In-situ Stochastic Training of MTJ Crossbar based Neural Networks

Ankit Mondal
University of Maryland
College Park, Maryland 20742
amondal2@terpmail.umd.edu

Ankur Srivastava
University of Maryland
College Park, Maryland 20742
ankurs@umd.edu

ABSTRACT
Owing to high device density, scalability and non-volatility, Magnetic Tunnel Junction-based crossbars have garnered significant interest for implementing the weights of an artificial neural network. The existence of only two stable states in MTJs implies a high overhead of obtaining optimal binary weights in software. We illustrate 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 both with and without access transistors and perform simulations on them to demonstrate the effectiveness of our techniques. The results reveal that stochastically trained MTJ-crossbar NNs achieve a classification accuracy nearly as same as that of real-valued-weight networks trained in software and exhibit immunity to real variable weight variations.

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

1 INTRODUCTION
Deep Neural Networks (DNNs) have become a popular choice for tasks such as image classification, face recognition, and Natural Language Processing. This has however been at the cost of massive computations on von Neumann architectures exhibiting high energy and area requirements [11]. The emergence of novel devices and special-purpose architectures has called for a shift from conventional digital hardware for implementing neural algorithms [19].

Attempts have been made towards dedicated hardware designs and realization of the synaptic weights (and neurons) of a Neural Network (NN) by using CMOS transistors in an analog fashion [14]; 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 [17]. However, the fabrication of multilevel memristors with stable states is still a challenge [8].

Another choice is the Magnetic Tunnel Junction (MTJ), an emerging binary device (since it has 2 stable states) which has shown its potential as storage elements and is a promising candidate for replacing CMOS in memory chips [15]. Its non-volatility and scalability makes it a particularly lucrative choice for logic-in-memory type architectures for neural networks. MTJs and memristors can be connected in a crossbar configuration which allows greater scalability and higher performance due to their inherent parallelism [1, 7, 17]. Several studies have investigated how the crossbar arrays with memristors [5, 22], MTJs [1, 8] and domain-wall ferromagnets [2, 22] can implement Spiking Neural Networks (SNN) trained using Spike-Timing Dependent Plasticity (STDP), both. Hasan et al. [20] and Soudry et al. [6] have implemented multi-layer NNs on memristive crossbars trained on-chip using the backpropagation algorithm and demonstrated on supervised learning tasks.

Continuous weight networks can be simplified into discrete weight networks without significant degradation in classification accuracy while achieving substantial power benefits [18]. The use of discrete weight networks, such as BinaryConnect [16] and in [9], also stems from the challenge to address the high storage and computational demands of a large number of full-precision weights. The existence of only 2 stable states in MTJs makes them a good candidate for the realization of binary weight networks. One way of training such NNs is to perform weight updates stochastically, which is justifiable from evidences that learning in human brains also has some stochasticity associated [19]. That such a method can lead to convergence with high probability in a finite time has been shown in [23].

Obtaining optimal weights for a binary network in software can be impractical because its discrete nature requires integer programming. Also, when physically realizing an NN on hardware, the underlying device variations 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 paper, we explore the use of MTJ crossbars for the hardware implementation of the synaptic weight matrices of a neural network. We propose the in-situ training of such an MTJ crossbar NN, which allows us to exploit its inherent parallelism for significantly faster training and also accounts for device variations. We advocate a probabilistic way of updating the MTJ synaptic weights 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. Finally, we support our proposed techniques with data by modeling device and circuit properties and running simulations.

2 BACKGROUND
In this section we describe the basics of neural networks and the parallelism offered by the crossbar architecture, and introduce the characteristics of Magnetic Tunnel Junctions.

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, $x \in \mathbb{R}^M$ is the input to a layer and $W \in \mathbb{R}^{N \times M}$ represents the synaptic weight matrix, then the output $y \in \mathbb{R}^N$ is

$$y = f(Wx)$$  \hspace{1cm} (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

$$\Delta W_{ji} = -\eta \frac{\partial E}{\partial W_{ji}} = -\eta x_i \delta_j$$  \hspace{1cm} (2)

where $E$ is the cost function of the presented input sample $x$, $\eta$ is the learning rate and $\delta_j$ 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. The weight
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. Fig 1(a) shows a simplified crossbar with each row corresponding to an input and each column to an output neuron. Let \( V_i \in [-V, V] \) be the voltage applied at the \( i^{th} \) input terminal and \( G_{ij} \) be the conductance of the synapse connecting it to the \( j^{th} \) output. By Ohm’s Law, the current through that synapse is \( G_{ij} V_i \) and by Kirchhoff’s law the total current at the output is

\[
I_j = \sum_i G_{ij} V_i
\]

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

Since the outputs are obtained almost instantaneously after the inputs are applied, the matrix-vector multiplication of eqn. (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 2 physical quantities derivable from the inputs and the errors. In this way, the \( O(MN) \) operations can be done in parallel using the \( M \times N \) synapses.

2.3 Magnetic Tunnel Junction

The Magnetic Tunnel Junction (MTJ) is a 2-terminal spintronic device consisting primarily of 2 ferromagnetic layers separated by a thin tunnel barrier (typically MgO). The magnetic orientation of one of the magnetic layers is fixed, whereas that of the other is free, as shown in fig 1(b). MTJs possess 2 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 which flips the magnetization of the free layer through the mechanism of spin-transfer torque [26]. The time required to switch is heavily dependent on the magnitude of the switching current. Not only that, 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 [26].

3 MTJ CROSSBAR BASED NEURAL NETWORKS

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 2 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 2 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 [2], the maturity of such technology is not at par with that of the binary version [22].

**Training Binary Networks**: Obtaining optimal binary weights for an NN is an NP-hard problem with an exponential time complexity, 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 [19], stochastic update of binary weights is computationally equivalent to deterministic update of multi-level weights at the system level.

In [1], Vincent et al. 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 [21]. We propose using a probabilistic learning approach by training using the gradient descent method (which requires weight updates of the form in eqn. (2)) as demonstrated in section 4.2.

3.1 The Motivation for In-situ Training

There are 2 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 fig 1(c)), resulting in \( O(M \times N) \) transistors in the crossbars.

Figure 1: (a) A crossbar (b) Magnetic Tunnel Junction (c) A 2 x 2 crossbar

Figure 2: (a) P vs t, for different values of I (b) P vs I, for different values of t
Table 1: Write phase. Signs of $\delta$, $\Delta W$, and $\omega$ depend on the input $x$, $\delta$, and the directed network of MTJ Synapse

| Input $x$ | Error $\delta$ | $\Delta W$ | $W$ and $\omega$ | Switch $P$ |
|-----------|----------------|------------|-----------------|------------|
| $x > 0$   | $\delta > 0$  | $\Delta W < 0$ | Decreases $P \rightarrow A_P$ |
| $x > 0$   | $\delta < 0$  | $\Delta W > 0$ | Increases $A_P \rightarrow P$ |
| $x < 0$   | $\delta > 0$  | $\Delta W > 0$ | Increases $A_P \rightarrow P$ |
| $x < 0$   | $\delta < 0$  | $\Delta W < 0$ | Decreases $P \rightarrow A_P$ |

(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 fig. 1(a). While 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 behaviour, 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). But, despite circumventing this issue, this architecture can suffer from performance degradation due to the intrinsic device variations which only aggravate with scaling. On the other hand, 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 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

Simply using $\pm 1$ 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 $L_2$ loss between the real-valued weights $W$ and quantized ones, as was done in [18]. This provides a solution $b = \|W\|_1/n$ (the mean of absolute values of $W$). Thus an MTJ in the $P$ ($AP$) state would signify a weight of $+b$ ($-b$).

4 IN-SITU TRAINING OF MTJ CROSSBARS

We first provide a high-level understanding of how an MTJ synaptic crossbar implementing an ANN 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 the gradient descent method can be used for the stochastic weight update of MTJs, and finally describe the in-situ training procedure for the 2 crossbar architectures.

4.1 Overview of Operations

The training process is carried out as follows.

Read Phase: Upon receiving a training input $x \in \mathbb{R}^M$, the input terminals are applied with voltages $V_i \in [-V, V]$ for $i \propto x_i$, whereas the output terminals are maintained at ground potential. Current $i_j = g_{ij}V_i$ flows through the $(i, j)$ 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 $\delta$. Table 1 lists the 4 possible cases of weight update depending on $x$ and $\delta$. The gradient descent algorithm requires a weight update of the form of eqn. (2). An appropriate way to realize this, as suggested in [12], is to set switching probabilities proportional to (the magnitude of) $\Delta W$ calculated in (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 gradient descent method. Just as the weight update in eqn (2) is a function of 2 variables (the input and the error), the probabilistic switching of MTJs can be controlled by 2 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 $\delta_i$. However, as can be evidenced from eqn (5) and fig 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 $\Delta W$, is a linear function of $x_i$ and $\delta_i$. 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 $\delta$ 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 $\eta$). 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 $\delta$) as $t_{wr} = t_0 + t_1|\delta|$

Similarly, the write current ($I_{wr}$) would be a minimum of $I_0$ and increase linearly with $|x|$ as $I_{wr} = I_0 + I_1|x|$

We now wish to find coefficients $I_0$, $I_1$, $I_0$ 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 fig. 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 $I_{wr} = t_0$ 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 $I_{wr}$ starts rising at $I_0$. That is

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

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 $P = 0$ if $I_{wr} = I_0$, (because $\Delta W = 0$ for $x = 0$). But $P$ should start increasing as soon as $I_{wr}$ increases, that is

$$P(I_{wr}, I_0 + t_1) = \begin{cases} < P_0 & \text{for } I_{wr} < I_0, \\ \geq P_0 & \text{for } I_{wr} \geq I_0 \end{cases}$$

Fig 3 shows how well the linear model approximates the required $AP \rightarrow P$ switching probabilities (similar curve fitting for $P \rightarrow AP$ as

![Figure 3: P vs. $t_{wr}$.](image)

Table 2: Boundary values of the parameters in the weight update eqn. (2) and their counterparts in probabilistic switching of MTJ

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

Table 3: The coefficients that fit the model for both $AP \rightarrow P$ and $P \rightarrow AP$ switching
well). Table 2 shows the write currents and duration for boundary values of $|x|$ and $|δ|$ and table 3 lists the values of the coefficients in eqns. (6) and (7). One could use non-linear models for mapping $|δ|$ and $|x|$ to $τ_{wr}$ and $τ_{rd}$, respectively, in order 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.

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 above.

### 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 which lead to unnecessary power consumption at a minimum. Fig 4(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 $τ_{wr}$.

Fig 4(b) plots the signals during both the read and write phases. During the read phase ($0 ≤ t ≤ τ_{rd}$), 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 $V_R$. Output currents $I_O$ are processed by the output CLs.

**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 sec. 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 $(i, j)$ synapse, the write pulse width depends on only $|δ_j|$, 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 eg., 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 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 2 stages, with the columns updated in a given stage depending on the signs of $δ$. Each phase has a duration of $τ_{rd}$ (which need not be more than $t_{rd}$, see eqn. (6)). The voltage signals in each phase are plotted in fig. 4(b) and detailed below -

**Figure 4:** The 1T1R crossbar. (a) Schematic (b) Read & write phases signals

1. **Phase 1:** $τ_{rd} ≤ t ≤ τ_{rd} + τ_{wr}$. Update the weights of the columns which had $δ > 0$. Then, the transistor control signals would be

\[
c_j = \begin{cases} 
V_{DD}, & \text{for } δ_x > 0 \text{ and } 0 ≤ t ≤ τ_{rd} \text{ or } τ_{rd} ≤ t ≤ τ_{wr}, \\
0, & \text{for } δ_x < 0 \text{ or } τ_{wr} ≤ t ≤ τ_{rd} ≤ τ_{wr}.
\end{cases}
\]

And the write voltages applied at the input terminals would be

\[
V_{wr, i} = VP(x_i)u(x_i) + V_{AP}(x_i)u(-x_i)
\]

where $u$ is the unit step function.

2. **Phase 2:** $τ_{rd} + τ_{wr} ≤ t ≤ τ_{rd} + 2τ_{wr}$. Update the weights of those columns which had $δ < 0$. Here, the signals are opposite to those in phase 1 as shown in fig. 4(b).

Here $VP(V_{AP})$ is the voltage applied to switch from $P → AP$ ($AP → P$) and can be obtained using (7) and $RP(RAP)$. $V_P$ and $V_{AP}$ still depend on $|x_i|$, but for brevity explicit mention will be omitted henceforth. Let $MTJ$s in the crossbar be arranged in a way that positive (negative) current from the $j^{th}$ input terminal to $i^{th}$ output terminal can switch $S_{j,i}$ from $P → AP$ ($AP → P$); 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 $τ_{rd} + 2τ_{wr}$ time which is $O(1)$. Due to limitations on the scalability of 1T1R architecture, it is worth exploring the feasibility of transistorless 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’s analyze the impact of sneak paths on the 1R crossbar with the 2-phase update strategy used previously. We first demonstrate the presence of sneak paths with a small example. Fig 5(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 a certain input $x$ with $x_1 > 0$, $x_2 < 0$ produced errors $δ_1 > 0$, $δ_2 < 0$ at the outputs. The equivalent circuit during write phase 1 is drawn in fig. 5(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 $P → AP$ and $S_{22}$ from $AP → P$ 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 $(V_P - AP)$. The current through such an MTJ, if in the $P$ state, is $I = (VP - V_{AP})/RP$ and is high enough (recall $V_P < 0$) to switch it from $P → AP$. In the other extreme case, a potential difference of $(V_{AP} - VP)$ leading to current $I = (V_P - VP)/RP$ through an MTJ in the $AP$ state will switch it from $AP → P$.

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

**Table 4:** 4-phase weight update for the 1R configuration in fig 5(c).

| Input | Error | $|x|_i$ | $V_P$ | $V_{AP}$ | Switch |
|-------|-------|--------|-------|----------|--------|
| Phase 1 | $x > 0$ | $δ > 0$ | $u(x_i)V_{DD}$ | $u(x_i)V_P$ | $u(x_i)V_{DD}$ | $P → AP$ |
| Phase 2 | $x < 0$ | $δ > 0$ | $u(-x_i)V_{DD}$ | $u(-x_i)V_P$ | $u(-x_i)V_{DD}$ | $AP → P$ |
| Phase 3 | $x > 0$ | $δ < 0$ | $u(x_i)V_{DD}$ | $u(x_i)V_P$ | $u(x_i)V_{DD}$ | $P → AP$ |
| Phase 4 | $x < 0$ | $δ < 0$ | $u(-x_i)V_{DD}$ | $u(-x_i)V_P$ | $u(-x_i)V_{DD}$ | $AP → P$ |
4.4.2 Four-phase Update: The large sneak currents in the 2-phase writing strategy, potentially resulting in false switching, is 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 2 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 $x$, and then further clustering the rows according to the sign of $x$. This proposed 4-phase writing scheme would require additional transistors to choose the rows to be updated in a given phase as shown in fig 5(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.

Let us now see how bad the issue of sneak-path leakage is with this strategy. Fig 5(d) shows the equivalent circuit for the $2 \times 2$ 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 $V_{AP}$. Hence, the 4-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 trade-off is not only worth but also necessary for satisfactory performance of the training process.

4.5 Multi-layer NNs

Multi-layer 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. Consider a 2-layer NN with weight matrices $W_1$ and $W_2$. For an input $x$, the final output $y_0$ is given as

$$y_2 = f(a_2) = f(W_2 y_1) \text{ where } y_1 = f(a_1) = f(W_1 x) \quad (12)$$

If $\delta_2$ is the error of the second layer (output), then that of the first layer (hidden) is $\delta_1 = (W_2^T \delta_2) \times f'(a_1)$ where $f'$ is the derivative of activation function $f$, and $\times$ represents a component-wise product. This operation can be done on the crossbar (of the output layer) itself by reversing the roles of its input and output terminals: $\delta_1$ is now fed as the input and out comes $W_2^T \delta_2$, which, when multiplied by $f'(a_1)$, gives $\delta_1$ as the error to be used for updating the weights of the hidden layer.

For the MTJ crossbar NN we described, during forward propagation, the total duration of the read phase would be $nT_{rd}$ 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. Lastly, 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 which share inputs or outputs. All these crossbars can still be read and written in parallel, thanks to the locality of the weight update operations.

5 EXPERIMENTAL SETUP AND RESULTS

To see how successfully the MTJ crossbar NNs 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 with supervised learning. To capture the MTJ device parameters, we used an HSPICE model [13] and included thermal fields in its LLG equations for obtaining the stochastic switching characteristics [3]. Certain device parameters used in and obtained from this model were then incorporated into the simulations of the crossbar.

The performance of the neural network was evaluated in the following scenarios (code-named for further reference). All training processes used the Mean Square Error cost function and neurons had the tanh activation function.

1. RV: We first train and evaluate a neural network with real-valued weights in MATLAB. Binary quantization step (b) is obtained from this trained network as shown in sec. 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 device variations 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 sec. 4.2 for the (a) 1T1R architecture, with the 2-phase write strategy (sec. 4.3). (b) 1R architecture, with both the 2-phase (to see the effects of sneak currents) and the 4-phase update strategies (sec. 4.4).

4. DV: Device variations of different extent are introduced in the stochastic training of both the 1T1R and 1R crossbars. It reflects in the variations in the resistance of the $P$ and $AP$ states, which usually doesn’t exceed 10% as per experiments [4].

We use the following datasets for evaluation.

SONAR, Rocks vs Mines[27]: Three different NN architectures are considered - one with 1 layer (1L), and two with 2 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[24]: Three 2-layer networks of 50, 100 and 150 hidden units respectively and a 3-layer network of 50+25 hidden units were evaluated on the 10000 images of the test dataset.

Wisconsin Breast Cancer (Diagostic)(WBCCD)[27]: A single-layer network (1L) and 2 two-layer networks (2L10 and 2L20) were considered, and the test dataset had 200 samples.

Table 5 summarizes the accuracy obtained with these networks under the different training scenarios mentioned above. The effect of device variations of different extents on the in-situ stochastic training is highlighted for some of the networks in table 6, with fig. 6 plotting the mean square error as the training progresses for the 1R crossbar. Additionally, fig. 7 compares the error for the two write strategies. It doesn’t converge with the 2-phase writing scheme due to higher instances of undesired weight changes, but does so with 4 phases. It is evident from these results that
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 real-valued weights (RV, 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.

In-situ training also benefits crossbars with transistors (ST-1TIR against DP-1TIR) in the presence of device variations by slightly improving accuracy (by about 0.5% – 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 device variations, owing primarily to the fault-tolerant nature of NN and its learning algorithms. As can be seen in table 6, increase in misclassification rates remain within 2% even with 20% variation.

The accuracy degradation of 2% – 3% that we achieve (on going from RV to ST) is comparable to the 3.7% reported by [8] and the 0.8% – 3.5% in [1]. However, it must be mentioned and emphasized that any comparison is fair only if they are on the same dataset and network architecture. The benefit of using in-situ training can also be seen when we compare our work with that of [7] (which performs offline learning). On the MNIST 21.00 network, we obtained an error rate of 10.20%, whereas [7] had a much higher value of 30% on the same network, although it must be mentioned that the latter were at a disadvantage due to linear activation units.

### 6 CONCLUSION

In this work, we show how MTJ crossbars representing weights of an ANN 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 learning algorithm can be implemented on crossbars with and without transistors. Results show 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] A. F. Vincent 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), 166–175.

[2] A. Sengupta et al. 2016. Hybrid Spintron–CMOS Spiking Neural Network with On-Chip Learning: Devices, Circuits, and Systems. Physical Review Applied 6 (2016).

[3] A. Sengupta et al. 2016. Probabilistic deep spiking neural systems enabled by magnetic tunnel junction. IEEE Transactions on Electron Devices 63 (2016), 2963–70.

[4] D. C. Worledge et al. 2010. Switching distributions and write reliability of perpendicular spin torque MRAM. In Electron Devices Meeting (IEDM), 2010 IEEE International.

[5] D. Querlioz et al. 2013. Immunity to device variations in a spiking neural network with memristive nanodevices. IEEE Transactions on Nanotechnology 12 (2013).

[6] D. Soudry et al. 2015. Memristor-based multilevel neural networks with online gradient descent training. IEEE transactions on neural networks and learning systems 26, 10 (2015), 2408–2421.

[7] D. 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.

[8] D. Zhang et al. 2016. Stochastic spintronic device based synapses and spiking neurons for neuromorphic computation. In Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on. IEEE, 173–178.

[9] F. Li et al. 2016. Ternary weight networks. arXiv preprint arXiv:1605.04711 (2016).

[10] H. Tomita et al. 2011. High-speed spin-transfer switching in GMR nano-pillars with perpendicular anisotropy. IEEE Transactions on Magnetics 47, 6 (2011), 1599–1602.

[11] J. Dean et al. 2012. Large scale distributed deep networks. In Advances in Neural Information Processing Systems (NIPS). 1223–1231.

[12] J. H. Lee et al. 2007. Defect-tolerant nonvolatile electronic pattern classifiers. International Journal of Circuit Theory and Applications 35, 3 (2007), 259–264.

[13] J. King et al. 2015. A technology-agnostic MTJ SPICE model with user-defined dimensions for STT-MRAM scalability studies. In Custom Integrated Circuits Conference (CICC), 2015 IEEE. IEEE, 1–4.

[14] J. Misra et al. 2010. Artificial neural networks in hardware: A survey of two decades of progress. 71, 1 (2010), 239–255.

[15] K. L. Wang et al. 2013. Low-power non-volatile spintronic memory: STT-RAM and beyond. Journal of Physics D: Applied Physics 46, 7 (2013), 074003.

[16] M. Courbariaux et al. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In Advances in NIPS. 3123–3131.

[17] M. Prezioso et al. 2015. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. Nature 521, 7550 (2015), 61–64.

[18] M. Rastegar et al. 2016. Xnor-net: Imaget classification using binary convolutional neural networks. In European Conference on Computer Vision. Springer.

[19] M. Suri et al. 2013. Bio-inspired stochastic computing using binary CBRAM synapses. IEEE Transactions on Electron Devices 60, 7 (2013), 2402–2409.

[20] R. Hasan et al. 2014. Enabling back propagation training of memristor crossbar neuromorphic processors. In Neural Networks (IJCNN), 2014 International Joint Conference on. IEEE, 21–28.

[21] R. Legerstee et al. 2005. What can a neuron learn with spike-timing-dependent plasticity? Neural computation 17, 11 (2005), 2357–2382.

[22] S. Saïghi et al. 2015. Plasticity in memristive devices for spiking neural networks. 9 (2015).

[23] W. Senn et al. 2005. Convergence of stochastic learning in perceptrons with binary synapses. Physical Review E 71, 6 (2005), 061907.

[24] V. LeCun et al. 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 11 (1998), 2278–2324.

[25] Y. Zhang et al. 2012. Asymmetry of MTJ switching and its implication to STT-RAM designs. In Proceedings of the Conference on Design, Automation and Test in Europe.

[26] Z. Li et al. 2003. Magnetization dynamics with a spin-transfer torque. Physical Review B 68, 2 (2003), 024404.

[27] M. Liebman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml