/3061639.3062310>

the hardware efficiency, through the following three steps.

- S1: *Learning weight significance*. We will classify the synaptic weights in a neural network into *significant* and *insignificant* categories based on their impact on the network’s performance. Our preliminary experiment shows the accuracy degradation induced by the classified insignificant weights can be less than 1%.
- S2: A *retraining algorithm* is developed to compensate the SBF caused computation error by re-tuning the trainable weights. Two major constrains in *weight initialization* and *weight updating* are involved in accelerating the retraining process and mimicking the SBF defects in a memristor array. Our experiment on a two-layer network shows that the retraining can recover the classification accuracy to 98.1% at 20% random SBF.
- S3: A *remapping algorithm* that utilizes a redundancy scheme can further improve the computation accuracy, especially when a large number of SBF defects fall in the significant weights category. Only the defects corresponding to the most significant weights will be remapped to the redundancy columns. Our results on the two-layer network show that by remapping only 5% defects of the most significant weights, the recovery rate further increases to 99.3%.

2. OBSERVATION & MOTIVATION

2.1 Random SBF in a Memristor Array

In a neuromorphic application of memristor crossbar arrays, multiple stable conductance states are necessary for each memristor to represent synaptic weights in neural networks. . Reported by HPE Labs recently, at least 64 conductance levels (6 bits) can be successfully programmed in Tantalum Oxide (TaO_x) based crossbar arrays in a one-transistor-one-memristor (1T1M) design [17]. Figure 2(a) shows a 44×44 pattern in a 64×64 1T1M array. It can be seen that the SBF defects distribute randomly across the array and blur the programmed pattern. In fabrication, memristor arrays demonstrate very different defect patterns and yields. Among all the measured arrays, this example has the lowest yield of 84%. However, across the full lifetime of a memristor array used for neural network inference and training, the memristor cells can be heavily exhausted and damaged by aggressive programming and testing cycles.

Figure 2(b) shows the stuck-on/off SBF distribution of the 44×44 sub-region. For ease of illustration, stuck-on defects are displayed as +1 and stuck-offs are presented as -1. It is worth pointing out that a defect is not fixed to the highest conductance g_H or the lowest value g_L . As our

Figure 2: (a) The conductance distribution (in μS) and (b) the measured stuck-on (+1) and stuck-off (-1) defects of a 64×64 memristor crossbar array.

design utilizes the full analog conductance range of memristor, a device is taken as defect when its g_{error} is out of tolerance after programming, where $g_{error} = g_{final} - g_{target}$ denotes the conductance difference between the real programmed and the target values. In this work, a device with $g_{error} > 30\mu S$ is considered as a stuck-on. The measurement shows that the conductance of stuck-on devices ranges from $330\mu S$ to $1200\mu S$. The devices with $g_{error} < -30\mu S$ are defined as stuck-offs, most of which demonstrate less than $1\mu S$ conductance value. For the example in Figure 2(b), 18.4% of defects are stuck-off and the remaining are stuck-on.

The measurement data depicts a random distribution of the SBF defects. When utilizing a memristor-based functional unit for data storage, the data at defective cells can still be read and corrected if necessary. Traditional methods such as redundancy and error correcting codes (ECC) can effectively solve these issues for storage. For in-memory computations, however, the situation is more complex and errors in the conductances aggregate in a non-trivial manner, requiring new correction schemes.

2.2 Impact of SBF on Neuromorphic Systems

We take a two-layer neural network for MNIST classification as the example to evaluate the impact of random SBF defects on the memristor-based neuromorphic design. Figure 3(a) depicts the network model which consists of an input layer, a fully-connected synaptic weight matrix W and an output layer. The well-tuned floating-point values of the synaptic weight matrix W_I will be mapped to the analog conductances that a memristor array can afford (e.g., 64 levels). Besides the slight precision loss during the mapping, unpredictable defects shall be considered. So we generate defect matrices W_D randomly and apply them to test the impact of SBF on the system performance.

Figure 3(b) shows the classification performance statistics when 5%~30% stuck on/off SBF are injected in a 784×10 memristor-based array. For each setup, the impact of defects is evaluated on 1,000 test cases that are generated by randomly spreading defects. Without defects, the network achieves a classification accuracy of 92.64%. The normalized accuracy rate defined as Acc_{real}/Acc_{ideal} in Figure 3(b) demonstrates the significant performance degradation due to defects: adding 10% SBF defects results in an average normalized classification accuracy rate of 59.7%, that is, 55.3% real classification accuracy for MNIST dataset.

Our observation of the large performance degradation here is different from previous research conclusion that comparable accuracy can be retained with weights penalties in deep neural networks (DNNs) [18, 19]. In the methods such as

Figure 3: (a) An illustration of synaptic weight defects; (b) the impact of SBF on the classification accuracy of a two-layer neural network.

L1/L2 regularization and dropout, only those weights with near-to-zero parameters are penalized as they are less important. For example, the near-to-zero synaptic weights could be removed through network sparsification to realize recognition systems with high computation accuracy and efficiency [20, 21]. The SBF defects, however, can occur anywhere in a memristor array. Many of them could greatly affect the network performance. Moreover, the sparsification method completely removes some connections by forcing the corresponding matrix entry to 0. A defect could still contribute certain error to the computation result.

3. DESIGN METHODOLOGY

We propose a *defect rescuing design methodology* to improve the efficiency of memristor-based neuromorphic system. The impact of synaptic weights on system performance first is quantified and analyzed. We then develop a retraining algorithm in network learning and a remapping scheme in hardware implementation.

3.1 Weight Significance

We first quantify the impact of each weight on system performance, denoted as *weight significance*. Back-propagation is the key step in network training, which updates synaptic weights according to the errors received at neurons. A gradient decent algorithm is usually adopted in which the weight updating process can be formulated as [22]:

$$\Delta w_{j,i} = -\eta \frac{\partial E}{\partial w_{j,i}} = \eta f' \sum_{k \in \text{downstream}(j)} \left(w_{k,j} \frac{\partial E}{\partial w_{k,j}} \right) x_{j,i}. \quad (1)$$

Where E represents the global error, η is the learning rate, f' denotes the deviation of the activation function of the hidden layer, $w_{j,i}$ is the weight associated with the i th input to neuron j , and $\Delta w_{j,i}$ is the updated weight computed by propagating the error back from the downstream units j .

During the learning, each weight demonstrates different sensitivity to E , resulting in different Δw . Therefore, we can analyze the sensitivity of $w_{j,i}$ through network training, i.e., $\partial E / \partial w_{j,i}$, to classify the weight significance. The weight significance can also be ascertained directly from experiments by inserting defects in a well-tuned network, e.g., the two-layer neural network for MNIST classification in this work. When inputting an image to the network, we insert a single-bit defect and compare the real and the ideal accuracy rates Acc_{real}/Acc_{ideal} .

Figure 4(a) shows the statistical result of the network's first layer, the trend of which is consistent to the normalized sensitivity $\partial E / \partial w_{j,i}$ in Figure 4(b). We divide all the weights into *significant* and *insignificant* groups based on the value of $\partial E / \partial w_{j,i}$ and re-characterize their impacts at the system level. Figure 4(c) shows the accuracy degradation that induced respectively by the significant and insignificant weights when 10% SBF is considered. Here, 55% weights are characterized to be insignificant with a threshold t of $\partial E / \partial w_{j,i}$. 1,000 random test cases are utilized and results in three scenarios are depicted in the Figure 4(c): *best*, *worst*, and *average* correspond to the highest accuracy, the lowest accuracy, and the mean of the highest and lowest values, respectively. As expected, defects on significant weights dramatically affect the system performance. Oppositely, insignificant weights are more tolerable to defects. It indicates that defects in some of the classified insignificant weights can be tolerated by network itself and induce negligible impact in accuracy degradation.

Figure 4: (a) Acc_{real}/Acc_{ideal} from experiments; (b) $\partial E / \partial w_{j,i}$ during training; (c) Accuracy degradation due to defects on significant and insignificant cells.

3.2 Network Retraining

Retraining is usually carried out to optimize the recognition accuracy after the a sparsification step of DNNs [20]. In this work, a hardware specific retraining methodology is developed to recover the accuracy loss induced by SBF defects.

Through a normal training process for a neural network, parameters are learned and the weight matrix is generated for a given application. Memristor cells in the crossbar array will be programmed to the corresponding conductance values, determined by the mapping algorithm in the memristor-based neuromorphic system. Some memristor cells will then be identified as stuck at certain conductance levels and not adjustable. Therefore, incorrect outputs are generated from the stuck weights. Our proposed retraining methodology attempts to recover the accuracy by re-tuning all remaining weights that are adjustable. Figure 5 illustrates a simple recovery model that is assisted with other trainable weights. The network retraining includes two major steps:

- *Weight initialization*: Instead of assigning random values to synaptic matrices, the pre-trained weight matrix W_{Ideal} will be used to initialize and accelerate the re-training process. The defect map obtained from chip testing will be applied, providing the initial values to the defected cells.
- *Weight updating*: Backpropagation is adopted to up-

Figure 5: The illustration of the weight distributions before and after retraining.

date the synaptic weights. In the retraining, $\Delta w_{j,i}$ at a defected location will be forced to be 0 so $w_{j,i}$ remains unchanged in iterations.

In this way, the behavior of the memristor array with some non-adjustable weights is mimicked in the retraining. Weights with defects maintain constant values according to the measured mapping. Accuracy is then recovered by updating the trainable weights.

3.3 Defect Rescuing Design Flow

Based on the weight significance analysis and the network retraining, a defect rescuing flow presented in Figure 6 was developed for the memristor-based neuromorphic systems.

First, a pre-training procedure is executed. This targets an initial weight matrix W_{Ideal} through normal neural network training consisting of forward and back-propagation. W_{Ideal} will then be mapped to a conductance matrix implemented on a memristor array as described in [12]:

$$G = \alpha \cdot W + \beta, \quad (2)$$

where, α and β are two linear mapping coefficients. These follow the relationship $\alpha = (g_H - g_L)/(w_H - w_L)$ and $\beta = g_H - \alpha \cdot w_H$. $[g_L, g_H]$ is a selected conductance range for a linear computation in matrix-vector calculations. w_L and w_H are the minimum and maximum synaptic weight values in the well-tuned W_{Ideal} . In this way, a conductance within the range is normalized to a weight in $[-1, +1]$. Through array testing, defect information including stuck-on and -off conductances and their locations is obtained, forming a defect conductance matrix G_{defect} . Again, by following Eq. (2), G_{defect} will be translated to matrix W_{defect} . The recognition accuracy will be retested after considering W_{defect} .

In accuracy checking, it is possible that only defects occur at insignificant weights. As shown in Figure 4(c), this results in negligible accuracy loss. In such a case, the retraining procedure may be omitted. Of most interest is when a certain accuracy degradation is caused by SBF defects. The proposed retraining is executed to rescue the accuracy loss by generating a new weight matrix, including the existing SBF defects. Then, the neural network is tested again with the new weight matrix. If the accuracy is recovered successfully after the retraining, the synaptic weights are finalized.

In the worst-case scenario when there are too many defects or the performance loss cannot be compensated by retrain-

ing, a remapping algorithm utilizing redundant memristor columns will be used. This additionally requires peripheral circuits such as the TIA and sample-and-hold blocks to support re-routing and operation from the redundant columns. In contrast to redundancy in the memory domain, only a small portion with the most significant defect weights will be mapped to redundancy columns in neuromorphic computing systems.

4. EVALUATIONS

In this work, the efficiency of the defect rescuing design is evaluated on feed-forward neural networks for MNIST handwritten digits classification [23]. Two networks with two-layer and three-layer structures are trained and tested. Here, 60,000 digital patterns from MNIST are used for training, and a test set of 10,000 examples are selected randomly.

One weight matrix W is included in the two-layer network with the array size of 784×10 . The implementation of the three-layer neural network utilizes $W1$ with the array size of 784×256 between the input layer and the hidden layer, and $W2$ with the array size of 256×10 between the hidden layer and the output layer. The two networks obtain 92.64% and 97.82% classification accuracy at the software level, respectively.

Through neural network training, the synaptic weights of any array, i.e., W_{Ideal} , can be obtained and mapped to the conductances of memristor arrays, i.e., G_{Ideal} . The memristor normal operation range $[g_L, g_H]$ is set to be $[1\mu S, 300\mu S]$ to guarantee a linear matrix-vector computation based on the measurement data. The stuck-on and -off conductance ranges are respectively considered to be $[300\mu S, 1200\mu S]$ and $[0.01\mu S, 1\mu S]$ based on the measured worst-case conductances. As mentioned in Section 3.3, a defect weight matrix W_{defect} can be obtained from the measured G_{defect} . Based on this information, the impact of the SBF defects are evaluated in the two feedforward neural networks and our described defect rescuing flow is adopted to restore the accuracy.

4.1 Robustness of Retraining

Figure 7 summarizes the normalized accuracy defined as Acc_{real}/Acc_{ideal} , the ratio of the retrained accuracy with SBF defects compared with the ideal accuracy without defects. The results before and after applying the proposed retraining process on the two-layer and three-layer feedforward neural networks are presented. Under each condition, the result is obtained from 1,000 test cases of defects at random locations, stuck mode, and conductance values.

In the two-layer neural network, large accuracy degradation can be observed due to the random stuck-on/off SBF. The degradation increases with number of defects as shown in Figure 7(a). For example, only 42.5% accuracy can be achieved on average when considering 20% defects. The corresponding lowest and highest accuracy rates in the 1,000 tests are 21.2% and 63.7%, respectively. The reason for such a large variation between the best and worst cases is the highly varying significance of the defect weights, as previously discussed in Section 3.1. The retraining result for the same network is presented in Figure 7(b). This shows that the worst-case accuracy of 21.2% is recovered to 97.9%. On average, the accuracy can be rescued to 98.8% and 98.1%, considering 10% and 20% random SBF defects respectively. Moreover, the variations become much smaller after applying our retraining—less than 0.4%.

Figure 6: The proposed defect-rescuing neuromorphic design flow.

Figure 7: The impact of random stuck-on/off defects (a) and the recovered accuracy after retraining (b) in the two-layer neural network; The impact of random stuck-on/off defects to $W1$ and $W2$ (c) and the recovered accuracy after retraining (d) in the three-layer neural network.

Figures 7(c) and (d) show the impact of the SBF on the three-layer network before and after retraining in three scenarios, assuming a certain percentage of defects only in $W1$, only in $W2$, or in both $W1$ and $W2$. Similarly, 1,000 random cases are tested and the average accuracy is presented. Figure 7(c) indicates that $W1$, i.e., the weight matrix between the input layer and the hidden layer is more sensitive to defects than $W2$. This is because $W1$ is designed to learn image features so as to have more severe impact [22]. As expected, the largest accuracy degradation happens when both $W1$ and $W2$ have defects. At a defect rate of 20%, the network shows only 10% classification accuracy. Utilizing the proposed retraining can recover the accuracy to 94.5% from the most destroyed network. We also observe that $W2$ has higher resilience, with a retained accuracy of 99.6% compared to the ideal result without defects. The rescuing ability of retraining is dominated by the more significant weight matrix $W1$. The results demonstrate that the retraining is robust and efficient in rescuing the accuracy loss caused by the random SBF.

Figure 8 shows the conductance distributions of the two-layer network. Specifically, Figure 8(a) is the conductance distribution without defects, and Figure 8(b,c) show the weight distributions after retraining with 10% stuck-on or stuck-off defects. By comparing the distributions, we note that the trainable weights are re-tuned to compensate for the error caused by defects, consistent with our approach described in Section 3.2. The values of trainable weights shift toward the larger side when inserting stuck-on defects while it shifting to smaller when including stuck-off defects. In contrast to other work [24], our approach is able to accommodate both types of defects simultaneously with no observed challenges compared to a single type of defect.

Figure 8: (a) The distribution of G_{Ideal} ; (b) The conductance distribution with stuck-on defects after retraining; (c) The conductance distribution with stuck-off defects after retraining.

4.2 Resilience of (In)significant Weights

Synaptic weights in a neural network can be characterized to be significant or insignificant, yielding different sensitivity to defects. Figure 9(a) shows that insignificant weights have lower accuracy degradation. For example, when 20% defects all fall at significant or insignificant weights, the test on significant weights shows 37.5% more degradation in accuracy. Here, 55% weights are classified to be insignificant when taking the two-layer network as the example.

In both cases, retraining leads to improved accuracy. But Figure 9(b) shows that the insignificant weights have better resilience to defects. By retraining as described here, the normalized accuracy can be recovered to 99.9% even with 30% defects in the insignificant weights. Correspondingly, the normalized accuracy can be recovered to 95.1% when 30% defects happen at the significant weights.

4.3 Redundancy Memristor Design

The above evaluations prove that most of the accuracy, e.g., 98.1%, can be recovered by the retraining algorithm even with 20% SBF defects. In this work, a remapping process assisted with an efficient redundancy design is also developed to address the case where SBF defects arise at many significant weight locations that cannot be fully recovered by retraining. Figure 10(a) shows the illustration of the redundancy scheme: columns that are heavily polluted by defects will be replaced by additional memristor columns with a remapping algorithm. And outputs resulting from the new columns will be selected and utilized for the next step computation.

The results in Figure 9(b) prove that the defects with low significance have better resilience on retraining. Therefore,

Figure 9: (a) The accuracy degradation in the significant and insignificant weights; (b) The ability to recover accuracy for defects at significant and insignificant weights through retraining.

Figure 10: (a) A simple redundancy scheme; (b) Recovered accuracy with significant defects remapping at 20% SBF defects.

only the most significant defects weights are considered to be remapped to redundancy columns to decrease the design cost while improving the accuracy efficiently. Figure 10(b) shows the results when 0% ~ 5% of the most significant defects are remapped to the redundancy columns that without significant defects at 20% SBF defects, again taking the two-layer network as the example. The results show that 99.3% accuracy can be obtained with 5% significant defects being remapped, increasing from the 98.1% that was restored by retraining only. It is also observed that the accuracy improvement flattens out going from 4% to 5%, as the defects remapped are increasingly less significant. Hence, our proposed redundancy scheme is able to improve accuracy with minimal redundancy and design cost.

5. CONCLUSIONS

The computation accuracy for memristor-based neuromorphic systems can be degraded significantly by random defects across memristor arrays. In this work, we proposed a defect rescuing design to effectively restore the accuracy. The proposed design has three major aspects including a weight significance categorization, a robust retraining algorithm, and an efficient remapping process. Basing on experimental device testing data in memristor arrays, the rescuing ability of our proposed design was evaluated in feed-forward neural networks for MNIST digit recognition. Considering 20% random single-bit defects, our proposed retraining process recovered the recognition accuracy to 98.1% and 94.5% from 42.5% and 10% in the two-layer and three-layer feed-forward networks, respectively. Additionally combining this with a remapping process, 99.3% accuracy can be achieved overall by remapping only 5% of the most significant defects to redundant columns in the two-layer network having 20% defects.

6. ACKNOWLEDGMENTS

This work is supported in part by AFRL FA8750-15-2-0048, NSF 1337198, and IARPA via ODNI Contract number 2014-14080800008. Any Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL, NSF, IARPA, or its contractors.

7. REFERENCES

- [1] P. Simon, *Too Big to Ignore: The Business Case for Big Data*. Wiley, 2013.
- [2] D. C. Ciresan *et al.*, "Convolutional Neural Network Committees for Handwritten Character Classification," in

- 2011 International Conference on Document Analysis and Recognition, pp. 1135–1139, Sept 2011.
- [3] V. Vanhoucke *et al.*, "Improving the speed of neural networks on CPUs," in *Deep Learning and Unsupervised Feature Learning Workshop*, 2011.
- [4] C. Farabet *et al.*, *Large-Scale FPGA-based Convolutional Networks*. Cambridge university Press, 2011.
- [5] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.
- [6] P. Merolla *et al.*, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *CICC*, pp. 1–4, 2011.
- [7] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
- [8] M. Sharad *et al.*, "Proposal For Neuromorphic Hardware Using Spin Devices," in *Disordered Systems and Neural Networks*, 2012.
- [9] C. Liu *et al.*, "A Spiking Neuromorphic Design with Resistive Crossbar," in *DAC*, pp. 1–6, July 2015.
- [10] C. Liu *et al.*, "A Memristor Crossbar Based Computing Engine Optimized for High Speed and Accuracy," in *ISVLSI*, July 2016.
- [11] M. Prezioso *et al.*, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, pp. 61–64, 2015.
- [12] M. Hu *et al.*, "Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication," in *DAC*, pp. 1–6, 2016.
- [13] L. O. Chua, "Memristor-the missing circuit element," in *IEEE Transactions on Circuit Theory*, vol. 18, pp. 507–519, September 1971.
- [14] D. B. Strukov *et al.*, "The Missing Memristor Found," *Nature*, vol. 453, no. 12, pp. 28–35, 2008.
- [15] F. M. Bayat, M. Prezioso, B. Chakrabarti, I. Kataeva, and D. Strukov, "Advancing memristive analog neuromorphic networks: Increasing complexity, and coping with imperfect hardware components," *arXiv preprint arXiv:1611.04465*, 2016.
- [16] S. Kannan *et al.*, "Detection, Diagnosis, and Repair of Faults in Memristor-based Memories," in *VTS*, pp. 1–6, April 2014.
- [17] E. J. Merced-Grafals *et al.*, "Repeatable, Accurate, and High Speed Multi-level Programming of Memristor 1T1R Arrays for Power Efficient Analog Computing Applications," *Nanotechnology*, vol. 27, no. 36, p. 365202, 2016.
- [18] N. Srivastava *et al.*, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
- [19] S. Han *et al.*, "Learning both Weights and Connections for Efficient Neural Network," in *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.
- [20] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *ISCA*, pp. 243–254, 2016.
- [21] S. Han *et al.*, "DSD: Regularizing Deep Neural Networks with Dense-Sparse-Dense Training Flow," *arXiv preprint arXiv:1607.04381*, 2016.
- [22] T. M. Mitchell, *Machine Learning*. McGraw-Hill Science, 1997.
- [23] Y. LeCun *et al.*, "The MNIST database of handwritten digits." <http://yann.lecun.com/exdb/mnist/>.
- [24] I. Kataeva, F. Merrikh-Bayat, E. Zamanidoost, and D. Strukov, "Efficient training algorithms for neural networks based on memristive crossbar circuits," in *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–8, IEEE, 2015.