In Situ Stochastic Training of MTJ Crossbars With Machine Learning Algorithms

ANKIT MONDAL and ANKUR SRIVASTAVA, University of Maryland College Park

Owing to high device density, scalability, and non-volatility, magnetic tunnel junction (MTJ)-based crossbars have garnered significant interest for implementing the weights of neural networks (NNs). The existence of only two stable states in MTJs implies a high overhead of obtaining optimal binary weights in software. This article illustrates that the inherent parallelism in the crossbar structure makes it highly appropriate for in situ training, wherein the network is taught directly on the hardware. It leads to significantly smaller training overhead as the training time is independent of the size of the network, while also circumventing the effects of alternate current paths in the crossbar and accounting for manufacturing variations in the device. We show how the stochastic switching characteristics of MTJs can be leveraged to perform probabilistic weight updates using the gradient descent algorithm. We describe how the update operations can be performed on crossbars implementing NNs and restricted Boltzmann machines, and perform simulations on them to demonstrate the effectiveness of our techniques. The results reveal that stochastically trained MTJ-crossbar feed-forward and deep belief nets achieve a classification accuracy nearly the same as that of real-valued weight networks trained in software and exhibit immunity to device variations.

CCS Concepts: • **Computing methodologies** → **Neural networks**; *Supervised learning*; *Unsupervised learning*; • **Hardware** → **Spintronics and magnetic technologies**; *Emerging architectures*; Non-volatile memory;

Additional Key Words and Phrases: Magnetic tunnel junctions, neural networks, restricted Boltzmann machines, crossbar architecture, on-chip learning

ACM Reference format:
Ankit Mondal and Ankur Srivastava. 2019. In Situ Stochastic Training of MTJ Crossbars With Machine Learning Algorithms. *J. Emerg. Technol. Comput. Syst.* 15, 2, Article 16 (March 2019), 29 pages.
https://doi.org/10.1145/3309880

1 INTRODUCTION

Deep neural networks (DNNs) have become a popular choice for machine learning tasks such as image classification, face recognition, and Natural Language Processing. However, this has been at the cost of massive computations on von Neumann architectures exhibiting high energy and area requirements [7]. The emergence of novel devices and special-purpose architectures has called for a shift from conventional digital hardware for implementing neural algorithms [50].

A preliminary version [34] of this work was published at ISLPED 2018.
This work was supported by the National Science Foundation (NSF) grant 1642424.
Authors' addresses: A. Mondal and A. Srivastava, Department of Electrical and Computer Engineering, 8223 Paint Branch Drive, Room 1313, AV Williams Building, College Park, MD 20742; emails: amondal2@terpmail.umd.edu, ankurs@umd.edu.

1550-4832/2019/03-ART16 $15.00
https://doi.org/10.1145/3309880

Attempts have been made toward dedicated hardware designs and realization of the synaptic weights (and neurons) of a neural network (NN) by using CMOS transistors in an analog fashion [33, 37], but these have met with challenges of scalability and volatility. Parallel research work has focused on using post-CMOS devices such as memristors, which are non-volatile devices with a variable resistance [38]. However, the fabrication of multi-level memristors with stable states is still a challenge [36, 60]. Another choice is the magnetic tunnel junction (MTJ) [54], an emerging *binary* device (since it has two stable states) that has shown its potential as storage elements and is a promising candidate for replacing CMOS in memory chips [53]. Its non-volatility and scalability make it a particularly lucrative choice for logic-in-memory type architectures for NNs.

NN architectures can be realized using a crossbar configuration, which allows greater scalability and higher performance due to its inherent parallelism [32, 38, 45, 52, 59]. Several studies have investigated how a crossbar array with memristors [4, 39, 41, 55], MTJs [52, 60], and domain-wall ferromagnets [41, 42] can implement spiking neural networks (SNNs) trained using spike-timing-dependent plasticity (STDP). Srinivasan et al. [48] propose the use of a pair of MTJs for a synapse in an SNN: one for long-term and the other for short-term synaptic memory. They develop a stochastic learning algorithm based on STDP and demonstrate learning efficacy superior to a stochastic one-bit synapse. The literature also contains several works [8, 18, 57] considering supervised learning of SNNs for various reasons.

Many works have dealt with methods and algorithms for training networks by modifying their weights at the site of their occurrence [5] instead of doing it offline. Hasan and Taha [11] and Soudry et al. [47] have implemented multi-layer NNs on memristive crossbars trained on-chip using the backpropagation algorithm and demonstrated on supervised learning tasks. Gokmen and Vlasov [10] use stochastic computing techniques for parallel weight update on crossbar arrays—numbers that are encoded from neurons are translated to stochastic bit streams, with device conductance changing when the streams coincide. In Lee and Likharev [25], hybrid semiconductor/nanodevice technology neural nets with binary synapses were trained "in situ" using the error backpropagation rule, and the results obtained were almost at par with networks with continuous weights trained in software.

Efficient architectures for the realization of different types of network models, such as convolution and recurrent NNs [58], liquid state machines [17], and echo state networks [12], are also being investigated. Neftci et al. [35] construct a restricted Boltzmann machine (RBM) with integrate-and-fire neurons and present an event-driven variation of the contrastive divergence (CD) learning algorithm. Herein, the recurrent structure of the network is exploited to mimic the construction and reconstruction phases of CD weight update in a spike-driven fashion, and STDP is used to carry out the weight updates. In Sheri et al. [46], an approach to implement CD in one layer of an RBM with memristors as synapses is presented. However, the RBM has stochastic binary units and weight updates are ternary. Suri et al. [49] fabricate an HfO_x device and test it for synapse implementation, internal neuron-state storage, and stochastic neuron activation function of a hybrid RRAM-CMOS RBM architecture.

Continuous weight networks can be simplified into discrete weight networks without significant degradation in classification accuracy while achieving substantial power benefits [40]. The use of discrete weight networks, such as BinaryConnect [6] and in Li et al. [28], also stems from the challenge to address the high storage and computational demands of a large number of full-precision weights. Ni et al. [36] design a distributed in-memory computing architecture based on binary RRAM crossbars for memory and logic units. The existence of only two stable states in MTJs makes them a good candidate for the realization of binary weight networks. Obtaining optimal weights for a binary network in software can be impractical because its discrete nature requires integer programming. One way of training such NNs is to perform weight updates stochastically,

which is justifiable from evidence that learning in human brains also has some associated stochasticity [50]. That such a method can lead to convergence with high probability in a finite time has been shown in Senn and Fusi [44], although using the perceptron learning rule. In addition, when physically realizing an NN on hardware, the underlying device variations (DVs) can have a substantial impact on the model accuracy and need to be accounted for in the training process. Merely characterizing the variations in the hardware platform is not sufficient for overcoming this issue.

In this article, we explore the use of MTJ crossbars for the hardware implementation of the synaptic weight matrices of feed-forward NNs and RBMs. We propose the *in situ* training[1] of these MTJ crossbars, which allows us to exploit their inherent parallelism for significantly faster training and also accounts for DVs. We advocate a probabilistic way of updating the MTJ synaptic weights of an NN through the gradient descent algorithm by exploiting the stochasticity in their switching. We experiment with two crossbar structures: with and without access transistors. The latter poses the additional challenge of sneak-path currents during programming, which makes training in situ the only choice to achieve satisfactory performance. Then we go on to propose a modification of the CD algorithm that is to be adopted when the MTJ crossbar is used to implement an RBM, and a means of using MTJs for storing RBM hidden units' states. Finally, we support our proposed techniques with data by modeling device and circuit properties and running simulations.

2 BACKGROUND

2.1 Neural Networks

The computation performed by any layer of an NN during the inference (forward propagation) phase basically comprises a matrix-vector multiplication. Say, $x \in R^M$ is the input to a layer and $W \in R^{N \times M}$ represents the synaptic weight matrix, then the output $y \in R^N$ is

$$y = f(Wx), \tag{1}$$

where $f(\,)$ is an activation function. Training of the NN can be done by backpropagation using the *gradient descent* optimization method. The weight update of the synapse connecting the i^{th} input to the j^{th} output is given as

$$\Delta W_{ji} = -\eta \frac{\partial E}{\partial W_{ji}} = -\eta x_i \delta_j, \tag{2}$$

where E is the cost function of the presented input sample x, η is the learning rate, and δ_j is the error calculated at the j^{th} output using y and the desired output. It is worth noting that such a weight update is local in nature, in that it depends only on the information available at the synapse—the input to it and the error at its output. Thus, the weight update of the entire matrix is

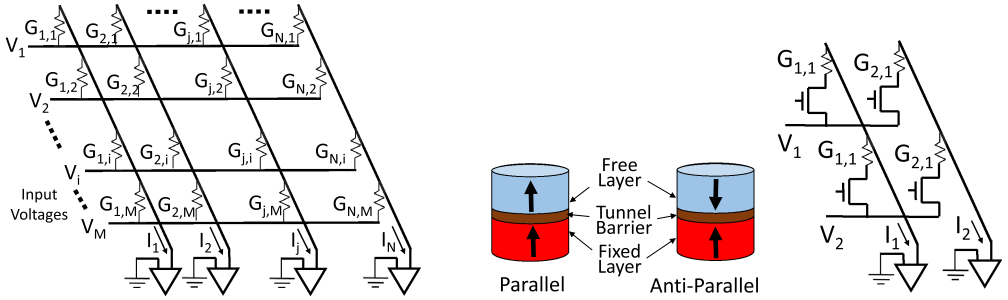
$$\Delta W = -\eta \delta x^T. \tag{3}$$

The major computational cost of this algorithm comes from the $O(M.N)$ complexity of Equations (1) and (3), whose implementation on general-purpose hardware requires time and memory of the same order, thereby not motivating their use for large-scale applications. Fortunately, the nature of computation in Equation (1) and the locality of weight update enable the design of highly parallel hardware that reduce the overall complexity to O(1).

2.2 The Crossbar Architecture

The physical realization of a synaptic weight matrix is possible using the grid-like crossbar structure where each junction has a resistance corresponding to one synapse. Figure 1(a) shows a

[1]It is worthwhile to mention that we do not actually perform online training where the training data becomes available during the in-field use of the system. Instead, our work deals with in situ/on-chip training where the entire training data is already available (in labeled/supervised form or unlabeled/unsupervised form).



(a) Structure of an $M \times N$ crossbar. **(b)** Two stable states of an MTJ. **(c)**

Fig. 1. (a) A crossbar. (b) Magnetic tunnel junction. (c) A 2×2 crossbar.

simplified crossbar with each row corresponding to an input and each column to an output neuron. Let $V_i \in [-V_{rd}, V_{rd}]$ be the voltage applied at the i^{th} input terminal and G_{ji} be the conductance of the synapse connecting it to the j^{th} output. By Ohm's law, the current through that synapse is $G_{ji}V_i$, and by Kirchhoff's law, the total current at the output is

$$I_j = \sum_i G_{ji} V_i, \tag{4}$$

which bears similarity to the dot products in (1). This can then be fed to suitable analog circuits for implementing the activation function [11, 19].

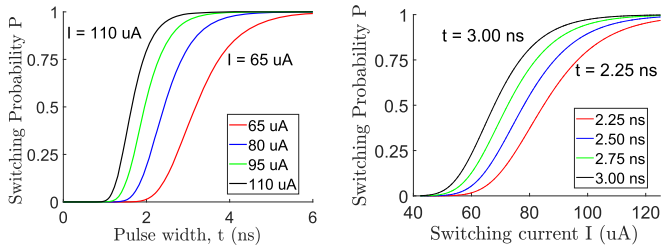
Since the outputs are obtained almost instantaneously after the inputs are applied, the matrix-vector multiplication of Equation (1) is performed in parallel with constant time complexity. As for the update phase, the crossbar resistances can be modified by suitably modeling the required change as the product of two physical quantities derivable from the inputs and the errors. In this way, the $O(M.N)$ operations can be done in parallel using the $M.N$ synapses.

2.3 Magnetic Tunnel Junction

The MTJ is a two-terminal spintronic device consisting primarily of two ferromagnetic layers separated by a thin tunnel barrier (typically MgO) [54]. The magnetic orientation of one of the magnetic layers is fixed, whereas that of the other is free, as shown in Figure 1(b). MTJs possess two stable states where the relative magnetic orientations of the *free* and *fixed* layers are parallel (P) and anti-parallel (AP), respectively, with the P state exhibiting a lower resistance than the AP state ($R_P < R_{AP}$). It is possible to switch the state of the MTJ by passing spin-polarized current of appropriate polarity that flips the magnetization of the free layer through the mechanism of spin-transfer torque [29]. The time required to switch is heavily dependent on the magnitude of the switching current. Not only that, but this switching process is a stochastic one in the sense that a pulse of given amplitude and duration has only a certain probability to successfully change the state. This stochasticity is due to thermal fluctuations in the initial magnetization angle and is an intrinsic property of the STT switching [29].

Depending on the magnitude I of the current and the critical current I_{c0} [60], the switching probability in the high-speed precessional regime ($I > I_{c0}$) is expressed as

$$P(a,t) = exp(-4f(a)\Delta exp(-2t/T)),\ with\ f(a) = \left(\frac{2a}{a-1}\right)^{\left(\frac{-2}{a+1}\right)}, \tag{5}$$



(a) P vs t, for different values of I. **(b)** P vs I, for different values of t.

Fig. 2. MTJ $AP \rightarrow P$ switching probability as a function of t and I.

where $a = I/I_{c0}$, t is the pulse width, Δ is the thermal stability, and T is the mean switching time (which is dependent on a) [51].

The spin transfer efficiency (θ) of an MTJ is different for the two switching directions, with $\theta^{P \rightarrow AP}$ having a smaller value than $\theta^{AP \rightarrow P}$ [61]. This makes $I_{c0}^{P \rightarrow AP} > I_{c0}^{AP \rightarrow P}$, which means that the same magnitude and duration of current will correspond to different switching probabilities for the two switching directions. Figure 2 shows the dependence of the switching probability on pulse width and switching current for the $AP \rightarrow P$ transition. Observe the similarity in the nature of variation with I and t. The $P \rightarrow AP$ transition depicts this kind of a behavior as well, albeit with different values of I and t.

3 MTJ CROSSBAR-BASED NNS

The stochastic switching nature of MTJs has necessitated the usage of high write currents or write duration in memory applications to ensure low write errors. Alternatively, one can also use them to implement the synaptic weights in a crossbar where each cross point would be an MTJ in one of its two states. They are capable of being programmed with high speeds and exhibit endurance of the order of 10^{15} write cycles. However, the inherently binary nature of MTJs implies that such synapses can represent only two weight values and hence can implement only binary networks. Although it is possible to have some continuous behavior with the inclusion of a domain wall in the free layer [42], the maturity of such technology is not at par with that of the binary version [41].

Training binary networks. Obtaining optimal binary weights for an NN is an NP-hard problem with an exponential time complexity [44], and hence a solution must involve training of the binary network of some form. This prompts the use of a probabilistic learning technique since the required weight update is continuous, whereas any possible change in the conductance of the MTJ could only be discrete, in fact binary. As stated in Suri et al. [50], stochastic update of binary weights is computationally equivalent to deterministic update of multi-level weights at the system level.

Vincent et al. [52] exploit the stochastic switching behavior of MTJs to propose its use as a "stochastic memristive synapse" in an SNN taught using a simplified STDP rule. However, there is no theoretical guarantee of the convergence of STDP for general inputs [27], and Lim et al. [31] believe that the learning performance using STDP in still in its early stages. We propose using a probabilistic learning approach by training using the gradient descent method (which requires weight updates of the form in Equation (2)) as demonstrated in Section 4.2.

3.1 The Motivation for In Situ Training

There are two ways (primarily) in which MTJs in the crossbar can be connected to their respective input and output terminals:

(1) *With selector devices* (1T1R): Here, each MTJ synapse is connected in series with an MOS transistor (as in Figure 1(c)), resulting in $O(M \times N)$ transistors in the crossbars.
(2) *Without selector devices* (1R): Synapses are directly connected to the crossbar terminals; there are no transistors within the crossbar, such as the one in Figure 1(a). Although a 1R structure provides greater scalability, it does so at the cost of reduced control of and access to individual synapses.

Stochastic learning can be done (simulated) offline, and the final weights obtained can be programmed on to the crossbar deterministically. But, since MTJs have an inherently stochastic switching behavior, deterministically programming them on a crossbar would require currents having high magnitude and duration to guarantee successful write operations. The possibility of selecting synapses to be written in the 1T1R architecture ensures no *side effects* of this method stemming from alternate current paths (because there would be none). Yet despite circumventing this issue, this architecture can suffer from performance degradation due to the intrinsic DVs that only aggravate with scaling. However, in a 1R architecture, such high programming currents, when they sneak through alternate paths, are bound to cause unwanted changes in neighboring synapses owing to which the weights may never converge. This necessitates in situ training of the crossbar in a probabilistic way for both 1T1R and 1R configurations, as only training on the hardware can account for both alternate paths and device variability.

3.2 Network Binarization and MTJ as a Synapse

Simply using ±1 as the binary weight values, represented by the P and AP states of an MTJ, is naive, and estimating a good scaling factor b is essential for overall network performance. An appropriate way to determine a suitable b is to minimize the L2 loss between the real-valued (RV) weights W and quantized ones, as was done in Rastegari et al. [40]. This provides a solution $b = \|W\|_1/n$, which is the mean of absolute values of W (n being the number of elements in W). Thus, an MTJ in the P (AP) state would signify a weight of $+b$ ($-b$).

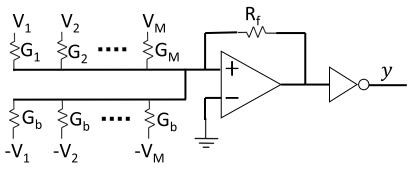
The weights of an NN are almost always bipolar, whereas the conductance of an MTJ or memristor is always positive. One method to realize negative weights is to effectively offset the conductance with a fixed bias; in the case of MTJs, we choose $G_{bias} = (G_P + G_{AP})/2$, as it brings symmetry to the effective conductance: $G = G_P - G_{bias}$ would correspond to the positive weight, say $+b$, and $G = G_{AP} - G_{bias}$ would correspond to the negative weight, say $-b$. These bias resistors are fed with the negative of the input voltages, and the output current in any column of the crossbar can be obtained by adding the bias currents to the current received from the MTJ synapses. In other words, the total output current can be written as

$$I = \sum_i (G_i - G_{bias})V_i = \sum_i (G_i V_i + (-G_{bias} V_i)) = \sum_i (I_i + I_{bias,i}). \tag{6}$$

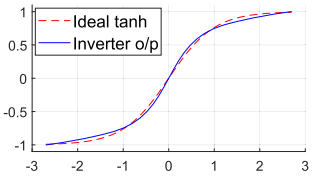
Figure 3(a) depicts the implementation of Equation (6), with the inverter producing the response of the *tanh* activation function (which we have used in all of our NNs) as shown in Figure 3(b). The average and maximum errors between the ideal value and the inverter output are 0.0327 and 0.0606, respectively, which are marginally better than the sigmoidal circuit proposed in Khodabandehloo et al. [19].

4 IN SITU TRAINING OF MTJ NN CROSSBARS

We first provide a high-level understanding of how an MTJ synaptic crossbar implementing a feed-forward NN should work. For the sake of simplicity, all operations are described for a single-layer NN and can be easily scaled to multiple layers (more details subsequently). We then illustrate how



(a) An M-input neuron with MTJ synapses and bias resistors, with an inverter for the activation function.



(b) Transfer characteristics of the inverter.

Fig. 3. (a) Synaptic weights and activation function in each column of the crossbar. G_b is the constant resistor that creates the bias. (b) Comparison of the output characteristics of the inverter and the actual *tanh* function. Producing this behavior requires the relation $MR_f V_{rd}(G_P - G_{AP}) = 7.2$ to be satisfied with inverter output load of $10k\Omega$. V_{DD} and V_{SS} of the inverter are $1.8V$ and $-1.8V$, respectively.

Table 1. Write Phase

Input	Error	ΔW	W and G	Switch
$x > 0$	$\delta > 0$	$\Delta W < 0$	*Decreases*	$P \rightarrow AP$
$x > 0$	$\delta < 0$	$\Delta W > 0$	*Increases*	$AP \rightarrow P$
$x < 0$	$\delta > 0$	$\Delta W > 0$	*Increases*	$AP \rightarrow P$
$x < 0$	$\delta < 0$	$\Delta W < 0$	*Decreases*	$P \rightarrow AP$

Signs of x, δ, and ΔW required change in weight W and conductance G, and the desired direction of switching of the MTJ synapse.

the gradient descent method can be used for the stochastic weight update of MTJs and finally describe the in situ training procedure for the two crossbar architectures.

4.1 Overview of Operations

The training process is carried out as follows.

Read phase. Upon receiving a training input $x \in R^M$, the input terminals are applied with voltages $V_i^r \in [-V_{rd}, V_{rd}]$ $\forall$ i proportional to x_i, whereas the output terminals are maintained at ground potential. Current $I_{ji} = G_{ji} V_i^r$ flows through the (j, i) synapse, and the total current I at the output terminals are suitably converted to output y.

Write phase. Using y and the desired output, calculate the error δ. Table 1 lists the four possible cases of weight update depending on x and δ. The gradient descent algorithm requires a weight update of the form of Equation (2). An appropriate way to realize this, as suggested in Lee and Likharev [26], is to set switching probabilities proportional to (the magnitude of) Δw calculated in Equation (2). Our way of achieving this is explained next.

The process of read and write are carried out for each input sample and repeated for several iterations until convergence is achieved.

4.2 Stochastic Learning of an MTJ Synapse

We will now describe how the stochasticity of MTJ switching can be used to perform weight updates with the gradient descent method. Just as the weight update in Equation (2) is a function of two variables (the input and the error), the probabilistic switching of MTJs can be controlled by two physical quantities: the magnitude and the duration of the programming current. We choose the magnitude of the write current to be dependent on the input x_i and the duration on the error δ_j. However, as can be seen from Equation (5) and Figure 2, the switching probability P is a highly non-linear function of the parameters a and t (recall $a = I/I_{c0}$), whereas the desired probability,

being proportional to ΔW_{ji}, is a linear function of x_i and δ_j. Further, the switching probability does not immediately rise with the pulse width and the write current as they increase from 0, indicating some kind of soft threshold. Note that the direction of switching can be decided by the polarity of the write current.

We therefore model switching probabilities by a linear mapping of x and δ to write current I_{wr} and duration t_{wr}, respectively, as follows. Usually $|x| \leq 1$, and henceforth assume for simplicity that $|\delta| \leq 1$ (can be ensured by normalizing and adjusting with η). The pulse width t_{wr} is set at a minimum of t_0 and increases linearly with $|\delta|$ (since t_{wr} needs to increase irrespective of the sign of δ) as

$$t_{wr} = t_0 + t_1|\delta|. \tag{7}$$

Similarly, the write current (I_{wr}) would be a minimum of I_0 and increase linearly with $|x|$ as

$$I_{wr} = I_0 + I_1|x|. \tag{8}$$

We now wish to find coefficients t_0, t_1, I_0, and I_1 that yield MTJ switching probabilities (P) close to the desired probabilities of weight update. A certain probability of switching can be obtained for different combinations of I and t, as is evident from Figure 2. We first fix the range of pulse widths by choosing suitable t_0 and t_1 (refer to Table 3). We want a nearly 0 switching probability for $t_{wr} = t_0$ irrespective of the value of I_{wr} because $\Delta W = 0$ for $\delta = 0$ regardless of x. We thus choose the maximum I_{wr} (which is $I_0 + I_1$) to be that value of I for which the plot of P against t_{wr} starts rising at t_0. In other words,

$$P(I_0 + I_1, t_{wr}) \; is \begin{cases} < P_0 & \text{for } t_{wr} < t_0, \\ \geq P_0 & \text{for } t_{wr} \geq t_0, \end{cases} \tag{9}$$

where P_0 is a small value. So now even if $|x|$ is (as high as) 1, $P = P_0$. In our experiments, we chose P_0 to be about 0.05.

A symmetric argument holds when $x = 0$. For $t_{wr} = t_0 + t_1$, we want $P \approx 0$ if $I_{wr} = I_0$, (because $\Delta W = 0$ for $x = 0$). But P should start increasing as soon as I_{wr} increases. In other words,

$$P(I_{wr}, t_0 + t_1) \; is \begin{cases} < P_0 & \text{for } I_{wr} < I_0, \\ \geq P_0 & \text{for } I_{wr} \geq I_0. \end{cases} \tag{10}$$

Figure 4 shows how well the linear model approximates the required $AP \rightarrow P$ switching probabilities (similar curve fitting for $P \rightarrow AP$ as well). Table 2 shows the write currents and duration for boundary values of $|x|$ and $|\delta|$, and Table 3 lists the values of the coefficients in Equations (7) and (8). One could use non-linear models for mapping $|\delta|$ and $|x|$ to t_{wr} and I_{wr}, respectively, to better fit the desired switching probabilities; however, that would complicate the analog circuit responsible for the conversion. Owing to this, and the closeness with which the linear model can replicate the stochastic switching characteristics, we stick to the linear version.

While training neural nets, it is necessary to have a small learning rate to avoid getting stuck in local minima. In our training strategy, this means having small probability values of MTJ switching. However, it is necessary to ensure that the probabilities are not so low that they barely cause any changes in the weight values. As can be seen from Figure 4, in the average case of having $|x| = |\delta| = 0.5$, the $AP \rightarrow P$ switching current and pulse width are $I = 75\mu A$ and $t_{wr} = 2.0ns$, respectively, and the switching probability stands at around 10%. The model parameters t_0, t_1, I_0, and I_1 have been adjusted so that probability values are within a reasonable range—that is, neither too small nor too large and help the training process to converge.

Next, we describe the 1T1R and 1R crossbar architectures implementing the NN. We show how these can be trained in situ using the stochastic learning technique described earlier.

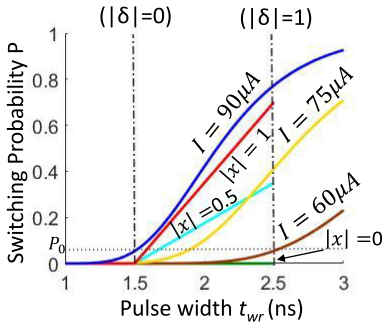


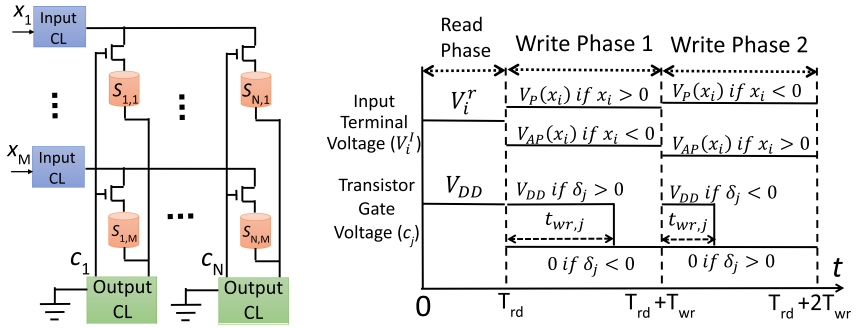
Fig. 4. P vs t_{wr} of the linear model and desired probabilities (obtained with $\eta = 0.7$) for the $AP \rightarrow P$ transition. The region between the dashed vertical lines is of interest. The dark green, cyan, and red straight lines plot desired probabilities for $|x| = 0, 0.5$, and 1, respectively. The brown, yellow, and blue plots correspond to the actual switching probabilities (obtained from the linear model) for the mapped currents $I = 60\mu A, 75\mu A$, and $90\mu A$.

Table 2. Boundary Values of the Parameters in the Weight Update Equation (2) and Their Counterpart in Probabilistic Switching of the MTJ

Weight Update	MTJ Switching
$\|\delta\| = 0$	$t_{wr} = t_0$
$\|\delta\| = 1$	$t_{wr} = t_0 + t_1$
$\|x\| = 0$	$I_{wr} = I_0$
$\|x\| = 1$	$I_{wr} = I_0 + I_1$

Table 3. The Coefficients That Fit the Model for Both $AP \rightarrow P$ and $P \rightarrow AP$ Switching

Direction	$AP \rightarrow P$	$P \rightarrow AP$
t_0	$1.5ns$	$1.5ns$
t_1	$1ns$	$1ns$
I_0	$60\mu A$	$140\mu A$
I_1	$30\mu A$	$60\mu A$



(a) An $M \times N$ crossbar. **(b)** Write voltages and control signals.

Fig. 5. The 1T1R crossbar. (a) Schematic. (b) Read and write phases signals.

4.3 The 1T1R Architecture

This is the conventional architecture for memory applications where each cell has a selection transistor. One major advantage of being able to selectively turn off certain cells is that it disallows the presence of undesired sneak currents that lead to unnecessary power consumption at a minimum. Figure 5(a) shows a 1T1R crossbar where each MTJ synapse is connected in series with an NMOS transistor. Input and output terminals are interfaced with necessary control logic (CL). All the transistors in a single column will have a common gate voltage since the corresponding synapses are connected to the same neuron output and hence will always have the same error "δ" and write pulse width t_{wr}.

Figure 5(b) plots the signals during both the read and write phases. During the read phase ($0 \leq t \leq T_{rd}$), all transistors are turned on: $c_j = V_{DD} \ \forall \ j = 1 \ldots N$ so that all columns (neuron outputs) are read simultaneously. Inputs x_i are provided to their respective input CLs, which convert them to read voltages V_i^r. Output currents I_j are processed by the output CLs.

4.3.1 Updating the Crossbar. Decide the write currents that should be provided to each input row and the pulse widths for each output column as described in Section 4.2. Recall that the former depend on x and the latter on δ. The direction of the currents would depend on the sign of the desired weight update. Apply suitable write voltages at the input terminals while grounding the output terminals to 0.

For the (j, i) synapse, the write pulse width depends on only $|\delta_j|$, and the write current magnitude depends on $|x_i|$. But the direction of switching depends on the signs of δ_j and x_i (see Table 1) and has to be decided by the polarity of current. For example, two MTJ synapses belonging to the same row but different columns may have opposite signs of δ. Thus, despite having the same input x_i, they are required to switch in opposite directions and hence need write voltages of the opposite sign. This requires us to split the write phase into two parts as explained next.

Since the transistor gate control signals are connected to the output CLs, we can select or deselect a certain column based on information at its respective CL, which is the error δ. We therefore program the crossbar sequentially in two stages, with the columns updated in a given stage depending on the signs of δ. Each phase has a duration of T_{wr} (which need not be more than $t_0 + t_1$; see Equation (7)). The voltage signals in each phase are plotted in Figure 5(b) and detailed as follows:

(1) *Phase 1*: $T_{rd} \leq t \leq T_{rd} + T_{wr}$. Update the weights of the columns that had $\delta > 0$. Then, the transistor control signals would be

$$c_j = \begin{cases} V_{DD}, & \text{for } \delta_j > 0 \text{ and } 0 \leq t - T_{rd} \leq t_{wr,j} \\ 0, & \text{for } \delta_j < 0 \text{ or } t_{wr,j} \leq t - T_{rd} \leq T_{wr}. \end{cases} \tag{11}$$

And the write voltages applied at the input terminals would be

$$V_{wr,i} = V_P(x_i)u(x_i) + V_{AP}(x_i)u(-x_i), \tag{12}$$

where u is the unit step function.

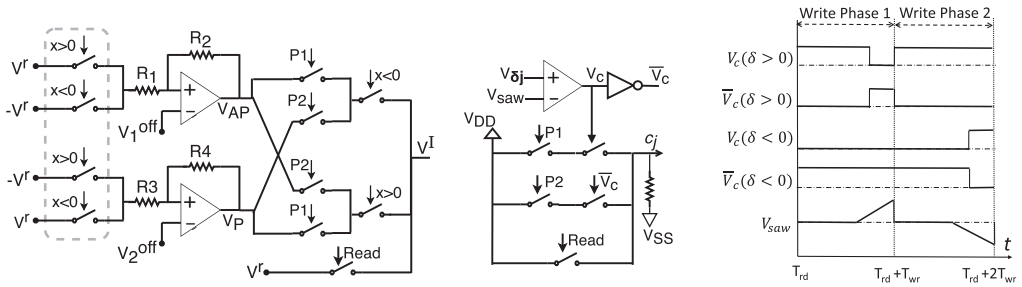
(2) *Phase 2*: $T_{rd} + T_{wr} \leq t \leq T_{rd} + 2T_{wr}$. Update the weights of those columns that had $\delta < 0$. Here, the signals are opposite to those in phase 1 as shown in Figure 5(b).

Here, V_P (V_{AP}) is the voltage applied to switch from P→AP (AP→P) and can be obtained using Equation (8) and $R_P(R_{AP})$. V_P and V_{AP} still depend on $|x_i|$, but for brevity explicit mention will be omitted henceforth. Let MTJs in the crossbar be arranged in a way that positive (negative) current from the i^{th} input terminal to j^{th} output terminal can switch $S_{j,i}$ from $P \rightarrow AP$ ($AP \rightarrow P$); hence, $V_P > 0$, ($V_{AP} < 0$). Parameters in Table 3 give $V_P \in [0.68, 0.98]$ volts and $V_{AP} \in [-0.81, -0.62]$ volts.

Thus, we can see that the read and update operations are completed in $T_{rd} + 2T_{wr}$ time, which is $O(1)$. The weight update is sequential with respect to the sign of δ, but it is done in parallel for all of those columns that have the same sign of δ.

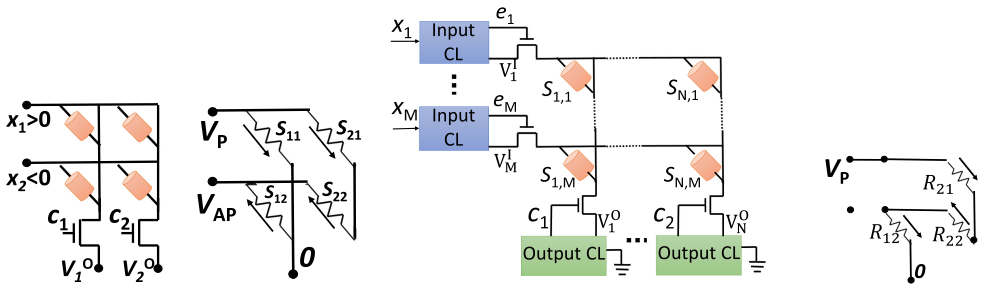
4.3.2 Control Circuits. Figure 6 shows the internals of the input and output CLs. In Figure 6(a), in the read phase, the read voltage V^r is directly passed on. The write voltages V_P and V_{AP} are obtained by suitably scaling V^r or $-V^r$, and shifting that by an offset to reach the desired range of values. Due to opposite polarities, V_{AP} is always obtained from a positive V^r, and V_P from a negative V^r, with the switches in the dashed green box thrown as per the sign of x. Switches controlled by P1 and P2 are "on" in write phases 1 and 2, respectively, and "off" otherwise. In our design, $V^r \in [-V_{rd}, V_{rd}]$ with $V_{rd} = 0.2V, R_2/R_1 = 0.95, R_4/R_3 = 1.5, V_1^{off} = -0.318V, V_2^{off} = 0.272V$.

Figure 6(b) depicts the CL in the output terminals to decide the duration of read and write phases by controlling the crossbar transistors' gate voltage c_j. We have $V_\delta \propto \delta$ (through circuits described in Section 4.5). If $\delta > 0$, V_C is high for some part of write phase 1 (as per Equation (11)), which pulls



(a) CL in the crossbar input terminals. **(b)** Output CL. **(c)** Write control signals.

Fig. 6. Circuit of the crossbar's input (a) and output CLs (b), and a write phase signals timing diagram (c).



(a) **(b)** **(c)** An $M \times N$ crossbar with 1R structure. **(d)**

Fig. 7. Alternate current paths in the 1R structure with a two-phase write strategy: a 2×2 crossbar (a); its equivalent circuit in write phase 1 with $c_1 = V_{DD}, c_2 = 0, V_1^O = 0, V_1^I = V_P, V_2^I = V_{AP}$ (b). (MTJ synapses are shown as resistors.) (c) Schematic of the proposed 1R architecture for the MTJ crossbar. (d) The equivalent circuit in phase 1 with four-phase writing.

c_j up to V_{DD} for the same duration. Similarly, for $\delta < 0$, $\overline{V}_C$ and c_j are high for a part of write phase 2. Figure 6(c) illustrates the timing diagram of V_C and its complement for the two possibilities of δ, and also of the sawtooth waveform for generating V_C. In phase 1, V_{saw} stays 0 until time $t_0 = 1.5ns$ and then rises linearly to the maximum value of V_δ. In phase 2, the behavior is the same but with an opposite polarity.

Due to limitations on the scalability of 1T1R architecture, it is worth exploring the feasibility of transistor-less crossbars to achieve even higher density of integration.

4.4 The 1R Architecture

Eliminating the need to have an access transistor for every synapse in the crossbar will allow for compact designs having an integration density of about $4F^2$/device. But the inability to select the synapses to be updated during programming results in leakage currents through alternate paths that not only waste energy but also can lead to undesirable changes in synaptic conductance. We first see the effect of such currents with the previously proposed write strategy and then suggest a modified strategy (and circuit) for the 1R architecture.

4.4.1 Two-Phase Update. Let us analyze the impact of sneak paths on the 1R crossbar with the two-phase update strategy used previously. We first demonstrate the presence of sneak paths with a small example. Figure 7(a) shows a 2×2 crossbar with transistors only at the output terminals (to choose columns to be written in any particular phase). Assume without loss of generality that

Table 4. Four-Phase Weight Update for the 1R Configuration in Figure 7(c): Condition on Input and Error for a Synapse to be Updated, Along with the Control Signals (e, c) and Write Voltages (V^I), for Each Phase

	Input	Error	e_i	V_i^I	c_j	Switch
Phase 1	$x > 0$	$\delta > 0$	$u(x_i)V_{DD}$	$u(x_i)V_P$	$u(\delta_j)V_{DD}$	$P \rightarrow AP$
Phase 2	$x < 0$	$\delta > 0$	$u(-x_i)V_{DD}$	$u(-x_i)V_{AP}$	$u(\delta_j)V_{DD}$	$AP \rightarrow P$
Phase 3	$x > 0$	$\delta < 0$	$u(x_i)V_{DD}$	$u(x_i)V_{AP}$	$u(-\delta_j)V_{DD}$	$AP \rightarrow P$
Phase 4	$x < 0$	$\delta < 0$	$u(-x_i)V_{DD}$	$u(-x_i)V_P$	$u(-\delta_j)V_{DD}$	$P \rightarrow AP$

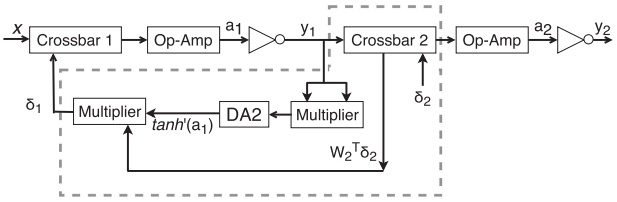
a certain input x with $x_1 > 0, x_2 < 0$ produced errors $\delta_1 > 0, \delta_2 < 0$ at the outputs. The equivalent circuit during write phase 1 is drawn in Figure 7(b). It depicts the currents through the synapses, with the ones through S_{21} and S_{22} being undesired. These *may* falsely switch S_{21} from $P \rightarrow AP$ and S_{22} from $AP \rightarrow P$ if they are in P and AP states, respectively.

We now state a worst-case scenario for a crossbar with M inputs. If M is large, analysis using Kirchhoff's current law shows that the potential difference across an MTJ synapse could go as high as $(V_P - V_{AP})$. The current through such an MTJ, if in the P state, is $I = (V_P - V_{AP})/R_P$ and is high enough (recall $V_{AP} < 0$) to switch it from $P \rightarrow AP$. In the other extreme case, a potential difference of $(V_{AP} - V_P)$ leading to current $I = (V_{AP} - V_P)/R_{AP}$ through an MTJ in the AP state will switch it from $AP \rightarrow P$.

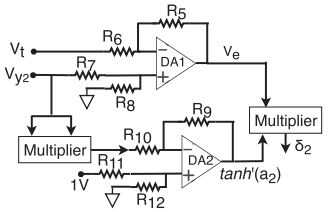
It is also necessary to mention an average (expected) case. Here, these currents reduce to $I = (V_P - V_{AP})/2R_P$ and $I = (V_{AP} - V_P)/2R_{AP}$, respectively, which are half of those found previously, but still have some probability of switching MTJs (because these currents are roughly the same as V_P/R_P and V_{AP}/R_{AP}). Thus, chances of unwanted flips of MTJs are quite significant, which calls for some modification in the circuit and/or in the programming method.

4.4.2 Four-Phase Update. The large sneak currents in the two-phase writing strategy, potentially resulting in false switching, are due to the high potential difference $V_P - V_{AP}$ between input terminals having different signs of inputs. One simple way to mitigate this issue is to further split the two phases of weight update so that, in a given phase, only rows having the same sign of input are updated at a time. This is equivalent to first clustering the columns according to the sign of δ and then further clustering the rows according to the sign of x. This proposed four-phase writing scheme would require additional transistors to choose the rows to be updated in a given phase as shown in Figure 7(c). It is summarized in Table 4 where each phase will have the same duration T_{wr}; thus, the total time for updating the crossbar is doubled to $4T_{wr}$. Note that this is still $O(1)$ time. The required write voltages and transistor gate voltages can be obtained with very similar circuits as in Figure 6(a) and (b). The V_{saw} here would be same in phases 1 and 2, and the opposite in phases 3 and 4. In this scheme, the programming currents for each row remain as they were in the two-phase update, just their time of appearance now differs. They still depend on the respective input and error.

Let us now see how bad the issue of sneak-path leakage is with this strategy. Figure 7(d) shows the equivalent circuit for the 2×2 crossbar with the same set of assumptions (only synapses providing alternate current paths are shown). For an $M \times N$ crossbar, in the worst-case scenario, sneak currents could be V_P/R_P and V_{AP}/R_{AP}, and can still result in false switching. This follows intuition as the potential difference between an input terminal and an output terminal is at most V_P or V_{AP}. However, in the average case, the sneak current values are found to be only $V_P/3R_P$ and $V_{AP}/3R_{AP}$. These currents are small and do not have the potential to cause undesired switching as is evident from the parameters listed in Table 3 and the range of values of V_P and V_{AP}. For example, the soft



(a) Error backpropagation requires two multipliers and a DA. The process involves the components within the dashed boundary.



(b) Computation of δ in the second layer.

Fig. 8. (a) Circuit for backpropagating errors to previous layers. (b) Circuit for finding the error at the last layer of the NN using the obtained value V_{y_2} and corresponding target V_t. In DA1, we need $R_5/R_6 = R_8/R_7 = 2$. For DA2, we use $R_9 = R_{10} = R_{11} = R_{12}$ to get $tanh'(a_2) = 1 - tanh^2(a_2)$.

switching threshold is about $45\mu A$ for $AP \rightarrow P$ switching with the maximum write pulse duration of $2.5ns$ (Figure 2(b)), whereas the average case sneak current is $30\mu A$. Similarly, for $P \rightarrow AP$, the threshold is about $105\mu A$, whereas the average sneak current is $67\mu A$.

Hence, the four-phase writing scheme significantly reduces the incidences of undesired switching at a small cost of increase in the duration of the write phase. As we shall see, this tradeoff is not only worthwhile but also necessary for satisfactory performance of the training process.

4.5 Multi-Layer NNs

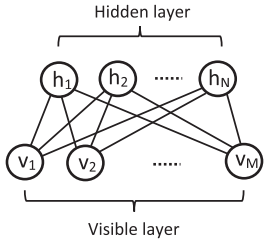
Multi-layer feed-forward NNs can be implemented on cascaded crossbars (each representing one layer) with the output of one fed as the input to the next. It is pretty straightforward to implement the backpropagation algorithm on such a structure, as demonstrated in Figure 8(a). Consider a two-layer NN with weight matrices W_1 (hidden layer) and W_2 (output layer) represented by crossbars 1 and 2, respectively. For an input x, the final output y_2 is given as

$$y_2 = f(a_2) = f(W_2 y_1) \;\; where \;\; y_1 = f(a_1) = f(W_1 x). \tag{13}$$

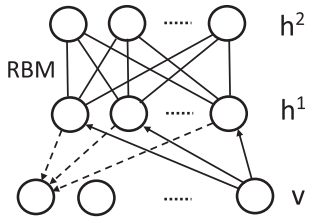
The op-amp and inverter following crossbar 1 (just as in Figure 3(a)) compute y_1 which is provided as an input to crossbar 2. With a mean square error cost function, the error of the second layer is given as $\delta_2 = 2(y_2 - t)f'(a_2)$, where t is the desired (target) value and f' denotes the derivative of activation function f. This is obtained as follows: y_2 and t, represented by V_{y_2} and V_t, respectively, are fed to the inputs of a differential amplifier (DA1) as shown in Figure 8(b) to obtain the difference $V_e = 2(V_{y_2} - V_t)$. We used $tanh$ as the activation function f; its derivative is given as $tanh'(x) = 1 - tanh^2(x)$, which is obtained using a multiplier, such as the Hilbert multiplier [21], followed by the differential amplifier DA2. Last, the outputs from DA1 and DA2 are multiplied to get δ_2.

The error of the first (hidden) layer is given as $\delta_1 = ({W_2}^T \delta_2) \times f'(a_1)$, where $\times$ represents a component-wise product. As depicted in Figure 8(a), the matrix-vector product can be done on crossbar 2 itself by reversing the roles of its input and output terminals: δ_2 is now fed as the input and out comes ${W_2}^T \delta_2$, which, when multiplied by $f'(a_1)$, gives δ_1 as the error to be used for updating the weights of the hidden layer.

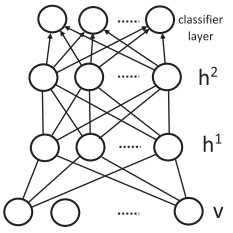
Note that DA1 is required only in the last layer of the NN to get the difference between the actual and target outputs, whereas the components for backpropagation, comprising the two multipliers and DA2, are present in all layers and are a part of the output CL. Also recall that the second layer error δ_2 has a dual role—deciding the MTJ write duration in crossbar 2 (with the circuit in Figure 6(b)), apart from being backpropagated to compute δ_1.



(a) An RBM with M visible and N hidden units.



(b) A DBN with two hidden layers.



(c) A DBN with a classifier layer at the end.

Fig. 9. (a) Schematic of an RBM. The absence of connections within the visible and hidden layers makes the units in any layer conditionally independent of each other. All weights are symmetric. (b) Schematic of a DBN. The top layer is an RBM being trained. In the lower layer, the dashed and solid arrows represent the top-down generative connections and bottom-up recognition connections, respectively [14]. (Note: All weights are still symmetric; some connections removed for clarity.) (c) The DBN in (b) with a layer at the end for classification.

For the MTJ crossbar NN we described, during forward propagation, the total duration of the read phase would be at most nT_{rd} for an n-layer NN. Backpropagation of errors to hidden layers would require an extra T_{rd}-long read phase for each such layer, during which the error at (the output of) a layer is fed as an input to its crossbar to obtain the error at its preceding layer. Last, all the layers can be updated simultaneously (in $2T_{wr}$ or $4T_{wr}$ time, as per the architecture).

Further, it must be mentioned that a large layer in an NN could be split into multiple crossbars, some of which share inputs or outputs. All of these crossbars can still be read and written in parallel, thanks to the locality of the weight update operations.

5 TRAINING OF RBMS

RBMs are a class of undirected graphical models used as generative models of data for the purpose of feature extraction, dimensionality reduction, and classification [14]. They form the fundamental building block of deep belief networks (DBNs), which have produced state-of-the-art results in learning tasks. In this section, we provide a simplified mathematical background of RBMs and DBNs, and describe the in situ training of RBM MTJ crossbars.

5.1 Basics of the RBM

RBMs consist of a set of visible and hidden units to represent the data and their features, respectively, and symmetric weighted connections between them as shown in Figure 9(a). The energy function of an RBM with visible and hidden units activations $\mathbf{v}$ and $\mathbf{h}$ and weights W is given as

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{h}^T W \mathbf{v} - a^T \mathbf{v} - b^T \mathbf{h}, \tag{14}$$

where a and b are vectors of biases for the visible and hidden units. The conditional probability of the hidden units, given a certain state of the visible units, is

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_i W_{ij} v_i\right), \tag{15}$$

where $\sigma(x) = 1/(1 + exp(-x))$ is the logistic sigmoid function. Similarly,

$$p(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_j W_{ij} h_j\right). \tag{16}$$

The marginal probability of observing a certain visible vector $\mathbf{v}$ is computed as

$$p(\mathbf{v}) = \frac{1}{Z}\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}. \tag{17}$$

Training of an RBM involves maximizing the log likelihood of the probability of the training data and gives rise to a gradient ascent rule in the weight space. The weight update is calculated as

$$\Delta W_{ij} = \epsilon \frac{\partial\, log\, p(\mathbf{v})}{\partial W_{ij}} = \epsilon(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model}), \tag{18}$$

where $\langle . \rangle$ denotes an expectation under the specified distribution and ϵ is the learning rate. Getting an unbiased sample of $\langle v_i h_j\rangle_{data}$ is simple because the absence of connections among the visible and hidden units lets us compute the probability with which the hidden units turn on using Equation (15). However, getting an unbiased sample for the model is difficult because it requires starting from a random training vector and performing alternate Gibbs sampling for a long time, where each iteration of the sampling process updates the hidden states in parallel using Equation (15) and then the visible states (again, in parallel) using Equation (16). The mathematical model of RBM considered only binary states, but this has long been extended to include continuous values and model different kinds of data distributions [22].

In Hinton [13], a much faster learning method was proposed wherein the first (positive) part of Equation (18) is computed using the hidden units' activations obtained from Equation (15). The second (negative) part is calculated by first reconstructing the visible units from the hidden states using Equation (16) and then the hidden units from these reconstructed visible units (Equation (15)). The weight update rule thus stands as

$$\Delta W_{ij} = \epsilon(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{recon}). \tag{19}$$

This works well even though it only roughly approximates the gradient of the log probability of the training data and is closer to the gradient of another objective function, the CD [13]. Observe that this learning rule is also local, just like the one in Equation (2). Further, since this CD weight update depends only on the training data and network parameters (the weights), and not on any labels, it comes under the category of unsupervised learning. The bias vectors a and b are also trained in a similar way.

One way to track the progress of learning is to measure the reconstruction error, which is the squared difference between the training data and its reconstructed version [15].

5.2 Deep Belief Networks

RBMs can be stacked to form deep generative models called *deep belief networks* [16]. The lower layers (which are close to the visible layer) capture low-level features, whereas the higher layers represent abstract concepts. Figure 9(b) illustrates a DBN with two hidden layers. In a DBN with l hidden layers, the joint distribution of the data $\mathbf{v}$ and the hidden layer variables $\mathbf{h}^1, \mathbf{h}^2, \ldots \mathbf{h}^l$ is expressed in terms of the conditional distributions [3].

$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \ldots \mathbf{h}^l) = P(\mathbf{v}|\mathbf{h}^1)P(\mathbf{h}^1|\mathbf{h}^2)\ldots P(\mathbf{h}^{l-1}, \mathbf{h}^l) \tag{20}$$

Hinton et al. [16] have proposed a greedy layer-wise training procedure for the DBN, starting with the lowest hidden layer $\mathbf{h}^1$. Once it has been trained using the CD formulation mentioned earlier, its weights are kept fixed and used to obtain the training data for the next layer $\mathbf{h}^2$. This is done by propagating the training samples $\mathbf{v}$ using the learned $P(\mathbf{h}^1|\mathbf{v})$ (computed using Equation (15)) and using either these probability values or samples from their distribution as training data for the second layer. This is repeated for all subsequent hidden layers up to $\mathbf{h}^l$.

Often, DBNs are not used on their own; rather, the features extracted by them are used for the purpose of classification by adding a classifier layer at the last hidden layer (Figure 9(c)) and using a supervised gradient descent algorithm to train the weights of this classifier. Another common practice is to use the weights of the trained DBN for initializing the hidden layers of a deep feed-forward NN. These weights are then fine tuned with the supervised training criterion, along with the weights of the classifier layer(s) appended at the end. This unsupervised pre-training of the hidden layers, before the data labels are used, is justified on the grounds that random initialization of the weights of hidden layers often leads to the network getting stuck at local minima when using only supervised gradient descent methods. This is specially problematic for the lower layers as their activations tend to get saturated, leading to vanishing gradients that slow down the learning process [9].

5.3 Adaptation of the CD Algorithm

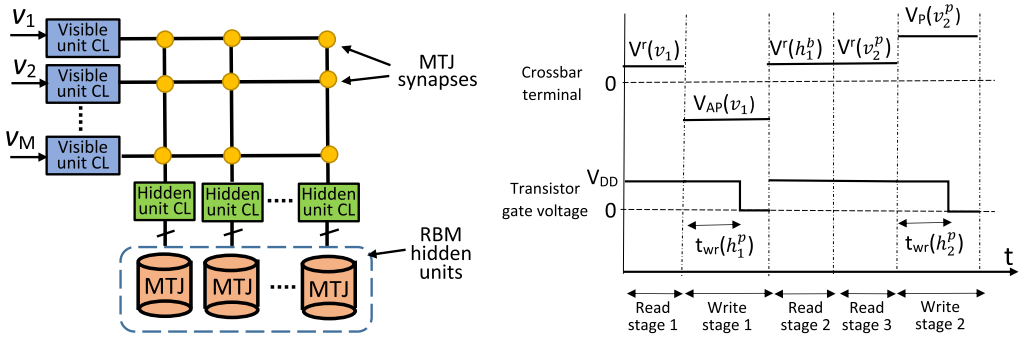
The standard CD algorithm is composed of the following steps:

(1) Clamp the visible nodes **v** to a training vector, say v_1.
(2) Find the probabilities with which the hidden units turn on using Equation (15). In other words, compute $h_1^p = \sigma(Wv_1)$ (ignoring the bias for simplicity).
(3) Obtain the binary states h_1^b of the hidden units by sampling from the probability distribution h_1^p. It is necessary to store the hidden states as binary values rather than using the RV probabilities themselves so that they can communicate a single-bit value during reconstruction, thereby acting as a strong regularizer [15]. This marks the end of a construction phase.
(4) Reconstruct the states of the visible units using those of the hidden units just as in Equation (16): $v_2^p = \sigma(W^T h_1^b)$. It is common to simply use these probability values as it reduces the sampling noise and hastens the learning process.
(5) Now reconstruct the hidden units as $h_2^p = \sigma(Wv_2^p)$. As per the recommendation in Hinton [15], it is *not* required to sample binary states from h_2^p so that unnecessary sampling noise can be avoided.
(6) Perform the CD weight update as

$$\Delta W = \epsilon\left(h_1^p v_1^T - h_2^p v_2^{pT}\right). \tag{21}$$

For the data-driven positive part of the weight update, it is better to use h_1^p because it eliminates the sampling noise present in h_1^b. Note that Equation (21) changes the weights with the statistics of only one training example and thus does not require the expectation operator. The aforementioned steps should be repeated for all training samples over several iterations.

Now we go on to explain how the CD algorithm would be adapted for implementation on the MTJ crossbar. The standard CD algorithm has a weight update in Equation (21) with two terms, each of which has activations of both the visible and hidden units. This makes it impossible to perform such a weight update on the crossbar without explicitly calculating and storing in memory at least the positive term. To avoid this, we choose to implement the updates from the construction and reconstruction phases separately. Further, since the v_1 and h_1^p are available at the end of step (2), the positive update can be done before the reconstruction. This further removes the necessity of storing v_1 and h_1^p while v_2^p and h_2^p are calculated. It has been observed that this two-step weight update does not quite affect the RBM's learning [46]. We shall verify this at a later stage.



(a) RBM MTJ crossbar and hidden layer. **(b)** Read and write voltages.

Fig. 10. (a) Crossbar implementation structure of the RBM with MTJs as synapses and hidden units. (b) Signals during the five stages of the CD update cycle. The quantities on which they depend are shown in parentheses. The crossbar terminal voltages are at the visible unit CL for all except read stage 2, where the hidden units provide an input for reconstruction of visible units. All reads and writes are of duration T_{rd} and T_{wr}, respectively.

In the construction phase, the binary states h_1^b of the hidden units are chosen by sampling from the probabilities h_1^p (step (3)). Since the probabilities are generated from the output of a sigmoid activation function, and the MTJ switching behavior (Figure 2(b)) bears close similarity to a sigmoidal response, we use an MTJ itself to produce and store the binary state of a hidden unit. The alternative to this would have been the use of some analog/digital random number generator to compare its output with h_1^p and generate a binary state; this is likely to have a higher overhead. We shall further discuss the implementation of this technique in the next section.

5.4 Training of the RBM MTJ Crossbar

Figure 10(a) depicts the RBM crossbar with CL for each visible and hidden unit, and an MTJ for storing the binary state of the latter. The MTJ synapses could be with or without selection transistors. Because the reconstructed values of the visible units are outputs of the sigmoid and restricted to the range $(0, 1)$, we would require inputs to the RBM to be normalized to the same range for better reconstruction. Each cycle of the CD algorithm implemented on the crossbar goes through five stages as listed next, and the signals interfacing the crossbar are shown in Figure 10(b):

- *Read stage 1*: The training starts with the crossbar visible terminals having a voltage $V^r \in [0, V_{rd}]$ proportional to the training input v_1. The current received at the hidden terminals would be used to flip the MTJ units storing the hidden states and simultaneously be converted to activations h_1^p.
- *Write stage 1*: For the positive weight update stage, since both v_1 and h_1^p are positive, we would only require to switch the MTJ synapses from $AP \rightarrow P$ with a suitable probability. Just as in Section 4.2, we linearly map v_1 and h_1^p to MTJ synaptic write current I_{wr} and pulse width t_{wr}, respectively, as

$$t_{wr} = t_0 + t_1 h_1^p, \tag{22}$$

$$I_{wr} = I_0 + I_1 v_1. \tag{23}$$

We use the same values of t_0, t_1, I_0 and I_1 as listed in Table 4, which give write voltages $V_{AP} \in [-0.81, -0.62]$ as previously.

- *Read stage 2*: The MTJs storing the binary states h_1^b of the hidden units are read, and the hidden terminals are applied a voltage V^r depending on the value read. Since h_1^b is binary (0 or 1), V^r is either 0 or V_{rd}. The reconstruction of visible units v_2^p is obtained using the current flowing into the other end.
- *Read stage 3*: A reconstruction of the hidden units (h_2^p) is obtained by feeding v_2^p to the crossbar. Unlike read stage 1, there is no need to sample binary states from h_2^p.
- *Write stage 2*: Last, the negative weight update, which would require MTJs to switch only from $P \rightarrow AP$, is carried out by passing currents with magnitude and duration proportional to v_2^p and h_2^p, respectively, just as in Equations (22) and (23). Only the polarity and current magnitudes are for $P \rightarrow AP$ switching, and $V_P \in [0.68, 0.98]$ volts.

The entire cycle thus takes $3T_{rd} + 2T_{wr}$ time. Since the logistic sigmoid σ is only a scaled and shifted version of $tanh$, the same circuit (i.e., the inverter) can be used to realize it, although with different parameters such as R_f and V_{SS}. The hardware required to implement the proposed training algorithm is also pretty much the same as that in Section 4.3.2 except h replaces δ as the quantity that decides write time of MTJs, the inverter in the output CL (Figure 6(b)) is not required, and V_{saw} is the same in both write stages.

In the training of the 1T1R NN crossbar in Section 4.3, the write stage had to be split into two because of the four possible combinations of the signs of the input and error. The RBM crossbar, however, does not require such splitting because the visible and hidden units' values driving the CD weight update are always positive, and weight updates of all synapses have the same sign in a given write stage.

The 1R crossbar in Section 4.4 had a four-way split of the write stage because a two-way split resulted in large sneak currents. However, an RBM crossbar with a 1R architecture would also have a single phase, for reasons the same as those of the 1T1R crossbar. In any given write stage, all synapses are updated, which means that all rows and columns are simultaneously active. Thus, transistors for selecting rows (the ones labeled e_i in Figure 7(c)) are not required, and only columns would have selection transistors (labeled c_i) to control their respective write pulse widths t_{wr}. Since there are no sneak paths during writing, the scalability of the 1R crossbar makes it the choice of architecture for an RBM.

At this stage, one may ask why training inputs to the general NN crossbar should be bipolar, as was considered in Sections 2.2 and 4.1. The explanation lies in the faster convergence of the training when inputs are bipolar or specifically have an average close to 0 [24]. If inputs x are normalized in $[0, 1]$, then the update of the weights connected to the j^{th} neuron ($x\delta_j$) would all have the same sign as that of error δ_j. Thus, these weights would always move together, making the training process inefficient and slow [24].

Figure 11(a) depicts two crossbars concatenated with each other forming a DBN with hidden layers h^1 and h^2. They would be trained sequentially using the procedure described earlier.

5.5 MTJs for Hidden Units

In read stage 1, the MTJ hidden units are provided with a switching current I_{sw} to switch them $AP \rightarrow P$ (say, P state is "on") with probability h_1^p. Their states are read in read stage 2 using a certain current I_{read}, and they are reset $P \rightarrow AP$ in either read stage 3 or write stage 3 in preparation for the next cycle. Figure 11(b) shows the circuit of the MTJ hidden units, and Table 5 summarizes its operation. The currents I_{sw} and I_{reset} flow in opposite directions to flip the MTJ from $AP \rightarrow P$ and $P \rightarrow AP$, respectively. The read current I_{read} could be in any direction.

We shall now provide a detailed description of how the stochastic switching behavior of MTJs is used for sampling the binary states of the hidden units. Table 6 summarizes the notations adopted.

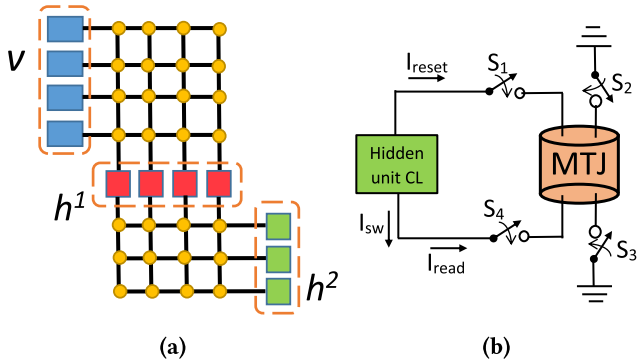


Fig. 11. (a) DBN crossbar structure: 4×4 and 4×3 crossbars for the first and second hidden layers. (b) Circuit of the MTJ as an RBM hidden unit connected to the respective CL.

Table 5. The Stages of CD Training When Different Currents (Figure 11(b)) Operate on the Hidden Units and the Switches That Are Active

Current	Switch	Stage
I_{sw}	S_4, S_2	Read 1
I_{read}	S_4, S_1	Read 2
I_{reset}	S_1, S_3	Read 3 or Write 2

Table 6. Notations for MTJ Curve Fitting

I_n	Neuron current at the hidden unit CL
I_{sw}	MTJ hidden unit switching current
a	Weighted sum input to sigmoid
I_0	Value of I_{sw} for $\sigma(0) = 0.5$ switching probability
I_{cf}	Value of I_{sw} for $\sigma(a_{cf})$ switching probability

We need to fit the transfer function of the sigmoid with the MTJ switching probability curve. A hardware-friendly method is a simple linear mapping of the incoming neuron current I_n to the MTJ's switching current, which requires us to match the characteristics at exactly at two points: say for values of the weighted sum $a = 0$ and a_{cf} ("cf" denotes curve fitting).

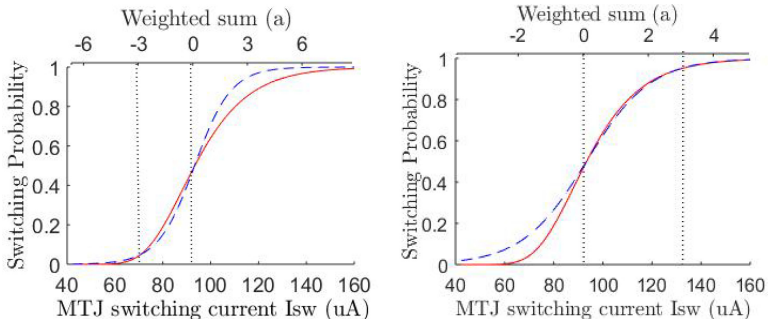
For the value of $a = 0$, we would have the neuron current $I_n = 0$. This should correspond to an MTJ switching current of I_0 and a probability of $\sigma(0) = 50\%$—that is, equal chances of the binary state to be 0 and 1. Recall from Section 3.2 that MTJ conductances G_P and G_{AP} correspond to synaptic weights b and $-b$, and read voltage V_{rd} to input of 1. Thus, for $a = a_{cf}$, we would have

$$I_n = a_{cf} \frac{V_{rd}(G_P - G_{AP})/2}{b}, \tag{24}$$

and the MTJ switching current should be $I_{sw} = I_{cf}$. This gives the relation

$$I_{sw} = I_0 + \frac{2bI_n(I_{cf} - I_0)}{a_{cf} V_{rd}(G_P - G_{AP})}, \tag{25}$$

which can be implemented using a differential amplifier. For our experiments, we perform the curve fitting at $a_{cf} = -3$. The reason behind choosing a large value is to cover a significantly wide range of values of activations. Figure 12(a) shows how close the MTJ switching probabilities are to the desired probabilities of activation as current I_{sw} and a are varied. The pulse width of I_{sw} has been chosen to be $2ns$, which is the duration of the read stages. In contrast, Figure 12(b) shows the same for $a_{cf} = 3$ wherein the probabilities match perfectly for positive values of a, but for $a < 0$ the MTJ switching probabilities obtained are significantly less than those of the desired values (i.e., the transfer curve of the sigmoid). It is crucial to capture the small probabilities; otherwise, values of $a < -2$ would produce currents I_{sw} that are too small to ever flip the MTJs and turn the



(a) With $a_{cf} = -3$. **(b)** With $a_{cf} = 3$.

Fig. 12. Switching probabilities of the MTJ (solid red line) meant to store the RBM hidden unit's states as a function of I_{sw}, and the desired hidden unit's activations as output of sigmoid (dashed blue line) for $a_{cf} = -3$ (a) and $a_{cf} = 3$ (b). Vertical dashed lines in the plots depict the matching at respective values of a_{cf}. In (a), the parameters are $I_0 = 93.28\mu A$, $I_{cf} = 70.38\mu A$. Maximum read voltage $V_{rd} = 0.2\ V$ has been used throughout.

hidden units "on." This will cause them to convey $h_1^b = 0$ in the reconstruction phase (step (4)) much more frequently than they should. In addition, one may consider not matching the curves at 0 and instead match at $a_{cf} = 3$ and -3; however, this results in poor fitting in intermediate values, with the difference in probabilities being higher than 0.15 for a wide range of values of a.

6 EXPERIMENTAL SETUP AND RESULTS

To see how successfully the MTJ crossbar NNs and RBMs[2] can be trained in situ, we performed system-level simulations by modeling the functionality of the crossbar architecture in MATLAB and training it on some datasets. To capture the MTJ device parameters, we used an HSPICE model [20] and included thermal fields in its LLG equations for obtaining the stochastic switching characteristics [43]. Certain device parameters used in and obtained from this model[3] were then incorporated into the simulations of the crossbar. We discuss the results obtained on feed-forward NNs and DBNs in that order.

6.1 Neural Networks

The performance of the NN was evaluated in the following scenarios (code named for further reference), in which all training processes used the mean square error cost function, and neurons had the *tanh* activation function:

(1) *RV*: First, we train and evaluate an NN with real-valued weights in MATLAB. Binary quantization step (b) is obtained from this trained network as shown in Section 3.2.
(2) *DP*: Suitable binary weights are obtained by doing probabilistic learning in software on a binary network. Then a 1T1R crossbar and a 1R crossbar are deterministically programmed to these weights. We see the effect of DVs on the former and of alternate current paths and resulting false switchings on the latter.
(3) *ST*: An MTJ synaptic crossbar is modeled and stochastically trained in situ using the linear model of stochastic weight update described in Section 4.2 for the

[2]A tutorial on DBNs, along with code in Theano, can be found in references [1] and [2].
[3]MTJ cell dimensions: $35nm \times 35nm \times 1.4nm$, $R_P = 4.86k\Omega$, $R_{AP} = 15.12k\Omega$, temperature $T = 300K$, saturation magnetization $M_s = 1029emu/cm^3$, damping constant $\alpha = 0.014$ yielded $\Delta = 40$, $I_{c0}^{P\to AP} = 64.5\mu A$, $I_{c0}^{AP\to P} = 21.2\mu A$.

Table 7. Classification Error Rates for the Three Datasets (on the Test Samples) With Various NN and Crossbar Architectures Under Different Training Scenarios

Dataset		SONAR			MNIST				WBCD		
Network		1L	2L15	2L25	2L50	2L100	2L150	3L	1L	2L10	2L20
RV		16.4	12.8	11.9	9.87	7.34	6.44	7.25	8.35	7.40	7.10
DP	1T1R	19.2	15.2	14.3	13.50	10.89	9.55	10.45	9.85	8.30	8.55
	1R	46.8	41.4	42.7	39.42	36.10	37.92	40.48	24.95	27.60	23.65
ST	1T1R	18.4	14.2	13.6	12.69	10.18	8.96	9.71	9.20	7.70	8.05
	1R	18.3	14.5	14.0	12.72	10.20	9.03	9.66	9.40	7.85	7.95

Here, ST-1R crossbar used a four-phase update. Ideal devices are assumed for all except DP-1T1R, where 10% variation was considered. SONAR and WBCD figures are an average of 10 runs. MNIST and WBCD figures are shown as a percentage (%).

(a) 1T1R architecture, with the two-phase write strategy (Section 4.3). and the
(b) 1R architecture, with both the two-phase (to see the effects of sneak currents) and the four-phase update strategies (Section 4.4). For the former, node voltages of output terminals not connected to the output CLs (i.e., columns not being updated) could be easily calculated using Kirchhoff's current and voltage laws, whereas for the latter, a mesh analysis of the crossbar was required and node voltages at both (unconnected) input and output terminals were obtained by solving a system of linear equations of KCL and KVL in MATLAB.

(4) *DV*: Device variations of different extents are introduced in the stochastic training (ST) of both the 1T1R and 1R crossbars. It reflects in the variations in the resistance of the *P* and *AP* states, the standard deviations of which usually do not exceed 10% of their mean values as per experiments [56].

We use the following datasets for evaluation:

SONAR, Rocks vs Mines [30]: Three different NN architectures are considered: one with one layer (1L) and two with two layers having 15 and 25 hidden neurons, respectively, and named 2L15 and 2L25. They were trained and then tested on 104 samples of the test dataset.

MNIST Digit Recognition [23]: Three two-layer networks of 50, 100, and 150 hidden units, respectively, and a three-layer network of 50 + 25 hidden units were trained on the first 10,000 samples of the training set and then evaluated on the 10,000 images of the test dataset.

Wisconsin Breast Cancer (Diagnostic) *(WBCD)* [30]: A single-layer network (1L) and two two-layer networks (2L10 and 2L20) were considered, and the test dataset had 200 samples.

Table 7 summarizes the accuracy obtained with these networks under the different training scenarios mentioned earlier. The effect of DVs of different extents on the in situ ST is highlighted for some of the networks in Table 8, with Figure 13 plotting the mean square error as the training progresses for the 1R crossbar. Additionally, Figure 14 compares the error for the two write strategies. It does not converge with the two-phase writing scheme due to higher instances of undesired weight changes but does so with four phases.

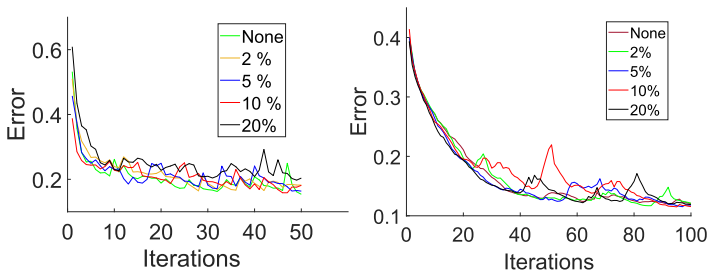
From these results, the following are evident:

- When an MTJ synaptic crossbar without access transistors is stochastically trained in situ (ST-1R), it shows classification accuracy only slightly lower (about 3% at worst) than when the same network is trained in software with RV weights (which can be considered to be the best achievable). However, it brings about significant improvement (up to 30%) in accuracy over a deterministically programmed crossbar (DP-1R) since the latter suffers from undesired weight changes arising from alternate current paths.

Table 8. Misclassification Rates of NNs With ST of 1T1R and 1R Architectures Under Different Levels of DVs Expressed in Terms of Standard Deviations of R_P and R_{AP} About Their Mean Values

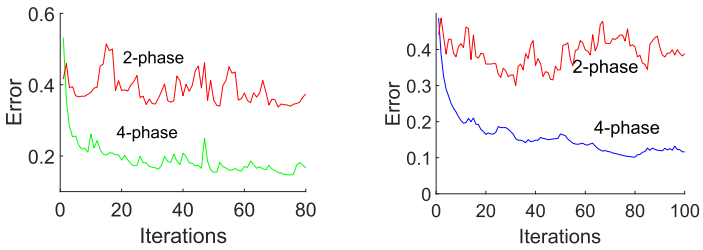
Dataset	SONAR				MNIST				WBCD	
Network	1L		2L15		2L100		3L		2L20	
Variation	1T1R	1R	1T1R	1R	1T1R	1R	1T1R	1R	1T1R	1R
2%	18.5	18.4	14.4	14.7	10.27	10.22	9.67	9.73	8.10	8.05
5%	18.7	18.7	14.7	14.8	10.28	10.29	9.78	9.80	8.25	8.30
10%	19.0	19.1	15.1	15.1	10.33	10.43	9.86	9.91	8.30	8.40
20%	19.3	19.5	16.0	15.9	10.42	10.72	10.15	10.28	8.60	8.75

MNIST gures are the worst of 3 runs. SONAR and WBCD are an average of the 10 worst runs.



(a) On SONAR for the 2L15 network. **(b)** On MNIST for the 3L network.

Fig. 13. NN training error with different extents of DVs on the 1R crossbar for two datasets.



(a) On SONAR for the 2L15 network. **(b)** On MNIST for the 2L100 network.

Fig. 14. Comparison of error during training of the 1R crossbar with two- and four-phase update schemes for two datasets. No variations are assumed.

- In situ training also benefits the crossbar with transistors (ST-1T1R against DP-1T1R) in the presence of DVs by slightly improving accuracy (by about 0.5% to 1%).
- It is possible to compensate for the loss in accuracy due to use of a binary network by increasing the size of the network (adding more hidden layers and/or neurons).
- Further, the trained crossbar has robustness even in the face of DVs, owing primarily to the fault-tolerant nature of the NN and its learning algorithms. As can be seen in Table 8, increase in misclassification rates remain within 2% even with 20% variation.

The accuracy degradation of 2% to 3% that we achieve (on going from RV to ST) is comparable to the 3.73% reported by Zhang et al. [60] and the 0.8% to 3.5% in Vincent et al. [52]. However, it must be mentioned and emphasized that any comparison is fair only if it is on the same dataset and network architecture. The benefit of using in situ training can also be seen when we compare our work to that of Zhang et al. [59] (which performs offline learning). On the MNIST 2L100 network,

we obtained an error rate of 10.20%, whereas Zhang et al. [59] had a much higher value of 30% on the same network, although it must be mentioned that the latter was at a disadvantage due to linear activation units. Further, the presence of a 20% device variability reduces our accuracy by less than 1.5%, which is competitive with that of Zhang et al. [60], Zhang et al. [59], and Vincent et al. [52].

There are a few similarities and differences between our work and the MTJ synapse–based STDP learning proposed in Srinivasan et al. [48] that we would like to first mention. In our work, all MTJ synapses from an input share the circuit that decides programming currents, and all synapses to an output neuron have the same programming duration. Similarly, in Srinivasan et al. [48], the STDP learning circuit for synaptic potentiation (and depression) are shared by the synapses that connect an input neuron (pre-neuron) to all excitatory neurons. However, they get programmed with different currents depending on their respective post-neuron spiking time, but their write durations are always the same and independent of any spike times; it is only the write current that varies from synapse to synapse. The STDP learning circuit consists only of two sets of two transistors and a capacitor. However, our implementation of stochastic learning would require more complex hardware (primarily two op-amps in the input CLs and one op-amp in the output CLs). Additionally, we need multiplier circuits [21] to back-propagate errors to hidden layers of the network.

However, Srinivasan et al. [48] have not described the hardware implementation of the neurons of the SNN and their functionality, although one possible configuration appears in Sengupta et al. [42]. This neuron in Sengupta et al. [42], although not being very complicated, is seemingly more expensive than an op-amp [21] in terms of area requirements. But overall complexity is perhaps higher for our design. However, Srinivasan et al. [48] achieve only about 75% classification accuracy on the MNIST dataset on an SNN with as many as 400 excitatory and 400 inhibitory neurons trained with 460 images. We could get higher accuracies on our MNIST networks, although we trained with many more (10,000) images.

6.2 Deep Belief Networks

For the training of the RBM crossbar, we consider only a 1R architecture since the absence of sneak currents (as discussed in Section 5.4) does not leave any difference in the training procedure of the 1T1R and 1R crossbars. The performance of the DBNs was evaluated in scenarios similar to Section 6.1: first with RV weights, then deterministically programming the MTJ crossbar to suitable binary weights, and finally performing ST of the crossbar without and with various extents of DVs. Two datasets were used for obtaining data:

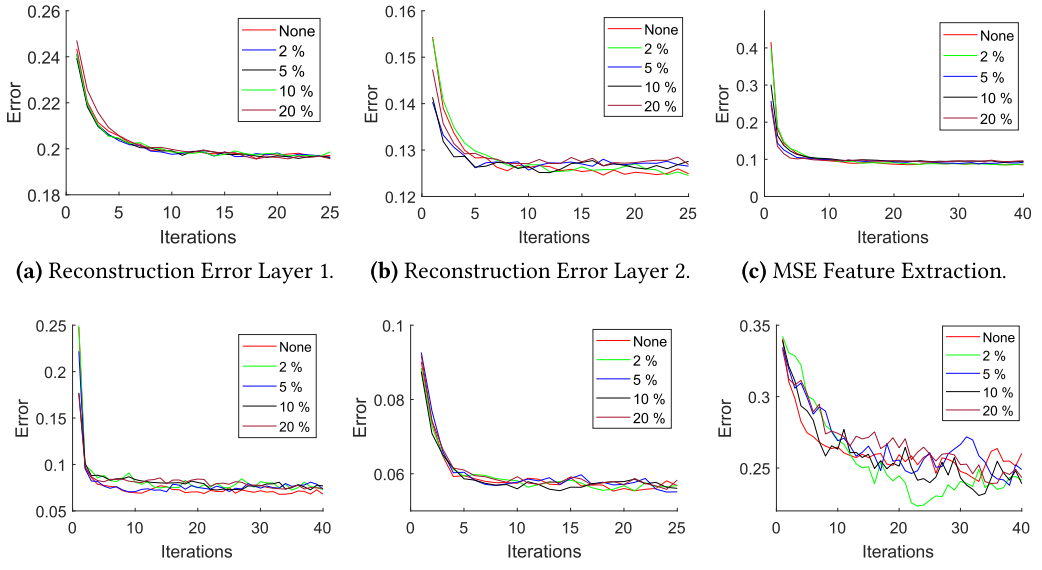
- ***MNIST***: Two two-layered networks with 150 and 200 hidden units, and two three-layered networks with 150 and 200 units in each hidden layer, were trained on the first 10,000 samples of the training set and then evaluated on all test samples.
- ***WBCD***: One two-layered network of 40 hidden units was considered.

The last (output) layer of all networks was used for classification purposes, either using only the Features Extracted (FE) at the last hidden layer as classifier inputs or performing Supervised Fine Tuning (SFT) of the hidden layer weights, along with training of the output layer, with the supervised training method used in NNs (refer Section 5.2). Table 9 lists the classification error rates obtained with all networks and training scenarios. As is clearly evident, it remains within 4% to 5% and 3% to 3.5% for FE and SFT, respectively, even with high levels of variations. Figure 15 depicts how the different kinds of errors converge both with and without variations. For MNIST, plots of only the DBN with 200 + 200 hidden units are shown.

Table 9. Misclassification Rates With Hidden Layers Trained as an RBM for the MNIST and WBCD Datasets for Different Levels of DVs

Dataset	MNIST								WBCD	
Network Size	150		150 + 150		200		200 + 200		40	
Purpose	FE	SFT	FE	SFT	FE	SFT	FE	SFT	FE	SFT
RV	8.85	6.63	8.07	5.12	8.73	5.30	7.98	4.27	7.90	1.30
DP	38.29	34.40	37.83	41.54	39.17	36.53	37.92	40.07	24.00	29.40
ST	12.72	8.82	11.74	8.23	12.69	8.09	11.40	7.08	11.60	4.10
DV 2%	12.93	8.97	11.77	8.33	12.75	8.21	11.58	7.19	11.80	4.20
DV 5%	13.05	8.96	11.89	8.55	12.84	8.34	11.76	7.27	12.20	4.50
DV 10%	13.22	9.18	12.15	8.70	13.02	8.71	12.00	7.34	12.50	4.50
DV 20%	13.56	9.29	12.44	9.12	13.35	8.87	12.39	7.72	12.60	4.70

For the figures reported with 2 to 20% variations, the ones for MNIST are the worst of 3 runs and those for WBCD are an average of 5 worst runs out of 10.

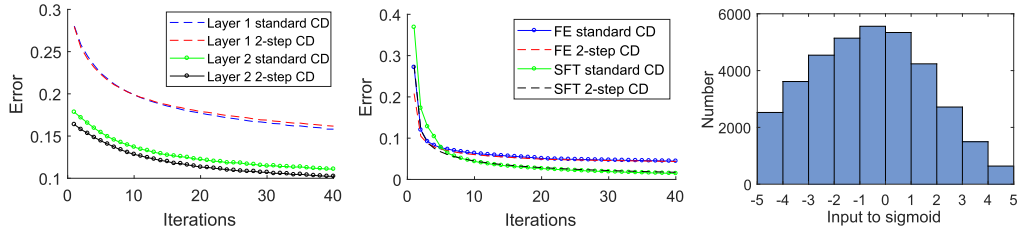


(a) Reconstruction Error Layer 1. **(b)** Reconstruction Error Layer 2. **(c)** MSE Feature Extraction.

(d) MSE Supervised Fine Tuning. **(e)** Reconstruction Error. **(f)** MSE Supervised Fine Tuning.

Fig. 15. Progress of training with RBMs as hidden layers for the MNIST dataset, with two hidden layers each of 200 units (a–d) and the WBCD dataset, one hidden layer with 40 units (e, f). (a, b) The reconstruction error on the first and second layers, respectively. (c, d) The classification mean square error with FE and SFT.

Additionally, Figures 16(a) and (b) compare the training with the standard CD algorithm and the two-step CD that we use to train the MTJ crossbars, as described in Sections 5.3 and 5.4, respectively. Real continuous weights were used for the sake of this comparison on the MNIST 200 + 200 network. Both the reconstruction errors and the classification MSE of the two different implementations of CD are barely distinguishable. Last, Figure 16(c) depicts the bias of the weighted sum inputs of the hidden layer toward negative values, which justifies the tight curve fitting for $a < 0$ done in Figure 12(a). Apparently, the reason for this bias was the average input value (across all input units and data samples) being less than 0.5 for both MNIST and WBCD. This required a reconstruction value (of visible units) of less than 0.5 for low errors, which tend to shift the



(a) Reconstruction Error Layer 1 & 2. **(b)** MSE - FE and SFT. **(c)** Distribution of weighted sums.

Fig. 16. (a, b) Comparison of standard CD and two-stage CD in terms of how the reconstruction errors (a) and classification MSE (b) converge with iterations of training. The small gap between the green and black dotted lines (representing standard and two-step CD, respectively) is only due to the different initializations of the weights of the second hidden layer. Importantly, this gap remains the same throughout, indicating a same rate of convergence. (c) Histogram showing distribution of inputs to the sigmoid activation function of the first hidden layer. All plots are on the MNIST 200 + 200 network.

weights to negative values during the learning process so as to obtain negative weighted sums on an average. Other datasets with different characteristics may be suited to a different fitting (e.g., the one in Figure 12(b)) that can be easily done using techniques described in Section 5.5.

In Neftci et al. [35], a spiking neuromorphic system trained with event-driven CD was used for learning MNIST digits. The architecture had additional neurons in the visible layer for class labels (since the RBM was discriminative [22]), and the weights connecting the 500 hidden neurons to these class neurons were also trained using CD. Their model had a recognition error of 8.1%. The hybrid RRAM-CMOS RBM architecture of Suri et al. [49] obtains an average error rate of about 11% with 100 neurons in the hidden layers and a separate classification layer (which is similar to our approach and unlike that of Neftci et al. [35]). Last, the memristor-based RBM in Sheri et al. [46], with 500 hidden neurons and 40 additional visible nodes, classifies 87.55% MNIST digits correctly but achieves convergence within only five epochs of 10,000 training samples.

7 DISCUSSION

We now analyze several other aspects of the in situ training method proposed by us:

- *Training time*: Training of an n-layer NN on one 1T1R crossbar using gradient descent will take $(2n-1)T_{rd} + 2T_{wr}$ time per training sample per iteration, as per Section 4.5. However, consider a DBN with $(n-1)$ hidden layers and a final classification layer. We saw that the two-step CD algorithm takes $3T_{rd} + 2T_{wr}$ per sample for an RBM. A DBN would be trained layer wise where the training data for the r^{th} layer would be first obtained by propagating the original training sample through the preceding $r-1$ hidden layers, assuming that there is no storage of data in on-chip or off-chip memory. Thereby, the total time for the r^{th} layer is $(r+2)T_{rd} + 2T_{wr}$; summing over r from 1 to $(n-1)$ gives a quadratic dependence on n for duration of training of the hidden layers. After this unsupervised learning, training of the classifier layer through gradient descent would take $nT_{rd} + 2T_{wr}$ if only the features extracted by the last hidden layer are used, whereas if we go for supervised fine tuning of the entire network, it is $(2n-1)T_{rd} + 2T_{wr}$.

  The higher time requirement for DBN training may be justified by the relatively smaller number of training iterations (typically 10 to 20) within which the reconstruction error and MSE converge as compared to the larger number of iterations required if the network is trained entirely in a supervised way (compare Figure 15 to Figure 13). If in the DBN the

training data for subsequent hidden layers is stored instead of calculated, then memory can be traded off for a linear dependence of training time.

- *Power consumption in the crossbar*: Let us estimate the expected power dissipated in the 1T1R MTJ crossbar NN by assuming an average case for all parameters. All inputs x and δ are half of maximum, that is, ± 0.5. Thus, read voltages are half of maximum, that is, $V_{rd}/2$, and write voltage are those for $x = 0.5$, that is, $V_P(0.5)$ or $V_{AP}(0.5)$, denoted $\overline{V_P}$ or $\overline{V_{AP}}$. At all times, half of the MTJs are considered to be in the P state and the other half in the AP state. This gives an average power in the read phase per MTJ synapse to be

$$P_{rd} = \frac{1}{2}(V_{rd}/2)^2 \left(\frac{1}{R_P} + \frac{1}{R_{AP}}\right) \tag{26}$$

and that in each of the two write phases to be

$$P_{wr} = \frac{1}{8}(\overline{V_P}^2 + \overline{V_{AP}}^2) \left(\frac{1}{R_P} + \frac{1}{R_{AP}}\right). \tag{27}$$

This yields the average power per device in a cycle of training to be $P_{rd} + 2P_{wr}$. Substituting values stated previously, this calculates to 82 μW. Taking $T_{rd} = 2ns$ and an average $t_{wr} = 2ns$, the energy consumed per device per cycle is 0.164 pJ. The 1R crossbar NN without transistors would have higher energy dissipation due to sneak currents. It must be noted that these values are heavily dependent on device parameters. Future MTJ technologies with scaled-down devices would consume lesser energy.

For the RBM crossbar, P_{rd} remains the same. Write stages 1 and 2 have average write voltages $\overline{V_{AP}}$ and $\overline{V_P}$, respectively, and current flows through all synapses in both stages. Thus, average power per synapse per cycle is $3P_{rd} + \frac{1}{2}(\overline{V_P}^2 + \overline{V_{AP}}^2)(\frac{1}{R_P} + \frac{1}{R_{AP}})$, which turns out to be 169μW, and the average energy is then 0.338 pJ.

- One very popular work with binary weights is BinaryConnect [6], wherein the weights used during the forward and backward propagation are binary and obtained stochastically from RV weights. However, the weight update step is not binarized to maintain a good precision of the weights, as in the updates are RV. The performance of BinaryConnect is reported to be as good as, or even better than, its counterparts with continuous weights. However, the MTJ crossbar (or any binary device weight array) would not allow for storing of RV weights, which perhaps explains a noticeable, although not significant, drop in classification accuracy when compared to floating-point weights.
- A drawback of in situ training is that every chip has to be trained separately, each requiring roughly the same amount of time. In addition, only the training algorithm for which the chip is designed (e.g., CD) can be used, unless extra hardware is added for the implementation of different techniques [58].
- *Dependence on temperature*: Higher operating temperatures reduce the thermal stability of the MTJs ($\Delta \propto 1/T$) and increase the switching probability for the same current magnitude and duration. The curves in Figure 2 shift to the left.
- The binary nature of MTJs severely limits the precision of each synaptic weight, thereby requiring larger crossbars with more hidden units to reach the accuracy exhibited by real continuous weights. However, although memristive devices do have several intermediate states, it is often difficult to program them reliably, so they too may end up being used in binary mode [36]. Further advances in materials of both magnetic and memristive devices will improve their prospects for use in memory and logic units.

8 CONCLUSION

In this work, we show how MTJ crossbars representing weights of ANNs and deep belief nets can be trained in situ by exploiting the stochastic switching properties of MTJs and performing weight updates in a way akin to gradient descent. We demonstrate how the machine learning algorithm can be implemented on crossbars with and without transistors. Results show that these stochastically trained binary networks can achieve classification accuracy almost as good as that of those trained in software and implemented on processors. This paves the way for the attainment of highly scalable neural systems in the future capable of performing complex applications.

REFERENCES

[1] CS Toronto. 2012. Training a Deep Autoencoder or a Classifier on MNIST Digits. Retrieved March 12, 2019 from http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html.

[2] Deep Learning. 2018. Deep Belief Network Tutorial. Retrieved March 12, 2019 from http://deeplearning.net/tutorial/DBN.html.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*. 153–160.

[4] Christopher H. Bennett, Djaafar Chabi, Theo Cabaret, Bruno Jousselme, Vincent Derycke, Damien Querlioz, and Jacques-Olivier Klein. 2015. Supervised learning with organic memristor devices and prospects for neural crossbar arrays. In *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'15)*. IEEE, Los Alamitos, CA, 181–186.

[5] Djaafar Chabi, Weisheng Zhao, Damien Querlioz, and Jacques-Olivier Klein. 2015. On-chip universal supervised learning methods for neuro-inspired block of memristive nanodevices. *ACM Journal on Emerging Technologies in Computing Systems* 11, 4 (2015), 34.

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. 3123–3131.

[7] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. 1223–1231.

[8] Brian Gardner and André Grüning. 2016. Supervised learning in spiking neural networks for precise temporal encoding. *PloS One* 11, 8 (2016), e0161335.

[9] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. 249–256.

[10] Tayfun Gokmen and Yurii Vlasov. 2016. Acceleration of deep neural network training with resistive cross-point devices: Design considerations. *Frontiers in Neuroscience* 10 (2016), 333.

[11] Raqibul Hasan and Tarek M. Taha. 2014. Enabling back propagation training of memristor crossbar neuromorphic processors. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'14)*. IEEE, Los Alamitos, CA, 21–28.

[12] Amr M. Hassan, Hai Helen Li, and Yiran Chen. 2017. Hardware implementation of echo state networks using memristor double crossbar arrays. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'17)*. IEEE, Los Alamitos, CA, 2171–2177.

[13] Geoffrey E. Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural Computation* 14, 8 (2002), 1771–1800.

[14] Geoffrey E. Hinton. 2007. To recognize shapes, first learn to generate images. *Progress in Brain Research* 165 (2007), 535–547.

[15] Geoffrey E. Hinton. 2012. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*. Springer, 599–619.

[16] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation* 18, 7 (2006), 1527–1554.

[17] Yingyezhe Jin, Yu Liu, and Peng Li. 2016. SSO-LSM: A sparse and self-organizing architecture for liquid state machine based neural processors. In *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'16)*. IEEE, Los Alamitos, CA, 55–60.

[18] Andrzej Kasiński and Filip Ponulak. 2006. Comparison of supervised learning methods for spike time coding in spiking neural networks. *International Journal of Applied Mathematics and Computer Science* 16 (2006), 101–113.

[19] Golnar Khodabandehloo, Mitra Mirhassani, and Majid Ahmadi. 2012. Analog implementation of a novel resistive-type sigmoidal neuron. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20, 4 (2012), 750–754.

[20] Jongyeon Kim, An Chen, Behtash Behin-Aein, Saurabh Kumar, Jian-Ping Wang, and Chris H. Kim. 2015. A technology-agnostic MTJ SPICE model with user-defined dimensions for STT-MRAM scalability studies. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'15)*. IEEE, Los Alamitos, CA, 1–4.
[21] Olga Krestinskaya, Khaled Nabil Salama, and Alex Pappachen James. 2018. Analog backpropagation learning circuits for memristive crossbar neural networks. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'18)*. IEEE, Los Alamitos, CA, 1–5.
[22] Hugo Larochelle and Yoshua Bengio. 2008. Classification using discriminative restricted Boltzmann machines. In *Proceedings of the 25th International Conference on Machine Learning*. ACM, New York, NY, 536–543.
[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.
[24] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural Networks: Tricks of the Trade*. Springer, Berlin, Germany, 9–48.
[25] Jung Hoon Lee and Konstantin K. Likharev. 2006. In situ training of CMOL CrossNets. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'06)*. IEEE, Los Alamitos, CA, 2749–2756.
[26] Jung Hoon Lee and Konstantin K. Likharev. 2007. Defect-tolerant nanoelectronic pattern classifiers. *International Journal of Circuit Theory and Applications* 35, 3 (2007), 239–264.
[27] Robert Legenstein, Christian Naeger, and Wolfgang Maass. 2005. What can a neuron learn with spike-timing-dependent plasticity? *Neural Computation* 17, 11 (2005), 2337–2382.
[28] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. arXiv:1605.04711.
[29] Z. Li and S. Zhang. 2003. Magnetization dynamics with a spin-transfer torque. *Physical Review B* 68, 2 (2003), 024404.
[30] M. Lichman. 2013. UCI Machine Learning Repository. Retrieved March 12, 2019 from http://archive.ics.uci.edu/ml.
[31] Suhwan Lim, Jong-Ho Bae, Jai-Ho Eum, Sungtae Lee, Chul-Heung Kim, Dongseok Kwon, Byung-Gook Park, et al. 2017. Adaptive learning rule for hardware-based deep neural networks using electronic synapse devices. In *Neural Computing and Applications*. Springer, 1–16.
[32] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S. Modha. 2011. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'11)*. IEEE, Los Alamitos, CA, 1–4.
[33] Janardan Misra and Indranil Saha. 2010. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing* 74, 1 (2010), 239–255.
[34] Ankit Mondal and Ankur Srivastava. 2018. In-situ stochastic training of MTJ crossbar based neural networks. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, New York, NY, 51.
[35] Emre Neftci, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. 2014. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience* 7 (2014), 272.
[36] Leibin Ni, Hantao Huang, Zichuan Liu, Rajiv V. Joshi, and Hao Yu. 2017. Distributed in-memory computing on binary RRAM crossbar. *ACM Journal on Emerging Technologies in Computing Systems* 13, 3 (2017), 36.
[37] Dong Pan and Bogdan M. Wilamowski. 2003. A VLSI implementation of mixed-signal mode bipolar neuron circuitry. In *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2. IEEE, Los Alamitos, CA, 971–976.
[38] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov. 2015. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 7550 (2015), 61–64.
[39] Damien Querlioz, Olivier Bichler, Philippe Dollfus, and Christian Gamrat. 2013. Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Transactions on Nanotechnology* 12, 3 (2013), 288–295.
[40] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*.
[41] Sylvain Saïghi, Christian G. Mayr, Teresa Serrano-Gotarredona, Heidemarie Schmidt, Gwendal Lecerf, Jean Tomas, Julie Grollier, et al. 2015. Plasticity in memristive devices for spiking neural networks. *Frontiers in Neuroscience* 9 (2015), 51.
[42] Abhronil Sengupta, Aparajita Banerjee, and Kaushik Roy. 2016. Hybrid spintronic-CMOS spiking neural network with on-chip learning: Devices, circuits, and systems. *Physical Review Applied* 6 (2016), 064003.
[43] Abhronil Sengupta, Maryam Parsa, Bing Han, and Kaushik Roy. 2016. Probabilistic deep spiking neural systems enabled by magnetic tunnel junction. *IEEE Transactions on Electron Devices* 63 (2016), 2963–2970.
[44] Walter Senn and Stefano Fusi. 2005. Convergence of stochastic learning in perceptrons with binary synapses. *Physical Review E* 71, 6 (2005), 061907.
[45] Jae-Sun Seo, Bernard Brezzo, Yong Liu, Benjamin D. Parker, Steven K. Esser, Robert K. Montoye, Bipin Rajendran, et al. 2011. A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'11)*. IEEE, Los Alamitos, CA, 1–4.
[46] Ahmad Muqeem Sheri, Aasim Rafique, Witold Pedrycz, and Moongu Jeon. 2015. Contrastive divergence for memristor-based restricted Boltzmann machine. *Engineering Applications of Artificial Intelligence* 37 (2015), 336–342.

[47] Daniel Soudry, Dotan Di Castro, Asaf Gal, Avinoam Kolodny, and Shahar Kvatinsky. 2015. Memristor-based multilayer neural networks with online gradient descent training. *IEEE Transactions on Neural Networks and Learning Systems* 26, 10 (2015), 2408–2421.

[48] Gopalakrishnan Srinivasan, Abhronil Sengupta, and Kaushik Roy. 2016. Magnetic tunnel junction based long-term short-term stochastic synapse for a spiking neural network with on-chip STDP learning. *Scientific Reports* 6 (2016), 29545.

[49] Manan Suri, Vivek Parmar, Ashwani Kumar, Damien Querlioz, and Fabien Alibart. 2015. Neuromorphic hybrid RRAM-CMOS RBM architecture. In *Proceedings of the 2015 15th Non-Volatile Memory Technology Symposium (NVMTS'15)*. IEEE, Los Alamitos, CA, 1–6.

[50] Manan Suri, Damien Querlioz, Olivier Bichler, Giorgio Palma, Elisa Vianello, Dominique Vuillaume, Christian Gamrat, et al. 2013. Bio-inspired stochastic computing using binary CBRAM synapses. *IEEE Transactions on Electron Devices* 60, 7 (2013), 2402–2409.

[51] H. Tomita, T. Nozaki, T. Seki, T. Nagase, K. Nishiyama, E. Kitagawa, M. Yoshikawa, et al. 2011. High-speed spin-transfer switching in GMR nano-pillars with perpendicular anisotropy. *IEEE Transactions on Magnetics* 47, 6 (2011), 1599–1602.

[52] Adrien F. Vincent, Jérôme Larroque, Nicolas Locatelli, Nesrine Ben Romdhane, Olivier Bichler, Christian Gamrat, Wei Sheng Zhao, et al. 2015. Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems. *IEEE Transactions on Biomedical Circuits and Systems* 9, 2 (2015), 166–174.

[53] K. L. Wang, J. G. Alzate, and P. Khalili Amiri. 2013. Low-power non-volatile spintronic memory: STT-RAM and beyond. *Journal of Physics D: Applied Physics* 46, 7 (2013), 074003.

[54] Mengxing Wang, Wenlong Cai, Kaihua Cao, Jiaqi Zhou, Jerzy Wrona, Shouzhong Peng, Huaiwen Yang, et al. 2018. Current-induced magnetization switching in atom-thick tungsten engineered perpendicular magnetic tunnel junctions with large tunnel magnetoresistance. *Nature Communications* 9, 1 (2018), 671.

[55] Qian Wang, Yongtae Kim, and Peng Li. 2016. Neuromorphic processors with memristive synapses: Synaptic interface and architectural exploration. *ACM Journal on Emerging Technologies in Computing Systems* 12, 4 (2016), 35.

[56] D. C. Worledge, G. Hu, P. L. Trouilloud, D. W. Abraham, S. Brown, M. C. Gaidis, J. Nowak, et al. 2010. Switching distributions and write reliability of perpendicular spin torque MRAM. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'10)*.

[57] Yan Xu, Xiaoqin Zeng, Lixin Han, and Jing Yang. 2013. A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Networks* 43 (2013), 99–113.

[58] Chris Yakopcic, Md Zahangir Alom, and Tarek M. Taha. 2016. Memristor crossbar deep network implementation based on a convolutional neural network. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'16)*. IEEE, Los Alamitos, CA, 963–970.

[59] Deming Zhang, Lang Zeng, Kaihua Cao, Mengxing Wang, Shouzhong Peng, Yue Zhang, Youguang Zhang, et al. 2016. All spin artificial neural networks based on compound spintronic synapse and neuron. *IEEE Transactions on Biomedical Circuits and Systems* 10, 4 (2016), 828–836.

[60] Deming Zhang, Lang Zeng, Youguang Zhang, Weisheng Zhao, and Jacques Olivier Klein. 2016. Stochastic spintronic device based synapses and spiking neurons for neuromorphic computation. In *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'16)*. IEEE, Los Alamitos, CA, 173–178.

[61] Yaojun Zhang, Xiaobin Wang, Yong Li, Alex K. Jones, and Yiran Chen. 2012. Asymmetry of MTJ switching and its implication to STT-RAM designs. In *Proceedings of the Conference on Design, Automation, and Test in Europe*. 1313–1318.

Received July 2018; revised November 2018; accepted January 2019