Journal of Systems Architecture 131 (2022) 102730

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Feed-Forward learning algorithm for resistive memories

Dev Narayan Yadav [a], Phrangboklang Lyngton Thangkhiew [b,*], Kamalika Datta [c], Sandip Chakraborty [a], Rolf Drechsler [c,d], Indranil Sengupta [a,e]

[a] *Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India*
[b] *Department of Computer Science and Engineering, Indian Institute of Information Technology Guwahati, India*
[c] *DFKI, German Research Centre for Artificial Intelligence, Bremen, Germany*
[d] *Institute of Computer Science, University of Bremen, Bremen, Germany*
[e] *Department of Computer Science and Engineering, JIS University, India*

ARTICLE INFO

Keywords:
Feed-Forward network
Memristor
Neural network
Resistive memory
Training algorithm

ABSTRACT

Resistive memory systems, due to their inherent ability to perform Vector–Matrix Multiplication (VMM), have drawn the attention of researchers to realize machine learning applications with low overheads. In resistive memory systems, each memory cell (synapse/neuron) stores a weight in the form of resistance/conductance value. Memristor-based resistive memory has been widely explored in this regard because of its small size and low power consumption. The inference quality of a neural network depends on how efficiently and accurately the weights are stored in the synapses. The weights are calculated using various training algorithms, like back-propagation (BP), least mean square (LMS), and random weight change (RWC). The training accuracy of existing algorithms is directly related to the algorithm complexity and the time devoted for training. This paper presents a training algorithm that requires an additional set of memristors and a threshold gate for training and achieves an accuracy similar to existing algorithms without using any complex circuitry. The method can update synapse weights in parallel and requires fewer epochs for training an application. Results on experiments with standard benchmarks reveal that the method can achieve an average speedup of 38× as compared to state-of-the-art methods.

1. Introduction

Neuromorphic computing has become popular in various applications such as pattern recognition, data analysis, audio and video processing, etc. and often demands low power consumption and high performance. The two key steps in the operation of a neural network are: (a) training phase, and (b) inference phase. The core operation during the training phase is *Vector-Matrix Multiplication* (VMM), where the weight values for synapses (neurons) are calculated. VMM operations are also required during the inference phase, but only to produce the final output. Mathematically, each VMM operation requires a large amount of data processing, and hence it is expensive both in terms of power consumption and latency when performed on conventional architectures [1,2].

Resistive random access memory (ReRAM) technologies allow both storage and processing to be carried out in the same hardware fabric. Consequently, such *in-memory computing* [3–5] architectures require fewer data transfer between processor and memory during computation. Hence VMM operations using ReRAM can be performed faster as compared to conventional systems [6–9]. In a ReRAM-based implementation, the weights are stored in the form of resistance/conductance values in the cells. Among the various competing technologies, memristors are considered as one of the most desirable candidates for this kind of application due to their non-volatile nature, in-memory computing capability, low power consumption, dense layout, and higher write endurance [10–12].

For a neural network implemented using memristive crossbars, inference quality depends on how accurately the weights can be stored in the memristors during training. In such systems, training can be performed in two ways: (a) Host System Based Training (HSBT), and (b) Hardwired Algorithm Based Training (HABT). In the HSBT approach, an external system (called host) carries out all necessary computations needed for weight calculation in software, and then the final weights generated are downloaded on the crossbar. The HSBT approach can be further classified into two types based on the working style: *direct downloading* and *chip-on-the-loop* approach. In contrast, for the HABT

* Corresponding author.
E-mail addresses: dev.narayan@iitkgp.ac.in (D.N. Yadav), phrangboklang.thangkhiew@iiitg.ac.in (P.L. Thangkhiew), kamalika.datta@dfki.de (K. Datta), sandipc@cse.iitkgp.ac.in (S. Chakraborty), drechsler@uni-bremen.de (R. Drechsler), indranil.sengupta@jisuniversity.ac.in (I. Sengupta).

https://doi.org/10.1016/j.sysarc.2022.102730
Received 7 March 2022; Received in revised form 29 July 2022; Accepted 29 August 2022
Available online 3 September 2022

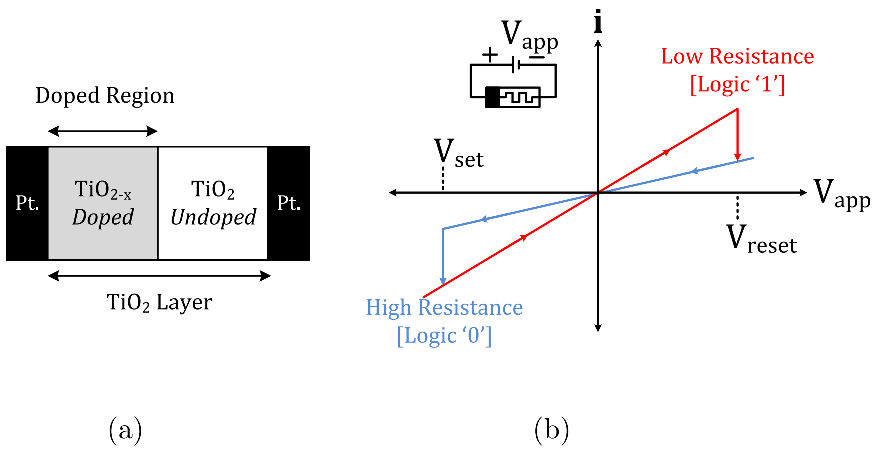


Fig. 1. (a) TiO_2-based memristor device, (b) Ideal V-I characteristic.

approach all computations are performed by hardware, and is considered efficient in terms of time and accuracy but expensive in terms of hardware complexity.

This paper proposes a training algorithm that updates constant weights only in the forward direction, and requires some additional memristors and threshold logic for training. The main contributions can be summarized as:

(a) propose a training algorithm to improve the convergence rate as compared to the state-of-the-art methods like BP, RWC, and OCTAN;
(b) reduce the hardware complexity by eliminating processes such as derivative, error calculation, and back-propagation;
(c) reduce the requirement of high precision memristors by using additional memristors; and
(d) update weights in parallel to reduce training time.

The rest of the paper is organized as follows. Section 2 describes the background and related works. Section 3 discusses the proposed learning scheme. In Section 4, performance analysis of the proposed scheme is carried out. Experimental results are presented in Section 5 along with comparison with state-of-the-art works. Section 6 concludes the paper with brief discussion on future extensions.

2. Preliminary

2.1. Memristor

Memristor is a passive circuit element [13] that is capable of remembering the total amount of charge that has passed through it. The first TiO_2-based memristor device was fabricated by HP Lab in 2008 as shown in Fig. 1(a) [14]. The device consists of two regions, *doped* and *undoped*, where the resistance of the doped region is lower as compared to that of the undoped region. The boundary between the two regions can be moved by applying a suitable voltage across the device, thereby changing the overall resistance. The V-I characteristics of the device depict a pinched hysteresis loop as shown in Fig. 1(b). Depending on the polarity of the applied voltage, the device switches between two distinct resistive states, *high resistance state (HRS)* and *low resistance state (LRS)*, which are treated as logic 0 and logic 1 respectively for logic design applications.

The threshold voltages V_{set} and V_{reset} are used to initialize the memristor to one of the two stable states. When a voltage V_{set} is applied, the width of the doped region expands, and the resistance decreases. Similarly, the voltage V_{reset} causes the doped region to shrink, thereby increasing the resistance.

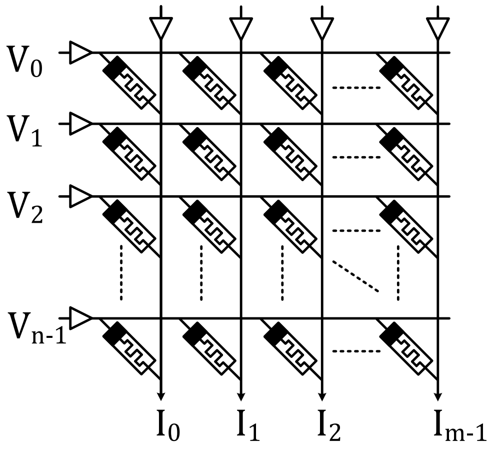


Fig. 2. An $n \times m$ memristor crossbar for VMM operation.

2.2. Memristor crossbar for neuromorphic computing

A crossbar is fabricated using a set of perpendicular nanoscale wires in two layers [3,4], with a memristor placed at each junction. This structure realizes a ReRAM system where both storage and computation can be performed. This computing paradigm is called *in-memory computing* [5,12]. The crossbar can also be used for neuromorphic applications as shown in Fig. 2(a), where the memristors act as synapses that store weights in the form of resistance/conductance.

During neuromorphic computation, the VMM operation can be directly carried out on the crossbar, as $I = V \times M$ [15,16]. Here, the vector $V = \{V_0, V_1, \dots, V_{n-1}\}$ represents voltages that are applied on the rows (i.e. the inputs), and the vector $I = \{I_0, I_1, \dots, I_{m-1}\}$ represents the currents that are flowing along the columns (i.e. the outputs). M represents an $n \times m$ matrix, where the (i, j)th co-efficient represents the conductance value of the memristor in row R_i and column C_j. The VMM operation does not require any explicit data transfer or arithmetic operation, as the data (weights) are already available in the crossbar. The architecture is scalable for reasonably larger applications as well. Several research works have used memristor crossbars to accelerate neuromorphic computation [6,8,9,15,16].

In neuromorphic applications, the memristors store the weights as conductance values. Such memristors are referred to as *multi-bit memristors*. The use of multi-bit memristors has been shown in various works [17,18], which provides a higher inference rate compared to single-bit memristors as analyzed in [19]. However, the non-linear switching characteristics of memristors makes it challenging to achieve

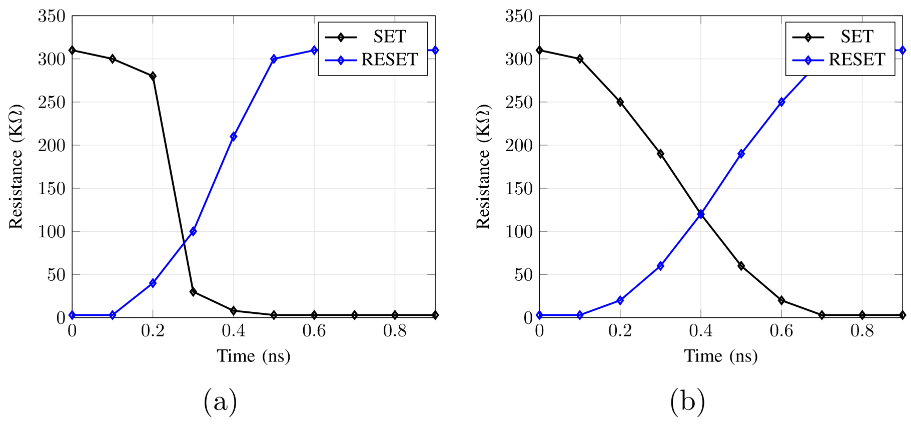


Fig. 3. Switching characteristics of memristor: (a) Stanford Model [20], (b) VTEAM model [21].

higher bit resolutions. The switching behaviour of two well-known memristor models is shown in Fig. 3.

From Fig. 3 we observe that for the VTEAM model [21], the switching behaviour is approximately linear, while for the Stanford model [20], it is nonlinear. For multi-bit storage, memristors with linear switching behaviour are more desirable as they require simpler control circuits for setting to intermediate states. As such not much study has been carried out to compare the linear and non-linear models of memristor for use in neuromorphic applications.

For non-linear models, as the bit capacity increases the design of the crossbar controller becomes more complex. The authors in [22] proposed a memristor-based bridge synapse structure that allows efficient programming for multi-bit storage. The bridge circuit is also capable of storing negative weights. However, it is not suitable for crossbar implementation, which is the natural way to fabricate larger memristor arrays. For multi-bit memristors, most of the prior works rely on the application of a short voltage pulse across the devices to change the weights [22–24].

2.3. Learning approach for memristor-based neural networks

We now briefly discuss the various learning approaches for crossbar-based neural networks, which can be trained in one of the following ways.

(a) *Host System Based Training (HSBT)*: Here, training is performed using an external system called the host. The key advantage is that we can use any suitable algorithm for weight calculation. This method can be further divided into the following two types.

  (i) *Direct Downloading Approach:* Here, training is carried out in a complete off-line mode where the crossbar does not interact with the host until the final weights are computed. The weights are computed by the host based on the training algorithm using software. The final calculated weights are then written into the crossbar. This method is straightforward; however, as the chip does not interact during training, the outputs of the actual VMM operation may differ and the network might need to be trained again in case of large differences [19].

  (ii) *Chip-on-the-Loop Approach:* Here, after each weight update in the host, the same update is also performed on the crossbar, and the error is calculated based on the outputs generated by the crossbar. This increases the quality of VMM operation as the host and the crossbar interact with each other, and the new weight is computed based on the error generated by the crossbar itself. However, the frequent interaction between the host and the crossbar slows down the training process. This approach also requires additional circuits for interfacing the crossbar with the host system [22,25].

(b) *Hardwired Algorithm Based Training (HABT)*: In this approach, training is performed using a learning algorithm implemented in hardware, which is efficient in terms of speed and accuracy. However, the main challenge is the physical implementation and adaptability [24,26,27].

2.4. Popular learning algorithms

The most popular algorithm for training neural networks is Back Propagation (BP), which provides higher accuracy as compared to other learning algorithms. However, the method requires a large number of read/write operations, making it expensive in terms of time and energy. The complex computational blocks such as derivative for the output, error-back propagation block, etc. make it difficult for physical implementation as well. The BP approach and its variants have been used in various works where the crossbar is used as a neuromorphic accelerator. However, very few works discuss hardware implementation due to the complexity involved [23,26,27].

The Random Weight Change (RWC) algorithm [28,29] is simple for hardware implementation as it only requires a random weight generator for training the network. The weight of each synapse is changed randomly by a fixed amount at every iteration. The change is accepted if the quality of VMM increases; otherwise, it is discarded. However, the algorithm requires a large number of training cycles to achieve a reasonable inference rate.

The work in [30] combines the BP and RWC algorithms. First, the network is trained using the BP algorithm using the chip-on-the-loop training approach, and then a RWC-based hardware algorithm is applied.

The Stochastic Least Mean Squares (SLMS) approach [31,32] trains the network by applying constant weights in the last layer of the network, and as such is simpler as compared to BP and RWC. However, in this approach the hidden layer weights are assumed to be random, and thus can be used only for neural networks with no hidden layers.

In the hardwired learning algorithm called OCTAN [24], the weights of each memristor are changed by a small value, and the effect of weight update is assessed immediately. If the change shows a positive effect, then the network accepts that change; otherwise, the change performed previously is performed twice in the reverse direction. The method provides faster convergence than BP and RWC approaches; however, it requires a high buffer space to store the update data (the location and magnitude of each update). If most of the changes are performed in the wrong direction, the number of cycles required may increase, resulting in higher energy consumption.

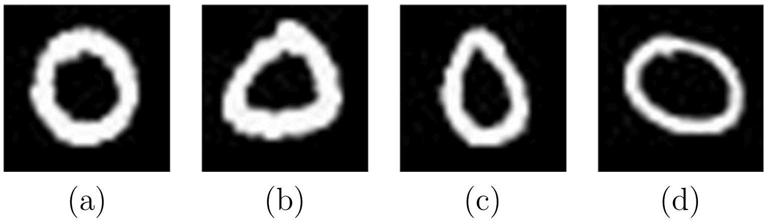
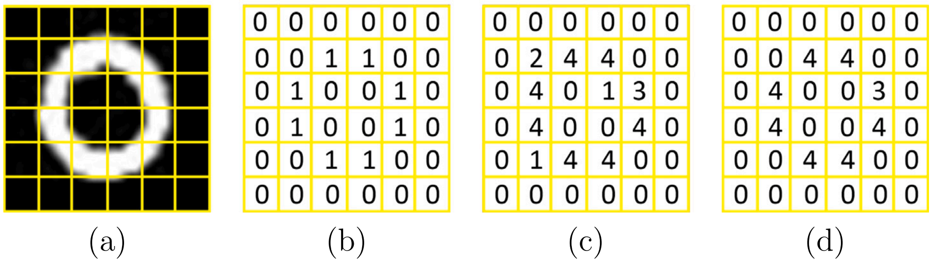


Fig. 4. Sample images of handwritten digit 0 (Training Set).



(b)

0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	0	1	0
0	0	1	1	0	0
0	0	0	0	0	0

(c)

0	0	0	0	0	0
0	2	4	4	0	0
0	4	0	1	3	0
0	4	0	0	4	0
0	1	4	4	0	0
0	0	0	0	0	0

(d)

0	0	0	0	0	0
0	0	4	4	0	0
0	4	0	0	3	0
0	4	0	0	4	0
0	0	4	4	0	0
0	0	0	0	0	0

Fig. 5. Illustration of pixelization of an image.

3. Proposed feed-forward training algorithm

In this section, we discuss the proposed training approach that requires additional memristors and threshold logic for implementation. The method allows parallel weight update in the crossbar, whereby the weights in multiple synapses (memristors) can be updated in a single cycle. The proposed algorithm takes the training set corresponding to each of the classes as input, and trains the network by computing the weights of the synapses.

3.1. Basic concept

We first discuss the basic functioning of the algorithm, where the aim is to extract useful information from the given training set. For the sake of illustration, we shall use the input patterns as shown in Fig. 4, which are various images of the handwritten digit '0'.

For the given images, the pixel values are the main attributes based on which the classification is carried out. During training, we look for similar pixel positions in the images that belong to the same class, since similarities in the attributes within the same class can be observed. The inputs are originally 28 × 28 MNIST handwritten digit dataset [33]; however, for the sake of illustration, the digits are shown as 6 × 6 images. Assume that each pixel represents a value of either 0 or 1, where the value 1 indicates that the region is covered by more than 50% of white colour; otherwise, it is 0. Thus, the image shown in Fig. 5(a) will have pixel values 0, 0, 0, 0, 0, 0 for row 1, pixel values 0, 0, 1, 1, 0, 0 for row 2, and so on, as shown in Fig. 5(b).

If we compute the 0/1 matrices for all the images given in Fig. 4 and add them up, we can get information about the common pixel locations. The 6 × 6 summed up matrix is shown in Fig. 5(c). If we now apply a threshold filter (say, with threshold 2), we can extract the highest common pixels (or discard the less matching pixels) as shown in Fig. 5(d). In this way, we can extract useful information from the input images.

We use a simple pattern recognition application to demonstrate the usefulness of the *information matrix* (summed up matrix). We compare a given test image with the information matrix. The output will be the sum of values stored in the information matrix for which the input images are covered with white pixels. Fig. 6 depicts the addition of input images for digits 2, 5, 1 and 0 to the information matrix, and all of these are compared with the information matrix that has been computed for digit 0. The outputs for respective images are computed as 12, 20, 0, 27. It can be observed that we are getting highest output for the last pattern, implying that it is capable of identifying the correct digit (here, 0).

3.2. The overall learning framework

The overall flow of the proposed learning algorithm is depicted in Fig. 7. We assume that N is the number of input attributes, and M is the number of classes in the classification problem. In the figure, NN-chip refers to the $N \times M$ memristor crossbar that is used in learning, and the additional memristors (AM) denote an $N \times 1$ memristor array used for thresholding. In the algorithm, inputs are applied one at a time from an input batch $IB = \{IB_0, IB_1, \ldots, IB_{M-1}\}$, where each batch IB_i consists of K sub-batches $IB_i = \{IB_{i1}, IB_{i2}, \ldots, IB_{iK}\}$. Let p denote the number of inputs in IB_{ij}, for all $0 \leq i < M$, $1 \leq j \leq K$.

The sub-batches are applied for training in sequential order of the classes. For each class i, every sub-batch IB_{ij} will first update the AMs. Based on the values stored in the AMs, the memristors in column C_i of the NN-chip will be updated. For the MNIST dataset, each image has 28 × 28 pixels, where each pixel is treated as a single attribute, and hence $N = 784$.

When an input $inp \in IB_{ij}$ is applied, it will be filtered by the threshold block. The use of the threshold block is optional; however, for applications like pattern recognition, it can be useful as by setting a threshold value, we can decide how much coverage of the pixel/pattern is to be considered. If the given input attribute is greater than the threshold, a voltage V_{sc} sufficient to increase the conductive state by one unit is applied to the respective memristor in AM. The process is repeated until there are no inputs left.

After this stage, the AMs are loaded with useful information in the form of weights. The weights of the NN-chip memristors in the column corresponding to the class in which input inp belongs are then updated. A threshold filter is applied at this level to achieve weight updation, which helps to classify the most and least matched information between the inputs of any sub-batch IB_{ij}. Based on the filtered values stored in AM after thresholding, the weights are updated in the respective column of the NN-chip. All the AMs are then reset to state 0, and the process is repeated for the next input batch. The weight updation is carried out in a manner similar to MAGIC [34] operation, where the memristors present in the same row or same column can be processed together. This process of weight updation in AM and NN-chip can be performed in parallel in a single cycle.

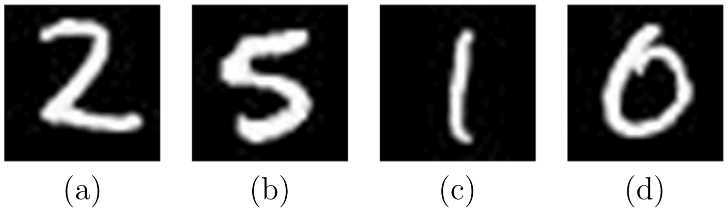


Fig. 6. Sample images for classification (Test Set).

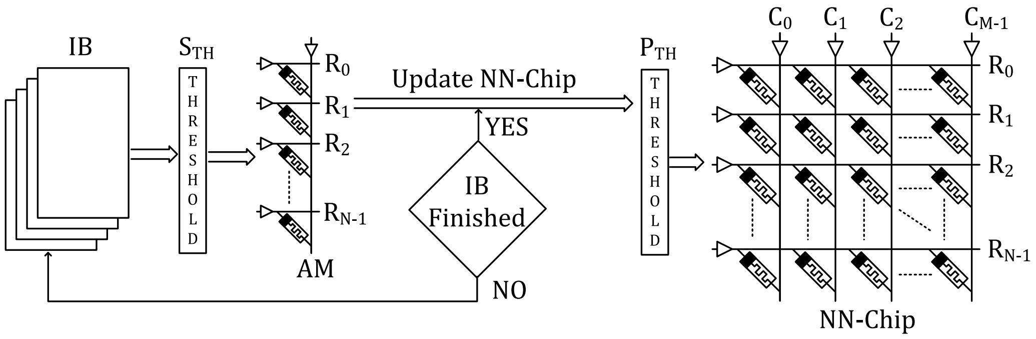


Fig. 7. Proposed architecture for learning.

Table 1
Micro-instruction for weight updation.

Cycle	V_{sc}	GND
1	R_0, R_2, R_3	AM_c

The steps in the learning process are depicted as Algorithm 1. The inputs to the algorithm will be $N \times M$ crossbar (NN-chip), AM, primary threshold P_{TH}, secondary threshold S_{TH}, and the input batches (IB). Algorithm 2 shows how the weights are updated in the AM and NN-chip. In the proposed algorithm, the weights are updated first in AM, and then based on the updated values, weight updation is carried out in the NN-chip. The algorithm generates the weights for the $N \times M$ crossbar as output. We assume that the number of IBs available for each class is K, and there are p inputs within each IB.

3.3. An illustrative example

We now illustrate the process of parallel weight updation with the help of an example. Consider a problem with 4 classes ($M = 4$), where each data has 5 attributes ($N = 5$). In this case, the size of NN-chip will be 5 × 4, and the number of AMs required will be 5. The actual attribute values shall vary depending upon the type of dataset. For our illustration, we assume that the attributes are numerical values in the range 0 – 100.

An example of weight updation in the AMs is shown in Fig. 8. Assume that the input attributes for class 1 are 40, 05, 85, 53, 09, and the values stored in the AMs are 0, 5, 4, 7, 1 (see Fig. 8(a)). The value of S_{Th} is set to 10. For all the attributes having value $\geq S_{Th}$, the voltage V_{sc} will be applied to the respective rows, and the column connected to GND as shown in Fig. 8(b). The operation performed is similar to that of Memristor Aided Logic (MAGIC) [34]. Various works [35–39] have used the MAGIC design style to map logic gates in the crossbar array, in which the gates can be evaluated in parallel. In a similar way, we have presented the micro-instruction for the parallel weight updation in the AMs as shown in Table 1. This will result in the weights of AMs being updated as 1, 5, 5, 8, 1 as shown in Fig. 8(c).

Now consider that the IBs for class 1 has been processed, and the final weight stored in the AMs are 2, 8, 7, 9, 1 as shown in Fig. 9(a).

Algorithm 1 Feed-Forward Learning Algorithm

```
    INPUT: N × M crossbar (NN-chip), AM, IB, $P_{TH}$ and $S_{TH}$
    OUTPUT: Weighted NN-chip
 1: $i$ ← [0 to M − 1]
 2: $j$ ← [1 to K]
 3: $inp$ ← [1 to p]
 4: $AM_c$ ← column of AM
 5: for each $i$ in IB do
 6:   $C_i$ ← $i^{th}$ column of NN-chip
 7:   for each $j$ in IB[$i$] do
 8:     for each $inp$ in IB[$i$][$j$] do
 9:       AM ← Weight_Update(AM, $inp$, $S_{TH}$, $AM_c$)
10:     end for
11:     $W_{AM}$ ← weights of AM
12:     NN-Chip ← Weight_Update(NN-chip, $W_{AM}$, $P_{TH}$, $C_i$)
13:     RESET AM
14:   end for
15: end for
16: return Weighted NN-chip
```

i. IB: input batch, where IB[i] represents the input batch ∈ class i, and IB[i][j] represents the sub-batch j ∈ class i.
ii. P_{TH}: Primary Threshold, is a model hyper-parameter which decides whether the memristors available at NN-chip will be updated or not.
iii. S_{TH}: Secondary Threshold, is a model hyper-parameter which decides whether the respective additional memristor will be updated or not.
iv. inp: input attribute, which have N number of attributes.
v. RESET AM (reset additional memristors): apply reset voltage V_{reset} to all rows. Upon applying RESET, all additional memristors will be set to low conductance value (minimum state).

Currently, the memristors available at column C_0 (corresponding to class 1) have weights 0, 3, 4, 2, 0 as shown in Fig. 9(b). The threshold value P_{TH} is set to 6. Thus for all the AMs having value $\geq P_{TH}$, a voltage V_{sc} will be applied to respective rows, and column C_0 will be connected with GND in NN-chip as shown in Fig. 9(c). The micro-instruction for the weight updation is shown in Table 2. The operation

Algorithm 2 Weight_Update(Crossbar, Weight, threshold, col)

```
1: $r \leftarrow$ number of rows in Crossbar
2: for (n ← 0 to r-1) do
3:     if Weight[n] ≥ threshold then
4:         $\text{Crossbar}_{row}$.append(n)
5:     end if
6: end for
7: $\text{Crossbar}_{row} \leftarrow V_{sc}$
8: col ← GND
9: RETURN Crossbar
```

i. V_{sc} : a voltage value which is sufficient to change the weight of a memristor.
ii. Crossbar_{row} : ∀ rows ∈ Crossbar for which the respective weights are greater than the threshold value.

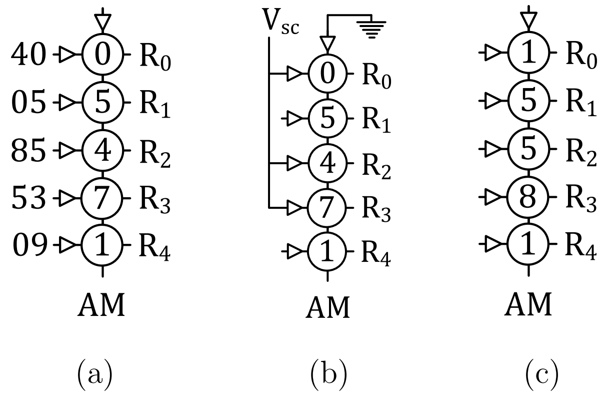


Fig. 8. Weight update in additional memristors: (a) Initial values with inputs, (b) Weight updation, (c) AM after weight updation.

Table 2
Micro instruction for weight updation in NN-chip.

Cycle	V_{sc}	GND
1	R_1, R_2, R_3	C_0

will update the weights in column C_0 to 0, 4, 5, 3, 0 as shown in Fig. 9(d).

It may be observed that weight updation in AM and NN-chip can be performed in parallel (i.e. in a single cycle). The number of cycles required for one batch will be $K \times (p+1)$, and for the complete training set we require $M \times K \times (p + 1)$ cycles.

4. Performance analysis of the algorithm

In this section, we analyse the performance of the proposed algorithm, and compare the results on various benchmark datasets with those for existing memristor-based crossbar learning algorithms. For analysis HSBT approach is used. The training in HSBT approach is performed using an external system called the host system. The key advantage is that we can use any suitable algorithm for weight calculation. This method can be further divided into two types, as discussed in the literature part, out of which we are using the direct-downloading approach. In the direct-downloading approach, training is carried out in a complete off-line mode where the crossbar does not interact with the host system until the final weights are computed. The weights are computed by the host system based on the training algorithm. The final calculated weights are then written into the crossbar for verification using Cadence Virtuoso. This method is straightforward; however, as the chip does not interact during training, the outputs of the actual VMM operation may differ, and the network might need to be trained again in case of large differences [19]. We have chosen the following benchmark datasets for analysis: *MNIST*, *Fashion MNIST*, *Cifar-10*, and *Breast Cancer Wisconsin (BCW)* [33,40–43]. In this study, we have carried out analysis on crossbars containing memristors with 2 – 8 bit storage resolution. In general, a memristor with k-bit storage resolution can have 2^k conducting states.

4.1. Characteristics of the proposed algorithm

The following properties of the proposed algorithm make it simpler as compared to state-of-the-art methods.

(a) *Circuit Requirement:* It does not require any additional circuits like error propagation block, derivative block, multiplicative block, etc.
(b) *Direction of Weight Update:* The direction of weight update is fixed in the forward direction (does not need to identify in which direction the weight needs to be updated).
(c) *Magnitude of Weight Update:* The weights are updated with a constant positive step value (by +1 towards higher conductive state), and does not require any circuitry to identify the magnitude.

4.2. Required bit resolution of the memristors

The proposed algorithm significantly simplifies the training process. However, the selection of the sub-batch size and bit-storage capability of the memristors can affect the accuracy of the training process. For example, consider a crossbar consisting of 3-bit memristors with three classes, and learning is performed on some dataset (say, D), which consists of 150 data out of which 126 (42 for each class) are used for training, and 24 (8 for each class) are used for testing. Using 3-bit memristors, the maximum forward update operation that can be performed in any memristor is $2^3 - 1 = 7$; thus, the size of a sub-batch is limited to $p = 7$. With this configuration, 6 sub-batches of each class can be used to train the network. We can also use 7 sub-batches each of size 6. However, with these configurations (7 sub-batches of size 6, or 6 sub-batches of size 7), the training data will be covered in one step (this is called *one epoch* in training terminology). With one epoch, it is challenging to minimize the error. As the proposed approach only applies forward update, shuffling the dataset and applying multiple epochs are required to achieve higher accuracy and avoid over-fitting. It is evident that the use of 3-bit memristors limits the number of epochs and hence reduces the overall accuracy. Therefore, to enhance the training accuracy we need to use memristors with higher bit resolution.

Now consider a crossbar with 4-bit memristors, which allows a maximum sub-batch size of $2^4 - 1 = 15$. For this configuration, the network can be trained with up to 3 epochs for the same dataset D (7 sub-batches of each class with sub-batch size 6). It is also possible to have 3 sub-batch of each class with sub-batch size 16, which will allow the training for up to 5 epochs. This configuration allows shuffling of data and also increases the number of epochs.

It is not easy to identify the exact number of epochs needed for any dataset to achieve the desired training accuracy, as it depends on the diversity of the dataset. Furthermore, it can be noted that it may be difficult to attain minimum training error with only one epoch. Thus, a crossbar with multi-bit programmability is desirable, which would allow the data to pass multiple times during training.

Table 3 shows the required bit resolutions of memristors in the crossbar, which can train a given application for multiple epochs. The first column shows the name of the application, and in the second column, a 4-tuple set indicating the bit-requirement, number of batches for each class, batch size and possible epochs.

For calculating the bit requirement we can either consider the full dataset or assume that ≈80% of the data will be used by the training algorithm. Since the number of training and test data for the *MNIST*,

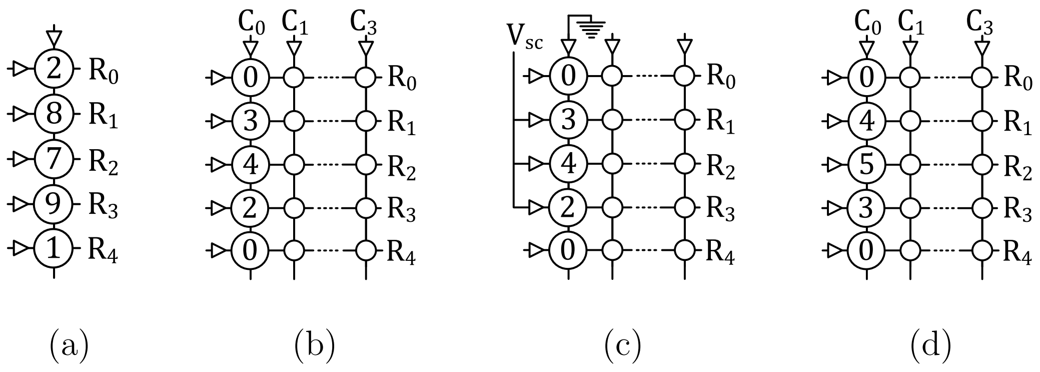


Fig. 9. Weight updation in NN-chip: (a) Weights available in AM, (b) Weights available in NN-chip, (c) Updating weights of NN-chip, (d) NN-chip with updated weight.

Table 3
Bit requirement for different applications.

Application	(Bit-Size::Num Sub-Batch:Sub-Batch-Size:Epochs)
BCW	(5-bit::10:32:3), (6-bit::10:32:6), (6-bit::6:40:10)
MNIST	(7-bit::60:100:2), (7-bit::44:128:3), (8-bit::60:100:4),
FMNIST	(8-bit::44:128:5), (8-bit::38:150:6), (8-bit::30:200:8),
CIFAR-10	(8-bit::24:250:10), (9-bit::30:200:17), (9-bit::24:250:22)

FMNIST and *CIFAR-10* datasets are approximately the same, their bit requirements are similar and are shown together in the same row.

From the table we can observe that if we minimize the number of batches and maximize the number of training data in a single batch as much as possible, we can run the dataset for more epochs using memristors with same precision.

The primary motivation of using additional memristors (AM) is to avoid using high precision memristors in NN-chip. Consider the training of the MNIST dataset directly in the crossbar (without considering the AMs and back-propagation approach) and assume that the algorithm only allows forward weight change. In this case, we will require a memristor that supports at least 14-bit storage capability (for ≈ 8 epochs). However, by using one level of AM, the storage resolution can be reduced to 8-bit as shown in Table 3. For larger applications with higher number of training sets, the number of AM levels can be increased in order to reduce the higher bit requirement. However, additional overhead will be incurred in the controller and threshold implementation.

4.3. Training of different applications

In the proposed approach, we do not use any error calculation circuitry to identify the training error. To check the progress of the training, we can apply some training sets randomly to the network after each epoch. Fig. 10 shows the training accuracy of different applications achieved during software training using this approach. The accuracy values are based on the progress data applied after each epoch to test if the algorithm is performing correctly. The figure illustrates that the algorithm performs as expected.

Fig. 11 shows the accuracy achieved by the algorithm for the trained applications with various bit storage capabilities of memristors. It can be observed that the multi-bit storage capability of memristors should be sufficient to cover all the input data to achieve good test accuracy. For example, the test accuracy of MNIST data trained in 2-bit memristor is only 63% because it can only cover 160–180 training sets. The accuracy increases to 94% with 8-bit memristor, as it allows ≈10 epochs to run. Increasing bit storage increases the accuracy, which can be observed for other datasets as well.

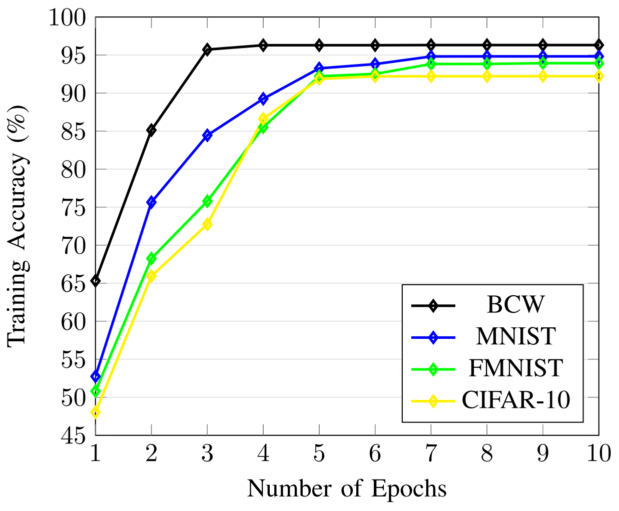


Fig. 10. Progress of training for application using our proposed method.

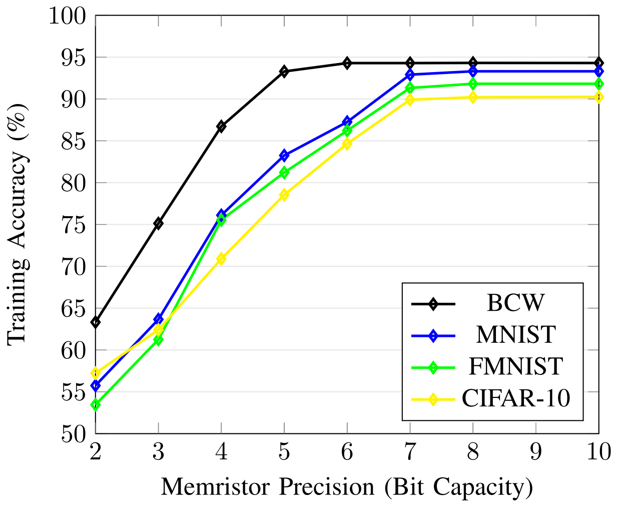


Fig. 11. Test accuracy for different applications using our proposed method.

4.4. Cycles and read/write analysis

The number of cycles required for the proposed algorithm depends on the size of the dataset and the size of sub-batches (IB_{ij}). We have already discussed the upper limit of sub-batch size and the minimum number of epochs required. Consider the data size as D, the data belonging to one class is considered as a single batch. Thus, the number

Table 4
Average cycles and write operations for various benchmarks.

Data-set	Property of FCNN			# Cycle	# Write
	Net-Size	# Mem	Precision		
BCW	9 × 2	27	6	3.2×10^3	7.68×10^4
MNIST	784 × 10	8624	8	4.3×10^5	11.45×10^9
FMNIST	784 × 10	8624	8	6.06×10^5	17.37×10^9
CIFAR-10	1024 × 10	10 304	8	6.32×10^6	2.06×10^{10}

Table 5
Parameters for simulation.

Parameter	X_{on}	X_{off}	W_{on}	W_{off}	R_{on}	R_{off}
Value	9 nm	0 nm	5 nm	0.5 nm	3 kΩ	2.66 MΩ

of class and number of IBs will be the same (i.e. M). We are assuming that each class will have the same number of training inputs, and thus the number of sub-batches for each IB_i will be K, and the size of each sub-batch IB_{ij} is p.

As each training input will require one cycle to update the AMs, p cycles will be required for a full sub-batch to update the AMs. One cycle will be required to update the NN-chip, and one cycle will be required to reset the AMs. The total number of cycles required to process a sub-batch can be given as:

$$C_{IB_{ij}} = p + 2 \tag{1}$$

The number of cycles required to process a full class (IB) can be estimated as the product of the number of cycles required for a single sub-batch and the number of sub-batches of that class:

$$C_{IB} = C_{IB_{ij}} * K \tag{2}$$

The number of cycles required to complete training of the full data set for one epoch can be given as:

$$C_D = C_{IB} * M \tag{3}$$

As the algorithm requires an average of 8 ≈ 10 epochs to achieve the required accuracy, the average number of cycles required for training will be $10 \times C_D$.

All the applications have been trained using a fully-connected neural network (FCNN). The proposed algorithm requires weight updation only in the forward direction. For this purpose, a short pulse generator circuit available in literature can be used [22,23]. We can count the number of write operations performed by the algorithm for a given dataset. However, the exact number of write operations required will depend on the threshold values of both levels, and hence an exact value cannot be estimated. We can also estimate the total number of memristors required as the sum of those required in NN-chip and AMs.

Table 4 summarizes the results for various datasets. The first column gives the name of the dataset, the second column shows the size of the network, the third column indicates the number of memristors required to train the application, the fourth column shows the precision (in bits) of the memristors, followed by the average number of cycles and number of write operations respectively.

4.5. Energy consumption of the crossbar

The proposed training algorithm does not require any crossbar read operations, and thus the total energy consumption can be estimated based on the number of write operations performed. Energy consumption for each write operation depends on the precision of memristors, weight value (conductance) which is variable, and amplitude and period of the voltage pulse used for weight updation. To estimate the energy, we have used the Stanford model [20] with parameters as given in Table 5.

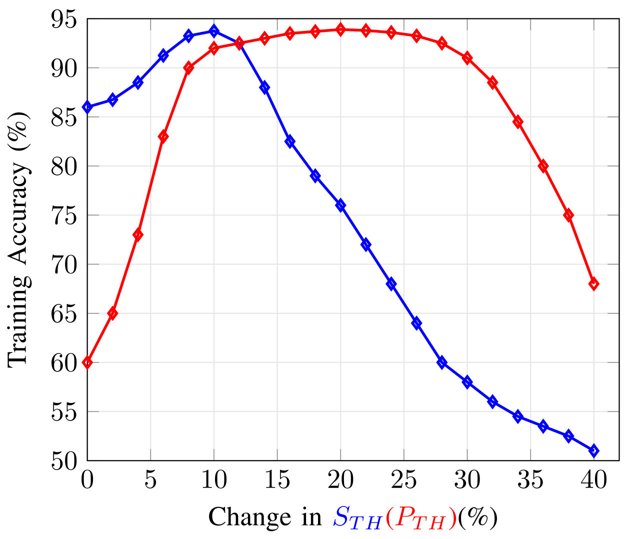


Fig. 12. Impact of S_{TH} and P_{TH} on training accuracy. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

For the proposed algorithm, we use a constant weight change factor, which depends on the precision of the multi-bit memristors. For instance, for a 5-bit memristor, we need to reduce the resistance value by ≈ 83 kΩ to change its state to the next higher level, whereas for an 8-bit memristor, we need to reduce the resistance by ≈ 10 kΩ for the same purpose.

The resistance values can be changed by applying a pulse of amplitude 1 V and pulse width ranging from 0.003–0.04 ns depending on the bit precision of the memristors. Through simulation studies, we have estimated that average energy of 6.93 nJ and 2.95 nJ is consumed for every write operation in memristors with 6- and 8-bit resolution respectively. The total energy consumption is calculated as the number of write operations multiplied by the average energy consumption for a single write operation. The overall energy consumption of the crossbar is 532.2 mJ, 33.77 J, 51.24 J and 60.77 J for BCW, MNIST, FMNIST and CIFAR10 datasets respectively. We have also observed that parallel write operations require less energy, but as the number of write operations depends on threshold and training input, we have not reported the values. The method requires A/D converter for reading the weights of AMs while updating the conductance value of the NN-chip, which may also increase energy consumption. As the primary aim of this work is to explore the feasibility of implementing neural network applications in crossbar array, energy consumption figures are provided only for the crossbar, excluding driver circuits, threshold circuits, A/D converters, etc.

4.6. Impact of parameters S_{TH} and P_{TH}

The inference rate of the proposed approach depends on the two external parameters, viz. the secondary threshold (S_{TH}) and primary threshold (P_{TH}) (refer algorithm 1).

We perform offline training-based analysis to select the values of S_{TH} and P_{TH}. The process starts by assigning smaller values to S_{TH} and P_{TH}, and by analysing the algorithm's performance the values are iteratively increased by a small amount (*i.e.* 5%). This iterative process is repeated until the algorithm's performance achieves a desirable outcome. Fig. 12 shows the impact of S_{TH} and P_{TH} for MNIST dataset. The blue and red lines show the impact of parameters S_{TH} and P_{TH}, respectively, on the algorithm's overall performance. The y-axis shows the change in accuracy, whereas the x-axis shows the % change of the S_{TH} and the P_{TH}.

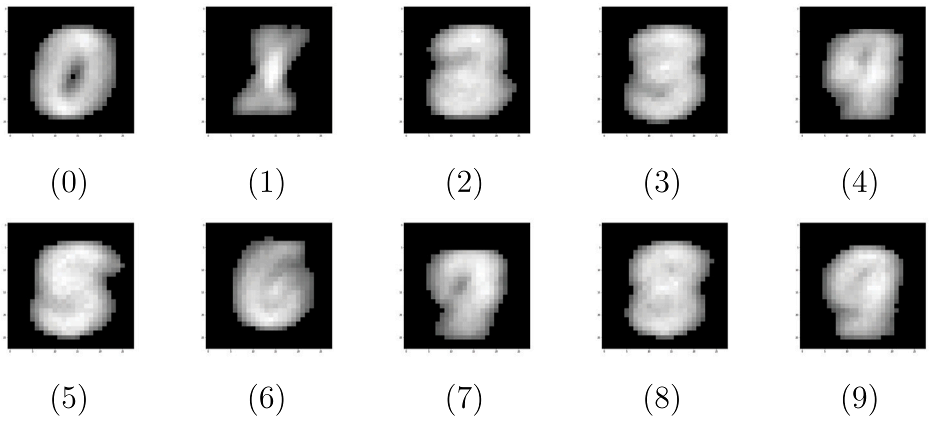


Fig. 13. Learnt weights for MNIST dataset.

Changing S_{TH} reflects how the attribute values/inputs affect the overall performance. For instance, a change from 0 to 10% gradually increases the performance as it does not allow smaller pixel values to change the weight of AM. On further increasing the S_{TH}, the performance degrades as the greater pixel values cannot change the AM as it is less than the threshold.

P_{TH} is the threshold that depends on the batch size and impacts how the memristors in AM update the NN-chip. For example, if the batch size is 100 and we set P_{TH} as 20%, this means that the memristor(s) in the AM crossbar that has been updated more than 20 times will update the corresponding memristor(s) in the NN-chip.

It can be observed that setting the value of P_{TH} beyond the range of 15%–25% does not improve the performance. P_{TH} less than 15% will allow most of the memristor in AM to update in NN-chip, which will degrade the performance. Likewise, if the P_{TH} value is greater than 25, this will restrict most of the AM updates to NN-chip, reducing the inference rate. Similar observation has also been seen for other datasets. We can say that by keeping the value of S_{TH} between 8%–12% and P_{TH} between 15%–25% we can achieve the desirable inference rate.

4.7. Detailed analysis for MNIST dataset

In this subsection, we carry out an analysis for MNIST handwritten digit dataset [33]. Since the application has been trained in 784 × 10 FCNN, the weight of each digit is stored in 784 memristors. We have considered 8-bit memristors; thus, by changing the conductance value to their respective decimal values, we can have values ranging from 0 – 255. Now we have 784 values, each ranging from 0 – 255, from which we can generate an image of 28 × 28 for the respective weights learnt. Fig. 13 shows the weights learnt for each digit as a grey-scale image.

From Fig. 13 we can visually identify the patterns that are similar and may lead to misclassification. For example, weights learnt for digits 4, 7, and 9 are quite similar, which may be misclassified. Table 6 reflects this observation that shows the result of testing with 10,000 patterns, out of which 595 patterns are misclassified. Table 6 shows the top 3% misclassified class for digits 0 – 9.

Table 6
Top 3% wrong classification.

Pattern	Top 3% mis-classification					
	Num	%	Num	%	Num	%
0	5	44%	8	24%	2	12%
1	8	56%	2	18%	3	12%
2	8	33%	3	18%	6	14%
3	5	30%	8	28%	2	16%
4	9	50%	2	16%	8	16%
5	3	35%	8	25%	4	11%
6	5	27%	0	20%	4	19%
7	9	38%	2	27%	3	13%
8	3	28%	5	26%	9	22%
9	4	30%	5	22%	3	20%

5. Comparison with other training algorithms

Prior works on memristor-based neural networks have used different application benchmarks for analysis. The most commonly used benchmark is the MNIST handwritten digit dataset [33]. The original image size of the MNIST dataset is 28 × 28. However, different sizes of the dataset have also been used as per network requirements and support.

Table 7 compares the proposed algorithm with respect to accuracy and number of epochs for different benchmarks. The first column shows the references of the work we are comparing with, while the second column shows the name of the benchmark. The third and fourth columns show the input and network sizes, respectively. The fifth and sixth columns show the accuracy achieved by different approaches and the number of times the data are processed. The last column shows the improvement in accuracy as compared to other approaches.

From Table 7 we observe that the works in [8,19] use a similar configuration as the proposed training algorithm (similar input size, network size, number of epochs), which has an accuracy of 89.9% and 83.85% respectively, which are 4.36% and 10.79% less as compared to the proposed algorithm. The work proposed in [27] uses a similar input size with hidden layer and achieves an accuracy of 92% with 1000 epochs which is 2.12% less as compared to the proposed algorithm. The work proposed in [24] uses a smaller input size with a single hidden layer and has an accuracy of 81.80% which is 12.97% less compared to the proposed algorithm. The LMS and SLMS approach proposed in [31] have an accuracy of 82% and 84% respectively, which is 12.76% and 10.63% less as compared to the proposed algorithm.

The work proposed in [29] shows an average of 1% higher accuracy as compared to the proposed algorithm. However, the work uses AlexNet for training the applications, with 6 hidden layers and 1000 epochs. The network proposed in [29] is sensitive to accuracy, with every single layer affecting the accuracy by ±2%. Overall, it can be observed that the proposed algorithm shows an average accuracy improvement of 5.03%. Thus using the proposed algorithm, it is possible to achieve higher/similar accuracy for the applications without using any complex training circuitry.

In this paper, we have not discussed the hardware implementation; however, an estimation of the total delay required to train the application is presented and compared with state-of-the-art works. Assume that N_e denotes the number of epochs, N_{tm} the number of total memristors in crossbar, N_{tml} the number of memristors in last layer, N_{ts} the number of total training samples, D_{rc} the delay for reverse change, D_{fp} the

Table 7
Performance comparison of the proposed algorithm.

Approach	Description	Input size	Net-Size	Accuracy	# of epochs	Accuracy %
Proposed	HSBT	784 (28 × 28)	784→10	94.00%	10	–
[8]	HSBT BP	760 (19 × 20)	760→10	89.90%	–	4.36
[16]	HSBT BP	49 (7 × 7)	50→10	92.00%	–	2.12
[27]	HABT BP	784 (28 × 28)	784→ 42 →10	92.00%	1000	2.12
[19]	HSBT BP	784 (28 × 28)	784→10	83.85%	10	10.79
[29]	HSBT RWC	784 (28 × 28)	AlexNet	94.79%	1000	−0.84
	HSBT BP + RWC			98.81%		−5.11
	HSBT Delta Rule			85.19%		9.37
	HSBT RWC + Delta			97.95%		−4.20
[24]	HABT (OCTAN)	64 (8 × 8)	64→ 100 →10	81.80%	52	12.97
[31]	HABT SLMS	25 (5 × 5)	–	84.00%	250	10.63
	HABT LMS			82.00%		12.76
Average accuracy improvement						5.03

delay for forward propagation, D_{th} the delay for threshold circuit, D_e the delay for error calculation, and D_w the delay for write operation.

The total delays of the proposed, OCTAN [24], RWC [24], and SLMS [32] algorithms can be given as:

$$D_{PROPOSED} = N_e N_{ts}(D_{th} + D_w) \tag{4}$$
$$D_{OCTAN} = N_e N_{tm} N_{ts}(1 + D_{rc})(D_{fp} + D_e + D_w) \tag{5}$$
$$D_{RWC} = N_e N_{ts}(D_{fp} + D_e + N_{tm} D_w) \tag{6}$$
$$D_{SLMS} = N_e N_{ts}(D_{fp} + D_e + N_{tml} + D_w) \tag{7}$$

The average number of epochs required by the proposed algorithm is 10. If we assume the cycle required for additional circuits $(D_{th} + D_w)$ is D_{ctrl} (delay of the controller), then the total delay of the proposed approach is:

$$D_{PROPOSED} = 10 N_{ts} D_{ctrl} \tag{8}$$

Similarly, for the works in [24,29,32] if we assume that the delay required by additional circuits is D_{ctrl}, and if we replace the N_e with the required number of epochs, then the delays for the different approaches will be:

$$D_{OCTAN} = 52 N_{tm} N_{ts} D_{ctrl} \tag{9}$$
$$D_{RWC} = 1000 N_{ts}(D_{ctrl} + N_{tm}) \tag{10}$$
$$D_{SLMS} = 250 N_{ts}(D_{ctrl} + N_{tml}) \tag{11}$$

The proposed approach is independent of the number of memristors available in the crossbar, as all the memristors for a training sample can be updated at once. The RWC and SLMS approaches are also capable of performing such parallel operations, while OCTAN updates the weights serially. The equations show that the proposed algorithm can train the network 5×, 100×, and 10× faster than the OCTAN, RWC, and SLMS algorithms, respectively. As the proposed algorithm requires fewer epochs, it will also require fewer write operations.

6. Exploiting the power of 3D crossbar

In the proposed algorithm, the number of columns and rows in the crossbar are determined by the number of classes and the number of attributes of the image/input, respectively. It can be observed that as size the of the dataset increases, the crossbar size also increases linearly, which causes significant area overhead. To reduce the area, the training can be performed in a 3D crossbar. For example, consider a dataset D having m classes and each element of the dataset having n attributes. The datasets can be trained in a 3D crossbar consisting of k layers where the size of each crossbars layer is $(i \times m)$ such that $i \times k = n$.

For the input attributes A_1 to A_i, the weights will be learnt in the layer 1 of the crossbar, for the attributes A_{i+1} to A_{2i} weights will be learnt in layer 2, and similarly for the attributes $A_{i(k-1)+1}$ to A_{ik} weights will be learnt in kth layer of the crossbar (see Fig. 14).

The output of each class is determined as the sum of the currents in the corresponding column of all the layers. For instance, for the weights of class-0, the total output current is $I_{01} + I_{02} + \dots + I_{0k}$. Thus, for any class c, the total output current will be given by:

$$I_c = \sum_{i=1}^{k} I_{ci} \tag{12}$$

The proposed algorithm is suitable for ReRAM-based networks where fully connected neural network architectures are modelled. This is because the proposed architecture can carry out VMM operations easily [8,16,19,24,27]. The proposed work acts as a benchmark where other neural network architectures can also be evaluated. For future work, we plan to implement an algorithm that supports convolutional neural networks (CNN). To implement such an algorithm, we need to include functionality to share weights among neurons of different classes, which is the basis of how CNN architectures work.

7. Conclusion

The VMM operation can be performed efficiently in resistive crossbar-based memory, accelerating the neural network's overall computation. In resistive crossbar-based memories, the weights are stored in the form of resistance/conductance. The accuracy of VMM computed by these devices depends on how efficiently the crossbar training has been done. For training the crossbar, various works have been proposed in the literature. The state-of-the-art solutions require additional computational blocks and circuitry and hence consume more power and incur higher delay. The proposed work shows that various applications can be learnt by performing constant weight change in the forward direction, which does not require any additional circuitry and can provide accuracy similar to existing training algorithms with fewer epochs. Thus the overall reduction in delay and energy consumption can be achieved using the proposed approach.

The analysis has been done for classification-based problems in single-layer FCNN. However, the algorithm can also be applied to other types of applications and can be used for multi-layer neural networks, which can be taken up as future work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data used in the manuscript have been cited.

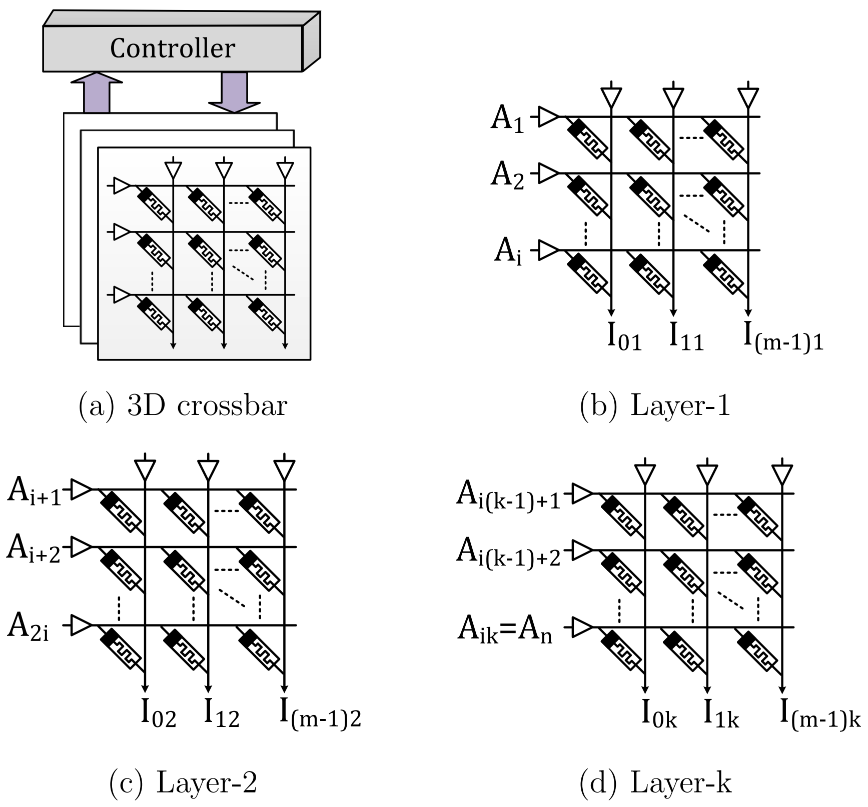


Fig. 14. Learning in 3D crossbar.

References

[1] W.A. Wulf, S.A. McKee, Hitting the memory wall: Implications of the obvious, SIGARCH Comput. Archit. News 23 (1) (1995) 20–24, http://dx.doi.org/10.1145/216585.216588.

[2] K.J. Kuhn, Considerations for ultimate CMOS scaling, IEEE Trans. Electron Devices 59 (7) (2012) 1813–1828, http://dx.doi.org/10.1109/TED.2012.2193129.

[3] K. Akarvardar, H.S.P. Wong, Ultralow voltage crossbar nonvolatile memory based on energy-reversible NEM switches, IEEE Electron Device Lett. 30 (6) (2009) 626–628.

[4] A. Bhola, G. Kanitkar, Memristors and crossbar latches, in: Proceedings of the International Conference and Workshop on Emerging Trends in Technology, ICWET '10, ACM, 2010, pp. 915–918.

[5] J.J. Yang, D.B. Strukov, D.R. Stewart, Memristive devices for computing, Nature Nanotechnol. 8 (1) (2013) 13.

[6] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, L. Jiang, Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, 2017, pp. 19–24.

[7] L. Song, X. Qian, H. Li, Y. Chen, Pipelayer: A pipelined ReRAM-based accelerator for deep learning, in: IEEE Intl. Symp. on High Performance Computer Architecture, HPCA, 2017, pp. 541–552.

[8] M. Hu, C.E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R.S. Williams, J.J. Yang, et al., Memristor-based analog computation and neural network classification with a dot product engine, Adv. Mater. 30 (9) (2018) 1705914.

[9] A. Ankit, I.E. Hajj, S.R. Chalamalasetti, G. Ndu, M. Foltin, R.S. Williams, P. Faraboschi, W.-m.W. Hwu, J.P. Strachan, K. Roy, et al., PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference, in: 24th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 715–731.

[10] W.-S. Lee, Future memory technologies, in: 2008 9th International Conference on Solid-State and Integrated-Circuit Technology, IEEE, 2008, pp. 1–4.

[11] H.-S.P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F.T. Chen, M.-J. Tsai, Metal–oxide RRAM, Proc. IEEE 100 (6) (2012) 1951–1970.

[12] X. Dong, C. Xu, Y. Xie, N.P. Jouppi, Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 31 (7) (2012) 994–1007.

[13] L.O. Chua, Memristor-The missing circuit element, IEEE Trans. Circuit Theory 18 (5) (1971) 507–519, http://dx.doi.org/10.1109/TCT.1971.1083337.

[14] D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found, Nature 453 (7191) (2008) 80–83.

[15] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J.P. Strachan, M. Hu, R.S. Williams, V. Srikumar, ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars, in: 43rd Annual Intl. Symp. on Computer Architecture, ISCA, 2016, pp. 14–26.

[16] L. Xia, P. Gu, B. Li, T. Tang, X. Yin, W. Huangfu, S. Yu, Y. Cao, Y. Wang, H. Yang, Technological exploration of RRAM crossbar array for matrix-vector multiplication, J. Comput. Sci. Tech. 31 (1) (2016) 3–19.

[17] N. Taherinejad, P.S. Manoj, A. Jantsch, Memristors' potential for multi-bit storage and pattern learning, in: IEEE European Modelling Symp., 2015, pp. 450–455.

[18] H. Hossam, G. Mamdouh, H.H. Hussein, M. El-Dessouky, H. Mostafa, A new read circuit for multi-bit memristor-based memories based on time to digital sensing circuit, in: 61st Intl. Midwest Symp. on Circuits and Systems, 2018, pp. 1114–1117.

[19] D.N. Yadav, K. Datta, I. Sengupta, Analyzing fault tolerance behaviour in memristor-based crossbar for neuromorphic applications, in: 2020 IEEE International Test Conference India, IEEE, 2020, pp. 1–9.

[20] J. Reuben, D. Fey, C. Wenger, A modeling methodology for resistive ram based on stanford-pku model with extended multilevel capability, IEEE Trans. Nanotechnol. 18 (2019) 647–656.

[21] S. Kvatinsky, M. Ramadan, E.G. Friedman, A. Kolodny, VTEAM: A general model for voltage-controlled memristors, IEEE Trans. Circuits Syst. II 62 (8) (2015) 786–790.

[22] S.P. Adhikari, C. Yang, H. Kim, L.O. Chua, Memristor bridge synapse-based neural network and its learning, IEEE Trans. Neural Netw. Learn. Syst. 23 (9) (2012) 1426–1435.

[23] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, S. Kvatinsky, Memristor-based multilayer neural networks with online gradient descent training, IEEE Trans. Neural Netw. Learn. Syst. 26 (10) (2015) 2408–2421.

[24] M. Ansari, A. Fayyazi, M. Kamal, A. Afzali-Kusha, M. Pedram, OCTAN: An on-chip training algorithm for memristive neuromorphic circuits, IEEE Trans. Circuits Syst. I. Regul. Pap. 66 (12) (2019) 4687–4698.

[25] S.M. Tam, B. Gupta, H.A. Castro, M. Holler, Learning on an analog VLSI neural network chip, in: IEEE Intl. Conf. on Systems, Man, and Cybernetics, 1990, pp. 701–703.

[26] O. Krestinskaya, K.N. Salama, A.P. James, Analog backpropagation learning circuits for memristive crossbar neural networks, in: 2018 IEEE International Symposium on Circuits and Systems, ISCAS, 2018, pp. 1–5.

[27] O. Krestinskaya, K.N. Salama, A.P. James, Learning in memristive neural network architectures using analog backpropagation circuits, IEEE Trans. Circuits Syst. I. Regul. Pap. 66 (2) (2018) 719–732.

[28] K. Hirotsu, M.A. Brooke, An analog neural network chip with random weight change learning algorithm, in: Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), Vol. 3, IEEE, 1993, pp. 3031–3034.
[29] S.P. Adhikari, C. Yang, K. Slot, M. Strzelecki, H. Kim, Hybrid no-propagation learning for multilayer neural networks, Neurocomputing 321 (2018) 28–35.
[30] C. Yang, H. Kim, S.P. Adhikari, L.O. Chua, A circuit-based neural network with hybrid learning of backpropagation and random weight change algorithms, Sensors 17 (1) (2017) 16.
[31] C. Merkel, D. Kudithipudi, A stochastic learning algorithm for neuromemristive systems, in: 2014 27th IEEE International System-on-Chip Conference, SOCC, 2014, pp. 359–364.
[32] T. Gokmen, M. Onen, W. Haensch, Training deep convolutional neural networks with resistive cross-point devices, Front. Neurosci. 11 (2017) 538.
[33] Y. LeCun, The MNIST database of handwritten digits, 1998, http://yann.lecun.com/exdb/mnist/.
[34] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E.G. Friedman, A. Kolodny, U.C. Weiser, MAGIC—Memristor-aided logic, IEEE Trans. Circuits Syst. II 61 (11) (2014) 895–899.
[35] P. Thangkhiew, R. Gharpinde, D.N. Yadav, K. Datta, I. Sengupta, Efficient implementation of adder circuits in memristive crossbar array, in: TENCON 2017-2017 IEEE Region 10 Conference, IEEE, 2017, pp. 207–212.
[36] P.L. Thangkhiew, R. Gharpinde, K. Datta, Efficient mapping of Boolean functions to memristor crossbar using MAGIC NOR gates, IEEE Trans. Circuits Syst. I. Regul. Pap. 65 (8) (2018) 2466–2476.
[37] D.N. Yadav, P.L. Thangkhiew, K. Datta, Look-ahead mapping of Boolean functions in memristive crossbar array, Integration 64 (2019) 152–162.
[38] P.L. Thangkhiew, A. Zulehner, R. Wille, K. Datta, I. Sengupta, An efficient memristor crossbar architecture for mapping Boolean functions using Binary Decision Diagrams (BDD), Integration 71 (2020) 125–133.
[39] P.L. Thangkhiew, K. Datta, Scalable in-memory mapping of Boolean functions in memristive crossbar array using simulated annealing, J. Syst. Archit. 89 (2018) 49–59.
[40] H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017, URL http://arxiv.org/abs/1708.07747.
[41] A. Krizhevsky, V. Nair, G. Hinton, Cifar-10 (canadian institute for advanced research), 5 (2009) 4. URL http://www.cs.toronto.edu/kriz/cifar.html.
[42] D. Dua, C. Graff, UCI machine learning repository, 2017, URL http://archive.ics.uci.edu/ml.
[43] O.L. Mangasarian, W.N. Street, W.H. Wolberg, Breast cancer diagnosis and prognosis via linear programming, Oper. Res. 43 (4) (1995) 570–577.

Dev Narayan Yadav received the M.Tech. degree in Computer Science and Engineering from National Institute of Technology Meghalaya, India, in 2018. He worked as a Junior Research Fellow at the National Institute of Technology Meghalaya during 2018–2019. He is currently pursuing his PhD degree from the Indian Institute of Technology Kharagpur, India. His research interests include memristor and its applications in logic design and neuromorphic computing.

Phrangboklang Lyngton Thangkhiew received the Bachelor of Engineering (B.E.) from Visvesvaraya Technological University, India, in 2014, and Ph.D. from National Institute of Technology Meghalaya, India, in 2019. He worked as an Assistant Professor at Vellore Institute of Technology Chennai, India, during 2020–2021 and as a Post-Doctoral Fellow at Indian Institute of Technology Guwahati, India, from October 2021 to January 2022. He is currently working as an Assistant Professor at the Indian Institute of Information Technology Guwahati, India. He has published several papers in peer-reviewed journals and conferences. His research interests include memristor and its applications in logic design, synthesis, and design optimization.

Kamalika Datta completed her Master of Science (MS) from Indian Institute of Technology Kharagpur, India in 2010, and Ph.D. from Indian Institute of Engineering Science and Technology (IIEST), Shibpur, India in 2014. She worked as Assistant Professor at the National Institute of Technology Meghalaya from 2014 to 2018, Post-Doctoral Research Fellow at the Nanyang Technological University Singapore from February 2019 to September 2020, and as an Associate Professor at JIS University, Kolkata from November 2020 to August 2021. She is presently working as a Senior Researcher at the Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) Bremen, Germany since October 2021. She has published more than 75 papers in peer reviewed journals and conferences. Her research interests include logic design using emerging technologies, synthesis and optimization of reversible and quantum circuit, and embedded systems.

Sandip Chakraborty received the B.E. degree from Jadavpur University, Kolkata, India, and the M.Tech. and PhD degrees from the Indian Institute of Technology Guwahati, India. He is currently an Associate Professor at the Indian Institute of Technology Kharagpur, India. He is an Associate Editor of the Ad-Hoc Networks Journal (Elsevier) and Pervasive and Mobile Computing Journal (Elsevier). He has authored more than 100 papers in peer-reviewed journals and conferences. His current research interests include applications of machine learning over computer systems and networks, broadly in network protocol design and performance optimization, network measurement, and sensing systems.

Rolf Drechsler received the Diploma and Dr. Phil. Nat. degrees in computer science from J.W. Goethe University Frankfurt am Main, Frankfurt am Main, Germany, in 1992 and 1995, respectively. He was with the Institute of Computer Science, Albert-Ludwigs University, Freiburg im Breisgau, Germany, from 1995 to 2000, and with the Corporate Technology Department, Siemens AG, Munich, Germany, from 2000 to 2001. Since October 2001, he has been with the University of Bremen, Bremen, Germany, where he is currently a Full Professor and the Head of the Group for Computer Architecture, Institute of Computer Science. In 2011, he additionally became the Director of the Cyber–Physical Systems group at the German Research Center for Artificial Intelligence (DFKI) in Bremen. His current research interests include the development and design of data structures and algorithms with a focus on circuit and system design. Rolf Drechsler was a member of Program Committees of numerous conferences including e.g. DAC, ICCAD, DATE, ASP-DAC, FDL, MEMOCODE, FMCAD, Symposiums Chair ISMVL 1999 and 2014, Symposium Chair ETS 2018, and the Topic Chair for “Formal Verification” DATE 2004, DATE 2005, DAC 2010, as well as DAC 2011. He received best paper awards at HVC in 2006, FDL in 2007 and 2010, DDECS in 2010 and ICCAD in 2013 and 2018. He is an Associate Editor of TCAD, JETC, and further journals. He is an ACM Distinguished Member and an IEEE Fellow.

Indranil Sengupta received his B.Tech., M.Tech., and PhD degrees in Computer Science from the University of Calcutta in 1983, 1985, and 1990 respectively. He joined the Indian Institute of Technology Kharagpur, India, as a faculty member in 1988, in the Department of Computer Science and Engineering. Currently, he is the Vice-Chancellor of JIS University, Kolkata, India. He had been the former Heads of the Department of Computer Science and Engineering and the School of Information Technology at Indian Institute of Technology Kharagpur. He has over 32 years of teaching and research experience, guided 22 PhD students, and published over 200 papers in peer-reviewed journals and conferences. He has served as the Program Chair / General Chair in several International Conferences in the areas of VLSI design/test, reversible computing, and information security. His research interests include reversible and quantum computing, VLSI design and test, and network security. He is a Senior Member of the IEEE.