Role of Memristive Device Variability in Governing the Accuracy and Conversion Time of Machine Learning Algorithms on Neuromorphic Hardware

Andrew J. Ford, *Member, IEEE,* and Rashmi Jha, *Member, IEEE*

***Abstract*—A vital issue regarding hardware implementations of machine learning algorithms with novel memristive devices is the concern of the proposed architecture's resilience to high device variability. We find that most algorithms have surprisingly high tolerance to variable weight updates and initializations. We also propose a simple method of quantifying the maximum variability an algorithm can handle without suffering loss of accuracy or increased training time. Finally, we show high level simulations of an RRAM cell with intermediate states, decay, and Gaussian variability.**

***Index Terms*—Memristive, RRAM, Device, Variability, Neuromorphic**

I. INTRODUCTION

With rise in Artificial Intelligence (AI) and Machine Learning (ML), neuromorphic architecture designs with memristive crossbar arrays have caught significant research attention [1]. This is because memristive devices, due to their non-volatile reconfigurable states with low programming voltages, offer tremendous opportunities for in-memory computing that addresses the memory bandwidth bottleneck faced by other computing architectures. However, when designing novel architectures implemented with memristive devices, it becomes clear that tolerance to device variability is not a recommendation; rather, it is a requirement of the research. In comparison with other traditional memory devices, these devices face the obstacle of matching the increasing read and write precision which entails substantial research and development. A natural inclination for designers might be to choose accurate memory write operations with a sacrifice regarding increased area, power consumption, or minimum operational frequency accompanying flash or SRAM relative to variable memristive devices. However, our findings indicate that the choice may not be so straightforward, and that the ML algorithms are quite tolerant to variability in states during read and write operations performed on these devices. In spite of numerous work reporting neuromorphic architectures with memristive crossbar arrays, a detailed study on acceptable variability on memristive devices for successful design, verification, and validation of neuromorphic systems design around memristive crossbar arrays is lacking. Due to this gap in the knowledgebase it is difficult to develop the verification and validation techniques on for these hardware designs. In this paper, we report our studies on the impact of variability in memristive devices on accuracy and conversion time for ML algorithms implemented on neuromorphic architectures implemented using memristive crossbar arrays. We also propose that ML accuracy vs. conversion time on benchmarking datasets, such as, MNIST [2], can also be used as successful validation technique of the neuromorphic hardware designs using memristive crossbar arrays.

Manuscript submitted on September 7th, 2020. This work is supported by Rindsberg Fellowship to Andrew Ford and National Science Foundation under awards # ECCS-1926465 and #CCF-1718428 .

A. J. Ford is a student in the Master of Science graduate program at the University of Cincinnati, Cincinnati, OH 45220 USA (email: fordaj@mail.uc.edu).

R. Jha is with the Electrical Engineering and Computer Science Department, University of Cincinnati, Cincinnati, OH 45220 USA (email: jhari@ucmail.uc.edu).

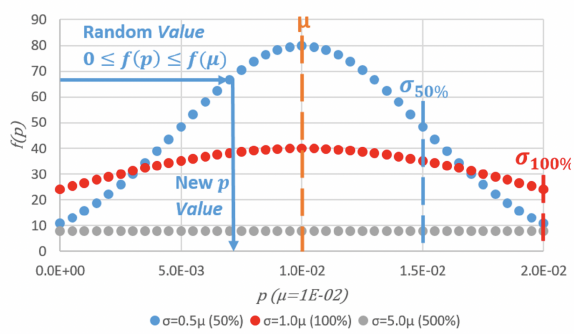

Fig. 1. Normal distribution created by the Gaussian function. (Figure from [1])

II. SIMULATION PLATFORM SETUP

During the domain specific neuromorphic hardware designs, various levels of abstractions are used based on design complexities and verifications are performed accordingly. Often, novel neuromorphic hardware architectures are first simulated at high level in languages like C++, MATLAB, or Python. To demonstrate the impacts of device variability in high level simulations, we chose to write a Single Layer Perceptron (SLP) in Python training against the popular MNIST dataset of handwritten numerical digits. The training and validation datesets were untouched, and left to 60,000 and 10,000 digits, respectively. In order to keep each trial comparable to other trials, the training and testing sets are not shuffled.

A. Representing Device Variability

Variability in resistive states of the device was simulated according to a normal function, shown in Fig. 1. The normal

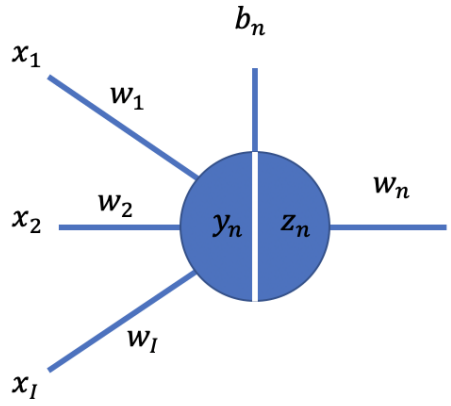

Fig. 2. SLP neuron diagram. b_n is the neuron's bias. y_n is the sum of the previous layer's weighted outputs calculated in eq. (3). z_n is the activation of y_n calculated in eq. (4).

distribution is modeled by the Gaussian function, shown in Eq. (1):

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{1}$$

The plot in Fig. 1 was created by selecting x values and computing the corresponding $f(x)$ values. The most likely $f(x)$ value to occur after variability is applied is $f(\mu)$, where μ is the ideal weight value, and $f(\mu)$ decreases as σ increases. In order to simulate random variability, we need to work backward from this. By randomly selecting $f(x)$ values ranging from $0 \leq f(x) \leq f(\mu)$, we can calculate x values using Eq. (2):

$$x = \mu \pm \sigma \sqrt{2\ln(\frac{1}{f(x)\sigma\sqrt{2\pi}})} \tag{2}$$

This process is applied in python simulations and referred to as $N(\mu, \sigma)$.

B. Network Properties

The MNIST handwritten digit dataset was chosen for its popularity as a benchmarking algorithm for evaluating machine learning algorithms. Static learning rates which provide the reliably highest accuracies for the SLP will contribute to the weight updates according to the Stochastic Gradient Descent (SGD) algorithm via backpropagation. [3]

C. SLP Neuron Properties

A diagram of a layered perceptron neuron is shown in Fig. 2. During testing, each neuron propagates forward through the network according to eq. (3).

$$y_n = b_n + \sum_{i=0}^{I} x_i \cdot w_i \tag{3}$$

b_n represents the individual neuron's bias, initialized randomly between -1 and 1 pre-training. x_i represents an element of the input vector (in this case, one pixel of an incoming MNIST digit). w_i represents each neurons coefficient which corresponds to a certain input element of an incoming input vector (in this case, the coefficient aligning with one MNIST input pixel, or the output value z_n from a previous layer in the case of a Multi-Layer Perceptron). Each neuron activates according to the sigmoidal activation function in eq. (4).

$$z_n = \frac{1}{1 + e^{-y_n}} \tag{4}$$

The cost function used for the layered perceptron networks was the Mean Squared Error (MSE) function in Eq. (5).

$$MSE = \sum_{n=0}^{N} \frac{1}{2}(ideal_n - z_n)^2 \tag{5}$$

During training, eq. (5) is used to calculate the error of the previous layer, Using Stochastic Gradient Descent (SGD), we can minimize the cost function by computing its gradient set to 0 in eq. (6).

$$\nabla MSE = 0 \tag{6}$$

This leads us to the partial derivatives in eq. (7).

$$\begin{aligned} A &: \frac{\partial MSE}{\partial z_n} = z_n - ideal_n \\ B &: \frac{\partial z_n}{\partial y_n} = z_n(1 - z_n) \\ C &: \frac{\partial y_n}{\partial w_i} = x_i \end{aligned} \tag{7}$$

These partial derivatives contribute to the outer layer's weight update algorithm in eq. (8).

$$\begin{aligned} w_i^{new} &= w_i^{old} - \alpha \cdot A \cdot B \cdot C \\ b_i^{new} &= b_i^{old} - \alpha \cdot A \cdot B \end{aligned} \tag{8}$$

Hidden neuron weight and bias updates in Multi-Layer Perceptrons (MLPs) can be calculated with eq. (9), but was not studied for the scope of this work.

$$\begin{aligned} w_i^{new} &= w_i^{old} - \alpha \cdot (\sum_{n=0}^{N} A_n \cdot B_n \cdot w_n) \cdot (z_n(1 - z_n)) \cdot x_i \\ b_i^{new} &= b_i^{old} - \alpha \cdot (\sum_{n=0}^{N} A_n \cdot B_n \cdot w_n) \cdot (z_n(1 - z_n)) \end{aligned} \tag{9}$$

Finally, the process described in Section II. A. can be applied to weight updates to simulate device variability in eq. (10).

$$\begin{aligned} w_i^{new} &= N(w_i^{new}, \sigma) \\ b_i^{new} &= N(b_i^{new}, \sigma) \end{aligned} \tag{10}$$

The σ found in eq. (10) is increased to simulate higher device variability, and decrease to simulate a device with more precise behavior, as is shown in (REFERENCE FIGURE).

D. Learning Rates

The static learning rate which, in the case of layered perceptrons, controls many aspects of backpropagation via stochastic gradient descent, can be different depending on a number of factors. Without diving deep into dataset characteristics and

σ	68.2%	95.4%	99.6%
0.5	0-1	-0.5-1	-1-1
0.1	0.4-0.6	0.3-0.7	0.2-0.8
0.05	0.45-0.55	0.4-0.6	0.35-0.65
0.01	0.49-0.51	0.48-0.52	0.47-0.53
0.005	0.495-0.505	0.49-0.51	0.485-0.515
0.001	0.499-0.501	0.498-0.502	0.497-0.502

Fig. 3. Impact of σ during a weight request of 0.5. At the most extreme, $\sigma = 0.5$, which results in weights occurring anywhere between the upper and lower bounds. A more realistic σ value, being $\sigma = 0.1$. This variability will cause the majority of new weight value update requests of 0.5 to occur between the values of 0.47 and 0.53.

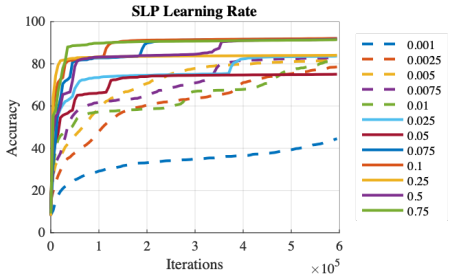

Fig. 4. Impact of learning rate on a SLP. Ideal learning rate was found to be $\alpha = 0.1$, which reached and consistently held accuracies above 90% quickly without over-training.

number of hidden layers and neurons, the learning rate is not the same for every application or network. For the purpose of device variability, it is not vital that the learning rate achieve a high peak accuracy; the learning rate should be chosen such that the network can reliably train to desired values. The trials to determine the ideal SLP learning rate can be found in Fig. 4. From Fig. 4, the ideal learning rate was found to be 0.1 for the SLP. Currently, the constraints of this particular experiment include 60,000 MNIST training inputs, and in the interest of minimizing trial time, only 10 epochs being run per trial, with a total of 600,000 iterations of training per trial. This same process could be used for any machine learning algorithm.

E. Successful vs. Unsuccessful Training

Given the constraints of 600,000 iterations during training and a static learning rate of $\alpha = 0.1$ for the SLP, it is also important to determine what constitutes a successful trial. In Fig. 5, 50 trials were run with our static learning rate. It is clear that these values range fairly wildly, depending on which randomized values the weights are set to. Since all trials in Fig. 5 were deemed successful, we can conservatively say that **any SLP trial which achieves and maintains at least 70% accuracy within the third epoch (180,000th iteration) of training is successful**. It is important to note that using an SLP for MNIST is not ideal - and far greater accuracies have been achieved much more reliably with with MLPs [4]. The SLP, however, is desirable for our purposes since it does not achieve 99%+ accuracies with ease, and as a result, any poor performances will be easily noticed.

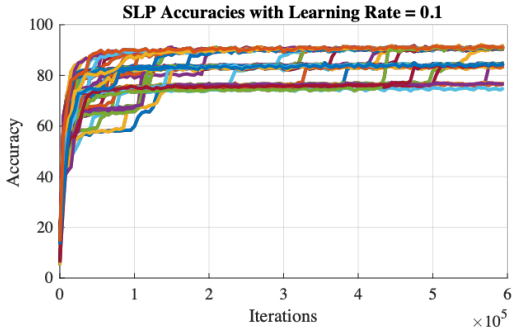

Fig. 5. Baseline variability from highly precise weight updates using Python's float variables. 50 trials run on an SLP with a learning rate of $\alpha = 0.1$ determined from Fig. 4.

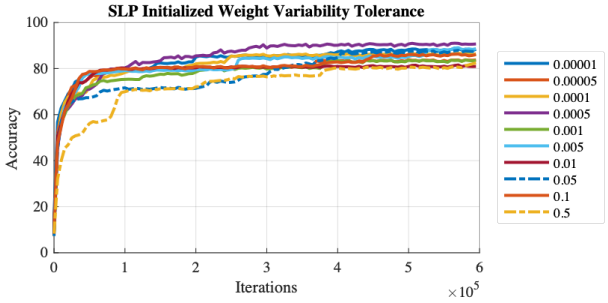

Fig. 6. SLP behavior when subjected to weight initialization variability depicted in eq. (10). Legend shows the variability in the form of σ.

III. Results

In neuromorphic hardware, memristive devices are used to represent the weights, therefore, device variability will manifest itself as variabilities in "weight" on NN. There were three phases of results in which the impact of device states variability on the SLP was assessed: weight initialization, weight update, and combined weight initialization and update.

A. Variability in Weight Initialization

The results of 10 trials of an SLP with a learning rate of $\alpha = 0.1$ where the weight and bias initializations were subjected to a Gaussian function to simulate device variability is shown in Fig. 6. An important consideration here is the behavior at boundary extremes. In the case of layered perceptrons, weights are initialized between -1 and 1. Take a weight initialization request of $w_0 = 0.98$. In a device with high variability, there is a reasonable change that the device may cause the actual weight to be set to $w_0 = 1.03$. Since weights need to vary between -1 and 1, upper and lower caps are necessary. This means that the initialized weights may not exceed $w_0 = 1$, and may not be below $w_0 = -1$. In this example, a weight request of $w_0 = 0.98$ is quite likely to exceed the upper bound, and be set to $w_0 = 1$. As seen in Fig. 6, too many weight values residing at their boundary extremes results in a substantial delay in the time required to constitute a successfully trained trial, in the cases of $\sigma = 0.5$ and $\sigma = 0.05$. However, all weight initializations with $\sigma \leq 0.01$ were uninhibited by their device variability. When comparing Fig. 6 to Fig. 4, there is a

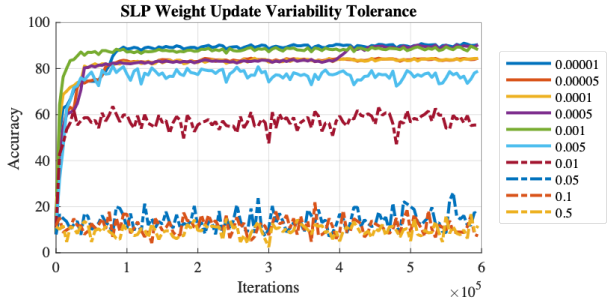

Fig. 7. Accuracy of a SLP with variability in weight updates. Legend shows the variability in the form of σ.

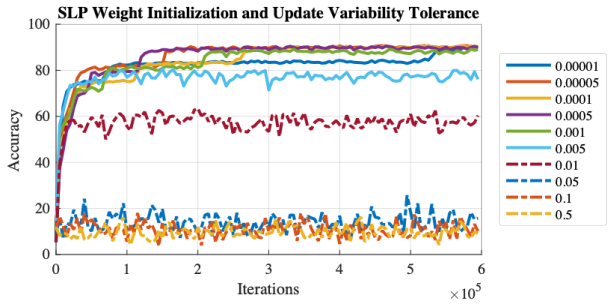

Fig. 8. Accuracy of a SLP with variability in weight updates and initializations.

noticeable delay at the beginning of training, likely resulting from behavior at the initialization boundary extremes.

B. Variability in Weight Updates

A phase heavily impacted by device variability would be weight updates during training. Results from 10 trials of an SLP with a static learning rate of $\alpha = 0.1$ and a variation in σ are shown in Fig. 7. The weight update phase is where variability has the most impact. From Fig. 10, all trials with $\sigma < 0.01$ were considered passing. However, $\sigma = 0.05$ did not show signs of improving above 80% accuracy, and this should be an absolute worst case amount of variability.

C. Combined Variability in Weight Initialization and Updates

The properties of the other two phases become most clear when combined and compared to their individual results. From Fig. 8, many aspects from the inclusion of device variability are noticeable. The delay introduced in training time from the weight initialization phase in Fig. 6 is present. Beyond that, it becomes clear that past the first epoch, the weight update phase (Fig. 10) becomes the dominant hindrance to the SLP training when too much device variability is present. However, as with the weight update phase, the only failures in training occured with variability of $\sigma \leq 0.005$. Again, the variability of $\sigma = 0.005$ did not show signs of surpassing 80% accuracy.

D. Variability in Offloaded Weights

In the case of an inferencing processor, the novel devices with higher variability may be utilized solely offline relative to the training device. For instance, a fleet of GPUs may be used

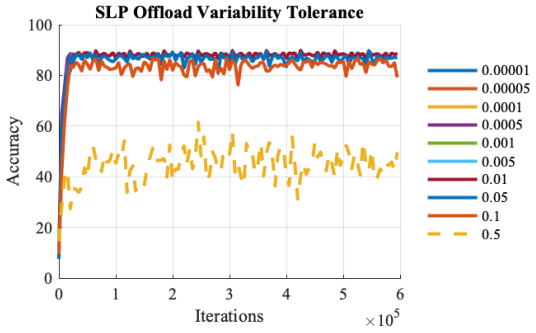

Fig. 9. Accuracy of a SLP with variability in weight and bias offloading from GPUs to an inferencing processor. Only trial to fail was $\sigma = 0.5$. When compared to Fig. 8, the tolerance to variability is extremely high, since the weight update and initialization phases are allowed to utilize high precision during weight updates.

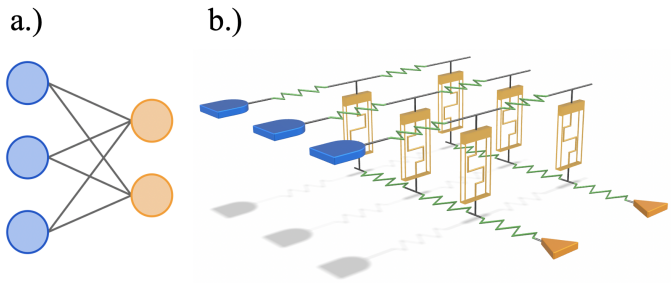

Fig. 10. Memristive RRAM implementation of a SLP. Part a.) shows a very high level SLP diagram, where 3 blue inputs are fully connected via the gray lines to 2 orange neurons. Part b.) shows 3 blue input voltages, encoded as currents as they route through the memristors, fully connected to the output neurons shown in orange.

to hone in on the ideal weight values, and those weight values could be copied over to a low-power high-speed inferencing processor, with no training capabilities. Fig. 9 shows how tolerant a single layer perceptron is to device variability during the offloading process from GPUs to a memristive processor.

IV. Gated RRAM: An Example

To offer a more realistic explanation and application of our machine learning tolerance to device variability, a SLP which uses Short-Term Memory (STM) and Long-Term Memory (LTM) Resistive Random Access Memory (RRAM) was also studied [8]. Fig. 10 shows the memristive array implementation of RRAM cells (part b.) of a single layer perceptron (part a.).

A. STM vs. LTM RRAM

Recently, research has been presented on STM and LTM RRAM which offers the opportunity to rely on a transferrance of weights from a training to a testing phase [5]. While RRAM is easy to use, quick, and low power, it has unique decay characteristics that could be viewed as worrisome for a designer looking to utilize RRAM in a machine learning processor. These decay properties can be seen in Fig. 11. When a weight value in an RRAM cell is set to a value of 1, it decays towards 0 constantly. It decays quickest at the beginning, and

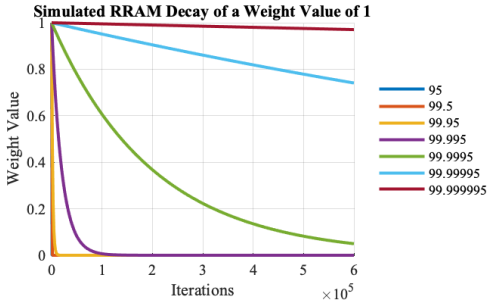

Fig. 11. Simulated exponential decay of an RRAM cell set to produce a weight value of 1. The x-axis shows the number of iterations, which corresponds to 10 epochs of MNIST training. The legend contains the percent of the weight preserved after one iteration. A decay value of 95% means that after one iteration, a weight value of 1 will become 0.95.

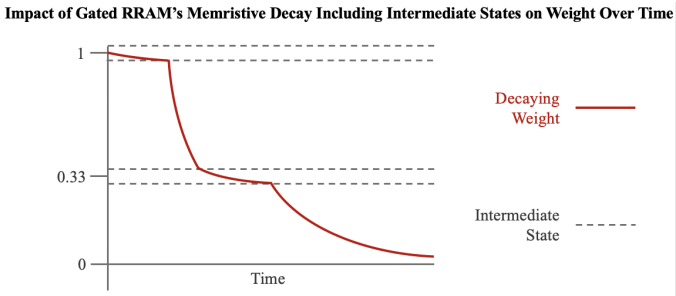

Fig. 12. Weight behavior inside and outside of an RRAM intermediate states. It decays quickly outside of the intermediate state window, and slowly when inside. Whether the weights have a negative or positive magnitude, they will decay towards 0 asymptotically. In this figure, this would represent $2^2 = 4$ intermediate states, at 1, 0.33, -0.33, and -1.

approaches 0 asymptotically. Testing and experiments have shown that decay can be modeled exponenetially [6]. Fig. 11 uses an exponential decay, where each iteration sees a weight decay of a certain value (95%, 99.5%, 99.95%, etc.) times the current weight.

Along with decay, it is important to model intermediate states. There are a varying number of intermediate states in an RRAM cell [7], usually 2^n. When an RRAM cell is set to one of these intermediate states, the decay rate becomes significantly smaller, as is shown in Fig. 12. In theory, there is zero decay when using these states. In reality, a small amount of decay is present. For our applications, the LTM RRAM will primarily reside in the intermediate states, and the STM will be modeled by an average rate of decay.

Another important consideration is the type of RRAM's specific decay rate. STM RRAM cells offer low power consumption rates at high operating frequencies, and prove ideal for training, since it should occur so quickly. Once weight values converge to values which yield a high network accuracy, the STM will continue to decay at a relatively rapid rate. These training weights could be offloaded to a different part of the processor into LTM RRAM cells, which will not decay quickly, and have a sufficient number of intermediate states which will not hinder the network's ability to classify accurately.

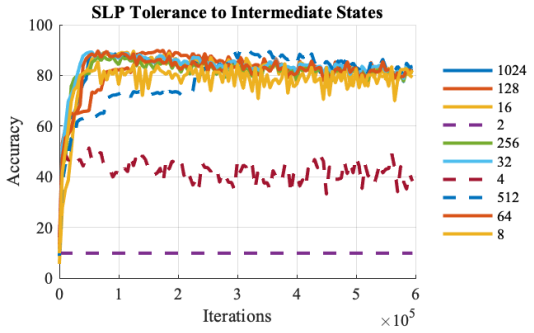

Fig. 13. Minimum number of intermediate states required in which the classification accuracy of an SLP is not inhibited. Dashed lines show the lowest performing trials. The exception was 512 states, which took longer to converge, but still reached satisfiable accuracy levels. Within completion of the 4th epoch, 512 states converges on classification values higher than any other, which implies it likely had to do with unlucky weight randomization, resulting in delayed convergence.

B. Intermediate States and Accuracy

In an examination of the behavior of the SLP with RRAM's high level characteristics, including device variability, an important first step is to determine the minimum number of required minimum states which the LTM testing portion of a processor would require without harming the SLP's classification accuracy. All parameters from the previous sections were used, and the results for 2^n states are shown in Fig. 13, where $1 \leq n \leq 10$. We find that any trial with more than $n = 2^3 = 8$ intermediate testing states is sufficient for this particular algorithm. $n = 2^3$ intermediate states still classified fairly accurately, but may be too volatile for some applications.

C. Realistic Intermediate Decay

In theory, there is no decay when RRAM cells are conducting in one of their intermediate states. In practice, there is still some decay, but not quite as much as when it is between intermediate states.

The testing phase realistically proceed as follows:

1) Offload weights from STM to LTM, which sets RRAM cells to their intermediate states
2) Between each testing inference, weight values decay slowly away from their intermediate states towards 0

Since testing is done in LTM, the RRAM cells do not decay quickly enough for the weight values to leave their intermediate state windows. On the contrary, the training process is different and more difficult to simulate:

1) Ideally, train continuously, testing in parallel as needed
2) If parallelized testing/training is not feasible, then keep testing sessions as quick as possible, so STM weights do not decay
3) With a forced potentiation and depression model, the decay is rapid enough that an average decay constant can be implemented. The idea here is that weight updates are so frequent that STM RRAM doesn't have any time to decay substantially enough to have any impact on training quality

With the testing and training intermediate states and decay implemented, a few trials with variable decay were run and

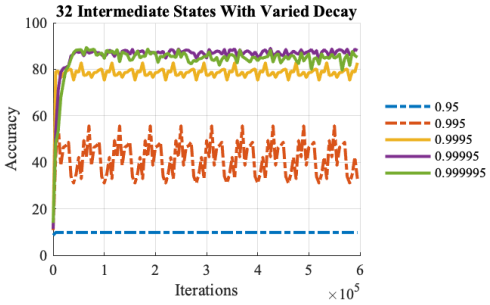

Fig. 14. SLP behavior when 32 intermediate states are implemented with decay. STM decay is shown in the legend, and LTM decay is one tenth of STM decay. The only trials to fail in this case were 95% STM decay between iterations and 99.5% STM decay between iterations. The dashed lines indicate failures.

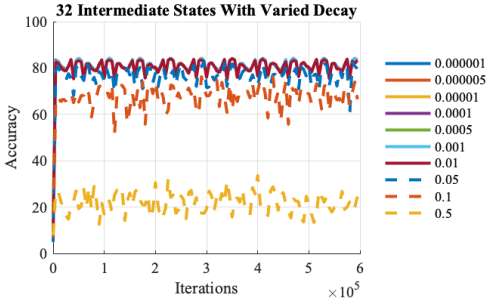

Fig. 15. The classification accuracy of a SLP behaving as though its weights are comprised of gated STM and LTM RRAM. 32 intermediate states are used, with 99.95% decay during training, and 99.995% decay during inferencing. Dashed lines indicate failed trials, which fell below 70% accuracy after the third epoch of training.

are compiled in the plot in Fig. 14. Given that the only 2 trials to fail had decay rates of 95% STM decay between iterations (99.5% LTM) and 99.5% STM decay (99.95% LTM), this is very promising with only 32 intermediate testing states.

D. Results

In an effort to simulate a gated RRAM SLP at high level, we attempt to demonstrate proof of concept by offering a quantifiable tolerances to device variability, intermediate states for weights, and memristive device decay, both from STM and LTM. The results for 32 states are shown in Fig. 15. The amount of space between the weights here is not trivial; each weight must occupy a value what cannot fall below a resolution of 0.0625. 32-bit precision. With a typical 32-bit floating point number, 23 bits are used to represent whole numbers, and 8 bits for the decimal points. That is an astounding 0.00000001 resolution! Having 8 bit decimal resolution low may not hinder the classification accuracy of the SLP, however, it implies more computing time and power consumption would be required. As seen in Fig. 15, 32 intermediate states and a substantial amount of decay, and variability of $\sigma \leq 0.01$, the SLP was able to successful classify MNIST with a mean of 80% accuracy.
A slight improvement in mitigating decay, as well as increasing the number of states from 32 to 64, yeilds the results in Fig. 16. .

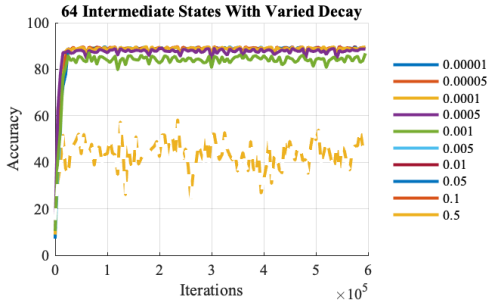

Fig. 16. SLP behavior with 99.5% STM decay, 99.95% LTM decay, 64 intermediate states, and varied device variability. Small developments reducing decay and intermediate states improve the SLP's resistance to high device variability.

V. Conclusions

Device variability is a major consideration each time a new electrical component is created. As time goes on and the devices can be improved, variability and decay rates can all be reduced, but that is not guaranteed. However, rather than ignore devices with a noticeable device variability, it is worthwhile to analyze a machine learning algorithm's tolerance to device variability. Modeling the devices by subjecting the weights to a Gaussian variability is useful to understand how much variability the algorithm is capable of withstanding without sacrificing accuracy. As an example, a SLP was analyzed with a varying σ. With too high of a variability, the accuracy fell below 70%. However, once σ fell below 0.01, the SLP performed uninhibited. To further illustrate the benefits of examining device variability, the SLP was adapted to mimic the behavior that STM and LTM RRAM. It was simulated with intermediate states, memristive decay, and device variability. The SLP was still able to converge to reliably classify MNIST with 80% accuracy. It was simulated with 32 states, a high rate of decay, and substantial device variability. When lower power consumption, higher operating speeds, and other positive aspects are available for consideration, it is worthwhile to examine the algorithm's resilience before prioritizing precise weight updates.

References

[1] T. J. Bailey, A. J. Ford, S. Barve, J. Wells and R. Jha, "Development of a Short-Term to Long-Term Supervised Spiking Neural Network Processor," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, doi: 10.1109/TVLSI.2020.3013810.

[2] LeCun, Yann and Cortes, Corinna and Burges, CJ, "MNIST handwritten digit database", ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist, vol. 2, 2010.

[3] Ketkar N. (2017) Stochastic Gradient Descent. In: Deep Learning with Python. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2766-4_8

[4] Ou, J., Li, Y. & Shan, C. Two-dimensional perceptrons. Soft Comput 24, 3355–3364 (2020). https://doi.org/10.1007/s00500-019-04098-w

[5] Mark R. Rosenzweig, Edward L. Bennett, Paul J. Colombo, Diane W. Lee, Peter A. Serrano, Short-term, intermediate-term, and long-term memories, Behavioural Brain Research, Volume 57, Issue 2, 1993, Pages 193-198, ISSN 0166-4328, https://doi.org/10.1016/0166-4328(93)90135-D. (http://www.sciencedirect.com/science/article/pii/016643289390135D)

[6] W. Wang, A. Bricalli, M. Laudato, E. Ambrosi, E. Covi and D. Ielmini, "Physics-based modeling of volatile resistive switching memory (RRAM) for crosspoint selector and neuromorphic computing," 2018 IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, 2018, pp. 40.3.1-40.3.4, doi: 10.1109/IEDM.2018.8614556.

[7] Castán,H. and Dueñas,S. and García,H. and Ossorio,O. G. and Domínguez,L. A. and Sahelices,B. and Miranda,E. and González,M. B. and Campabadal,F.,Analysis and control of the intermediate memory states of RRAM devices by means of admittance parameters, Journal of Applied Physics,vol. 124, no. 15, pg. 152101, 2018, doi 10.1063/1.5024836, https://doi.org/10.1063/1.5024836

[8] Jones, Alexander and Jha, Rashmi ,A Compact Gated-Synapse Model for Neuromorphic Circuits, 2020, 2006.16302, arXiv, cs.NE

Andrew J. Ford received a B.S. degree in electrical engineering from the University of Cincinnati, Cincinnati, OH, USA, in 2019, where he is currently pursuing a M.S. degree in electrical engineering. He works as a Graduate Research Assistant at the University of Cincinnati, where he researches neuromorphic computing, emerging memory devices, and applications of electroencephalography data.

Rashmi Jha (M'05) Rashmi Jha (M'05) received the B.Tech degree in electrical engineering from IIT Kharagpur, Kharagpur, India, in 2000, and the M.S. and Ph.D. degrees in electrical engineering from North Carolina State University, Raleigh, NC, USA in 2003 and 2006 respectively.

She is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, University of Cincinnati, Cincinnati, OH, USA.