Memristive Stochastic Computing for Deep Learning Parameter Optimization

Corey Lammie, *Student Member, IEEE*, Jason K. Eshraghian, *Member, IEEE*, Wei D. Lu, *Fellow, IEEE*, and Mostafa Rahimi Azghadi, *Senior Member, IEEE*

***Abstract*—Stochastic Computing (SC) is a computing paradigm that allows for the low-cost and low-power computation of various arithmetic operations using stochastic bit streams and digital logic. In contrast to conventional representation schemes used within the binary domain, the sequence of bit streams in the stochastic domain is inconsequential, and computation is usually non-deterministic. In this brief, we exploit the stochasticity during switching of probabilistic Conductive Bridging RAM (CBRAM) devices to efficiently generate stochastic bit streams in order to perform Deep Learning (DL) parameter optimization, reducing the size of Multiply and Accumulate (MAC) units by 5 orders of magnitude. We demonstrate that in using a 40-nm Complementary Metal Oxide Semiconductor (CMOS) process our scalable architecture occupies 1.55mm^2 and consumes approximately 167μW when optimizing parameters of a Convolutional Neural Network (CNN) while it is being trained for a character recognition task, observing no notable reduction in accuracy post-training.**

***Index Terms*—Memristors, Stochastic Switching, Stochastic Computing, Deep Learning**

I. Introduction

Embedded Resistive Random Access Memory (RRAM)-based neuromorphic and DL accelerators have attracted significant attention due to their promise to revolutionize computing [1]. Such devices are capable of in-memory computation, and can be used to perform near-sensor high-speed and low-power computation at the Internet of Things (IoT) edge [2].

Current research efforts are primarily focused towards the realization of scalable and reliable memristive architectures for in-memory computing applications [1], [3]. For example, memristive crossbar architectures can be used to efficiently implement MAC or dot-product accelerators to perform 2D Vector-Matrix Multiplications (VMMs), which are prominent in both neuromorphic and DL systems, in $\mathcal{O}(1)$ [4]. Resistive memories, however, are still considered an emerging technology [5] that are prone to device-to-device variability, endurance challenges, and stochastic behavior [6], which make reaching the maximum gain of RRAM technology, currently unfeasible.

While the commercial long-term viability of using RRAM devices for such purposes is yet to be properly determined [7], rather than treating the non-ideal stochastic behavior of RRAM devices as a hindrance, some researchers have sought to exploit the well-characterized stochasticity of RRAM devices for alternative applications such as chaos [8], [9] and random number generation [10], [11]. Traditionally, large hardware costs associated with stochastic number generation have hindered stochastic processors, as they require large bit stream lengths to mitigate undesirable computational errors [12], rendering them largely ineffective for most applications. In [13], a hybrid CMOS-memristor stochastic processor was proposed, which used digital CBRAM devices for efficient stochastic number generation. It was demonstrated that when using large bit-stream lengths, the processor was able to perform gradient descent optimization and k-means clustering in a low-power and high-speed mode of operation.

In this brief, we expand upon efforts in [13], and exploit the well characterized switching stochasticity of probabilistic CBRAM devices to efficiently generate stochastic bit streams in order to perform deep learning parameter optimization using a hybrid CMOS-memristor stochastic processor. By nature, such a processor is highly tolerant to external noise and relaxes many of the stringent hardware requirements needed in generating distinct voltage levels (Fig. 1). While most prior DL-based SC research applies SC to the feed-forward processing (inference) stage, as it is known to be tolerant to noise, we instead explore if it is at all possible to

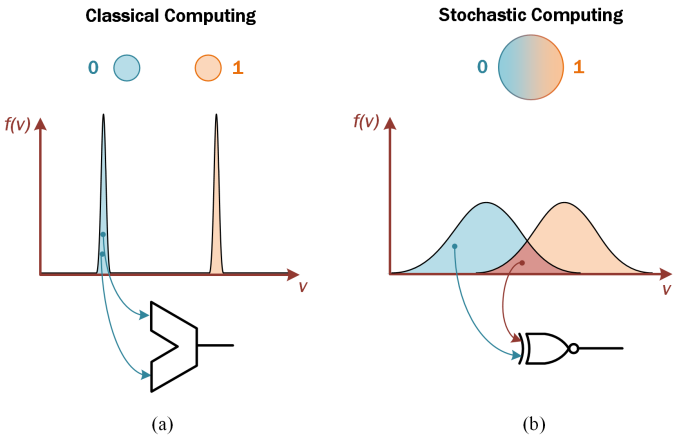

Fig. 1. Conceptual difference between classical and stochastic computing in terms of probability density functions. (a) Classical or deterministic computing requires well-defined margins. (b) Stochastic computing is error-tolerant, and can thus exploit quantum models of uncertainty that occur at the particle level. This simplifies the hardware required for downstream processing. In our case, a large arithmetic logic unit containing a multiplier can be replaced by a single logic gate.

Corey Lammie and M. Rahimi Azghadi are with the College of Science and Engineering, James Cook University, Australia (e-mail: {corey.lammie, mostafa.rahimiazghadi}@jcu.edu.au).

Jason K. Eshraghian and Wei D. Lu are with the Department of Electrical Engineering and Computer Science, University of Michigan, USA (email: {jasonesh, wluee}@umich.edu).

perform parameter optimization, that typically requires high precision, using probabilistic bits. Our specific contributions are as follows:

1) We are the first to exploit the switching stochasticity of CBRAM devices to perform deep learning parameter optimization using SC;
2) We evaluate our architecture by training a Deep Neural Network (DNN) for the MNIST character recognition task using Mini-Batch Stochastic Gradient Descent (SGD) and Mini-Batch SGD with Momentum;
3) We investigate the tradeoff between latency, area and power consumption, and demonstrate that our architecture can be used to reduce the size of MAC units.

II. Preliminaries

A. Stochastic Computing

In SC, numbers are represented using bit streams. The frequency of 0s and 1s in stochastic bit streams determines their value depending on a range, R, denoted as the priori. The priori can be unipolar, i.e., $R \in [0,1]$ to represent unsigned operands, or bipolar to represent signed operands, i.e., $R \in [-1,1]$. The value represented by unipolar bit streams can be determined using the mean, x, whereas the value represented by bipolar bit streams can be determined using $2x-1$. Conventional SC circuits use Linear Feedback Shift Registers (LFSRs) to generate pseudo-random numbers for stochastic bit stream generation, and popcount and up-down counter circuits to decode bit streams. SC blocks for bit stream generation, addition, multiplication, and stochastic bit stream to binary converters are depicted in Fig. 2.

B. Memristor-based Native Stochastic Computing

Digital memristor devices have a large dynamic range between logic levels that is associated with the formation and rupture of a dominant, nanoscale conducting filament [14]. On account of device-device variation, experimental studies have demonstrated that the switching time between logic levels is stochastic, and that the time-to-switch can be accurately modelled using a Poisson distribution [15], [16], as depicted in Fig. 3. By exploiting this property, both unipolar and bipolar stochastic bit streams can be efficiently generated using digital memristors by applying programming pulses with variable pulse widths to memristor cells. Memristor-based native SC systems can be realized by using memristor cells for stochastic bit stream generation, and using CMOS for stochastic arithmetic circuits.

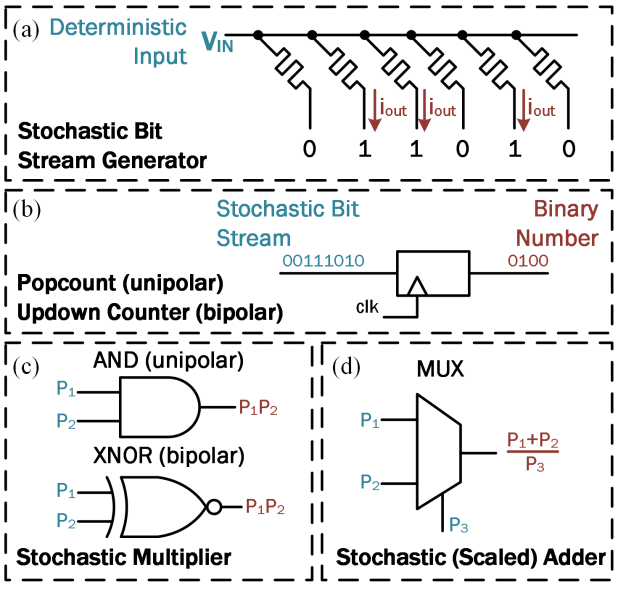

Fig. 2. Stochastic computing blocks for (a) bit stream generation, (b) stochastic bit stream to binary converter, (c) multiplication, and (d) scaled addition.

C. Deep Learning Parameter Optimization

SGD [17] is a parameter optimization method that is described by (1), which is commonly used to train DNNs.

$$\theta_n = \theta_{n-1} - \eta \nabla_\theta J(\theta, y_i', y_i), \tag{1}$$

where, η is the learning rate, θ denotes a trainable parameter, y_i represents the class label, y_i' denotes the predicted class label, and $J(\theta, y_i', y_i)$ describes the objective function. Momentum [18], or SGD with momentum, described by (2), is a method that improves SGD and accelerates gradients in relevant directions towards a local minima by dampening oscillations using γ and v, i.e., momentum and velocity parameters.

$$\begin{aligned} v_n &= \gamma v_{n-1} + \eta \nabla_\theta J(\theta, y_i', y_i), \\ \theta_n &= \theta_{n-1} - v_n \end{aligned} \tag{2}$$

Mini-Batch SGD and Mini-Batch Momentum are variations of the SGD and Momentum optimization algorithms which split training datasets into small batches that are used to perform parameter optimization. From this stage forward, we implicitly refer to mini-batch variations.

III. Proposed Architecture

A block diagram of our proposed architecture is depicted in Fig. 4. We confine operating in the stochastic domain to

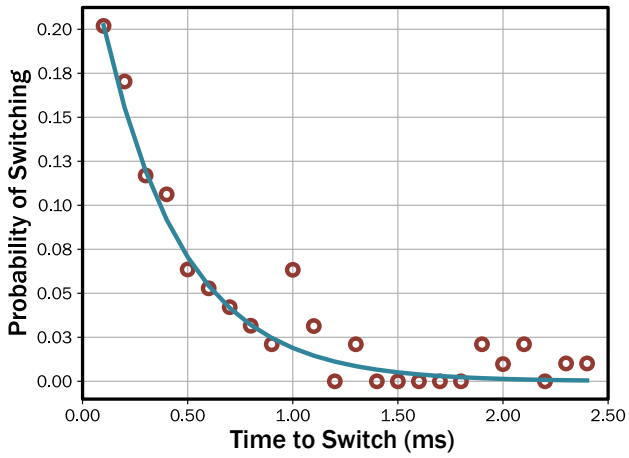

Fig. 3. Distribution of switching time of a fabricated CBRAM device [14] under an applied voltage of 4.5 V. Switching events (red circles) fit a Poisson distribution (blue line), $P(t) = (\Delta t/\tau)e^{-t/\tau}$ for Δt = 0.5, $V = 0.4$ and $\tau = 0.38$ms.

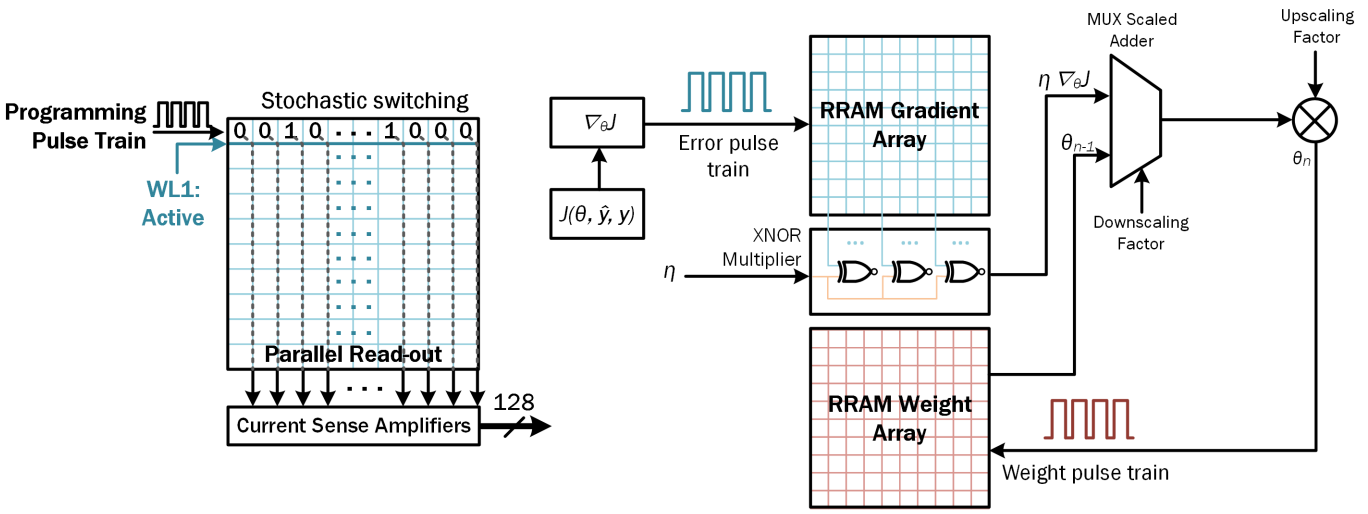

Fig. 4. Native stochastic parameter update process. (a) The word line (WL) is activated to open one row at a time. A pulse train is applied to induce stochastic switching in the RRAM. The current sense amplifiers are used to sense which devices are drawing current, indicating which cells have been switched on. (b) The parallel-generated bit stream is XNOR multiplied by the negative learning rate and added to the previous weights to perform a single parameter update. The overhead of requiring an additional array to compute the weight bit stream is partially offset by removing the need for ADCs.

the parameter update stage during training. The architecture consists of scalable crossbar tiles of probabilistic CBRAM devices (RRAM input and weight arrays) and CMOS systolic stochastic arithmetic circuits (multipliers and scaled adders). Stochastic bit streams are generated by writing pulse trains to columns of 2D crossbar architectures. Given an applied voltage, V, and a programming pulse width, t, the switching probability can be expressed using (3)

$$P(t, V) = 1 - e^{(-te^{V/V_0})/\tau_0}, \tag{3}$$

where V_0 and τ_0 are fitting parameters [13]. By fixing the applied voltage and varying the programming pulse width applied to each column, bipolar bit streams can be efficiently generated in-memory. The bits are read out from the array in parallel using current sense amplifiers (Fig. 4(a)).

IV. Training and Validation Methodologies

To evaluate our architecture, we trained a CNN described in Table I using MNIST. All convolutional, and the first fully connected layer were sequenced with batch normalization layers, and the ReLU activation function was used for all layers. All networks were trained for 10 epochs with a batch size of 256 and a fixed learning rate. Gradients were clipped between -1.0 and 1.0. Cross entropy loss was used in conjunction with SGD and SGD with momentum. For all networks trained using momentum, Nesterov was disabled, and a fixed momentum value of 0.9 was used, which has demonstrated significant performance for a variety of DL tasks [18].

TABLE I
Adopted Network Architecture[1]

Layer	Output Shape
Convolutional, $f = 10, k = 5, s = 1, p = 0$	$(10 \times 24 \times 24)$
Max Pooling, $k = 2, p = 2$	$(10 \times 12 \times 12)$
Convolutional, $f = 20, k = 5, s = 1, p = 0$	$(20 \times 8 \times 8)$
Max Pooling, $k = 2, p = 2$	$(20 \times 4 \times 4)$
Fully Connected, $N = 50$	(50)
Fully Connected, $N = 10$	(10)

V. Results

A. Performance

The MNIST test set accuracy for all training epochs, and the MNIST test set accuracy and training loss for all mini-batches during the first training epoch are reported in Fig. 5, for network architectures trained using our native SC-based parameter optimization with SGD and SGD with momentum. These are also shown for a baseline implementation using conventional training. The learning rate was varied from 0.01 – 0.5 and the stochastic bit stream length (N_{bit}) was varied from 2-Kbits – 64-Kbits, near the range investigated in [13]. Each training instance was repeated 10 times, and the mean and standard deviation across instances were determined. From Fig. 5, it can be observed that for both SGD and SGD with momentum a bit stream length larger than 8-Kbit and a learning rate of $\geq$0.1 is required for stable training performance. We attribute the performance degradation when smaller learning rates are used to the fact that positive values near zero can be encoded negatively in the stochastic domain, and note that such implementations may eventually converge if trained for $\gg$10 epochs.

B. Power and Area Requirements

Power and area estimates were calculated based on a 40 nm CMOS process integrated with RRAM in the back end of

[1]For each convolutional and pooling layer, f denotes the number of filters, k determines the filter size, s is the stride length, and p denotes the padding. N is the number of output neurons for each fully connected layer.

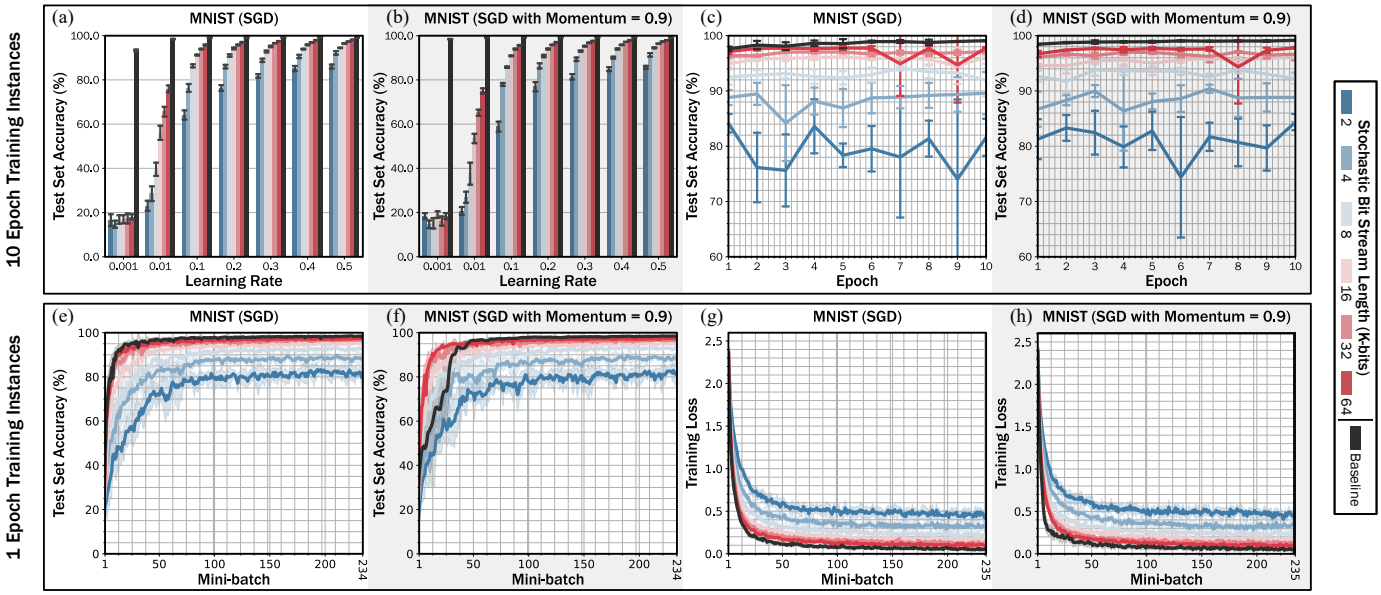

Fig. 5. The MNIST test set accuracy across seperate training instances of 10 epochs (a–d) and 234 mini-batches (1 epoch) (e–h), respectively, for CNNs trained using our native SC-based parameter optimization with different learning rates for (a) SGD, and (b) SGD with momentum; for a fixed learning rate with (c) SGD, and (d) SGD with momentum; for a fixed learning rate with (e) SGD, and (f) SGD with momentum. The training loss during the first training epoch for (g) SGD, and (h) SGD with momentum. In (c–h) a learning rate of 0.5 was used. Each baseline implementation uses 32-bit floating point weights.

the line [19]. Specifically, a pre-configured IP is used for the encoder and decoder, and a full-custom approach was taken to laying out the RRAM array, sense-amplifiers, registers, MUX-based scaled adder, and XNOR accumulator, with verified DRC/LVS compliance. Power draw is modified to suit the device distribution in Fig. 3. The area of each 128×128 RRAM tile is $2.77\times10^3\mu\text{m}^2$. Assuming that the entire bit stream must be accessible at once, and that only one row may be read out at a time, a bit stream length of 16-Kbits requires 2^7 tiles. The gradient bit stream is multiplied by a learning rate bit stream of equivalent length and subtracted from the weight. To improve throughput, the number of arrays is doubled to allocate half of the tiles to be reset while the other half are generating bit streams. This gives a total of 2^9=512 RRAM tiles, occupying a total area of 1.42mm^2.

Static power consumption is estimated based on the pre-sumption that half of the tiles are generating bit-streams while the other half are being reset, one row at a time. Balancing the latency between reading and resetting is feasible because reading requires stochastic programming. As a conservative estimate, we assume an input voltage of 4.5V, which is tolerable at 40nm using Laterally-Diffused Metal Oxide Semiconductor (LDMOS) transistors, that draw approximately 10nA from a device that is on, and 100pA from a device that is off [20]. The corresponding static power dissipation is 45nW and 0.45nW, respectively. The power consumed for one read out can be measured by (4):

$$P_{\text{read}} = N_{\text{bit}}[P_{\text{on}}\mathbf{E}(\nabla_\theta\mathbf{J}) + P_{\text{off}}(1-\mathbf{E}(\nabla_\theta\mathbf{J}))], \qquad (4)$$

where N_{bit} is the bit stream length, P_{on} and P_{off} are the power dissipated from an on and off device, and $\mathbf{E}(\nabla_\theta\mathbf{J})$ is the expected value of the gradient $\nabla_\theta J(\theta, y_i', y_i)$. We measured $\mathbf{E}(\nabla_\theta\mathbf{J})$=0.5367 at the start of the training process. Gradient updates are generally larger at the start of training than at convergence. Therefore, more cells are switched on for steeper gradients. This means that worst-case power consumption should be measured at the start of training. For normalized bipolar weights, $\mathbf{E}(\theta)$=0.5. Power dissipation from the RRAM array is estimated to be 43.0 μW per gradient, and 40.6 μW per weight. These estimates are doubled to account for additional tiles used for resetting and enhancing throughput, giving a total result of 167 μW.

The layout of the peripheral CMOS circuitry was used to generate accurate area and power estimates. The area of a single XNOR gate is 670nm×355nm. The pitch of the RRAM array is approximately 410nm. The XNOR multiplier can be oriented such that it is pitch-matched to the RRAM columns. It can then fit under the bottom row without additional area overhead. The total area occupation of the XNOR gate of 0.031mm^2 can thus be merged with the RRAM area. The total area overhead from accumulation consists of the MUX-based scaled adder and a register which consume 0.0735mm^2. Each sense-amplifier built from a pair of cross-coupled inverters occupy $0.41\mu\text{m}^2$. Although the shorter dimension can be optimized to be pitch-matched to the array, a reference signal is required to distinguish on and off cells. This reference must be obtained from an adjacent column, which requires time multiplexing the bit stream generation process into two steps. With pairwise column sharing, the total area of all current sense amplifiers is 0.0267mm^2. Time multiplexing also halves the maximum static power dissipation to 83.5 μW. Total static power dissipation of the CMOS elements is from subthreshold current draw, and thus dominated by resistive dissipation.

VI. Discussion

The length of the parallel-generated bit stream may cause large power draw as throughput is increased. But this drawback

is offset by three factors. Firstly, the MAC process of non-quantized weights can now be completed in one single step using an XNOR gate and a multiplexer (with a simulated post-layout delay of 58 ps). A 16-bit MAC unit designed in a comparable process [21] is approximately five orders of magnitude larger and two orders slower than an XNOR gate and a MUX. Secondly, the power estimates are derived from the peak value at the end of bit stream generation. All devices are initialized to be off and only switch on during the course of the time to switch (Fig. 3). This peak only occurs at the end of the read out process. Finally, sampling inputs and weights from a normal distribution tracks gradient descent pathways that avoid saddle points that riddle high-dimensional problems.

Numerous options are available to further optimize the power dissipation. Significant resources have been dedicated to developing low-voltage memristors well below the supply limitation of the 40nm process used. At one extreme end, the work in [22] uses 100 mV programming pulses. This would reduce resistive dissipation by a larger factor, though must be balanced with the average switching time. Our work has shown that lengthy bit streams are required to obtain acceptable convergence in the training process. This is on par with literature [13], however, we expect that the use of quantization-aware training can effectively place a constraint upon the set of permissible values, thus reducing the bit length.

Alternatively, the generation of long bit streams can be time multiplexed to free up arrays to increase throughput. Although this slows down bit stream generation, we have eliminated the large delays associated with MAC operations by one order of magnitude [21]. SC also uses noise to avoid being trapped by saturated gradients during training, known to significantly slow down the training process in conventional computing. Endurance concerns can be overcome by substituting RRAM for Magnetoresistive Random Access Memory (MRAM), which exhibits endurance of over 10^{15} cycles, compared to that of RRAM ($\approx 10^5 - 10^{12}$ cycles) with faster write times, at the cost of increased fabrication complexity.

VII. Conclusion

In this brief, we demonstrated that it is indeed possible to perform DL parameter optimization using stochastic bits by exploiting the stochasticity during switching of probabilistic CBRAM devices to efficiently generate stochastic bit streams. This new insight to stochastic computing is valuable for the following reasons:

1) For an end-to-end stochastic computing system, the gradient update step can share resources with the feed-forward step; the alternative to long bit-streams would be a floating point unit, which offsets this disadvantage;
2) The multiply-and-accumulate steps now rely only on two combinational logic gates. This means the propagation delay for MAC is reduced by orders of magnitude.

An investigation of switching probability variation on account of device-to-device variation and an end-to-end timing analysis at the circuit and system level for a variety of configurations to train more complex networks using larger data sets and forms the basis of future work.

References

[1] M. Rahimi Azghadi, Y.-C. Chen, J. K. Eshraghian, J. Chen, C.-Y. Lin, A. Amirsoleimani, A. Mehonic, A. J. Kenyon, B. Fowler, J. C. Lee, and Y.-F. Chang, "Complementary Metal-Oxide Semiconductor and Memristive Hardware for Neuromorphic Computing," *Advanced Intelligent Systems*, vol. 2, no. 5, 2020.

[2] O. Krestinskaya, A. P. James, and L. O. Chua, "Neuromemristive Circuits for Edge Computing: A Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 1, pp. 4–23, 2020.

[3] A. Mehonic, A. Sebastian, B. Rajendran, O. Simeone, E. Vasilaki, and A. J. Kenyon, "Memristors—From In-Memory Computing, Deep Learning Acceleration, and Spiking Neural Networks to the Future of Neuromorphic and Bio-Inspired Computing," *Advanced Intelligent Systems*, 2020.

[4] C. Lammie, O. Krestinskaya, A. James, and M. R. Azghadi, "Variation-aware Binarized Memristive Networks," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genova, Italy., Nov. 2019, pp. 490–493.

[5] Y. Chen, "ReRAM: History, Status, and Future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[6] Y. Li, Z. Wang, R. Midya, Q. Xia, and J. J. Yang, "Review of Memristor Devices in Neuromorphic Computing: Materials Sciences and Device Challenges," *Journal of Physics D: Applied Physics*, vol. 51, no. 50, Sep. 2018.

[7] S. Mittal, "A Survey of ReRAM-based Architectures for Processing-in-memory and Neural Networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 75–114, 2019.

[8] B. Muthuswamy and P. P. Kokate, "Memristor-Based Chaotic Circuits," *IETE Technical Review*, vol. 26, no. 6, pp. 417–429, 2009.

[9] C. Zheng, J. K. Eshraghian, A. James, and H. H. Iu, "Chaotic Oscillator Using Coupled Memristive Pairs," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genova, Italy., Nov. 2019, pp. 462–465.

[10] H. Jiang, D. Belkin, S. E. Savel'ev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao, M. Barnell, Q. Wu, J. J. Yang, and Q. Xia, "A Novel True Random Number Generator Based on a Stochastic Diffusive Memristor," *Nature Communications*, vol. 8, no. 1, p. 882, Oct. 2017.

[11] T. Zhang, M. Yin, C. Xu, X. Lu, X. Sun, Y. Yang, and R. Huang, "High-speed True Random Number Generation Based on Paired Memristors for Security Electronics," *Nanotechnology*, vol. 28, no. 45, Oct. 2017.

[12] J. M. de Aguiar and S. P. Khatri, "Exploring the Viability of Stochastic Computing," in *IEEE International Conference on Computer Design (ICCD)*, New York City, NY., Oct. 2015, pp. 391–394.

[13] P. Knag, W. Lu, and Z. Zhang, "A Native Stochastic Computing Architecture Enabled by Memristors," *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 283–293, 2014.

[14] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic Memristive Devices for Computing and Neuromorphic Applications," *Nanoscale*, vol. 5, pp. 5872–5878, 2013.

[15] W. Sun, B. Gao, M. Chi, Q. Xia, J. J. Yang, H. Qian, and H. Wu, "Understanding Memristive Switching via In Situ Characterization and Device Modeling," *Nature Communications*, vol. 10, no. 1, Aug. 2019. [Online]. Available: https://doi.org/10.1038/s41467-019-11411-6

[16] R. Naous, M. Al-Shedivat, and K. N. Salama, "Stochasticity Modeling in Memristors," *IEEE Transactions on Nanotechnology*, vol. 15, no. 1, pp. 15–28, 2016.

[17] S. Ruder, "An Overview of Gradient Descent Optimization Algorithms," *CoRR*, vol. abs/1609.04747, 2016. [Online]. Available: http://arxiv.org/abs/1609.04747

[18] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the Importance of Initialization and Momentum in Deep Learning," ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3, Atlanta, GA., Jun. 2013, pp. 1139–1147.

[19] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A deep neural network accelerator based on tiled rram architecture," in *2019 IEEE Int. Electron Devices Meeting (IEDM)*. IEEE, 2019.

[20] S. Gaba, F. Cai, J. Zhou, and W. D. Lu, "Ultralow sub-1-na operating current resistive memory with intrinsic non-linear characteristics," *IEEE Electron Device Letters*, vol. 35, no. 12, pp. 1239–1241, 2014.

[21] M. Shoba and R. Nakkeeran, "Energy and area efficient hierarchy multiplier architecture based on vedic mathematics and gdi logic," *Int. J. Engineering Science and Technology*, vol. 20, no. 1, pp. 321–331, 2017.

[22] X. Zhu, Q. Wang, and W. D. Lu, "Memristor networks for real-time neural activity analysis," *Nature Communications*, vol. 11, no. 1, 2020.