Training Memristive Neural Networks

1st Ermanno Fiorillo 2nd Paul Roters 3rd Vaishnavi Chaturvedi

***Abstract*—Conventional digital computers face challenges such as the Von Neumann bottleneck and limitations in transistor scaling. These challenges are compounded by the end of Dennard scaling [1], which has halted further reductions in supply voltage due to constraints on noise margin. Furthermore, Moore's Law [2], historically driving increases in digital CMOS circuit operating frequencies, has significantly slowed. As a result, the role of CMOS device scaling in achieving high-performance and low-power computing has diminished [3]. Overcoming these obstacles necessitates exploring alternative computing paradigms. While advancements like high parallelization and near-memory computing alleviate some data communication bottlenecks, transformative approaches such as in-memory computing (IMC) are essential for achieving substantial improvements in computational efficiency and reducing power consumption.**

Modern artificial intelligence relies heavily on neural networks that demand rapid, parallel operations for optimal performance, often achieved through hardware acceleration. Memristor-based neuromorphic computing systems effectively address these needs by employing memristor crossbar arrays for parallel multiply-add operations in the analog domain. This approach not only enhances throughput but also reduces energy and area consumption significantly.

This project aims to reproduce a simple paradigm for training *on-chip* using a single crossbar per layer and integrating two memristors per synapses. The system leverages the backpropagation (BP) algorithm [4] for weight updates.

I. Introduction

Memristive crossbars act as the basic layer for neuromorphic hardware, simultaneously storing and processing information like neurons and synapses in the human brain. Unlike conventional computers that run commands sequentially, neuromorphic computers process and store data in parallel, making them faster and more energy-efficient.

A. Biological Counterpart

The foundation of the neuromorphic revolution is the memristor, theorized by Leon Chua [4] and phisically implemented by Williams and Strukov [5]. The charachteristic 'memristance' [5] of the analog memristor can continuously change according to the input voltage and recall the altered resistance value, akin to synaptic plasticity in biological synapses. Structurally, memristors' top electrodes, middle insulating layer, and bottom electrodes correspond to the presynaptic membrane, synaptic cleft, and postsynaptic membrane of biological synapses [6]. Memristors, due to their nanoscale size, can potentially match the number of synapses in the human brain on very-large-scale integration (VLSI) circuits.

B. Challenges in Computing in Memory (CIM)

Training models using Computing in Memory (CIM) faces significant challenges due to non-idealities that affect performance. Contemporary memristors often inaccurately represent synaptic weights due to their stochastic behavior, which stems from ionic transport dynamics [8]. Typical current-voltage (I-V) hystereses exhibit an abrupt SET event necessitating current compliance limits and a gradual RESET event controlled by applied voltage, affecting the OFF state. This behavior arises from amplified or suppressed drift and diffusion processes of charge carriers during switching events [9], [10].

To mitigate these issues, iterative corrections are employed at the expense of time and energy, particularly weight mapping can be compensated through iterative programming using a write-verify scheme, where conductance is read and rewritten if necessary [11]. Previous approaches applied this practice to every weight of a Deep Neural Network (DNN), resulting in prolonged programming times [12]. However, recent studies indicate that only a small subset of weights require write-verify to maintain DNN accuracy, enabling significant speedups. Techniques like SWIM (Second-order Weight Importance Measurement) further optimize this process, requiring only a single pass of forward and backpropagation to efficiently identify weights needing write-verify [11], while there are other techniques that train neural networks for memristor programming to map desired conductance updates to the required voltage pulses, reducing memristor programming delays compared to traditional methods [12].

Another critical source of error is *'sneak-path'* currents, causing cross-talk interference between adjacent memory cells in memristor crossbar arrays. This interference can lead to misinterpretation of stored information, particularly problematic as crossbar size increases. Solutions targeting leakage currents aim to reduce power consumption and enhance reliability and stability of crossbar arrays [13], [14], considerations that were also factored into our project's design.

C. Training Methods: Ex-Situ, In-Situ and Hybrid

Each of these training methods has its own advantages and challenges:

- **Ex-Situ Training**: Allows any algorithm to be implemented in software, updating memristor weights once. This approach handles circuit non-linearities that increase with the weights switches, making it the safest method from this perspective. However, it struggles with modeling device variations and is limited by hardware processor bottlenecks due to dependence on von Neumann architecture. Moreover, since the weights are updated only

once, it suffers from accuracy losses when the network is applied to new datasets or devices [15]–[17].

- **In-Situ Training**: This method avoids the need for a duplicated digital system, eliminating the processor-memory bottleneck. In-situ training, automatically adapts network parameters to minimize the impacts of hardware non-idealities, such as wire resistance and conductance drift. However, there are two main factors that complicate the implementation of *in-situ* training [18]–[20]:
  - Resolution and Endurance: Devices require high resolution to program weight updates accurately and high endurance due to the frequent SET/RESET operations during the training process.
  - On-Chip Learning Algorithm: Fully exploiting in-situ learning in practical applications requires not only performing the VMM in the crossbar but also carrying out the learning algorithm on-chip. Additionally, the development of the necessary on-chip electronics is not straightforward and entails significant research costs. A trade-off solution is to implement peripheral circuit electronics *off-chip*.
- **Hybrid Training**: Combines *ex-situ* and *in-situ* training. The model is trained ex-situ, then weights are transferred to the memristor processing elements (PEs), with the last fully connected layer trained in-situ to adjust for device variations, enhancing efficiency. With only one layer updated, no gradient storage is needed and biases can be adjusted for device-to-device imperfections [21].

The focus of the present project was to implement a paradigm for *on-chip* training after a thorough review of the literature. To achieve this goal an initial idea of the circuit was implemented on *LTSpice* [22], as shown in Fig. 2, particular focus should be directed towards the single crossbar's cell: researchers often encounter the need for a transistor to control the current flow in memristors (recent advances have introduced memristor that inherently include a selector, that are called memtransistors [23]), that is why we opted for a 1T1M cell architecture, where each cell includes a switch transistor controlled by an external source (Figure 1). However, the complexity of weight updates and training time led us to simplify the schematic, a decision further detailed in the Results section.
The choice of the training algorithm relied on backpropagation since it stands as one of the most used algorithm even for Memristor networks [24]–[26].

II. Related Literature

Our training was focused on a pattern recognition task, which is one of the most researched evaluation criteria in AI, as highlighted in surveys like [27], [28]. This task is fundamental to human cognition; our cortex implements a pattern recognition algorithm every day: the human visual system (HVS) integrates perceptual and processing tasks, followed by parallel high-level image processing in the visual

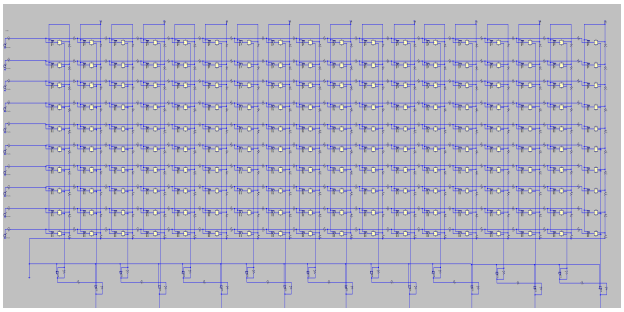

Fig. 1. First layer of the Network, 10x18 crossbar based on 9 inputs and 1 bias, the columns are doubled to have positive and negative weights.

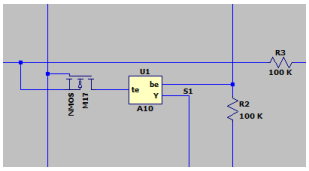

Fig. 2. Single cell of the memristor crossbar in LTSpice, the memristor is the A14 Model from [20]

cortex [29]. Throughout human evolution, pattern processing capabilities have become increasingly sophisticated, mainly due to the expansion of the cerebral cortex, particularly the prefrontal cortex and regions involved in image processing [30].

A. Network Architectures Overview

Over the last decades, various network architectures for neuromorphic hardware have been proposed [31], including Reservoir Computing and Liquid State Machines [32], Recurrent Neural Networks [33] and Hopfield networks [34]. Current research also explores embedding attention mechanisms and transformers in memristor crossbars [35].
For this project, we opted for a network based on 2 fully connected layers.

B. Training of Memristor Weights

The most relevant section of the project relies on the training algorithm. Here, we present a summary of training paradigms closely related to our project's idea.

- **Backpropagation** Numerous studies on Memristor Neural Networks are based on backpropagation [24]–[26], [36]. Generally implemented ex-situ to avoid hardware derivative implementation issues [36], straightforward implementation of backpropagation in memristive hardware is impractical because each weight must be uniquely adjusted according to the delta rule. Analog adjustment of weights could be difficult, especially considering the large number of training epochs required and the memristor endurance. Nevertheless, in-situ variations of backpropagation have been developed over the past decade [25].

- **Manhattan Rule** An example of in-situ training is based on the Manhattan Rule [37], [38], a coarse-grain variation of backpropagation. In this approach, only the sign information of weight adjustment is used:

$$\Delta W_{Mij}(n) = \text{sgn}[\Delta W_{ij}(n)], \tag{1}$$

and

$$\Delta W_{M'ij} = \text{sgn}[\Delta W'_{ij}]. \tag{2}$$

The primary appeal of this algorithm is that all weights are updated by the same amount, simplifying weight updates and facilitating efficient in-situ hardware implementation.

- **Perturbation Algorithms** Memristor-based neural networks have explored various perturbation algorithms to simplify weight adjustment and enhance circuit implementation. The Random Weight Change (RWC) [39] algorithm, despite its simplicity, does not utilize the error variance and only considers the sign of error variation, resulting in low convergence speed. Conversely, the Weight Simultaneous Perturbation (WSP) [40] algorithm considers both the value and sign of error variation for weight adjustments. However, it simultaneously perturbs all weights, complicating the circuit implementation of multi-layer neural networks. The Weighted Sum Simultaneous Perturbation (WSSP) [41] algorithm refines this approach by updating the weights based on the difference between unperturbed and perturbed error functions. Unlike the BP algorithm, WSSP avoids complex derivative calculations and error backpropagation. Furthermore, compared to WSP, WSSP only applies perturbations to the weighted sum, leading to more concise and efficient circuit implementations.

III. Research Question

The core objective of our project was to explore and combine different approaches for in-situ training, centered on the backpropagation algorithm, with the aim of solving a pattern recognition task using a simple double-layer fully connected network. The main doubts were focused on the actual implementability of the network in LTSpice:

- **Memristor model**: 'Which model should we use?' This was the most critical question, as it is the foundational element of the circuit and the entire project. A thorough research led us to find an adequate model that was less sensitive to weight change and had high endurance. Our crossbar was based on verified memristor models, particularly selected for their performance across various operating conditions, including low, middle, and high frequencies signal levels and nonlinearity, chosen amongst one of the model in this book [20]. However, further simplification was necessary to suit our implementation needs.
- **ADC + DAC**: A significant challenge anticipated from the beginning was the reliance on ADCs and DACs in existing methods. Finding the right component for the conversion could lead to several errors. Generally, alternatives to DAC and ADC involve complex network configurations, such as the reservoir computing paradigm, which introduces high levels of randomness not easily implementable in hardware.
- **Backpropagation**: While assembling the forward path of the circuit was straightforward, implementing a working backward path that could properly change the memristors' conductances posed a major challenge. The literature provided guidance for the forward path, but practical implementation of the backward path required innovative solutions.

IV. Results

The core idea of a circuital implementation passed through several simplifications. The pattern were encoded in 3×3 pixelated matrixes, mapped into a one-dimensional vector format and feeding them into a specialized nine-input memristive neuron circuit. Each pixel in the image is represented by a logic "1" (1 V) for black and logic "0" (0 V) for white, facilitating the recognition process, we also opted for a -0.1 V 0.1 V for faster convergence.

Each layer of the network was based on a 10×18 crossbar, the tenth row was reserved to the bias, the columns were doubled in order to double the precision [16] The choice of memristor model, crucial for the system's performance, considered various nonlinear models including those proposed by Strukov, Joglekar, and Biolek, all available at this github folder [43].

To streamline LTSpice simulations and effectively manage the dynamic nonlinearity of memristors, a simplified memristor model was adopted. This decision aimed to optimize simulation time while maintaining accuracy in evaluation of the weights update procedure. Our time decided to approximate their behavior to a linear relationship between input and output terminals. The circuit implementation of backpropagation training for the neural network was based on the research by Hasan and Yakopcic [26], [36], leveraging the idea of using memristor crossbars to implement matrix-vector multiplication based on Kirchhoff's law:

$$I_j = \sum_i G_{ji} V_i.$$

The entire weight update procedure was implemented in Python. First, the circuit .net file is read, and an initializing function introduces the input voltages and bias to the circuit, randomly initializing memristor conductances (we observed significant mismatch in simulation depending on the order of magnitude of these parameters) around $1\,\text{M}\Omega$. After the first simulation (run in LTSpice using the PyLTSpice library [44]), the process can be broken down into three major steps:

- **Record layer 2 neuron output errors:** Error terms ($\delta_{L2,i}$) were generated using comparators between the positive and negative weights columns outputs (Figure 3).

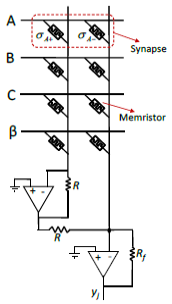

Fig. 3. Schematic from [26] of the first 2 columns of the crossbar

- **Back-propagate layer 2 errors through second crossbar weights and record layer 1 errors:** The errors from layer 2 ($\delta_{L2,i}$) are applied to the second layer weights using the formula:

$$\delta_j = (t_j - y_j) f'(DP_j)$$

where t_j are the target outputs, y_j are the actual outputs from the comparators circuit, and $f'(DP_j)$ is the derivative of the activation function. We opted for the *tanh* function to avoid overflow errors, as it scales the outputs to the range $[-1, 1]$ and its derivative can be expressed in terms of the function itself.
The choice to adapt the circuit in Python was influenced by the selection of the f function and its derivative.
To generate the errors for the first layer, it is mandatory to update the second layer weights first. In the hardware implementation, we considered using control switches, but in the software version, it was easier to control the flow. The errors $\delta_{L1,i}$ are computed in the rows direction to mimic the transposition of the weight matrix present in the backpropagation algorithm. This method ensures accurate error propagation and weight updates throughout the network.

- **Update layer 1 synaptic weights based on layer 1 errors:** The layer 1 errors $\delta_{L1,i}$ from the second crossbar's rows are applied to the first layer with:

$$\delta_j = \left(\sum_k \delta_k w_{k,j} \right) \times f'(DP_j)$$

where neuron k is connected to the previous layer neuron j and $w_{k,j}$ is the corresponding synaptic weight.
The main difference here with respect to the update rule of the second layer is based on $\delta_{L1,i}$, because here:

$$\sum_j \delta_j w_{j,i} = \sum_j \delta_j (\sigma_{ij}^+ - \sigma_{ij}^-)$$

The term DP_j refers to the j-th column resulting dot product. In our case, we decided to estimate this value directly in the netlist by looking at the voltage at the final node of the column. The key impact of δ and x is to determine the direction of the weight update, while the magnitude is determined by DP_j and is consistent for all synapses of a particular neuron j.

$$\mathrm{DP}_j = \sum_i x_i w_{j,i} \tag{3}$$

$$w_{j,i,\text{new}} = w_{j,i,\text{old}} - \Delta w_{j,i} \tag{4}$$

The actual weight update value was based on this formula:

$$\Delta w_j = 2\eta \times \delta_j \times x$$

where η stands for the learning rate and x is the actual input voltage for that single memristor.

V. Discussion

We successfully implemented a method to update memristor weights using our proposed paradigm.
The continuous weight adjustments in LTSpice after each simulation allowed us to overcome some limitations of a fully hardware-implemented network. This training method marks a significant step towards achieving fully in-situ circuit training.
In the final part of the project period, our team evaluated the effectiveness of our method using two different approaches. First, we measured Mean Squared Error (MSE) loss, based the difference between the final output patter and the target. The results, as shown in Figure 4, were biased towards lower values, leading us to the conslusion that the initial memristor conditions and the parameter η could definitely mark substantial variations: we observed that even small variations (e.g., $10\,\Omega$) in memristor resistance can lead to disproportionately large deviations in the linear model.
Furthermore, we investigated convergence issues inherent again for the linear model. Figure 5 illustrates significant differences between the final outputs and the expected targets, highlighting the challenges we faced.
To introduce non-linearity and potentially improve convergence, we experimented with various memristor models, including those from [20]. Despite these efforts, we did not achieve satisfactory results in our training algorithm.
Amongst several challenges, our modifications provide valuable insights for future work. Additionally, we successfully bypassed the need for Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs), which was also one of our initial concern, nonetheless one of the biggest issue in hardware training algorithm since remarkably increase the area and power consumption.

A. Lessons Learned

This project imparted several crucial lessons:

- **Circuit Analysis**: We learned to analyze and manipulate circuits without solely relying on schematic components, deepening our understanding of circuit behavior.
- **Research and Curiosity**: Our exploration of memristive neural networks unveiled a vast array of possibilities, fueling our curiosity and drive to innovate.

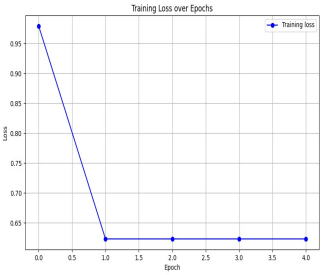

Fig. 4. MSE loss after 5 epochs, each epochs was aimed to learn a single pattern.

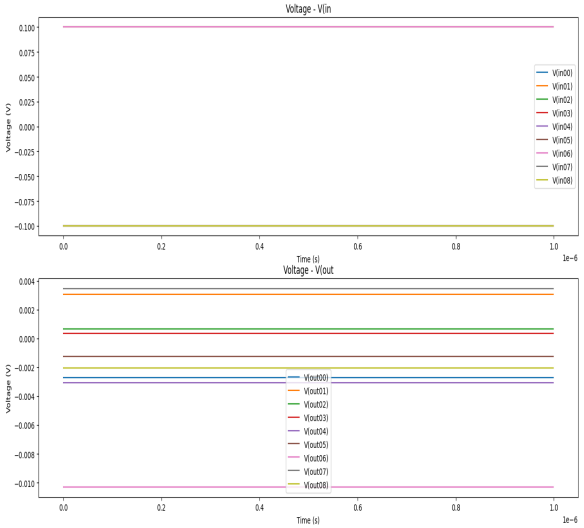

Fig. 5. Comparison between inputs and outputs voltages using the linear memristor model

- **Algorithm Adaptation**: We discovered that learning algorithms can vary significantly between software and hardware implementations.

Although we observed changes in conductances and output voltages, the limited training time and the limited linear memristor model prevented us from achieving optimal training. However, the insights gained lay a solid foundation for future endeavors.

B. Possible Next Steps

Looking ahead, several paths can enhance this work:

- **Hardware Integration**: Apply different hardware circuit that could enhance the hardware training capability, like the one shown in [43], trying to reproduce a more realistic circuit (with wire resistance, transistors and more realistic memristor model). Adding more layers, such as a single readout layer in order to scale the circuit for different tasks, like classification.
- **Training Procedures**: Investigating alternative training algorithms may yield better results and further validate our approach.
- **Reducing Software Dependencies**: Our ultimate goal is to eliminate software dependencies, achieving a fully hardware-implemented training mechanism. Initially, Python served as a preliminary control step to understand weight update processes. With more time, this understanding can guide the development of a fully hardware-based implementation, adapting components to handle non-idealities.

The potential for a fully hardware-based memristive neural network is immense, and we are excited about the prospects of further hardware integration. This project has provided a valuable learning experience and set the stage for future advancements in this field.

REFERENCES

[1] R. H. Dennard, F. H. Gaensslen, H. -N. Yu, V. L. Rideout, E. Bassous and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid State Circuits*, vol. SC-9, no. 5, pp. 256-268, Oct. 1974.

[2] G. E. Moore, "Cramming More Components Onto Integrated Circuits," in *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82-85, Jan. 1998.

[3] M. T. Bohr and I. A. Young, "CMOS scaling trends and beyond," *IEEE Micro*, vol. 37, no. 6, pp. 20–29, 2017.

[4] Rumelhart, D., Hinton, G. and Williams, R. "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[5] L. Chua, "Memristor-The missing circuit element," in *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507-519, September 1971.

[6] Strukov, D., Snider, G., Stewart, D. Williams, R. Stanley. "The missing memristor found," *Nature*, vol. 453, pp. 80–83, 2008.

[7] Li Y, Su K, Chen H, Zou X, Wang C, Man H, Liu K, Xi X, Li T. "Research Progress of Neural Synapses Based on Memristors," *Electronics*, vol. 12, no. 15, article 3298, 2023.

[8] Sun W., Gao B., Chi M., Xia Q., Yang J. Joshua, Qian H., Wu, H. "Understanding memristive switching via in situ characterization and device modeling," *Nat Commun*, vol. 10, article 3453, 2019.

[9] N. Schmitt et al., "Exploration of Bistable Oscillatory Dynamics in a Memristor from Forschungszentrum Jülich," in *Proceedings of the 12th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, Athens, Greece, 2023, pp. 269-273.

[10] A. Hardtdegen, C. La Torre, F. Cüppers, S. Menzel, R. Waser and S. Hoffmann-Eifert, "Improved Switching Stability and the Effect of an Internal Series Resistor in HfO2/TiOx Bilayer ReRAM Cells," *IEEE Transactions on Electron Devices*, vol. 65, no. 8, pp. 3229-3236, Aug. 2018.

[11] Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi. "Swim: Selective write-verify for computing-in-memory neural accelerators," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 277–282, 2022.

[12] Yu, Z., Yang, M., Finkbeiner, J., Siegel, S., Strachan, J.P., Neftci, E. "The Ouroboros of Memristors: Neural Networks Facilitating Memristor Programming," ArXiv, abs/2403.06712, 2023.

[13] M. Kasongo, T. Ytterdal, J. Lee, M. Rizkalla and M. Kumar, "Evaluation of Leakage Currents in Memristor Crossbar Arrays," in *NAECON 2023 - IEEE National Aerospace and Electronics Conference*, Dayton, OH, USA, 2023, pp. 269-273.

[14] L. Shi, G. Zheng, B. Tian, B. Dkhil and C. Duan, "Research progress on solutions to the sneak path issue in memristor crossbar arrays," *Nanoscale Advances*, vol. 2, no. 5, pp. 1811-1827, 2020.

[15] A. Bala, A. Adeyemo, X. Yang and A. Jabir, "Learning method for ex-situ training of memristor crossbar based multi-layer neural network," in *2017 9th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, Munich, Germany, 2017, pp. 305-310.

[16] Hasan, R., Yakopcic, C., Taha, T.M. "Ex-situ training of large memristor crossbars for neural network applications," *Analog Integr Circ Sig Process*, vol. 99, pp. 1–10, 2019.

[17] Alibart, F., Zamanidoost, E., Strukov, D. "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nat Commun*, vol. 4, article 2072, 2013.

[18] Mondal A., Srivastava A. "In Situ Stochastic Training of MTJ Crossbars With Machine Learning Algorithms," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, article 16, 29 pages, April 2019.

[19] Haghzad Klidbary, S., Javadian, M. "Hardware architecture and memristor-crossbar implementation of type-2 fuzzy system with type reduction and in-situ training," *J Supercomput*, 2024.

[20] Mladenov, V. "Advanced Memristor Modeling," MDPI: Basel, Switzerland, 2019.

[21] Peng Yao, Gao B., Tang J., Zhang Q., Zhang W., Yang, J. Joshua, Qian He. "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, pp. 641–646, 2020.

[22] LTSpice Website: https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html

[23] Jihong B., Jongbum W., Wooyoung S. "The rise of me transistors for neuromorphic hardware and In-memory computing," *Nano Energy*, vol. 126, p. 153698, 2024.

[24] O. Krestinskaya, K. N. Salama and A. P. James, "Analog Backpropagation Learning Circuits for Memristive Crossbar Neural Networks," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, 2018, pp. 1-5.

[25] Yang, L., Ding, Z., Xu, Y. et al. "Memristive crossbar-based circuit design of back-propagation neural network with synchronous memristance adjustment," *Complex Intell. Syst.*, 2024.

[26] R. Hasan and T. M. Taha, "Enabling back propagation training of memristor crossbar neuromorphic processors," in *2014 International Joint Conference on Neural Networks (IJCNN)*, Beijing, China, 2014, pp. 21-28.

[27] Schuman, C.D., Potok, T.E., Patton, R.M., Birdwell, J.D., Dean, M.E., Rose, G.S., Plank, J.S.. "A Survey of Neuromorphic Computing and Neural Networks in Hardware," ArXiv, abs/1705.06963, 2017.

[28] A. Shrestha, H. Fang, Z. Mei, D. P. Rider, Q. Wu and Q. Qiu, "A Survey on Neuromorphic Computing: Models and Hardware," in *IEEE Circuits and Systems Magazine*, vol. 22, no. 2, pp. 6-35, Secondquarter 2022.

[29] Wang, L., Meng, Q., Wang, H. et al. "Digital image processing realized by memristor-based technologies," *Discover Nano*, vol. 18, p. 120, 2023.

[30] Mattson MP. "Superior pattern processing is the essence of the evolved human brain," *Front Neurosci.*, 2014 Aug 22;8:265. PMID: 25202234; PMCID: PMC4141622.

[31] Liu, X., Zeng, Z. "Memristor crossbar architectures for implementing deep neural networks," *Complex Intell. Syst.*, vol. 8, pp. 787–802, 2022.

[32] Guo, M., Sun, Y., Zhu, Y. et al. "Pruning and quantization algorithm with applications in memristor-based convolutional neural network," *Cogn Neurodyn*, vol. 18, pp. 233–245, 2024.

[33] A. Henderson, C. Yakopcic, C. Merkel, H. Hazan, S. Harbour and T. M. Taha, "Memristor Based Liquid State Machine With Method for In-Situ Training," *IEEE Transactions on Nanotechnology*, vol. 23, pp. 376-385, 2024.

[34] Wang, Z., Li, C., Lin, P. et al. "In situ training of feed-forward and recurrent convolutional memristor networks," *Nat Mach Intell*, vol. 1, pp. 434–442, 2019.

[35] Sun, J., Xiao, X., Yang, Q., Liu, P., Wang, Y. "Memristor-based Hopfield network circuit for recognition and sequencing application," *Aeu-international Journal of Electronics and Communications*, vol. 134, article 153698, 2021.

[36] H. Xiao, Y. Zhou, T. Gao, S. Duan, G. Chen and X. Hu, "Memristor-Based Light-Weight Transformer Circuit Implementation for Speech Recognizing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, no. 1, pp. 344-356, March 2023.

[37] Hasan, R., Yakopcic, C., Taha, T.M. "Ex-situ training of large memristor crossbars for neural network applications," *Analog Integr Circ Sig Process*, vol. 99, pp. 1–10, 2019.

[38] L. T. Chabane, D. -K. G. Pham and P. Desgreys, "Analysis of The Manhattan Update Rule Algorithm," in *Proceedings of the 2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Glasgow, United Kingdom, 2022, pp. 1-4.

[39] E. Zamanidoost, F. M. Bayat, D. Strukov and I. Kataeva, "Manhattan rule training for memristive crossbar circuit pattern classifiers," in *2015 IEEE 9th International Symposium on Intelligent Signal Processing (WISP) Proceedings*, Siena, Italy, 2015, pp. 1-6.

[40] Adhikari, S.P., Kim, H., Budhathoki, R.K., Yang, C. and Chua, L.O., 2014. "A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 215-223.

[41] Wang, C., Xiong, L., Sun, J. et al. "Memristor-based neural networks with weight simultaneous perturbation training," *Nonlinear Dyn*, vol. 95, pp. 2893–2906, 2019.

[42] Cong Xu, Chunhua Wang, Yichuang Sun, Qinghui Hong, Quanli Deng, and Haowen Chen. "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications," *Neurocomput*, vol. 462, pp. 581–590, October 2021.

[43] Mladenov, V.; Kirilov, S. "A Memristor Neural Network Based on Simple Logarithmic-Sigmoidal Transfer Function with MOS Transistors". *Electronics* 2024, 13, 893.

[44] Memristor Models in LTSpice folder link: https://github.com/knowm/memristor-models-4-all

[45] PyLTSpice library: https://pypi.org/project/PyLTSpice