Memristor-Spikelearn: A Spiking Neural Network Simulator for Studying Synaptic Plasticity under Realistic Device and Circuit Behaviors

Yuming Liu* Angel Yanguas-Gil† Sandeep Madireddy† Yanjing Li*
*The University of Chicago, Chicago, IL, USA †Argonne National Laboratory, Lemont, IL, USA

***Abstract*—We present the *Memristor-Spikelearn* simulator (open-sourced), which is capable of incorporating detailed memristor and circuit models in simulation to enable thorough study of synaptic plasticity in spiking neural networks under realistic device and circuit behaviors. Using this simulator, we demonstrate that: (1) a detailed device model is essential for simulating synaptic plasticity workloads, because results obtained using a simplified model can be misleading (e.g., it can overestimate test accuracy by up to 21.9%); (2) detailed simulation helps to determine the proper range of conductance values to represent weights, which is critical in order to achieve the desired accuracy-energy tradeoff (e.g., increasing the conductance values by $10\times$ can increase accuracy from 70% to 83% at the price of $20\times$ higher energy); and (3) detailed simulation also helps to determine an optimized circuit structure, which is another important design parameter that can yield different accuracy-energy tradeoffs.**

***Index Terms*—spiking neural network, synaptic plasticity, memristor-based designs**

I. Introduction

Memristor-based implementations of synaptic plasticity in spiking neural networks (SNNs) provide a promising approach towards energy-efficient online learning [1]–[7]. Such designs are typically structured as crossbars, and the synaptic weights of a SNN are represented as memristor conductance values. To achieve online learning, conductance values need to be updated based on the corresponding weight value changes required by the synaptic plasticity algorithm.

A key challenge of designing these memristor-based designs is that the change in conductance (ΔG) of a memristor device may not exactly match the desired change in the corresponding weight (ΔW_d). This is because ΔG depends not only on the input voltage V, but also on the current conductance G_i of the same device. To be specific, $\Delta G = h_i(V)$, where the h_i functions can be different for different G_i values and are determined uniquely by the specific device characteristics. Therefore, we must design memristor crossbars together with synaptic plasticity algorithms carefully to mitigate any discrepancies between the desired and actual weight update amounts; otherwise, the desired accuracy targets may not be achieved.

A naive solution is to determine V such that ΔG and ΔW_d always match (by using the inverse of the correct h_i given G_i). However, this approach can incur prohibitive costs. For example, in many existing memristor crossbar designs [2]–[4], [6], [7], one input voltage signal is shared across all cells in a row or column to avoid the high overhead of generating a separate signal for each cell. Since the current conductance values G_i's, and thus the h_i functions, can be different for the cells sharing the same input voltage, calibrating the voltage for individual cells will not work. Even if a signal could be dedicated to each cell, for real-world tasks it would be extremely inefficient to re-calibrate every voltage signal every time the conductance is required to change.

Other potential solutions include (1) mapping weight values in higher memristor conductance ranges, which reduces the dependence of ΔG on G_i at the price of increased energy costs; (2) devising more complex circuit structures to offset the discrepancies between ΔW_d and ΔG; (3) creating hardware-aware synaptic plasticity algorithms so that the discrepancies between ΔW_d and ΔG can be tolerated. Although only a few examples are listed, there are many opportunities for inventing new, novel solutions. To facilitate research for tackling this important co-design challenge, it is crucial to support realistic simulations of real-world SNN synaptic plasticity workloads while taking into consideration realistic memristor device and circuit behaviors. However, currently no methods or infrastructures exist for this purpose (more details in Sec. II-E).

To close this gap, we present a new simulator, called Memristor-Spikelearn, which is capable of incorporating device and circuit details to enable the study and benchmarking of different memristor crossbar designs (using various device configurations and circuit structures) and synaptic plasticity algorithms under realistic conditions. Using this simulator, we simulated two SNN online learning workloads, both are used to classify the MNIST dataset but they use different synaptic plasticity algorithms. We experimented with two device models: a simplified model and a detailed, realistic model, as well as two different circuit structures: 1R and 1T1R, where each cell in the crossbar consists of one memristor and one transistor together with one memristor, respectively. The key results are:

1. Models that do not capture realistic device characteristics can produce misleading results. For example, when the simplified device model is used, accuracy can be overestimated by up to 12.8% and 21.9% for the two workloads used in our experiments, for the 1R circuit structure. Moreover, for the first workload, the energy cost required to achieve a 82% accuracy target is underestimated by $5.3\times$ when the simplified device model and the 1R circuit structure are used.

2. To obtain the desirable accuracy/energy targets, it is critical to determine the proper range of conductance values to represent the weight values. For example, for the first workload

978-3-9819263-7-8/DATE23/© 2023 EDAA

in our study, when the 1R circuit and the detailed device model are used, mapping the unit weight 1 to the conductance value $10^{-5}S$ yields 70% accuracy while incurring an energy cost of $0.85nJ/image$, while mapping the same weight value to $10^{-4}S$ increases the accuracy to 83%, but it also increases the energy cost by $20\times$.

3. Different circuit structures can yield different accuracy-energy tradeoffs, which provides another dimension for design space exploration. For example, while the structure of the 1T1R circuit is larger and consumes higher power than the 1R circuit, it can partially offset the discrepancies between ΔG and ΔW_d, thereby allowing weight values to be mapped to lower conductance ranges to minimize energy costs for online learning workloads. For example, for the first workload in our experiments, the 1T1R design consumes $17\times$ less training energy than the 1R design without sacrificing test accuracy.

In summary, the main contributions of this paper are:

(1) The key observation, confirmed by our results, that it is essential to study memristor-based designs together with synaptic plasticity algorithms under realistic device and circuit behaviors.

(2) The design and implementation of Memristor-Spikelearn, the *first* simulator that incorporates detailed device and circuit models in the simulation of realistic synaptic plasticity workloads. We have open sourced this simulator [8].

(3) The accuracy and energy results obtained using Memristor-Spikelearn for different workloads, device models, and circuit structures, which provide new insights.

II. Background and Related Work

A. Spiking Neural Network (SNN)

A SNN consists of neurons that communicate to each other by exchanging spikes, similar to biological neurons, to accomplish various tasks such as image recognition, object tracking, navigation, and motion detection. Neurons are connected by unidirectional synapses, and each synapse carries a weight value. A synapse accepts spikes generated by the pre-synaptic neuron, multiplies them by its weight, and propagates the weighted spikes to its post-synaptic neuron. The potential of a neuron is typically determined by accumulating the values of all in-coming spikes. Once its potential reaches a threshold, the neuron will generate a spike, and its potential will drop to the reset level [9].

B. Synaptic Plasticity

Synaptic plasticity algorithms change the synaptic weights in a SNN based on spike activities collected during the training process (and, optionally, external information such as golden prediction results for supervised learning) to enable online learning. Common forms of synaptic plasticity include STDP (spike timing dependent plasticity), modulated STDP, and fully-supervised plasticity [10]. We discuss some examples of synaptic plasticity algorithms (the ones used in our experiments) in more details in Sec. IV-B.

C. Memristor Crossbar Designs for SNN Workloads

In crossbars designed for SNN workloads, the inputs of each row and outputs of each column correspond to the pre- and post-synaptic neurons for a set of synapses, respectively, and the conductance of memristors correspond to the synaptic weights. Typical circuit structures of a cell include 1R [1], [5], 1T1R [2]–[4], and 2T1R (two transistors, 1 memristor) [6], [7].

Figure 1 illustrates the circuit structures (1R and 1T1R), SNN to crossbar mapping, along with the circuit activities for a 2×2 fully-connected layer for three operating modes: (1) Spike propagation mode used for both inference and training, which generates spikes in post-synaptic neurons given the spikes from pre-synaptic neurons. In Fig. 1, we use the orange texts to annotate how a spike from pre-synaptic neuron 1 propagates to both post-synaptic neurons. (2/3) weight update mode used in training, which is triggered by spikes in post-/pre-synaptic neurons (the synaptic plasticity algorithm decides whether one or both update modes are required). Conductance (weight) updates are only applied to the columns/rows that receive post-/pre-synaptic spikes, and the amounts of conductance change is specified as input voltages to the rows/columns (so this amount is the same for all memristors in the same row/column). In Fig. 1, the blue/green texts annotate an example where weight updates are triggered after pre-synaptic neuron 1/post-synaptic neuron 2 generates a spike.

A synaptic weight (w_{ij}) is linearly mapped to the conductance of the corresponding memristor element in the crossbar: $G_{ij} = w_{ij} * G_1$, where G_1 specifies the conductance for unit weight 1. The choice of G_1 is an important consideration because it affects not only the energy consumption, but also the accuracy of a workload – as discussed in Sec. I, since the change in conductance depends on the current conductance state, certain G_1 values can lead to larger discrepancies between the actual and desired weight values than others.

D. Memristor models

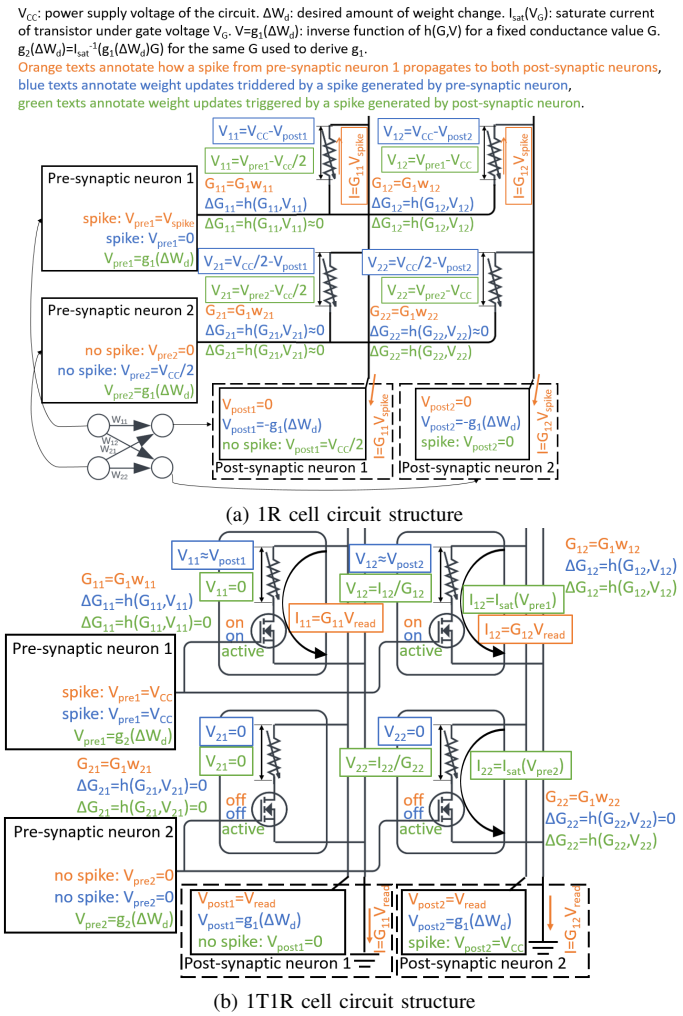
We classify existing memristor models into three types based on their respective levels of details, as summarized in Table I. Simplified device models, which are widely adopted in previous work, are inadequate for the purpose of studying memristor-based implementation of synaptic plasticity (see Sec. V-A).

TABLE I
MEMRISTOR MODEL TYPES

Type	Examples	How realistic?	Captures ΔG's dependence on G?
Physical	PDE-based [11]	Most realistic	Yes; validated under fast voltage sweeps
Semi-empirical	Compact model for HfO_x [12] and TiO_2 [13] devices	Reasonable	Yes; validated under voltage pulses or fast voltage sweeps
Simplified	TEAM [14], VTEAM [15]	Unrealistic	No

E. Prior Work

Multiple SNN simulation frameworks exist; however, they are not adequate for studying synaptic plasticity implemented in memristor-based designs. Nengo-dl [16] supports the training of SNNs as conventional ANNs (Artificial Neural Networks)



(a) 1R cell circuit structure

(b) 1T1R cell circuit structure

Fig. 1. Memristor crossbar designs and operations to execute SNN workloads, using a 2×2 fully-connected SNN layer as an example.

using back-propagation. SnnToolbox [17] is only capable of converting ANNs to SNNs. Neither Nengo-dl nor SnnToolbox supports synaptic plasticity algorithms developed for SNNs. Both Nengo [18] and Brian [19] allow various forms of synaptic plasticity to be specified, but they lack the proper interface for incorporating device and circuit details. In BindsNET [20], synaptic plasticity algorithms are tied to network topology, making it difficult to flexibly implement any given algorithms and to incorporate device and circuit details.

Previous work on synaptic plasticity algorithms assumes that weights are always updated exactly as specified without considering whether inexact updates can be tolerated [21]–[25]. In [3], [4], [6], [7], special synaptic plasticity algorithms are tailored for the specific memristor device and circuit designs. However, the workloads used are quite simple (e.g., memorizing one or several input patterns), and the generality/scalability of such algorithms has not been demonstrated.

III. The Design and Implementation of Memristor-Spikelearn

We devise a new simulation framework, Memristor-Spikelearn, to overcome the limitations of previous work discussed above. The key features of the simulator include: (1) It allows crossbar designs using different devices and circuit structures to be flexibly modeled. (2) It provides modular design, isolating device/circuit modeling from SNN modeling to achieve extensibility. (3) It allows automated procedures to be integrated to perform large-scale design space exploration and benchmarking studies.

A. Base Simulator Design: Spikelearn

We designed Memristor-Spikelearn based on the Spikelearn simulator [26], implemented in Python. Figure 2 shows a block diagram of Memristor-Spikelearn, where the original Spikelearn components are outlined in black boxes.

SNN is the top-level base class, which instantiates (1) *Layer*, which represents neurons, and (2) *BaseSynapse*, which represents synapses that connect two groups of neurons with any given topology. From these classes, various subclasses can be defined. For example, the *LIF* class implements the leaky-integrate and fire neurons [9], [27], and the *PlasticSynapse* class implements synapses with STDP functionalities. Input spikes are passed through input ports defined in the *SNN* class to connect to the *BaseSynapse* instances. The *step* method in *SNN* is used to advance simulation by one time step. At each time step, spikes generated in the previous time step along with input spikes in the current time step are passed to the *call* method in the *BaseSynapse* class, and the *step* method of neurons (in the *Layer* class) are also invoked to simulate spike activities. The output spikes of neurons are then recorded for the next time step, or as network outputs.

B. New Features Essential for Simulating Memristor-based Synaptic Plasticity

The original Spikelearn optimistically assumes that ΔG always exactly match the desired ΔW_d. To incorporate realistic device and circuit behaviors in simulation, we augmented the original simulator with three new classes: *PlasticSynapseCircuit*, *SynapseCell*, and *Data_driven*. Together they serve as a flexible interface to allow different device models and circuit structures to be specified, as shown in Fig. 2.

To add a new device model to present a synapse using one memristor (1R), a new class that inherits *SynapseCell* will need to be defined. The key methods in this class (*deltaG_to_signal* and *update*) are used to calculate and apply the actual ΔG based on the corresponding ΔW_d, so ΔG is different for different device models for the same ΔW_d.

To perform simulation using more complex circuit structures (e.g., 1T1R or 2T1R) than 1R, we created the *Data_driven* class, derived from *SynapseCell*, in which a lookup table is provided to store ΔG values under various combinations of memristor voltage, current, and conductance values. The information required by the table may be obtained through detailed circuit/device simulations. Memristor-Spikelearn refers to this lookup table throughout the simulation to apply the actual weight changes to each synapse in the network.

Furthermore, each *PlasticSynapseCircuit* object is capable of estimating its total energy consumption by accumulating the energy consumed for every spike propagation and weight

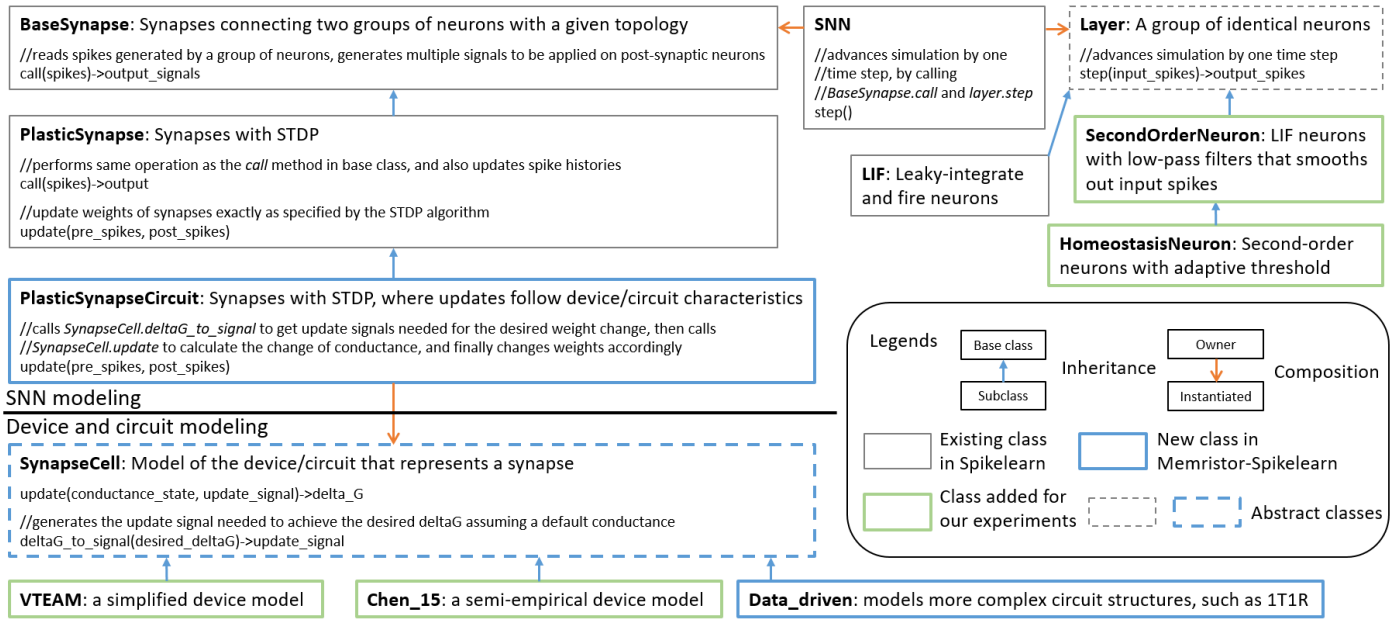


Fig. 2. The block diagram of Memristor-Spikelearn, showing the major classes and key methods.

update operation, which is obtained based on the device/circuit models.

IV. Experiment Setup

To showcase the capability of Memristor-Spikelearn and also to obtain new insights on memristor-based implementations of synaptic plasticity in SNNs, we implemented and conducted various experiments using different device/circuit models and workloads. All experiments were run on Intel(R) Xeon(R) Silver 4116 servers.

A. Device and Circuit Models

We experimented with the following crossbar designs: (1) 1R cells with a semi-empirical HfO_x memristor model [12] (Chen_15 in Fig. 2); (2) 1R cells where memristors are modeled using VTEAM [15]; and (3) 1T1R cells with the same semi-empirical device model as (1). We specified the information required by the lookup table in the *Data_driven* class by sweeping the memristor conductance, the voltage on the gate of transistor, and the voltage on the top terminal of the memristor in HSPICE simulation to determine the actual conductance change under each combination of these circuit parameters. A Verilog-A implementation of the semi-empirical model is used to model the memristor in HSPICE simulation. In addition, a software baseline is implemented where all weight updates exactly match the amounts specified by the algorithms.

B. Workloads

We ran experiments using two workloads, both of which are used to classify the MNIST dataset. The first workload (Workload 1) implements a STDP-based unsupervised algorithm [24]. The SNN consists of three layers. The input layer encodes each pixel as a Poisson spike sequence. Then a set of excitatory synapses fully connect the input layer to 400 excitatory neurons in second layer . The weights of the excitatory synapses are trainable. The third layer consists of 400 inhibitory neurons. Each excitatory neuron is connected to one inhibitory neuron, and each inhibitory neuron is connected to all excitatory neurons except its input excitatory neuron to provide lateral inhibition [24]. The weights of the synapses connecting the second and third layers are treated as hyper-parameters and are not changed during training. To train the SNN, the full MNIST training/test sets of $60,000/10,000$ images are used for training/testing, respectively. First, the weights of the excitatory synapses are updated in an unsupervised manner: each training sample is presented as input spikes for $1,000$ time steps, during which the amount of weight change of a synapse upon a spike of its pre- or post-synaptic neuron is exponential wrt. the time since the last spike. Afterwards, a supervised procedure assigns labels to the excitatory neurons: each neuron is assigned the digit to which it fires the most frequently during the last $10,000$ training samples. For inference, the digit predicted for a sample is the one with the highest average spike count among all excitatory neurons of the same label.

To simulate this workload in Memristor-Spikelearn, the *SecondOrderLayer* and *HomoeostasisLayer* classes were added (outlined by green boxes in Fig. 2), as they are required to model neuron behaviors required by the algorithm.

In the second workload (Workload 2), the SNN is a fully-connected network with 784 input neurons, $1,500$ hidden neurons, and 10 output neurons (corresponding to the 10 classes in MNIST). During inference, each pixel of an input image is converted into a series of spikes spanning 49 time steps and is fed to each input neuron. The output neuron that generates the most spikes during the next 49 steps provides the predicted digit. To train the weights of all synapses in this SNN, an algorithm that approximates gradient descent [25] is used, and $33,000$ training samples and $1,000$ testing samples are randomly selected from the MNIST training and testing

datasets, respectively. Each training sample is presented as input spikes for 49 time steps. At the end of every update interval (7 time steps), an error value is assigned to each hidden and output neuron. The error of an output neuron is set to 1 if it corresponds to the correct label but does not generate any spike in this interval, -1 if it corresponds to an incorrect label but generates spike(s) in this interval, and 0 otherwise. The error vector of the hidden neurons is the product of the output neuron error vector and the weight matrix between the hidden and output layers (except that if no spike is generated by a hidden neuron during this interval, its error value is adjusted to 0). After assigning error values, all pre-synaptic spikes from the first two layers in this interval are replayed. During replay, if a synapse receives a spike, its weight update amount is calculated by multiplying the learning rate with the error value of its post-synaptic neuron.

C. Parameter Mapping

As discussed in Sec. II-C, G_1 (the conductance value used to represent unit weight 1) is an important design parameter. In our experiments, we swept different values of G_1. For each G_1 value, we also experimented with various learning rates. The accuracy reported for each G_1 value is the the best accuracy that is achieved across different learning rates.

V. Results

A. Accuracy Results using Different Memristor Models

For the 1R design, when either VTEAM or the semi-empirical device model is used, the accuracy of Workload 1 cannot reach the same accuracy as the software baseline (Fig. 3). For Workload 2, accuracy comparable to the software baseline is only achieved if weights are mapped to high conductance ranges (Fig. 4). Moreover, the test accuracy obtained when using the semi-empirical model can be quite different from that obtained using VTEAM. For Workload 1, when $G_1 = 10^{-5}S$, simulation using the semi-empirical device model shows significantly lower test accuracy than the software baseline (70%, down from 86%), while simulation using VTEAM indicates a test accuracy of 79%. The maximum accuracy gap between the two models is 12.8%, which occurs at $G_1 = 7 \times 10^{-6}S$. In higher conductance ranges (e.g., when $G_1 = 10^{-4}S$), the accuracy gaps are much smaller.

Similar trends also exist in the results for Workload 2, as shown in Fig. 4. When G_1 is mapped to a lower conductance value (e.g., $1.4 \times 10^{-5}S$), simulation with the semi-empirical model shows a significant test accuracy degradation compared to the software baseline, while using VTEAM overestimates the test accuracy by up to 21.9%. For $G_1 \geq 5 \times 10^{-5}S$, simulations using these two models lead to similar test accuracy results.

To explain these differences, we analyzed the device models. In both models, the maximum resistance is $R_{off} = 2.2M\Omega$. w_{min}, the minimum weight value that can be represented in the circuit, can be derived accordingly (it is 0.0044 and 0.044 when G_1 is $10^{-4}S$ and $10^{-5}S$, respectively). A higher w_{min} value contributes to lower accuracy because weight values smaller than w_{min} are required by the workloads. The significant accuracy loss observed using the semi-empirical model when G_1 is small (e.g., when it is $10^{-5}S$) is due to the larger degree of switching non-uniformity (i.e., the rate of conductance change depends more heavily on the current conductance state). VTEAM is not capable of revealing this problem as it assumes the same conductance change rate for all conductance values; therefore, it incorrectly suggests that setting G_1 to a small value (e.g., $10^{-5}S$) leads to reasonable accuracy for both workloads. These results clearly confirm our key observation that it is critical to incorporate detailed device models that capture realistic device behavior.

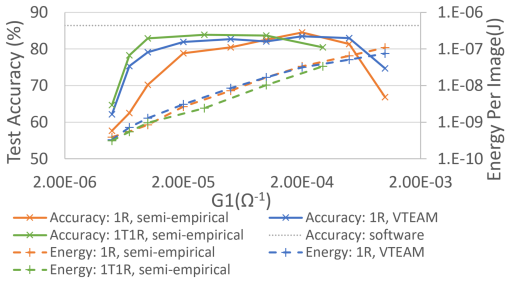


Fig. 3. Energy and accuracy results of Workload 1. The network is trained for 2 epochs (i.e. 2 iterations through all training samples), and then evaluated for test accuracy.

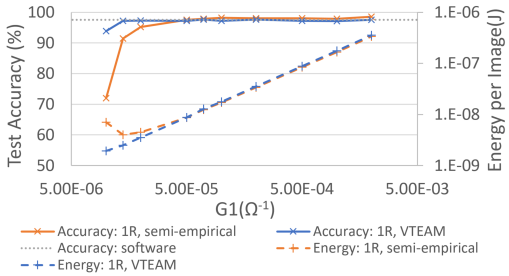


Fig. 4. Energy and accuracy results of Workload 2. The network is trained for 3 epochs, and then evaluated for test accuracy.

B. Accuracy Results using Different Circuit Designs

For Workload 1, high test accuracy ($\sim$ 82%) is achieved even when G_1 is as low as $10^{-5}S$ for the 1T1R design (below this range, the accuracy drops because the minimum weight value that can be presented becomes too high). For the 1R design, a similar accuracy is achieved only with a much higher G_1 of $10^{-4}S$, while a lower G_1 value results in much lower accuracy. This is due to significant non-uniformity of switching for the 1R synapses as discussed in Sec. V-A. This effect is less significant in the 1T1R design, because for weight updates upon post-synaptic spikes (as shown in Fig. 1b with green annotations), a desired ΔW_d is translated into the current applied on the corresponding memristor device. Since voltage changes linearly wrt. current, it partially offsets the non-uniform switching effect.

C. The Impact of Weight-Conductance Mapping on Accuracy-Energy Tradeoffs

We performed experiments with different values of G_1 to investigate its impact on accuracy-energy tradeoffs. Results for Workload 1 using the 1R design is shown in Fig. 3. When $G_1 = 10^{-5}S$, the test accuracy is only 70%, and the training energy is $0.85nJ/image$. For $G_1 = 10^{-4}S$, the test accuracy increases to 83%, but the energy ($16.6nJ/image$) also increases significantly (by $20\times$). For Workload 2 (Fig. 4), when $G_1 = 2 \times 10^{-5}S$, the test accuracy is 95.2% and the training energy is $4.5nJ/image$. When $G_1 = 7 \times 10^{-5}S$, both the test accuracy and energy are higher (97.9% and $12.4nJ/image$, respectively).

Comparing the semi-empirical model with VTEAM, they largely agree on the energy consumption for various G_1 values, showing an approximately linear relationship. The only exception is for Workload 2, when the accuracy is low, simulations using the semi-empirical model indicate higher energy consumption results than those obtained from simulations using VTEAM. This is because, the number of weight update operations is inversely proportional to the accuracy for this workload (note that this does not apply to Workload 1). Since VTEAM suggests higher accuracy, this means less weight update operations, and thus lower energy consumption.

However, VTEAM cannot be used to accurately determine accuracy-energy tradeoffs because it can over-estimate accuracy significantly. For example, in Workload 1, if an accuracy of 82% is required, simulations using VTEAM suggest that G_1 can be set to $2 \times 10^{-5}S$, yielding a low energy consumption of $3.1nJ/image$. This result is too optimistic – it is $> 5\times$ lower than that obtained from the realistic semi-empirical model.

Comparing the 1R design with the 1T1R design, the latter is more complex and generally less energy efficient for inference [28]. However, for training, the 1T1R design may allow G_1 to take on a much smaller value than what is required by the 1R design without sacrificing accuracy, thereby resulting in lower training energy (e.g., by $17\times$ for Workload 1 with a 82% accuracy target as shown in Fig. 3). The takeaway here is that different circuit structures can yield different accuracy-energy tradeoffs, which provides another dimension in design space exploration for memristor-based implementations of synaptic plasticity in SNNs.

VI. Conclusion

We present the Memristor-Spikelearn simulator to enable the study of memristor-based implementations of synaptic plasticity in SNNs under realistic device and circuit behaviors. Memristor-Spikelearn is designed with extensibility and scalability in mind. We have open-sourced this framework, and our eventual goal is a powerful infrastructure encompassing rich and diverse libraries of device and circuit models, SNN architectures, synaptic plasticity algorithms, and online learning tasks to facilitate future research.

Acknowledgment

This work was supported by DOE ASCR and BES Microelectronics Threadwork. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

[1] Y. Wang *et al.*, "Energy efficient rram spiking neural network for real time classification," in *GLSVLSI*, 2015.
[2] S. Ambrogio *et al.*, "Spike-timing dependent plasticity in a transistor-selected resistive switching memory," *Nanotechnology*, sep 2013.
[3] S. Ambrogio *et al.*, "Neuromorphic learning and recognition with one-transistor-one-resistor synapses and bistable metal oxide rram," *IEEE Transactions on Electron Devices*, 2016.
[4] G. Pedretti *et al.*, "Stochastic learning in neuromorphic hardware via spike timing dependent plasticity with rram synapses," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2018.
[5] D. Querlioz *et al.*, "Simulation of a memristor-based spiking neural network immune to device variations," in *IJCNN*, 2011.
[6] S. Kim *et al.*, "Nvm neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," in *IEDM*, 2015.
[7] Z. Wang *et al.*, "A 2-transistor/1-resistor artificial synapse capable of communication and stochastic learning in neuromorphic systems," *Frontiers in neuroscience*, 2015.
[8] Y. Liu *et al.* https://github.com/YLab-UChicago/Memristor-Spikelearn.
[9] F. Ponulak *et al.*, "Introduction to spiking neural networks: Information processing, learning and applications.," *Acta neurobiologiae experimentalis*, 2011.
[10] J. C. Magee *et al.*, "Synaptic plasticity forms and functions," *Annual review of neuroscience*, 2020.
[11] S. Larentis *et al.*, "Resistive switching by voltage-driven ion migration in bipolar rram—part ii: Modeling," *IEEE Transactions on Electron Devices*, 2012.
[12] P.-Y. Chen *et al.*, "Compact modeling of rram devices and its applications in 1t1r and 1s1r array design," *IEEE Transactions on Electron Devices*, 2015.
[13] M. D. Pickett *et al.*, "Switching dynamics in titanium dioxide memristive devices," *Journal of Applied Physics*, 2009.
[14] S. Kvatinsky *et al.*, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2013.
[15] S. Kvatinsky *et al.*, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2015.
[16] D. Rasmussen, "Nengodl: Combining deep learning and neuromorphic modelling methods," *Neuroinformatics*, 2019.
[17] B. Rueckauer *et al.*, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, 2017.
[18] T. Bekolay *et al.*, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, 2014.
[19] M. Stimberg *et al.*, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, aug 2019.
[20] H. Hazan *et al.*, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in neuroinformatics*, 2018.
[21] S. R. Kheradpisheh *et al.*, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, 2018.
[22] G. Srinivasan *et al.*, "Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing," *Frontiers in neuroscience*, 2019.
[23] P. Ferré *et al.*, "Unsupervised feature learning with winner-takes-all based stdp," *Frontiers in Computational Neuroscience*, 2018.
[24] P. U. Diehl *et al.*, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, 2015.
[25] A. Tavanaei *et al.*, "Training spiking convnets by stdp and gradient descent," in *IJCNN*, 2018.
[26] A. Yanguas-Gil *et al.*, "Automl for neuromorphic computing and application-driven co-design: asynchronous, massively parallel optimization of spiking architectures," in *ICRC*, 2022.
[27] M. Bouvier *et al.*, "Spiking neural networks hardware implementations and challenges: A survey," *J. Emerg. Technol. Comput. Syst.*, apr 2019.
[28] D. Niu *et al.*, "Design of cross-point metal-oxide reram emphasizing reliability and cost," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.