SAND2015-3287C

Training neural hardware with noisy components

Fred Rothganger, Brian R. Evans, James B. Aimone, Erik P. DeBenedictis
Sandia National Laboratories
Albuquerque, New Mexico 87185

***Abstract*—Some next generation computing devices may consist of resistive memory arranged as a crossbar. Currently, the dominant approach is to use crossbars as the weight matrix of a neural network, and to use learning algorithms that require small incremental weight updates, such as gradient descent (for example Backpropagation). Using real-world measurements, we demonstrate that resistive memory devices are unlikely to support such learning methods. As an alternative, we offer a random search algorithm tailored to the measured characteristics of our devices.**

I. Introduction

Future applications can no longer rely on the various types of CMOS scaling to gain more computing power [1]. The industry is actively seeking other paths to go "Beyond Moore". One such path is neuromorphic computing, where the hardware is specialized to run neural networks of some variety.

An attractive approach is to arrange memristors in a crossbar to do analog matrix-vector multiplication. The DARPA SyNAPSE project invested in such an approach [2], [3], though several of their performers moved away from analog computation. The IBM TrueNorth chip is in fact entirely digital [4], [5]. Hewlett-Packard focused on a software architecture that runs on conventional CMOS [6]. This suggests how difficult it is to implement a neural algorithm using memristors. We show below that some of this difficulty stems from the limited precision of analog computation.

The emphasis in the field has been on devices that perform efficiently, as opposed to devices that learn. For example, TrueNorth packs 10^6 integrate-and-fire units on a chip in part because it does not include the hardware support for online learning. All training must be done in a conventional computer. Resistive memory devices ("memristors") are attractive because of their potential for low-cost online learning.

Highly adaptive behaviors–that which we call "intelligence" in humans–require online learning by definition. New applications which communicate more naturally with humans and help them solve problems will need to be adaptive rather than static [7]. Certainly conventional computers can do this, but to scale up will require new generations of efficient hardware.

At present no such hardware is in hand, and there should be a clear advantage before developing it. We explored whether neural learning methods would indeed benefit from memristor crossbars, and which algorithms would work best. Below we detail the journey. We first characterized the devices (Section II), then developed a simulation to train them using conventional methods (Section III). When this proved impossible (Section IV) we developed a random-walk approach (Section V).

II. Memristor modeling

A simulation of neural algorithm running on a memristor crossbar consists of three levels: the device, the crossbar, and the algorithm. We detail the latter two in Section III. Here we focus on modeling the memristor itself.

Analytic models of memristors based on first principles are an active area of research [8], [9]. This difficult problem is not yet fully solved. In particular, analytic models tend to underestimate the variability of a given device's behavior. Thus we opted for a data-driven approach: collect enough measurements in a table and use nearest-neighbor lookup and interpolation to emulate the physical device.

We have several tantalum oxide (TaO_x) test chips produced by HP and Sandia. HP also developed a programmable test board, which they kindly provided to us. (Only a small number of these boards have been produced, primarily for internal use.) The board consists of a PIC microcontroller, digital-to-analog and analog-to-digital converters, and an array of electronic switches to select row and column on the chip.

Barring any hidden state, a memristor can be viewed as a function $f(R_0, V) \rightarrow R_1$, where R_0 is pre-pulse resistance, R_1 is post-pulse resistance, and V is the pulse voltage. All pulses in our tests were $2\mu s$ wide. The goal of the data collection was to sample this function.

A. Controlled start values

The first challenge was to move the device into a known initial resistance R_0. Memristors are quite noisy, both in to read and to write. It is generally not possible to set one within a small enough tolerance in a single pulse. Closed-loop control is necessary. We programmed the HP board to apply a series of pulses, measuring the resistance after each one and adjusting the sign and magnitude to track toward the target R_0. Our naive implementation used four cycles with a maximum of 30 pulses each. All the voltage pulses in a given cycle used the same sign, which was determined by the sign of the difference between measured resistance and R_0. At the start of the cycle the magnitude of the voltage was small, then gradually increased until the measured resistance passed R_0, or 30 pulses were completed. The standard deviation of resistance error with this procedure was 6.2Ω.

The test consisted of a resistance read, a specific voltage pulse, and a final read. We wrapped this in a pair of loops that systematically explored a region of (R_0, V) space.

B. Random-pulse sampling

The above procedure is inefficient. Setting the pre-pulse resistance involves a large number of reads and pulses in itself. Those reads and pulses are also perfectly valid data. We developed a simplified procedure that interleaved pulses and reads: $R\ V\ R\ V\ R\ V\ R$... The resistance readings immediately before and after a voltage pulse V provide the R_0 and R_1 values, respectively.

In order to cover a large portion of the parameter space (R_0, V), the test program drew voltages from a uniform distribution over $[-7, 4.3]$. However, the domain of the distribution changed when the resistance dipped too low or rose too high. This was necessary to protect the device from damage and to concentrate the samples within a useful ranges of resistances.

Using this method we collected several million samples from a small handful of devices. The collect time per device was under 24 hours. A 10 megasample collect from one device forms the basis for the simulations below. The other devices gave qualitatively similar results, thus we consider this data representative. Figure 1 shows a subset of the collected data points, decimated by a factor of 100 for ease of viewing.

C. Table construction

The primary use of the data is to emulate a memristor in a non-parametric manner. One option is to find the nearest neighbor in (R_0, V) space and use the third member of the tuple as the resulting resistance. That approach would be slow and overly sensitive to input values. A better approach is to extract statistics from the data and store them in a table that allows direct indexing.

We constructed a table that ranged from $-10V$ to $10V$ in $0.1V$ increments. Resistance ranged from 0Ω to $20k\Omega$ in 100Ω increments. We distributed each sample to its four nearest bins in (R_0, V) space with bilinear weighting. For each sample we calculated $\Delta = R_1 - R_0$, and for each bin summed $weight$ and $weight \cdot \Delta$. From these we calculated the mean Δ in each cell. In turn we used bilinear interpolation to compute the smoothed mean value for every position in (R_0, V) space. From the difference with Δ we computed the standard deviations for each cell.

The resulting table still required a number of post-processing steps to be useful for simulation. First we determined the write voltage threshold to be $\pm 1.6V$ by examination. Voltages near zero showed a small amount of resistance increase, even though in theory they should produce no change at all. The cause of this is undetermined, but possibilities include an actual increase in resistance due to the read protocol of the HP board. The lower-level software samples the resistance many times, and we have observed some cumulative effect from sub-threshold pulses. We forced these entries to be zero, and subtracted their average from other entries in the same row (associated with a 100Ω-wide bin), effectively removing the read drift. Figure 2 shows the resulting table.

We applied a similar procedure to the standard deviation values. There were nonzero entries in cells below the write threshold, which we interpreted as read noise. We subtracted the average read noise on a per-row basis, and recorded it in another table for use during simulation. We interpreted the remaining standard deviation values as write noise. Figure 3 shows the resulting table.

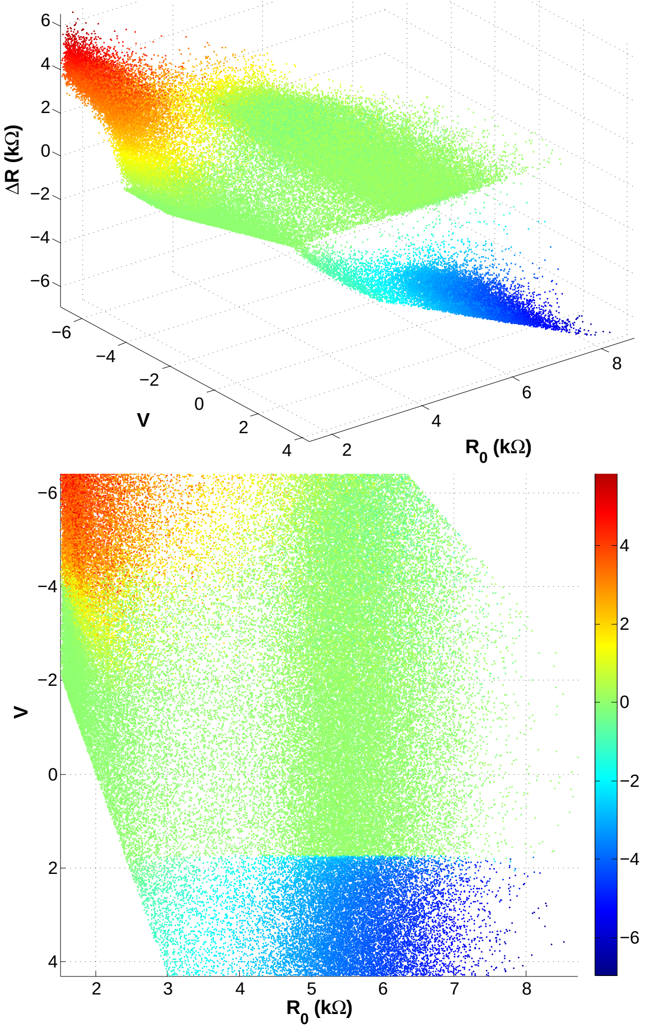


Fig. 1. A decimated subset $(100,000$ out of $10,000,000)$ of the random-pulse samples, plotted in (R_0, V, Δ) space. R_0 is pre-pulse resistance. V is the random-pulse height. Δ is the change in resistance. Colors indicate Δ in $k\Omega$. The 3D plot is shown from two viewpoints to give a better understanding of coverage.

Note that these two tables $(\mu\Delta,\ \sigma\Delta)$ do not fully capture the behavior of the device. We subjected the data to several analyses outside the scope of this paper (see [9] for related work). We observed that R_1 is not strictly a function of R_0 and V, but also the history of previous voltage pulses. That is, the memristor contains hidden state. The table-based approach treats the hidden state as noise, which is sufficient for this work.

D. Results

The Δ (resistance change) values showed some noteworthy patterns. Unlike the typical model of a memristor as a sym-

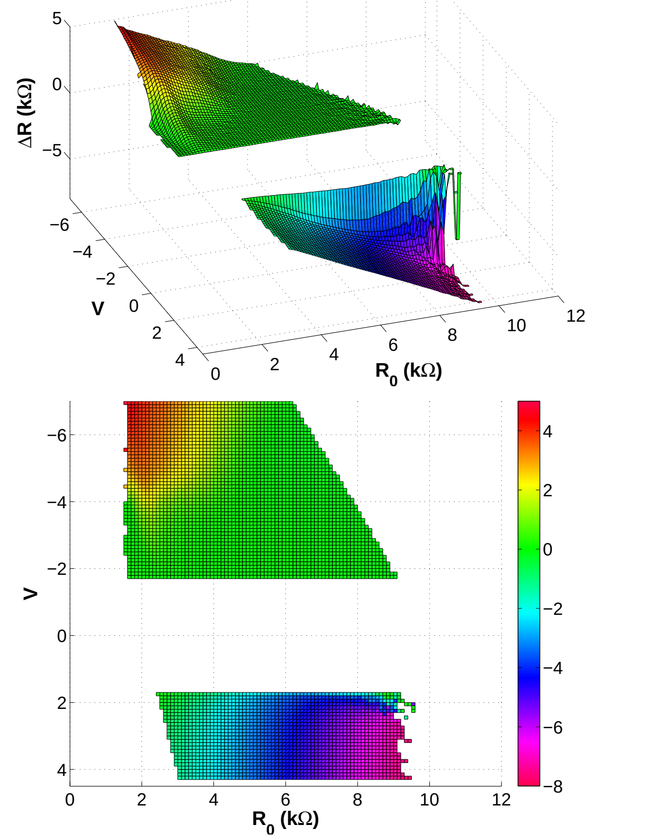


Fig. 2. Two views of the $\mu\Delta$ table. These are the average amounts of change in resistance for each combination of pre-pulse resistance and pulse voltage. The band between $-1.6V$ and $1.6V$ has been moved by the data cleaning process. These are read as zero.

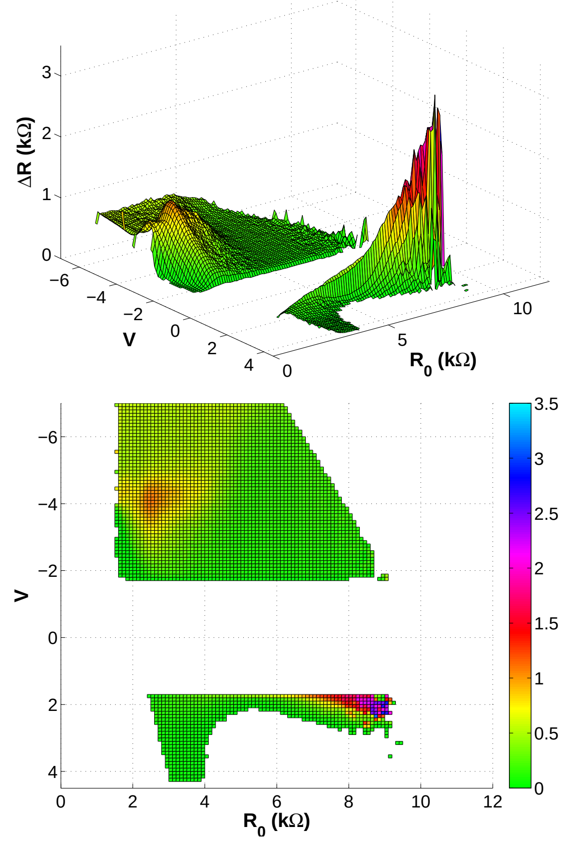


Fig. 3. Two views of the $\sigma\Delta$ table. These are the standard deviation in resistance change for each combination of pre-pulse resistance and pulse voltage. The band between $-1.6V$ and $1.6V$ has been moved by the data cleaning process. These are read as zero.

metric pair of exponentials, we found significant asymmetry. Figure 4 plots a row of the $\mu\Delta$ table at $4k\Omega$, showing Δ in response to pulse voltage. Note the "cliff" in response to positive voltages. We expect this from a circuit-based argument. When a memristor is subject to a voltage sufficient to cause a decrease in resistance, the current will increase according to Ohm's law, creating a feedback loop that drives resistance down even faster.

Read and write noise varied across the space. Focusing on the regions that were useful for our simulations, read noise (one cycle) was about 1.5% of the resistance value. Write noise for positive pulses (decreasing resistance) was about 10% of R_0. Write noise for negative pulses was about 0.5% of R_0.

With the finished table, we simulated a pulse to the memristor by retrieving the four cells closest to (R_0, V) and interpolating between them. The new resistance resulting from the pulse was $R_1 = R_0 + \Delta + G(\sigma_W \cdot N_W)$, where $G(\cdot)$ is the Gaussian distribution with norm zero, σ_W is the interpolated standard deviation, and N_W is an optional scaling for write noise. Similarly, we simulated a noisy read as $R_0 + G(\sigma_R \cdot N_R)$, where N_R is an optional scaling for read noise.

This model approximates the function $f(R_0, V) \rightarrow R_1$ with a piecewise linear (ruled) surface. In rapidly changing regions

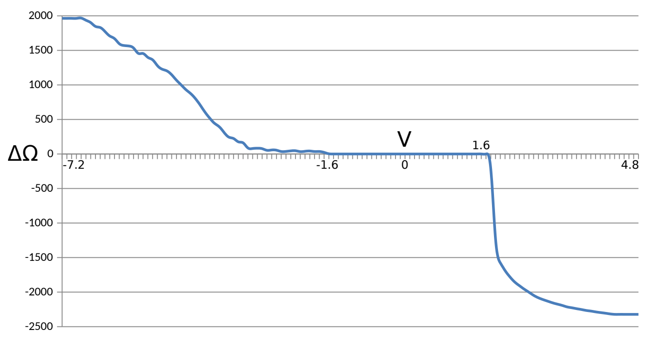
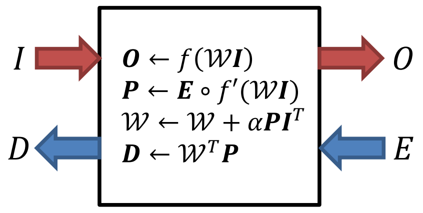


Fig. 4. Resistance change in response to voltage, given a fixed starting resistance of $4k\Omega$. This is effectively a slice through the data in Figure 2.

such as near the "cliff", the model introduces systematic error due to large differences between the actual slope and straight-line fit. Sometimes this difference exceeds 100Ω. One solution would be to use finer binning.



I
$O \leftarrow f(\mathcal{W}I)$
$P \leftarrow E \circ f'(\mathcal{W}I)$
$\mathcal{W} \leftarrow \mathcal{W} + \alpha P I^T$
$D \leftarrow \mathcal{W}^T P$
O
D
E

Fig. 5. One stage of a multi-layer perceptron. This module performs thee key functions: 1) compute the output $\boldsymbol{O}$ of the layer based on input $\boldsymbol{I}$, 2) update weights $\mathcal{W}$ based on error feedback $\boldsymbol{E}$, 3) send error feedback $\boldsymbol{D}$ to preceding layers.

III. Training a Multi-Layer Perceptron with Backpropagation

For clarity, we give a very brief summary of multi-layer perceptrons (MLPs) and the Backpropagation algorithm (Backprop). One stage of an MLP has the form of a matrix-vector multiply followed by a differentiable squashing function: $\boldsymbol{O} = f(\mathcal{W}\boldsymbol{I})$, where $\boldsymbol{I}$ is the input vector, $\mathcal{W}$ is the weight matrix, $\boldsymbol{O}$ is the output vector, and typically $f(x) = \frac{1}{1+e^{-x}}$ (the logistic sigmoid function). Simple function composition expresses multiple layers: $\boldsymbol{O} = f(\mathcal{W}_2 f(\mathcal{W}_1 \boldsymbol{I}))$. The venerable Backprop algorithm is an application of gradient descent to the weights $\mathcal{W}_i$ where the derivatives are determined by the chain rule. Specifically:

$$\begin{aligned}
\boldsymbol{O}_1 &= f(\mathcal{W}_1 \boldsymbol{I}) \\
\boldsymbol{O}_2 &= f(\mathcal{W}_2 \boldsymbol{O}_1) \\
\boldsymbol{E}_2 &= \boldsymbol{T} - \boldsymbol{O}_2 \\
\frac{\partial}{\partial \mathcal{W}_2}(\tfrac{1}{2}\boldsymbol{E}_2^2) &= -\boldsymbol{E}_2 \circ f'(\mathcal{W}_2 \boldsymbol{O}_1)\boldsymbol{O}_1^T \\
\frac{\partial}{\partial \mathcal{W}_1}(\tfrac{1}{2}\boldsymbol{E}_2^2) &= -(\boldsymbol{E}_2 \circ f'(\mathcal{W}_2 \boldsymbol{O}_1))^T \mathcal{W}_2 \circ f'(\mathcal{W}_1 \boldsymbol{I})\boldsymbol{I}^T \\
&= -\boldsymbol{E}_1 \circ f'(\mathcal{W}_1 \boldsymbol{I})\boldsymbol{I}^T
\end{aligned}$$

where $\boldsymbol{T}$ is the "truth" or "target" vector for the network output, $\boldsymbol{E}_i$ is the error feedback to a given layer, and $\circ$ is elementwise multiplication. Each layer is updated using a step size α:

$$\mathcal{W} \leftarrow \mathcal{W} - \alpha \frac{\partial}{\partial \mathcal{W}}(\tfrac{1}{2}\boldsymbol{E}^2). \qquad (1)$$

The step size is typically a positive constant much smaller than 1. Alternately, a line search can find the step size that gives the largest improvement in each cycle. We used a constant $\alpha = 0.01$ in all the tests reported below.

A. Crossbar implementation

A memristor crossbar (abbreviated "xbar") could be part of an electronic module that does all the work of one perceptron layer (Figure 5). These modules could be chained together to form an MLP (Figure 6).

A module may be implemented in several different ways. Figure 7 shows a highly abbreviated xbar. In addition to a dense 2D array of memristors, a certain amount of peripheral circuity would be needed to drive the devices and perform the functions described in Figure 5. In the design shown here, each perceptron weight is represented by one memristor device. In order to have signed weights, there is an additional row of conventional (non-programmable, low noise) bias resistors. These produce a bias current which is fed to each row circuit. The rows and columns also have driver circuits which pulse the array to program the memristors.

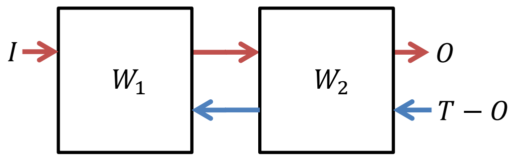


Fig. 6. An example of how two blocks might be wired together. It would also be possible to route wiring to several different blocks.

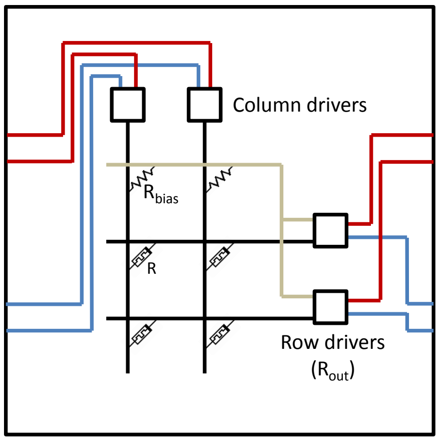


Fig. 7. How an xbar might implement a perceptron block. Column and row drivers link into previous and subsequent stages. Conventional bias resistors enable signed weights. Memristors represent the weights themselves.

Omitting the electronic details, an output circuit contains two key resistors that determine the operating range of the memristor devices. R_{bias} is the fixed conventional resistor that determines the value of a "zero" weight. R_{out} is an output resistor that scales current to voltage. A weight w translates to a conductance (inverse of resistance) in this system, which has the following relationship with its associated memristor value R:

$$w = R_{out}\left(\frac{1}{R} - \frac{1}{R_{bias}}\right) \qquad (2)$$

Ultimately, R contributes to a voltage that represents the linear (non-squashed) output of the perceptron. This passes through some electronic implementation of the sigmoid function. In our simulations $R_{bias} = 4k\Omega$, as this is roughly in the center of the cleanest data in the table. The scale factor was $R_{out} = 40k\Omega$, which gives us weights in $-3.\bar{3}$ to 10 for resistances in $6k\Omega$ to $2k\Omega$ respectively. We observed that conventional MLPs trained on the problems presented here typically had weights in $[-3, 3]$, so this configuration was sufficient.

B. Learning on an xbar

Each training cycle produces a set of weight changes (Equation 1). Learning would be most efficient if there existed a set of row and column voltages that could fully update the xbar in a single pulse. Unfortunately, the asymmetry in memristor response (Figure 4) prevents a single pulse that correctly delivers both positive and negative changes. Furthermore, the shape of the response curve does not allow the set of column voltages suitable for one row to work with a different row. Thus, the best we can do is pulse the array row-by-row.

Alternately, it may be possible to pulse the whole array using voltages right at the programming threshold (in our case $\pm 1.6V$) as described in [10]. The idea is to rely on a large number of small pulses that add up in the correct direction for each weight. Unfortunately, this approach failed to converge in our test setup.

As a consequence of Backprop itself, both pulsing methods are forced to change the resistances by amounts that are orders of magnitude smaller than the observed noise in memristor devices. We explore this in Section IV.

We developed software that implemented weights in a modular way, so that the table-lookup memristor emulation could easily replace conventional floating-point weights. This allowed us to validate network structure and performance before testing memristor behavior. The software was not intended to be a full electrical simulation, simply a means to explore the feasibility of these algorithms on potential future hardware.

C. Datasets and Results

We tested using the following datasets:

1) The MNIST handwritten digit set, as preprocessed by LeCun et al. [11]. This set has 28×28 pixel input cells. The pre-selected training set has $60,000$ items, and the test set has $10,000$ items.
2) The "Optical Recognition of Handwritten Digits" dataset from the UCI repository [12]. This set has 8×8 pixel input cells. The pre-selected training set has $3,823$ items, and the test set has $1,797$ items.
3) A Boolean function with 4 bits of input and one output bit. The Boolean data consisted of all 16 possible inputs plus their ground-truth output. We used the full set for both training and testing, and required that the resulting network achieve 100% accuracy.

Training had three possible stop conditions. Define an "epoch" as one pass through all the training data. We could set a limit T_{epoch} on the number of epochs. We could limit the maximum error on any element of the output vector, that is, $\|\boldsymbol{E}\|_\infty \leq T_{max}$. Finally, we could limit the average error, $\frac{\|\boldsymbol{E}\|_1}{n} \leq T_{avg}$, where n is the number of elements in $\boldsymbol{E}$. Note that $\boldsymbol{E}$ is different than the actual classification, and that classification accuracy is measured on the test set, which may

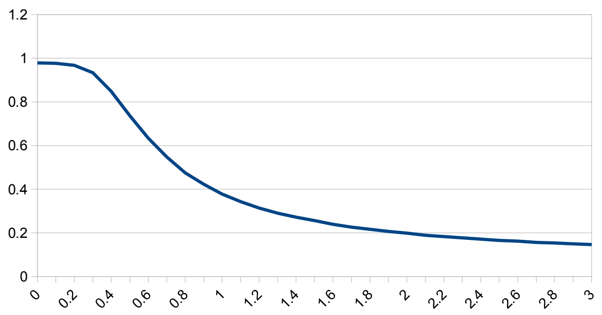


Fig. 8. Accuracy of a converted xbar on dataset 1 as N_R (multiple of observed noise) varied between 0 and 3. Based on three trials per noise level.

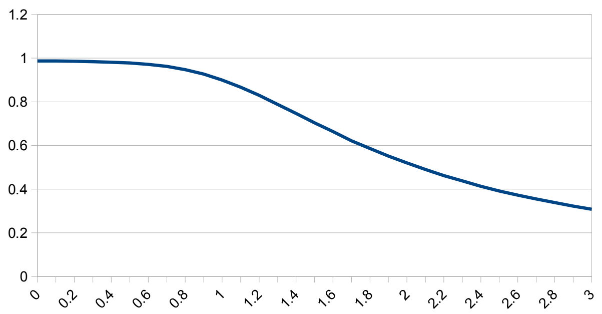


Fig. 9. Accuracy of a converted xbar on dataset 2. Based on 100 trials per noise level.

be different than the train set. Thus, T_{avg} and T_{max} are indirect predictors of classification accuracy.

We trained conventional (float) MLPs with 300 hidden units on dataset 1. It took 695 epochs to satisfy $T_{avg} = 0.1$, resulting in an accuracy of 96.5%. See Table I for a full listing of results and their statistics. Note that when a σ column is blank, the column to its left is exact.

We did not attempt to fine-tune the network for better accuracy, because the goal was to test the feasibility of a memristor-based network, not to set a new classification record. We converted this network to resistance values using Equation 2. Without any noise, the resulting memristor simulation achieved the same level of accuracy as the float version. As noise increased, the accuracy dropped off (Figure 8). At a realistic level ($N_R = 1$) it fell to 37.8%.

We trained conventional MLPs with 36 hidden units on dataset 2. The only stop condition was $T_{epoch} = 1,000$. The resulting network achieved 96.6% accuracy. A memristor simulation with converted weights and no noise achieved the same level of accuracy as the float version. With $N_R = 1$ the accuracy of the converted network dropped to 90% (Figure 9).

Because of the higher cost of simulation (about $10\times$ to $20\times$ the cost of the float version, depending on the noise model), we only attempted xbar training on the smaller datasets. We trained on dataset 2 using the row-by-row pulsing method, $N_R = 1$ and $N_W = 1$. The network achieved 80.9% accuracy after $1,000$ epochs.

TABLE I
SUMMARY OF SIMULATIONS.

Dataset	Method	Runs	Epochs	σ	Accuracy (%)	σ	Time (s)	σ
1	Backprop/float	4	695	24	96.5	0.29	14143.2	595.7
2	Backprop/float	4	1000		96.6	1.48	108.8	2.1
2	Backprop/xbar ($N_W = 1$, $N_R = 1$)	5	1000		80.9	2.22	2077.4	20.8
3	Backprop/float	10	11310	1046	100		1.97	0.23
3	Backprop/xbar ($N_W = 1$, $N_R = 1$)	4	100000		79.7	5.98	45.6	0.66
3	Backprop/xbar ($N_W = 0$, $N_R = 0$)	4	100000		60.9	9.38	27.9	0.62
3	LOTTO/xbar ($N_W = 1$, $N_R = 0$, converged)	22	14162	16840	100		1.51	1.83
3	LOTTO/xbar ($N_W = 1$, $N_R = 0$, not converged)	11	101851	457	83	10.49	11.5	0.61
3	LOTTO/xbar ($N_W = 1$, $N_R = 1$, 100 reads, converged)	4	36006	28312	100		337.6	260.4
3	LOTTO/xbar ($N_W = 1$, $N_R = 1$, 100 reads, not converged)	8	101116	166	76.6	4.42	972.2	70

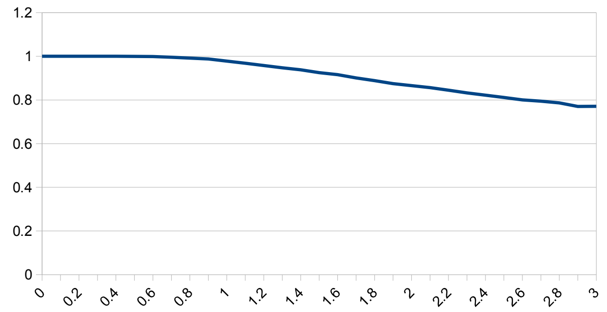


Fig. 10. Accuracy of a converted xbar on dataset 3. Based on 1000 trials per noise level.

Dataset 3 was more difficult. An MLP with one hidden layer of sufficient size should be able to learn any Boolean function perfectly. One way to enforce this is to train until all output errors fall below the binary decision threshold (which in this case is 0.5). We trained a conventional network with $T_{max} = 0.4$. It converged in $11,405$ epochs, with 100% accuracy. We converted the learned network to an xbar. Without noise it attained 100% accuracy. With $N_R = 1$ it only fell to 99.6% (Figure 10).

We attempted to train the network directly on an xbar with zero noise, and it repeatedly failed to reach the required 100% accuracy. When trained with the observed level of noise ($N_R = 1$ and $N_W = 1$) it did about 20% better. This is consistent with the observation that some noise in neural-network training can improve the result (for example [13]).

IV. The problem of precision

Our tests showed that the emulated memristor crossbar with a realistic level of noise trained to a lower level of performance than the equivalent conventional ("float") network. In particular, they were incapable of reaching 100% accuracy on the simple Boolean function of dataset 3.

It is well known that noise places a lower bound on precision. We argue here that the precision necessary for Backprop to converge is finer than that allowed by the observed noise of the device. First we examine the precision used by the conventional network.

An IEEE 754 single-precision float has a 23-bit mantissa, the equivalent of about 6 decimal digits. A cursory analysis of Backprop shows that this is just enough. Specifically, suppose $\alpha = 0.01$, $\boldsymbol{I} = \mathbf{1}$, and $\mathcal{W}\boldsymbol{I} = \mathbf{0}$ so $f'(\mathcal{W}\boldsymbol{I}) = f'(\mathbf{0}) = 0.25$. Substituting these into Equation 1 gives:

$$\begin{aligned} \mathcal{W} &\leftarrow \mathcal{W} + \alpha \boldsymbol{E} \circ f'(\mathcal{W}\boldsymbol{I})\boldsymbol{I}^T \\ w &\leftarrow w + 0.01 \cdot e \cdot 0.25 \cdot 1 \\ w &\leftarrow w + 0.0025 \cdot e \end{aligned}$$

Since the typical range of network weights is $[-3, 3]$, we assume no more than 5 significant digits after the decimal point. This implies that $e < 0.001$ will be lost in the addition operation. Late in training, observed values of e are on the order of 0.1 to 0.01, so single-precision floats are sufficient for our purposes. See [14] for a more detailed analysis of precision required by Backprop.

However, the xbar emulation brings more loss of precision. Weights are represented by resistances. From Equation 2 it is easy to derive the relationship between weight change and resistance change:

$$w_a - w_b = R_{out}\left(\frac{1}{R_a} - \frac{1}{R_b}\right)$$

If we let $R_a = R_b - 100\Omega$ and $R_b = 4k\Omega$, we get a weight change of 0.244 per 100Ω. This amount varies between 0.11 and 0.95 depending on where we set R_b, but that does not change this order-of-magnitude argument. Given the observed values of e, we can expect resistance changes as small as $0.0025 \cdot 0.01 \cdot \frac{100\Omega}{0.244} \approx 0.01\Omega$. Our resistance values are on the order of 10^4, so such updates are right at the limit of single-precision floats. Furthermore, they evaluate to voltage pulses that are only a few microvolts above threshold, again straining the limits of single-precision. We could improve the simulation by increasing to double-precision, but all this would accomplish is to more accurately compute the noise that overwhelms the algorithm.

It is extremely unlikely that a practical electronic implementation of an xbar will support microvolt pulses or sub-Ohm resistance changes. Even under closed-loop control, the smallest step is on the order of 10Ω, two orders of magnitude larger than the needed value. The fact that Backprop leads naturally to such absurd scales suggests that it is not the appropriate algorithm.

Consider that we observed $\sigma_W \approx 400\Omega$ in the "cliff" area, four orders of magnitude larger than the needed resistance step size. If the goal were simply to train the network offline and then burn it to an xbar, then this would be acceptable, as we could use closed-loop control to set the resistances. The only barrier to practical use would be read noise. The observed $\sigma_R \approx 60\Omega$ degraded the performance in our simulations by various degrees. It may be possible to mitigate this with more redundant network structures. This is the subject of ongoing work.

V. Training with LOTTO

However, the goal is to use an xbar to learn directly and efficiently. We may rely on the CPU for some tasks, but the sum of the work done by the xbar+CPU should be substantially less than the equivalent algorithm done entirely on the CPU. Otherwise there is no value in hardware-based learning.

We look at the characteristics of the system, and try to take advantage of them. We should minimize expensive operations, and make cheap operations do as much work as possible. The xbar operations, in descending order of time cost, are:

- Load data from CPU. – Assume the use of direct memory access (DMA) to cycle through blocks of training or test data in conventional memory. This will be slightly more efficient than fetching them to the CPU core, but similar enough that we should ignore the difference. The key constraint is that the xbar-based learning algorithm not use substantially more fetches than a CPU-based algorithm.
- Set a resistance with closed-loop control. – The number of iterations, and thus the cost, scales inversely with the accuracy bound ϵ.
- Matrix-Vector multiply.
- Write pulse to entire xbar.

Memristors have an asymmetric response to voltage pulses. Raising the resistance involves low noise and low slope, and thus is easy to control. Lowering resistance involves high noise and high slope. It makes sense to think of the device as controllable in one direction, with reset in the other [15].

A. Algorithm

The learning method presented here (Algorithm 1) is whimsically named "Lazy Optimization Through Targeted Oversampling" (LOTTO). It is a variant of random search [16] specifically adapted to the characteristics of memristors. The key idea is to incorporate write noise as a central part of the algorithm. The "targeted" aspect refers to the fact that sometimes it is necessary to use closed-loop control to set weights, but if we relax ϵ we reduce the cost and also get random sampling around the desired weight.

The algorithm alternates exploration and exploitation steps until convergence. An exploration step picks a new random point in the weight space. An exploitation step goes back to the best point seen so far and samples a random location near it. In either case, LOTTO then iterates through the training data (a full epoch) to measure the goodness of the new point. If it is an improvement, LOTTO accepts the new point, then takes a small random step in the direction of increasing resistance. As long as each new point is an improvement, LOTTO continues the random walk (Figure 11).

Algorithm 1 LOTTO

```
repeat
  {Alternate explore and exploit steps}
  if explore then
    Random (low cost) initialization
  else
    Set to best point, using $\epsilon$-bounded precision
  end if
  {Random search}
  loop
    for all data do
      Test classification accuracy
    end for
    if accuracy is better than current best point then
      Set new best point
    else
      Break inner loop
    end if
    Apply small negative pulse to whole xbar
  end loop
until convergence
```

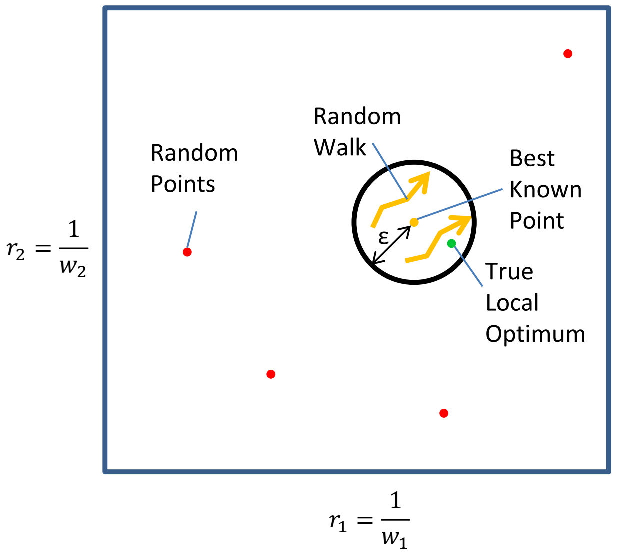


Fig. 11. A random search in the parameter space involves setting the current point with limited precision, along with small steps in the controllable direction.

A random set of weights (exploration step) can be created in a manner similar to the random-pulse sampling method described in Section II-B. The small steps of the random walk are made by pulsing the entire xbar with a negative voltage at the write threshold. Due to the inherent randomness, all resistances will increase by small but varying amounts.

B. Results

We implemented LOTTO within the same framework described earlier and tested it on dataset 3. We set $T_{epoch} = 100,000$ and $T_{max} = 0.4$. Table I separates runs that converged due T_{max} from those that used up the full allotment of epochs. The latter are labeled "not converged".

LOTTO is sensitive to read noise, as it needs to accurately measure the quality of points in weight-space. This becomes more problematic with small numbers of data, such as the 16 items in dataset 3. We trained using full write noise ($N_W = 1$), but relaxed the read noise in two different ways. First, we trained with $N_R = 0$. The average number of epochs ($14,162$) was competitive with conventional Backprop ($11,310$). Second, we tested with $N_R = 1$ but oversampled by $100\times$. This approach was substantially more expensive and less accurate. However, even in the cases where it did not converge, its average accuracy (76.6%) was competitive with Backprop running on an xbar with full noise (79.7%). Note that the reported number of epochs for the "not converged" case is slightly greater than $100,000$ due implementation details.

C. Discussion

Clearly, LOTTO is not a satisfactory algorithm. A random search provides no guarantee of time bound. In fact, it provides no guarantee of convergence, even to a local minimum. Its chances are improved with lower-dimensional or highly-degenerate parameter spaces. Thus, LOTTO is less likely to be useful for larger networks. A simple improvement in the current scheme would be to run Backprop until it stalls, then switch to LOTTO to finish converging.

The ideal memristor learning algorithm would work with resistance changes that are no smaller than the device noise levels, and it would measure its fitness online (that is, using only one or a few current data). There is plenty of space to explore for better algorithms. Categorically these are likely to be non-gradient-based and non-supervised. Examples include self-organizing maps [17], winner-take-all networks [18] and sparse-code learners [19].

VI. Conclusion

We explored the feasibility of direct training of shallow MLP networks using memristor crossbars. We characterized several real memristors and used a non-parametric model of one as an emulator. This demonstrated some promise for hardware-accelerated neural networks. However, noise and asymmetry in the device's response to programming make direct training problematic. We offered a random search approach that, while not ideal in many respects, shows the possibility that hardware accelerators may eventually compete with CPU-based learning.

Acknowledgment

This research was possible in part by LDRD program support from Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] "International technology roadmap for semiconductors." [Online]. Available: http://www.itrs.net

[2] "DARPA SyNAPSE program." [Online]. Available: http://www.darpa.mil/Our_Work/DSO/Programs/Systems_of_Neuromorphic_Adaptive_Plastic_Scalable_Electronics_(SYNAPSE).aspx

[3] "SyNAPSE." [Online]. Available: http://en.wikipedia.org/wiki/SyNAPSE

[4] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, E. McQuinn, B. Shaw, N. Pass, and D. S. Modha, "Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores," in *IJCNN*, Aug 2013, pp. 1–10.

[5] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: http://www.sciencemag.org/content/345/6197/668.abstract

[6] A. Gorchetchnikov, M. Versace, H. Ames, B. Chandler, J. Lveill, G. Livitz, E. Mingolla, G. Snider, R. Amerson, D. Carter, H. Abdalla, and M. S. Qureshi, "Review and unification of learning framework in cog ex machina platform for memristive neuromorphic hardware," in *IJCNN*, 2011.

[7] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high interest credit card of technical debt," in *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[8] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor spice model and crossbar simulation based on devices with nanosecond switching time," in *IJCNN*, 2013.

[9] P. R. Mickel, A. J. Lohn, C. D. James, and M. J. Marinella, "Isothermal switching and detailed filament evolution in memristive systems," *Advanced Materials*, vol. 26, no. 26, pp. 4486–4490, 2014. [Online]. Available: http://dx.doi.org/10.1002/adma.201306182

[10] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor crossbar-based neuromorphic computing system: A case study," *IEEE Transactions on Neural Networks and Learning Systems*, 2014.

[11] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist

[12] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[13] K. Audhkhasi, O. Osoba, and B. Kosko, "Noise benefits in backpropagation and deep bidirectional pre-training," in *IJCNN*, 2013.

[14] J. L. Holt and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Transactions on Computers*, vol. 42, no. 3, March 1993.

[15] G. W. Burr, R. M. Shelby, C. di Nolfo, J. W. Jang, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element," in *International Electron Devices Meeting*, 2014.

[16] J. C. Spall, *Introduction to Stochastic Search and Optimization*. Wiley, 2003.

[17] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59–69, 1982.

[18] P. Sheridan, W. Ma, and W. Lu, "Pattern recognition with memristor networks," in *ISCAS*, 2014, pp. 1078–1081.

[19] R. P. N. Rao and D. H. Ballard, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects," *Nature Neuroscience*, vol. 2, pp. 79–87, January 1999.