SANDIA REPORT
SAND2015-10868
Unlimited Release
Printed December 2015

Cognitive Computing for Security

Erik P. DeBenedictis, Fred Rothganger, Brad Aimone, Matt Marinella,
Brian Evans, Christina Warrender, Alex Hsia, Patrick Mickel

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@osti.gov
Online ordering:	http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.gov
Online order:	http://www.ntis.gov/search

SAND2015-10868
Unlimited Release
Printed December 2015

Cognitive Computing for Security

Erik P. DeBenedictis, Fred Rothganger, Brad Aimone, Matt Marinella,
Brian Evans, Christina Warrender, Alex Hsia, Patrick Mickel
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1319

Abstract

Final report for Cognitive Computing for Security LDRD 165613. It reports on the development of hybrid of general purpose/neuromorphic computer architecture, with an emphasis on potential implementation with memristors.

Table of contents

List of figures

List of tables

Cognitive Computing for Security LDRD

Overview

This LDRD project evolved into a one of an interrelated set of projects responsive to the current interest in "Beyond Moore's Law" technology and as articulated by the emerging National Strategic Computing Initiative (NSCI). The network of projects is illustrated in Figure 1, with time flowing roughly top to bottom.

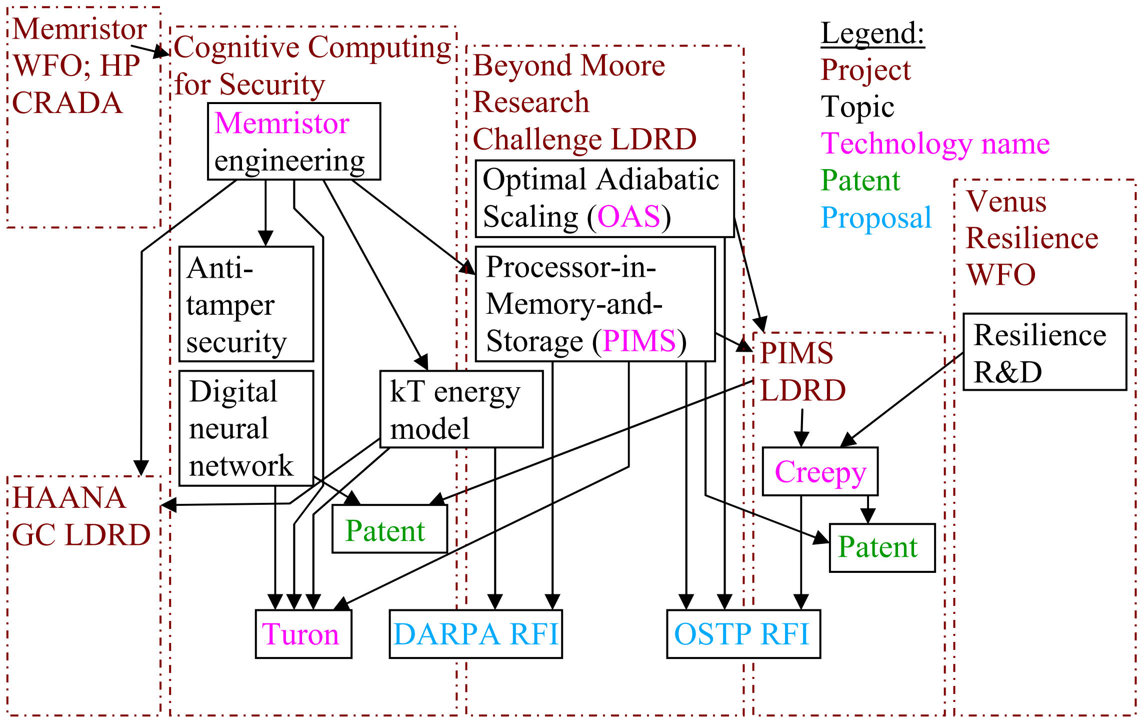


Figure 1: Network of Beyond Moore's Law projects

The set of projects started with a memristor research project funded by Work For Others (WFO). The project was associated with Hewlett Packard and eventually led to a Cooperative Research And Development Agreement (CRADA) with HP. This LDRD was one of several spin offs due to Sandia personnel having their own ideas for novel uses of memristors. Memristors were seen initially as an enabling technology for neural networks.

Sandia embraced Research Challenges in early 2013, funding the Beyond Moore Research Challenge LDRD. The scope of the research challenge was broader than the current LDRD, acting as a sort of "umbrella" for several projects. One of the Sandia people defining the Beyond Moore Research Challenge spent time in Washington helping to define the NSCI program.

Around the beginning of 2014, Sandia staff figured out the new prioritization of computing research at the Federal level. Not surprisingly, Sandia had many of the pieces in house ready. However, this led to reorganizing the ideas in the projects of Figure 1 to create a suite of ideas that (we believed) better match the emerging research directions. In Figure 1, these are:

1. Memristors as a new circuit element
2. The Turing-Neuron Architecture (this project)
3. Optimal Adiabatic Scaling (OAS), for 3D modules to succeed current 2D chips
4. Processor-In-Memory-and-Storage (PIMS) architecture for 3D modules
5. A theoretical model of evaluating energy efficiency of computing approaches, called the kT model
6. The "Creepy" ultra-low energy processor architecture

These were accompanied by two patent filings by the PI, one associated with this LDRD.

Project status

The situation is fluid as this document is being written. The administration announced new research direction a few months ago. We are in a three-month period where government agencies make a plan. Sandia is assisting this process, including by sending ideas in this document to one of the NSCI agencies to help make plans.

Remainder of this document

The remaining pages of this document are as follows:

A technical report on the Turon architecture. The technical report could be a stand alone document, but LDRD rules call for us to append it to the end of this brief project report.

However, Appendix IV (non-interrogatable logic) is a slightly reformatted version of the original anti-tamper security document that motivated this project.

Publications from this project

Publications

Marinella, Matthew J., et al. "Development, Characterization, and Modeling of a TaOx ReRAM for a Neuromorphic Accelerator." Meeting Abstracts. No. 42. The Electrochemical Society, 2014.

Rothganger, Fred, et. al., Training neural hardware with noisy components, IJCNN 2015
A.J. Lohn, P. R. Mickel, J. B. Aimone, E. P. Debenedictis, and M. J. Marinella, “ Memristors as synapses in artificial neural networks: Biomimicry beyond weight change,” in Cybersecurity Systems for Human Cognition Augmentation (Springer, 2014).

Presentations

Erik DeBenedictis, “Potential development plan to create a neuromorphic processor, with key steps including software and Alex's chip,” presentation at AFRL, Kirtland, April 16, 2013

Matt Marinella, “Recent work with Memristor devices and integration,” presentation at AFRL, Kirtland, April 16, 2013.

Alex Hsia, “Multi-project wafer run that went out about a month ago, years ago,: presentation at AFRL, Kirtland, April 16, 2013”

Rothganger, Fred, “Can Memristors Learn,” Presentation at NICE 2015

Making Smarter Memristor-Based Neurons, Erik DeBenedictis, et. al., NICE 2014.

E. DeBenedictis, “A New Approach to Memory: Theory and Computing Approach,” Response to ACCESS RFI: Analog and Continuous-variable Co-processors for Efficient Scientific Simulation

Erik DeBenedictis, “Memristor-based Neuromorphic Computing for Security,” visit to AFRL Rome, July 31, 2013

Patent

Erik DeBenedictis, “METHOD AND APPARATUS FOR MANAGING ACCESS TO A MEMORY,” filed August 20, 2015

The Turing-Neuron Architecture (Turon)

Erik DeBenedictis, Fred Rothganger, 9/15/2015

Abstract

Von Neumann and Neuromorphic computers have been viewed as operating on different principles, but this document shows how both can be viewed as special cases of a more general architecture. One portion of a Turing-neuron (Turon) system described here can use a specific artificial neuron to learn and recognize patterns as is common today in neural network R&D projects. Digital circuits can be trained into other neurons, which will result in a neural network that can execute the digital function by “imagining” the digital circuit. The combination of the two would enable self-contained systems that do both precise mathematical calculations and less rigid learning and recognition functions.

This document discusses the potential performance of various implementations, showing that Turon could possibly simulate CMOS at lower power than CMOS could do on its own. This would make Turon a candidate for a new approach to computing beyond CMOS and the von Neumann architecture.

Overview

A Turon computer would be manufactured as arrays of two types of unallocated resources, one type corresponding to synapses and the other corresponding to neuron bodies (soma). The architecture will allow resources to be partitioned into regions of various sizes, each of which could be a neural network, a Boolean logic circuit, or a hybrid of the two. The partitioning would need to occur on the fly for greatest generality, just as a microprocessor system can assign memory to different tasks on the fly. However, less ambitious, practical systems could have a reconfigurable design like a Field Programmable Gate Array (FPGA).

Duality between neurons and logic circuits

A portion of the resources could implement digital circuits designed with CAD tools or programmed with compilers. For example, the gate-level ALU illustrated in Figure 2 has been partitioned into green-outlined, neural network regions that would be trained to behave like the logic circuits shown within their boundary (with red lines indicating the division of the neural network into multiple layers). Inter-region connections convey signals just like the wires and busses in digital logic; a subset of the interconnect is shown in purple. The design process would create training sets for the input-output mapping of gates within each of the logic circuits. In some cases, training would be simulated in software and yield tables of synapse values.

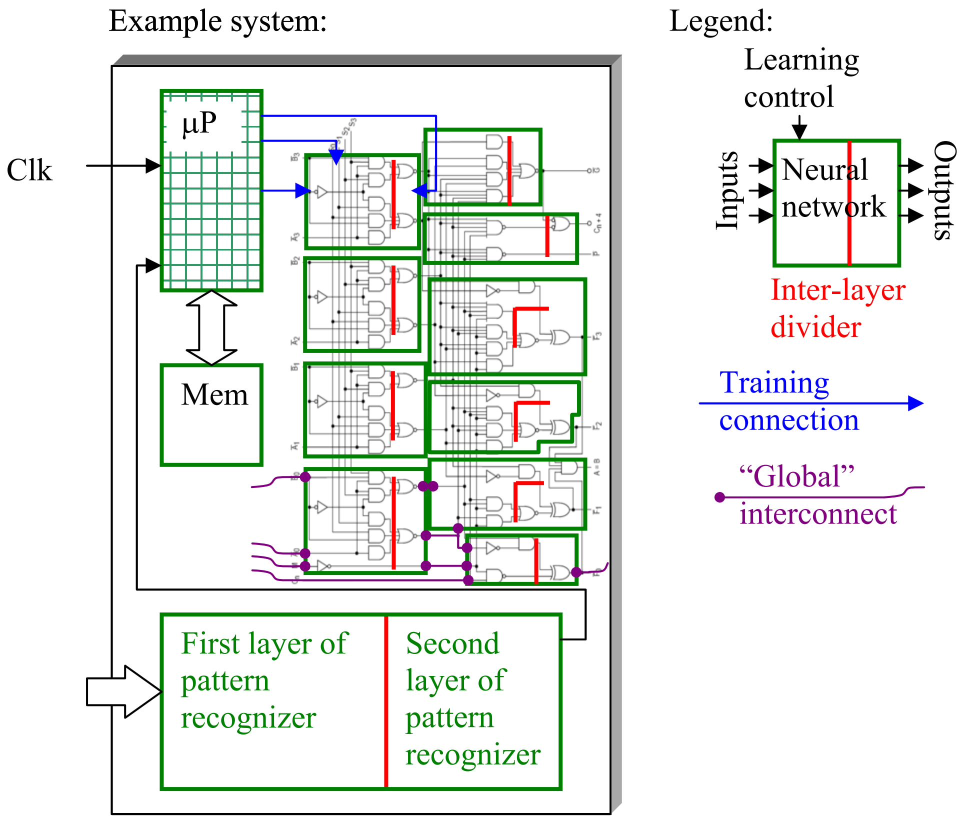


Figure 2: Architecture of a mixed neural-digital computer

Turon start up and operation

The example system in Figure 2 would contain only the controller in the upper left after manufacture or power-up. The controller would first use FPGA-like features in the chip to partition resources into the green regions defining neural networks of various sizes. The controller could also establish connections as shown by blue lines for training and the purple lines for execution. The controller would set up the behavior of the digital circuits either by applying the training sets to the neural networks or loading the pre-computed synapse values. Once programmed, the blue training connections would be removed and purple execution connections would be added (such as connection to an external clock). The system would have just added a digital logic circuit, which could become a new controller for the rest of the system in a type of self-replication. The controller could be a microprocessor, in which case it could execute any program and thereby demonstrate Turon to be a finite Turing machine.

However, the controller could also use the FPGA-like features to create traditional neural networks, such as the pattern recognizer at the bottom of Figure 2. While not fully trained in advance, these portions will be exposed to training patterns during operation. A digital logic circuit would provide control waveforms for the neural components.

For example, a digital logic circuit might, for example, produce waveforms on its output wires that cause an external source (for example, a TV camera) to apply 1000 patterns to the pattern recognizer, while other waveforms cause the patterns to be learned via back propagation. After the learning phase, the digital circuit could produce different waveforms that sequence an additional 100 patterns and attempt to recognize them.

Optimizations

Implementations of Turon are expected to be optimized for the following modes of operation:

<u>Memory</u> support would be specifically engineered for high speed and storage efficiency. The synapse resource can become computer-type memory at about a 1:1 ratio of synapses to bits or words. Memory implementations would potentially be implemented as a specialized neural network, where the memory cells at the intersection of rows and columns in the memory are equivalent to synapses at the intersection of horizontal axons and vertical dendrites.

<u>High throughput</u> would be a design objective, specifically when compared to a von Neumann computer. A von Neumann computer executes one instruction per clock cycle irrespective of the size of the computer, where Turon would have parallelism similar to a FPGA. FPGAs group a few hundred transistors into a Look Up Table (LUT), which can perform a logic operation each clock cycle. By analogy, Turon groups a few hundred synpases into a region equivalent to a handful of gates. These perform their function once each clock cycle. Since synapses are equivalent to memory in about a 1:1 equivalence of capacity, Turon's throughput is comparable but superior to a von Neumann computer

because the throughput per clock cycle is not constant but proportional to the amount of memory. This makes the throughput of Turon comparable with systolic arrays, FPGAs, and processor-in-memory approaches.

<u>Leaning</u> has no simulation overhead for Turon – a feature that may be timely because of current interest in machine learning. Neural circuits that can learn are valuable because they create pattern-recognition logic on the fly that would otherwise require human ingenuity. While the Church-Turing thesis holds that a von Neumann computer could simulate Turon, such a simulation could require many instruction cycles to simulate the learning of one synapse at a time. In contrast, many or all synapses in Turon can learn in parallel and implementations should be optimized to do this quickly and efficiently.

Implementation approaches

As an architecture, Turon is independent of physical implementation, but discussion of several implementations is helpful to understanding its possibilities. Figure 3A and Figure 3B illustrate an analog implementation based on memristors. However, memristors and other technologies can be used as memory cells in a 3D array storing synapse values digitally as in Figure 3C. Both the analog neuromorphic circuit in Figure 3B and the 3D memory array in Figure 3C can emulate either a neural network as in Figure 3A or a digital circuit as in Figure 3D.

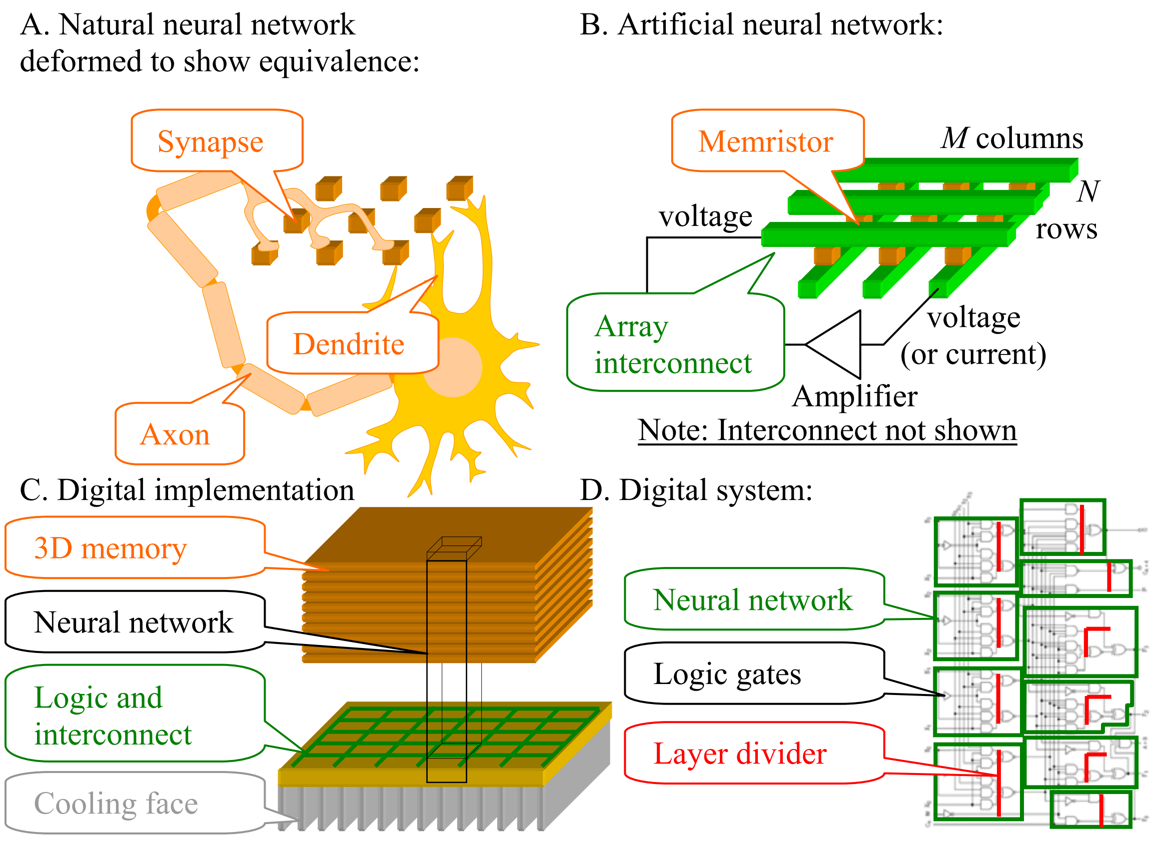


Figure 3: Explanatory implementations of the Turon architecture

Boolean logic

The ability of a neural network to act as a digital circuit is extremely well founded in both theory and experiment, yet the literature contains multiple interpretations. There are a variety of papers that show mathematical equivalence of a (non-learning) neural network and a Turing machine [Killian 93], yet we do not use this interpretation here. This document uses the equivalence between certain artificial neurons and the Boolean logic that underlies essentially all computers. The interpretation used here also embraces the learning behavior of the neurons.

Logic circuits can be embedded in perceptron-class neural networks with about a 1:1 ratio of gates to neurons. As illustrated in Figure 4, a perceptron-type neuron can create the input-output behavior of a logic gate by suitable choices of weights (w) and thresholds (θ) [Wing yy]. The diagram in Figure 4A is a two-level network like the ones in Figure 3D, yet also showing resistive weights like Figure 3B (negative weights are shown on resistors due to space limitations, yet they imply unrealistic negative resistors). For example, the three columns and two rows in Figure 4A implement NAND, OR, AND, and A OR NOT B functions along with interconnect to the necessary inputs. Figure 4B shows the multi-level logic network equivalent to the multi-layer network in Figure 4A, which is an example that generalizes. However, the number of neural network layers must match the number of logic levels. The number of neurons and synapses in each layer must be sufficient to hold the number of gates in the corresponding level.

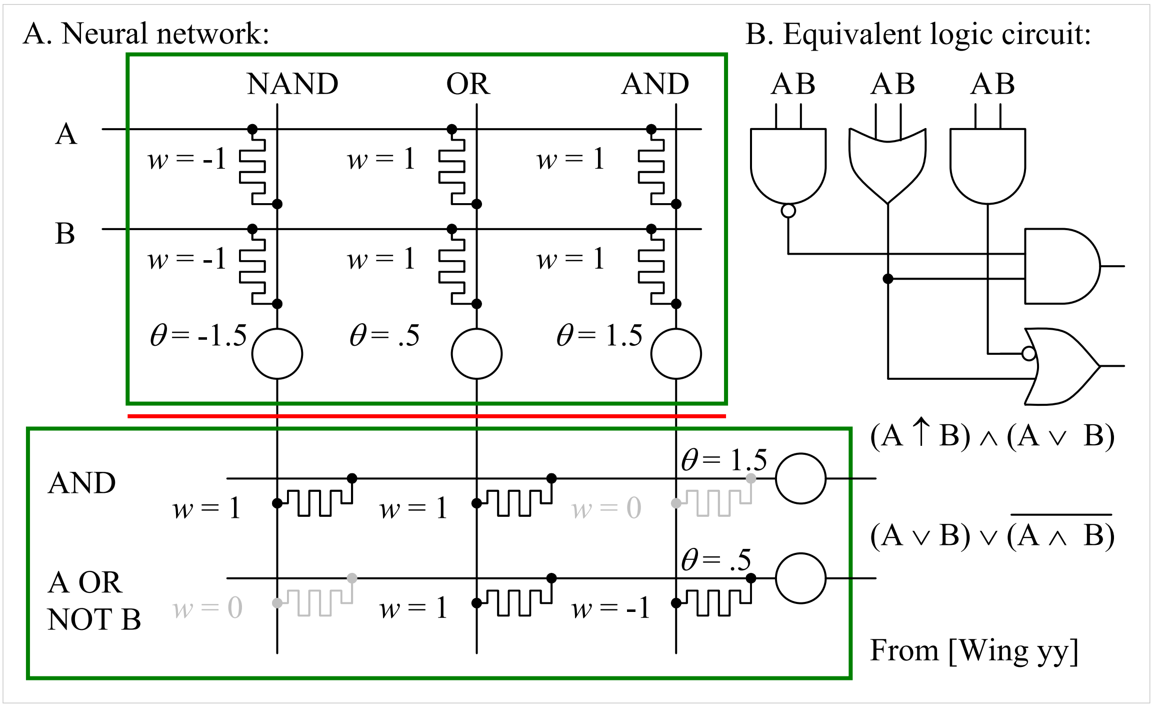


Figure 4: Equivalence between gates and perceptron-style neural networks

A Turing machine includes a state machine, which is implemented in Boolean logic as a combinational logic network with feedback. Neural networks often have feedback.

It might look like the training set would have to be very large and learned with 100% accuracy, but the techniques below tend to simplify training:

- A multi-layer neural network can be trained one or a few layers at a time, cutting the exponential growth in training vectors.
- Most logic designs have "don't care" input combinations. Eliminating these from training set could improve efficiency. However, logic design tools do not necessarily identify these.

Interconnect

The structure in Figure 4 duplicates both Boolean logic and its wiring pattern, with the wiring pattern being learned as opposed to being manufactured into the device. This effect can be viewed mathematically as a sparsity pattern in the array of Figure 3B. Zero-valued array entries assume a role similar to wires carrying signals long distances without their being involved in logic along the way. For the crossbar in Figure 3B, this would either mean synapses (memristors) would have zero weight (infinite resistance) or the crossbar would give way to a specialized interconnect structure based on the sparsity pattern. The former option is inefficient while the latter reduces the generality of the system. In contrast, the digital approach in Figure 3C would naturally support a signal routing network in the interconnect layer.

Semirings and other computing primitives

The structures in Figure 3B and Figure 3C can be applied to different types of data and operations. The functioning of many types of neural networks is described as a series of matrix operations. For example, the neuromorphic array in Figure 3B can be described as multiplication of a vector of voltages on the rows by a matrix of conductances (1/resistance) at the row-column intersections, producing a vector of currents at the amplifiers. While Figure 3B shows a single array with a single loopback of one signal, a mathematical description of Figure 3D would be a chain of vector-matrix multiplies where the result of one becomes input to the next.

Turon allows the data type of the elements of the vectors and matrices to vary over a considerable range. The data types and elements can be a semiring [Wikipedia SR], which is defined as a set of elements with add and multiply operations defined. Semiring

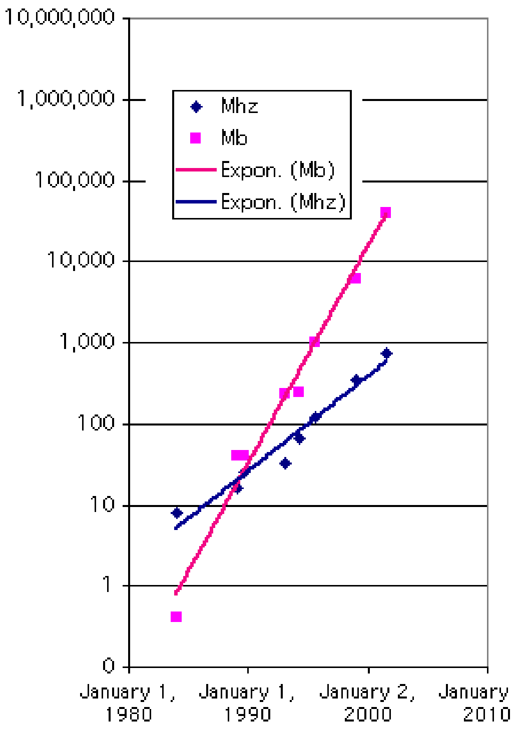
Analog implementations of Turon can readily function with two semirings. Perceptron-based neural networks such as Figure 4A use the semiring comprising natural numbers under ordinary addition and multiplications. It should be noted that the circuitry in Figure 4A would limit the range of the natural numbers and their precision. Logic circuits like Figure 4B use Boolean logic, where the values are just 0 and 1. However, mapping OR to addition, AND to multiplication, and NOT(x) to 1-x make these two systems compatible.

Digital implementations are adaptable to other semirings, enabling support for a wider range of applications. Floating point approximates a ring or semiring, subject to broader limits on precision and range than possible with analog signals. Turon with floating point would be able to do sparse matrix operations like finite elements, yet would require a digital implementation such as Figure 3C. Turon would be able to perform graph algorithms if [Shinn 13] min and + on, essentially, floating point values correspond to + and × of what is called the distance semiring [Shinn 13]. There are many other possibilities.

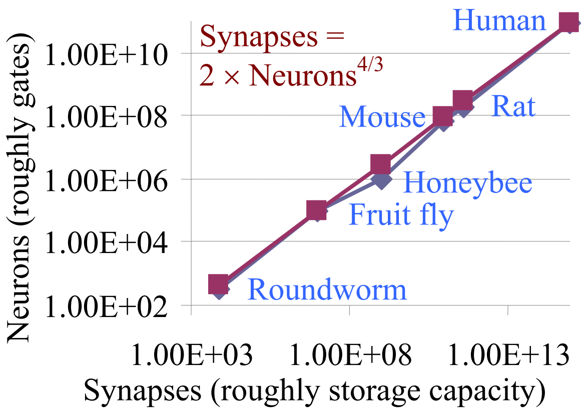
Scaling

Scaling rates differ for logic and memory, which impacts suitability of the approach in Figure 3B versus Figure 3C. The conventional wisdom is that logic and memory should be built from the same technology, thus subject to the same scaling rule. This is the basis of today's microprocessor and memory chip manufacturing. However, a fixed ratio of logic to memory is not found in computing when viewed broadly. For example, the capacity of disk drives is shown in Figure 5A and follows Kryder's Law [Walter 05], where disk drive capacity grows over time at double the slope of computer throughput (which was very closely tied to clock rate in the era covered by the plot). Nature has also created a family of information processing systems (brains) that follow the scaling rule shown in Figure 5B. Over the evolutionary scale up sequence from roundworm to human, the storage capacity (number of synapses) is proportional to the number of logic elements (number of neurons) raised to the 4/3 power [Herculano-Houzel 11] [Wikipedia YY].

A. Computer systems:



B. Brain scale up sequence:



← Growth rate of HDD storage space compared to computer clock rate using Apple consumer products (1984-2001). From Wikipedia, which cites the diagram to left as © Creative Commons.

Figure 5: Throughput vs. memory scaling across diverse computing systems

Turon is illustrated Figure 3C as logic on the 2D face of a 3D solid, where the 3D volume contains memory. This approach would cause memory capacity to grow as the amount of logic to the 3/2 power. An exponent of 3/2 is closer to the observed scaling rate for computer systems in Figure 5A (exponent 2) and biological brains in Figure 5B (exponent 4/3) than von Neumann computers (exponent 1).

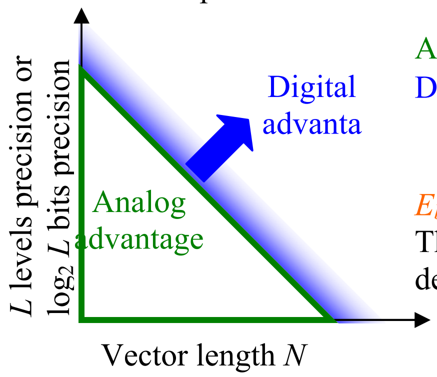
Security

The Turon concept developed from a security idea. A reformatted version of the original document is included in Appendix IV.

Analog vs. digital performance limits (non-sparse)

We developed a method of comparing computing approaches, which will show that analog and digital are each more power efficient in different regions of the general problem space. Figure 6A gives a simplified preview of the result, showing that the circuit Figure 3B has the advantage a small scale and low precision as shown in green. Digital approaches such as Figure 3C scale better, making them more advantageous in the blue zone.

A: Parameter space



B: Equations

Analog: $E_{min} \approx O(N^2L^2)$ kT

Digital: $E_{min} \approx \underbrace{O(N \log_2^2 L)\ \mathrm{kT}}_{\text{(logic energy)}} + \underbrace{N \log_2 L\ E_{bit}}_{\text{(memory energy)}}$

E_{bit} is the energy to access each bit from memory. There is no universal consensus on circuit definitions; see [3].

Figure 6: Best approach for dense dot product

Figure 6B includes complexity expressions for the ultimate physical limits of an N-element dot product calculation of L-level ≡ $\log_2 L$-bit values using analog/ digital approaches. We merge ideas from Shannon [Shannon 48], Landauer [Landauer 61], and computational complexity theory. Shannon tells us how raising signal energy as a multiple of kT increases Signal to Noise Ratio (SNR) and gives more reliable results. Landauer showed how energy in logic computation is also related to kT.

We merge these concepts to create a complexity measure that expresses energy for an algorithm-computer combination in the form $f(N, L)$ kT, where f is like a complexity expression but with significant constant factors (i. e. not big-O notation). The expressions are on the right of Figure 6 and documented in [DeBenedictis 14]. Digital E_{min} energy has separate terms for logic and memory energy, with E_{bit} being the energy to access one bit from the memory. E_{bit} will be discussed later.

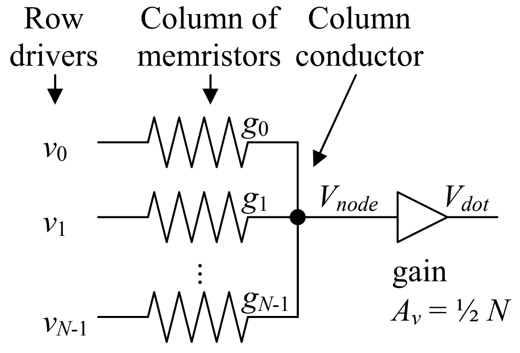
While the result for the digital circuit is consistent with ideas in the literature, readers may find two surprises in the analog result. The first is that the analog result has the same kT units as the digital result. The second is that the f's have different asymptotic behavior. For example, an analog N-element dot product consumes $O(N^2 L^2$ kT) energy whereas a digital one is $O(N \log_2^2 L)$ kT.

Energy efficiency limit of dense analog vector-matrix multiply

We first analyze an N-element dot product of two vectors $\boldsymbol{v}$ and $\boldsymbol{g}$, M copies of which are equivalent to the $N \times M$ dense vector-matrix multiply in Figure 3B. In preparation for comparison with digital approaches, we want the answer to be correct with probability 1-p_{error}.

The voltages of $\boldsymbol{v}$ and the synapse values of $\boldsymbol{g}$ are both defined to be L-level analog signals. Voltages v_i driving each row are assumed to be uniformly distributed in the range [-V, V]. The other vector $\boldsymbol{g}$ is defined by the state of memristors in a column and comprises conductances g_i uniformly distributed in the range [0, g_{max}]. (The V's will cancel algebraically in the energy consumption equation, but the fact that the V's are distributed symmetrically around zero will affect constant factors. The value g_{max} will cancel similarly, but becomes the definition of the numerical value 1 for the elements of $\boldsymbol{g}$.)

We use the resistive combining network in Figure 7, which actually forms the weighted average of the input voltages instead of the dot product. However, the equations show a weighted average is mathematically equivalent to a dot product divided by the sum of the weights. We can recover the dot product by amplifying (multiplying) the voltage, but the gain has to be chosen carefully.



$$V_{node} = \frac{\sum_i v_i g_i}{\sum_i g_i}$$

Assume
$\sum_i g_i = ½ \, g_{max} N$

$$V_{dot} = \textstyle\sum_i v_i \, (g_i/g_{max})$$

Figure 7: Dot product circuit

Since multi-level neural circuits use the output of one vector-matrix multiply as the input of the next, it would be best for the output and input signals to be compatible. We use voltage. However, when voltage v_i is multiplied by conductance g_i during the computation of the dot product, the result has units of current. Our remedy is to interpret g_i's in the range [0, g_{max}] as weights in the range [0, 1]. If we assume the average memristor conductance is ½ g_{max} (which is the statistical midpoint but strictly speaking implies normalization on each step), the sum of the weights will be ½ g_{max} N. If the amplifier's gain is set to A_v = ½ N, the circuit will compute the dot product subject to the considerations just mentioned.

Gain A_v = ½ N is controversial among reviewers. In subjective terms, a signal entering one input will leak backwards through the N-1 other inputs. As N grows, the amplifier must use more and more gain to counteract the reduced signal. However, gain in the amplifier also amplifies noise and creates other deleterious effects. Appendix I has more detail and the next section also analyzes a circuit with better scaling.

Transients will be due to Johnson-Nyquist noise in the limiting case, with the time available for transients to die out determined by system speed or clock rate. For the time being, let us assume the circuitry in Figure 7 is bandlimited to frequency *f*. The noise power according to the Johnson-Nyquist noise theorem will be 4kT *f* at the input to the amplifier. (*F* will cancel algebraically in the energy consumption equation.) Equating noise power to voltage^2 divided by resistance, which is equivalent to voltage^2 times the average conductance ½ g_{max} *N*, we get

$$\overline{P_{noise}} = 4\ \mathrm{kT} f = \overline{V_{noise}}^{2}\ ½\ g_{max} N, \tag{1}$$

Which yields

$$\overline{V_{noise}} = \left[\frac{8\ kT}{N} \frac{f}{g_{max}} \right]^{½} \tag{2}$$

In accordance with previous discussion, the noise will be amplified by A_v before appearing on the output. However, we are interested in the peak noise transients that occur with probability p_{error}, which requires multiplying the average noise voltage by $\ln(1/\sqrt{p_{error}})$.

$$\overline{V_{peaknoise}} = \overline{V_{noise}}\, A_v \ln(1/\sqrt{p_{error}}) = \overline{V_{noise}}\ ½\ N \ln(1/\sqrt{p_{error}}) \tag{3}$$

The number of reliably distinguishable resolution levels *L* will be the output range 2*V* divided by the peak noise voltage $V_{peaknoise}$.

$$L = \frac{2V}{V_{peaknoise}} = \left[\frac{N}{8\ kT} \frac{g_{max}}{f} \right]^{½} \frac{4V}{N \ln(1/\sqrt{p_{error}})} \tag{4}$$

Squaring (4) and rearranging yields a form that has units of energy and will be useful later

$$\frac{V^2 g_{max}}{f} = ½\ L^2 N \ln(1/p_{error})\ \mathrm{kT} \tag{5}$$

Energy per operation can be computed by the average amount of heat produced by the memristors. V_{node} moves asymptotically to zero as *N* increases, so we assume in this analysis that $V_{node} = 0$. If V_{node} is grounded, there will be *N* uniformly distributed voltages [-*V*, *V*] across resistors with average conductance $g_{max}/2$. This yields the base power of 1/6 V^2 g_{max} per resistor and total power

$$\overline{P_{neuron}}^{(B)} = 1/6\ V^2 g_{max} N \tag{6}$$

We will designate the base power with the superscript $^{(B)}$, which includes the assumption of large N and $V_{node} = 0$. However, a correction for small N is given in Appendix I.

We must now establish a connection between operating speed and the Johnson-Nyquist noise. We had previously assumed the circuitry would be bandlimited to f, but f has so far been just an algebraic symbol. We are now free to choose a specific value for f for lowest energy per operation. The Nyquist sampling theorem states that the maximum rate at which voltages could be applied to the rows would be $2f$. If so, the energy to evaluate a neuron would be P_{neuron} times the sample-to-sample interval $1/(2f)$.

Dividing equation (6) by $2f$ and substituting (5) yields

$$\overline{E_{neuron}}^{(B)} = \frac{P_{neuron}^{(B)}}{2f} = \frac{V^2 g_{max} N}{12f} = \ln(1/p_{error})/24\, L^2 N^2\, \mathrm{kT} \qquad (7)$$

The equation above is notable because implementation details V, g_{max}, and f cancel out algebraically, leaving an implementation-independent expression.

In conventional computer terminology, the system will perform N multiply operations. The energy per operation will be $E_{neuron}^{(B)}$ from (7) divided by N.

$$\text{Energy/op} = \ln(1/p_{error})/24\, L^2 N\, \mathrm{kT} \qquad (8)$$

Which tells us the energy per equivalent multiply operation is proportional to the number of elements in a column. This is a notable difference from digital summation.

The above expressions are for a dot product. If we multiply (8) by the number of output neurons M, we get the energy of an $N \times M$ vector-matrix multiply, as may occur in software-based methods such as Deep Learning.

$$\overline{E_{vmm}}^{(B)} = \ln(1/p_{error})/24\, L^2 N^2 M\, \mathrm{kT}, \qquad (9)$$

where E_{vmm} is the energy of a vector-matrix multiply.

Improving dense analog vector-matrix multiply

It is possible to improve the performance of the analog vector-matrix multiply through a simple method, albeit a method apparently not in the literature so far. The poor N^2 scaling is due to the circuit activating all the rows at once. Say that an N-element dot product were to be computed by dividing the dot product into j smaller dot products of N/j vector elements each. This would require an additional activity of adding the j resulting analog values, which we will designate as consuming power J. However, this would reduce the energy from

$$C\,N^2\,\mathrm{kT}, \tag{10}$$

for some constant C to

$$j\,C\,(N/j)^2\,\mathrm{kT} + J = C/j\,N^2\,\mathrm{kT} + J, \tag{11}$$

which reduces the energy by a factor of j in exchange for the duty to add up intermediate results at the end. It is possible that clever circuit design could yield $J = 0$ or nearly so.

If $j = N$, it reduces the exponent on N, leading to overall energy C N kT.

Energy efficiency of dense digital vector-matrix multiply

This section will discuss the minimum energy consumption of a digital implementation comparable to Figure 3B. Theory on the low energy limits of computation consider logic operations while data storage and movement are not considered. A later section will consider storage structures as well.

Figure 8 shows C-language software for dot product of two N-element vectors of ints, ints in C being data words of an implementation-dependent number of bits treated as an integer. Let us assume that the elements of $\boldsymbol{v}$ and $\boldsymbol{w}$ are $B = \log_2 L$ bits of precision. If so, the variable holding the sum should be $2B$ bits. Bit field specifications are shown in red (although not valid C syntax).

```
int v[N]:B;
unsigned w[N]:B;
int sum:2*B = 0;
for (int i = 0; i < N; i++)
    sum += v[i] * w[i];
```

Figure 8: Software dot product

Each B-bit multiplier is assumed to require B^2 gated one-bit full adders, each comprising about a half-dozen gates (there are different multiplier and adder designs). If we presuppose a 3× overhead for the additions and other control functions, this leads to energy consumption for a digital dot product of

$$E_{digital} = 3 \log_2^2(L)\, N\, E_{fulladd}, \qquad (12)$$

where $E_{fulladd}$ is the energy of a full adder. Example energies of full adders are given in Table 1. The definition of p_{error} is used from earlier in the document.

Table 1: Full adder energy. Top two entries from [Nikonov 13]

	Energy/32-bit Adder Fig 47	$E_{fulladd}$ units of kT
CMOS HP	3 fJ	22,000 kT
HomJTFET	.15 fJ	1100 kT
Thermal limit $\ln(1/p_{error})$ kT/gate		About 9 gates: $9 \ln(1/p_{error})$ kT

At the thermal limit used for the analog example, this yields

$$E_{digital} = 27 \ln(1/p_{error}) \log_2^2(L)\, N\, \mathrm{kT}. \qquad (13)$$

Many learning methods have been proposed for memristor and other analog synapse types, but idealized devices are sufficient for this document. Specifically, the ultimate limits of energy efficiency will not appear for many years, by which time device behaviors will be more ideal.

Comparison discussion

The energies for matrix-vector multiplication summarized in Table 2 are notable for both similarities and differences.

Table 2: Energy for $N \times M$ dense vector-matrix multiply at thermal noise limit

Approach	Energy per vector-matrix multiply						
Analog full parallel	1/24	$\ln(1/p_{error})$	L^2	N^2	M	kT	
Analog time sequential	1/24	$\ln(1/p_{error})$	L^2	N	M	kT	+addition
Digital	27	$\ln(1/p_{error})$	$\log_2^2(L)$	N	M	kT	

The most profound similarity is that the limits of all options can be expressed in units of kT. Radio signals are easily attenuated to kT levels of lower by moving the receiver further away from the transmitter. This yields audible static in many cases. Computer energy is well known to scale down with Moore's Law, with kT being a reasonable ending point of that scaling. However, analog computer circuits are unusual enough that most people do not have an experience base for their scaling over time. This analysis shows the limits of analog computing are the same as everything else, which is perhaps unexpected but not unreasonable.

The results are in Table 2 are close but not the same, which has implications for Turon.

Consider the last two rows of Table 2. Ignoring the energy of the last addition, the two energy expressions will be equal when $L \approx 190$ or about a $B \approx 8$-bit numbers. It is reassuring that these numbers are close in magnitude, but it would be unwise to develop performance expectations based on these numbers. In the opinion of the authors, not being close would probably mean we made a mistake. However, it took microelectronics 70 years and perhaps a trillion dollars to get CMOS to its current level of around 10,000× the physical limit. Analog computing has had much less investment.

The results can also be viewed as vastly different. Due to the scaling being different, Figure 6 shows that the different circuit types will be applicable in different regions.

We claim Table 2 supports the novel proposition behind Turon, which is that the analog neural network could simulate CMOS at similar energy levels to what CMOS could do on its own. The argument is that the digital circuits in Figure 2 and Figure 3D use L=2 and N in the range, say, 4-10. This is a region where the analog approach could have an advantage. However, this could change with sparsity.

Adiabatic memory

It is possible to be even more energy efficient than the above analysis suggests by recycling some of the energy expended to drive access lines to memory. Typical arrays operate by charging one row (of capacitance *C*) at a time to some voltage *V*. When access moves to a different row, the previous line is discharged to ground, the energy is dissipated as heat, and more energy is drawn from the power supply to charge a different row. The charging and discharging cycle ultimately consumes CV^2 energy. However, the adiabatic circuit in Figure 12 (derived from [Karakiewicz 12]) uses a different approach. A bi-directional switch connects an energy storage unit to one row at a time. To switch rows, the CV^2 energy is withdrawn from the row and put in the energy storage unit. The switch then shifts to a different row and the energy in the storage unit is put into the next row. With good technology, the losses will be a small fraction of CV^2.

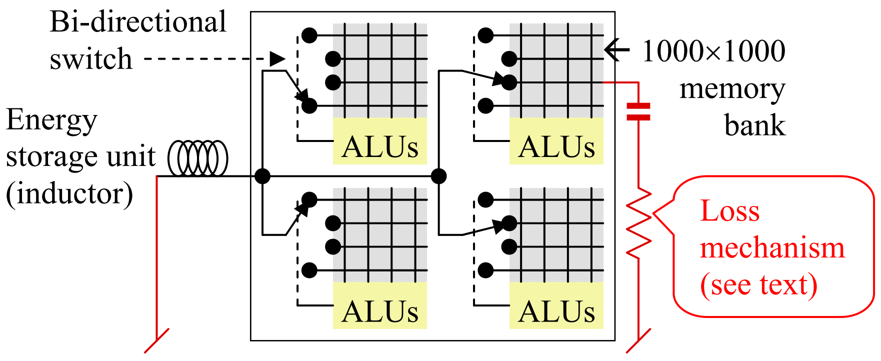


Figure 9: Adiabatic clocking for four PIMS replication units

If the energy storage unit is an inductor, a “tank” circuit is formed between the inductor and the collective capacitance of one row per memory bank. The tank circuit will oscillate sinusoidally. The bi-directional switch can change to a different row without affecting the oscillation, as long as the change occurs at a point in the cycle where there is no current flowing.

The circuit in Figure 12 can have substantially lower losses than a conventional memory. When viewed as a tank circuit, the circuit in Figure 12 can be characterized by a quality factor or “Q” in standard electrical engineering terminology. (Q can be viewed as the number of oscillations after which the energy is reduced to $1/e$ of the original value.) In a key demonstrated circuit [Karakiewicz 12], the Q value was about 85, which is equivalent to the fraction 84/85 of the energy in one row being recycled to the next.

The adiabatic circuitry in Figure 12 is applicable to both analog and digital memory. In the case of a memristor crossbar, it can be applied to both rows and columns. In the case of digital memory, it applies only to the rows.

Learning

The preceding section dealt only with a computer performing its programmed or otherwise intended function, with this section dealing with the process or specifying that function. Specification occurs at two levels. At the higher level, the resources are partitioned into regions like in Figure 2. This is almost identical mathematically to treating the resources as a one giant matrix and defining a sparsity pattern for it, a similarity that will be exploited shortly. The lower level trains the synapses within each region. Analog and digital implementations will have different properties at each level.

The analog memristor crossbar in Figure 3B has good and bad attributes with respect to learning. For Figure 3B as shown, the energy efficiency limit is very high. As the discussion below shows, energy is consumed only when and to the extent that a synapse changes. The fraction of synapses changing at any given instant is very small, and that fraction is effectively a multiplier for energy consumption. On the other hand, the structure in Figure 3B is inflexible. It is not possible to change the manufactured array structure into an array of different dimensions or into a sparse matrix.

The digital system in Figure 3C can accommodate sparsity at the expense of complexity and somewhat larger energy consumption. Figure 3C would not be very energy efficient if viewed as digital hardware for just dense matrix in Figure 3B. Digital electronics would evaluate mathematical formulae for the amount of synapse change in each learning cycle, most of the evaluations consuming energy but specifying zero change to a synapse. In other words, the dense matrix approach would not take advantage of the small fraction of synapses being updated each cycle. However, digital Turon implementations are expected to use sparse matrix computer algorithms that organize the activity so formulae are only evaluated for the synapses likely to have nonzero changes. While there is overhead in the management of lists and pointers, sparse matrix approaches often have a large net beneficial effect. The digital system in Figure 3C would need to alter the sparsity structure on the fly, which is feasible but adds complexity.

These ideas will be further developed below.

Analog Learning

The delta rule is a key operation in several neural learning algorithms, including classic back propagation. It updates the weight matrix ***W*** by the outer product of the input ***I*** and error ***E*** vectors:

$$\boldsymbol{W} \mathrel{+}= \alpha \boldsymbol{E}\boldsymbol{I}^{T}, \tag{14}$$

where α is the learning rate. The definition of ***E*** depends on the specific learning algorithm. The use of rows and columns here is arbitrary, that is, the transpose of this system is equally valid. We developed a system where the weight matrix ***W*** is defined by the memristor conductances in an array such as shown in Figure 3B.

Ideally we want to update all of ***W*** in a single pulse event, where the row and column voltages are carefully selected so that the voltage difference across each element is just the right amount to shift its resistance by the correct amount. If all the elements had the same symmetric exponential response curve $\mathrm{d}w/\mathrm{d}t=\exp(V)$, this would be possible. Specifically, the row and column voltages could be chosen to be the logarithms of their numeric values, such that $\mathrm{d}W_{jk}/\mathrm{d}t = \exp(\log r_j + \log c_k) = r_j c_k$. In practice we would also need to add an offset voltage to overcome the programming threshold, but this does not change the argument. This method is described in Appendix III.

Unfortunately, we show below that real devices do not cooperate with this scheme. There is variability in manufacturing and their response can be highly asymmetric. A number of alternate approaches have been introduced by various researchers. Several of these amount to varying the duty cycle during a write, such that the overlap of column and row pulses produce the desired amount of shift over a period of time:

- Vary voltage on columns and duty cycle on rows [Sapan Agarwal, unpublished].
- One long pulse on columns and series of shorter pulses on rows [Kadetotad 15].
- Random pulses on both rows and columns with approximate duty cycle [Merkel 14].
- Spike Timing Dependent Plasticity (STDP) [Snider 08].

Another approach involves delivering minimal (threshold level) voltage of the appropriate sign to the rows and columns, and relying on multiple training cycles to produce sufficient movement [Hu 14]. Provided that any single programming pulse rarely or on average does not exceed the amount of change needed, this method is consistent and will converge.

Memristors exhibit both read and write noise. (Some of this write noise can be explained as hidden state, as discussed in the *Memristor* section below.) It is important to choose a network configuration and learning method that can work within the precision limits of the available devices. For example, it has been shown theoretically that a minimum of about 12 bits is needed for backprop [Holt 93]. The specific number depends on the desired level of convergence.

Digital learning

The discussion below will discuss digital implementation of Turon initially as a machine that could simulate the neural network in Figure 3B, both for learning and execution. As the discussion proceeds, features will be added that accommodate sparse matrices. By the end of the discussion, the digital implementation will be able to accommodate matrices where the sparsity pattern changes on the fly. This would contrast with the analog implementation that require remanufacturing of the chip to change the sparsity pattern.

The exposition will start with a dense vector-matrix multiply in the form of dataflow [Dennis 80] and systolic arrays [Kung 79]. We start with Figure 10 showing a 2D systolic array rotated clockwise 45°, rotated so data flow direction corresponds to other diagrams in this document. The original systolic array in Figure 10 had a synchronous or systolic clock, which we will modify shortly. Numbers flowed according to the arrows, one step per clock. The blocks labeled “DPU” perform an algorithm-specific arithmetic operation on input data and one or a few locally stored numbers.

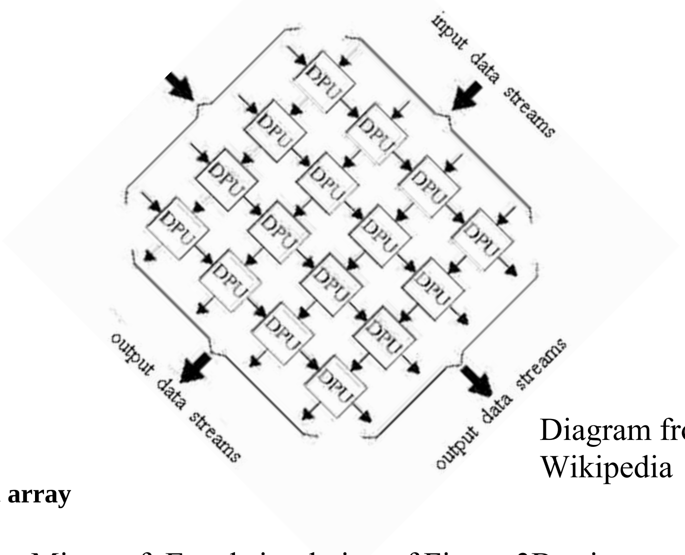
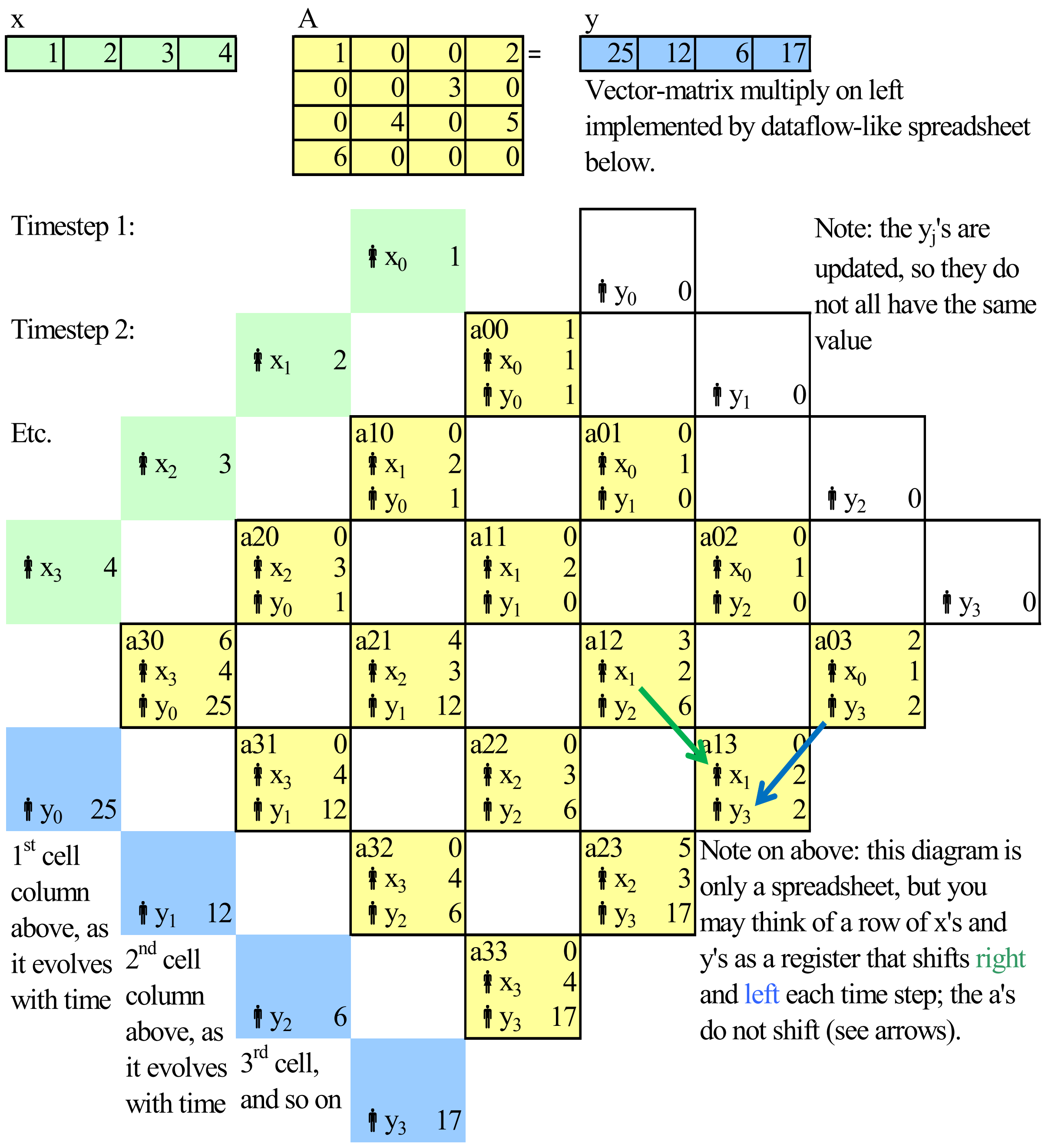


Diagram from Wikipedia

Figure 10: Systolic array

Figure 11 shows a Microsoft Excel simulation of Figure 3B using methods similar to Figure 10 – although using a 4×4 matrix. The top of Figure 11 shows the matrix equation $\boldsymbol{xA} = \boldsymbol{y}$ with the bottom of Figure 11 simulating the calculation of $\boldsymbol{y}$ using the systolic array in Figure 10. However, Figure 11 uses Excel equation dependencies as opposed to using clock cycles like a systolic array. Figure 11 originated as an Excel spreadsheet embedded in a Word document, so the reader may be able to access the formulae that do the matrix calculation if this document is in a suitable form or the reader can access the associated material.



Vector-matrix multiply on left implemented by dataflow-like spreadsheet below.

Note: the y_j's are updated, so they do not all have the same value

Note on above: this diagram is only a spreadsheet, but you may think of a row of x's and y's as a register that shifts right and left each time step; the a's do not shift (see arrows).

Figure 11: Diagram of the spreadsheet for 4×4 vector-matrix multiplication

A key property of Turon will be derived from the fact that the spreadsheet in Figure 11 can be easily changed from performing vector-matrix product to the delta rule. Figure 11 comprises a data flow layout plus operations in cells. Changing a single formula (albeit one that is repeated in each of the yellow cells) will change the function while retaining the data flow layout. The data flow layout captures the sparsity pattern of the matrix.

For the vector-matrix multiply illustrated, the cell at point of the green and blue arrows (matrix element a_{13}) contains a formula that updates y_3 by a multiply and add, specifically "=P21+N22*L20" corresponding mathematically to $y_3' = y_3 + a_{13} * x_1$.

However, the expression "N22 + α * L20 * P21" would update the value for the a_{13}, specifically computing "$a_{13}' = a_{13} + \alpha * x_1 * y_3$." α is in the role of a learning rate and could be replaced by, say, .01 or the contents of a cell. This expression would have to appear in all the yellow squares, albeit with the Excel coordinates changed so they represent the same relative positions.

However, the desire is to perform the same functions on sparse matrices. Users familiar with spreadsheets will understand that cells can be rearranged without changing the underlying mathematical calculation. So let us convert Figure 11 to an efficient form for sparse matrix calculation that we call an "operational layout." This requires two steps:

- Eliminate the formulae in each yellow square where the matrix element is zero. To maintain consistency, a formula elsewhere that references an x or y value that is deleted must be adjusted to use the x or y value in the formula that was deleted.

- Use a 2D graph layout algorithm to create a compressed or operational layout. The graph layout algorithm must be oriented so that all graph arcs point downward.

This will produce an operational layout like Figure 12, which the reader will see produces the same output value.

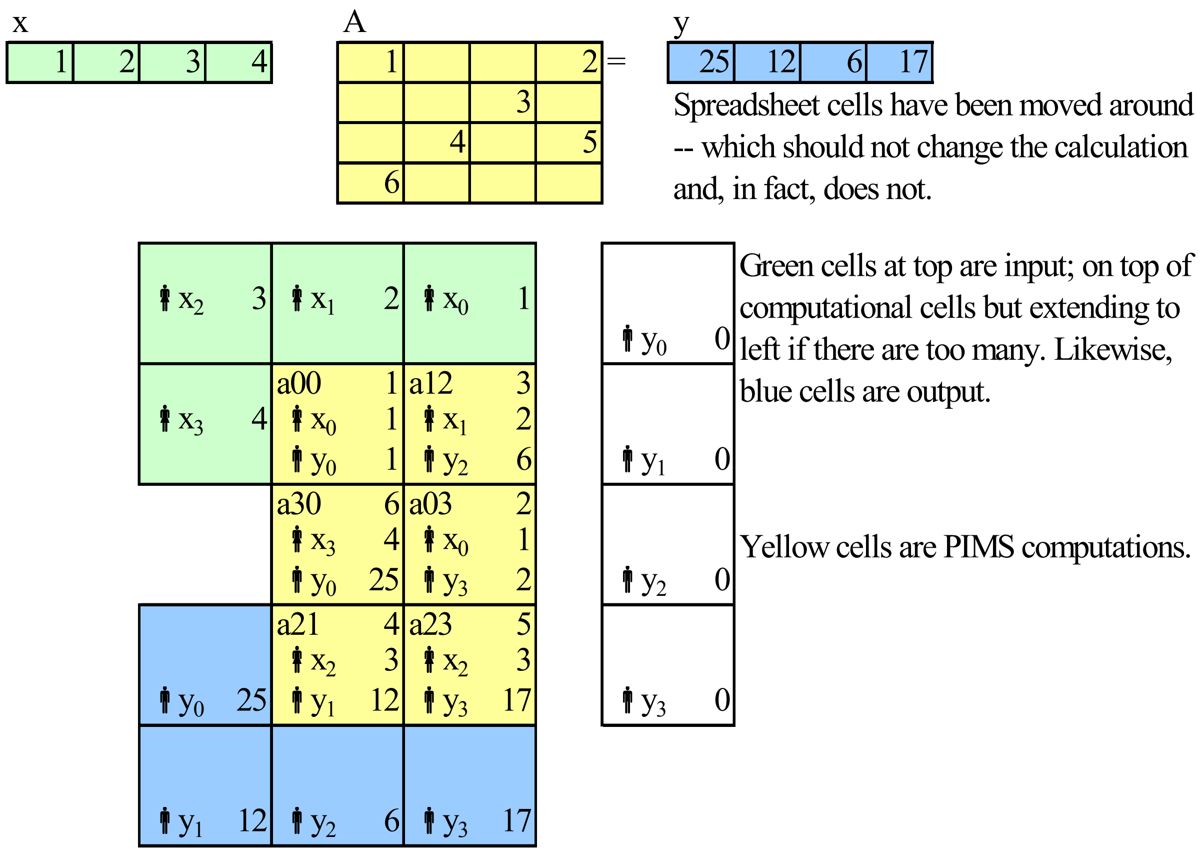

x
1 2 3 4
A
1 2
3
4 5
6
=
y
25 12 6 17
Spreadsheet cells have been moved around -- which should not change the calculation and, in fact, does not.
x2 3
x1 2
x0 1
x3 4
a00 1
x0 1
y0 1
a12 3
x1 2
y2 6
a30 6
x3 4
y0 25
a03 2
x0 1
y3 2
y0 25
a21 4
x2 3
y1 12
a23 5
x2 3
y3 17
y1 12
y2 6
y3 17
y0 0
y1 0
y2 0
y3 0
Green cells at top are input; on top of computational cells but extending to left if there are too many. Likewise, blue cells are output.
Yellow cells are PIMS computations.

Figure 12: Operational layout corresponding to Figure 11

An implementation of Turon can be constructed by loading the operational layout in Figure 12 in the memory of Figure 3C, as illustrated in Figure 13. Both the numeric values and information in the formulae of the Excel spreadsheet in Figure 12 would be loaded into the adiabatic memory, preserving the layout in terms of rows and columns. Instead of the Excel spreadsheet evaluating the dependencies, the adiabatic memory would be accessed in top to bottom order. This would send the numeric values and formulae to the ALUs in an order that could be used for computation. (This effectively changes the vertical dimension in Figure 12 into the time sequence of row access in Figure 13.) Since the original spreadsheet had all data flowing downwards, the ALUs will receive data in an order that can be processed – as long as they can shift values between themselves leftward and rightward. Graph layout algorithms attempt to minimize total arc length. Minimizing arc length will reduce the amount of ALU to ALU communications.

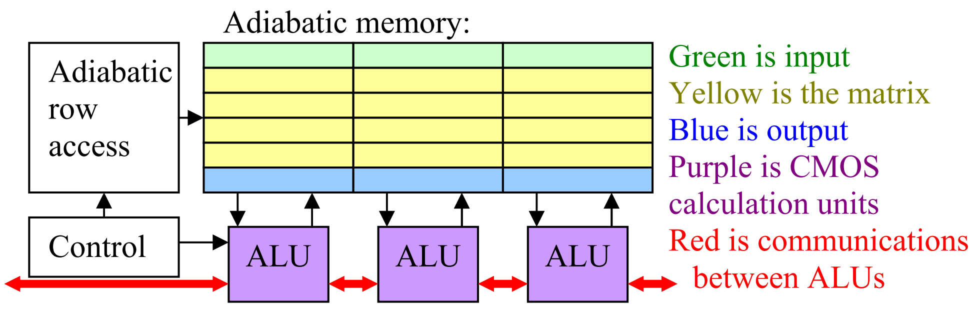


Figure 13: Turon digital architecture option

The approach described above can perform both vector-matrix multiply and delta rule update with the same sparsity pattern (although the layouts illustrated in Figure 11 and Figure 12 are not sufficient to illustrate this point).

Digital analog summary

Digital and analog implementation approaches differ in terms of flexibility and scalability. This suggests a future user should analyze the problem at hand and then use the least complex system that will solve the problem.

The approach using an analog crossbar is simple, elegant, and could be very energy efficient. However, the crossbar should be small because the energy consumption has an N^2 term whereas a digital approach has just an N term. Furthermore, the crossbar is not easy changed to a different structure. While it is certainly possible to embed any structure in a large crossbar by making entries zero, this is inefficient. Creating a different structure is also possible, but it would generally require remanufacturing the chip.

The digital approach addresses the limitations of the analog system, yet would lead down a path of increasingly sophisticated systems for addressing a more and more diverse problems – almost certainly with decreasing energy efficiency. Specifically, the energy consumption of digital dot product is lower when the vectors are long – although analog approaches should be more energy efficient when the vectors are short. Furthermore, a digital implementation could represent a sparsity or communication pattern as what we call an operational layout that is stored in memory. A chip could be repurposed to a different pattern by changing memory – which is a lot less cumbersome than redesigning the chip.

Memristor characterization

Sandia performed experiments to test the readiness of TaO_x memristors for Turon. While we found memristors were not ideal enough for neural applications, we made some advances in the technology for characterizing memristors.

Experiments performed on TaO_x memristors reveal that the state space is characterized by more than resistance. Millions of memristor tests were analyzed to assess the resistance change due to a probe pulse in a narrow voltage range. The amount of resistance change was found to depend on resistance as expected, but it was additionally found to depend on the history of how the memristor got to the initial resistance state. Figure 14 illustrates the key result: Each dot is a test colored by the memristor's history prior to the test. The chart would have a uniform color when viewed at a distance if memristors were characterized only by resistance, but there is a conspicuous blue-green area. This implies that a memristor's state can be expressed as $\{R, \text{extra state}\}$. The extra state is invisible to a resistance measurement but can be revealed by essentially applying a probe pulse and seeing how much change in resistance results.

A novel experimental protocol was used: A special memristor test fixture was constructed that applied a continuous stream of random pulses to a memristor, collecting over 9M samples in the data set use for this document. The resulting 300+ megabyte file was "data mined" in Matlab to extract a subset of the random sequences that met experimental criteria.

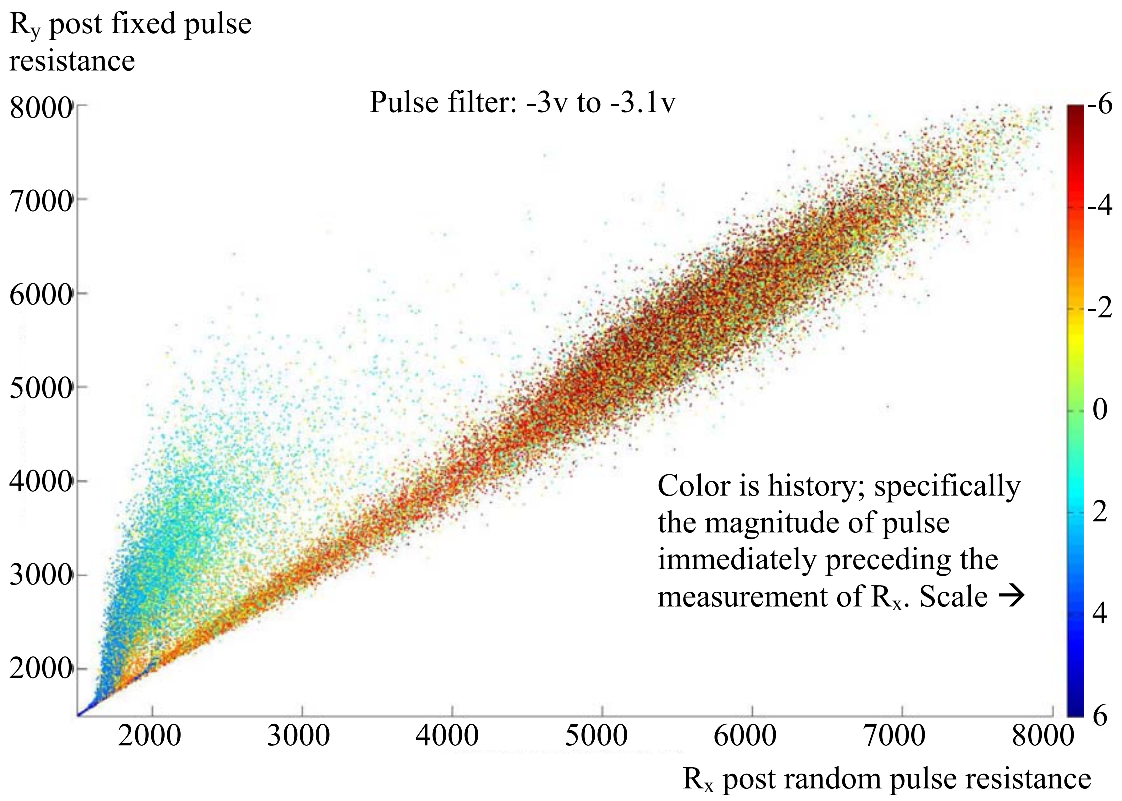


Figure 14: Memristor resistance change test due to a -3v pulse

Experimental method

Prior to this project, experimental technique involved applying a voltage or current sequence or waveform and observing the change in resistance. However, we found the process unreliable. Applying a voltage V_1 to a memristor with resistance R_1 does not always change the memristor to the same resistance R_2. This makes it impossible in general to create a test that moves a memristor through a resistance sequence R_1, R_2, R_3.

The new experimental method involves executing a very long sequence comprising (a) applying a random voltage pulse and then (b) measuring the resistance. The $\{V_k, R_k\}$ values are saved in a computer file. Experiments are performed by loading the data file into Matlab and doing what is essentially "data mining." With this method we can find resistance sequences like R_1, R_2, R_3 (to some resistance tolerance) within the random data set and then do statistical analyses on the voltages V_1, V_2, V_3 that produce the resistance sequence.

The experimental apparatus is shown in Figure 15. It comprises a custom PC board produced by a Sandia partner company. The board illustrated contains A-D and D-A converters to drive and sense memristors in a memristor array-containing chip on the left in a standard package. The bottom side of the board contains a PIC microcontroller on a mezzanine board, which we programmed in C to generate the random voltage sequence alternating with resistance measurements and transmit the data to a computer via a USB cable.

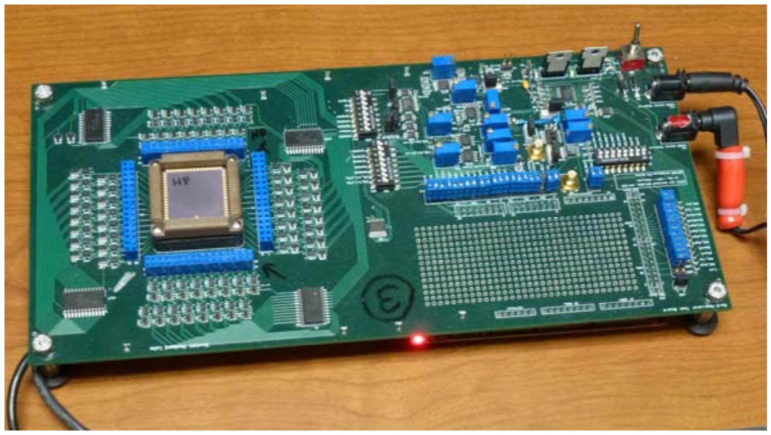


Figure 15: Experimental apparatus

The test apparatus has the ability to control the duration of voltage pulses. However, all pulses are 2 μS in this paper.

The voltage sequence is highly random, but we found a need to control distribution function. We use a software random number generator on the PIC microcontroller to generate a uniform distribution of voltages within a range. The range varies with the memristor's current measured resistance in accordance with the distribution in Figure 16. The distribution function has been chosen to avoid damaging the memristor and also collecting samples preferentially in the range we want to analyze. Fine details of the distribution function are not very important because random samples will later be selected by the data mining process.

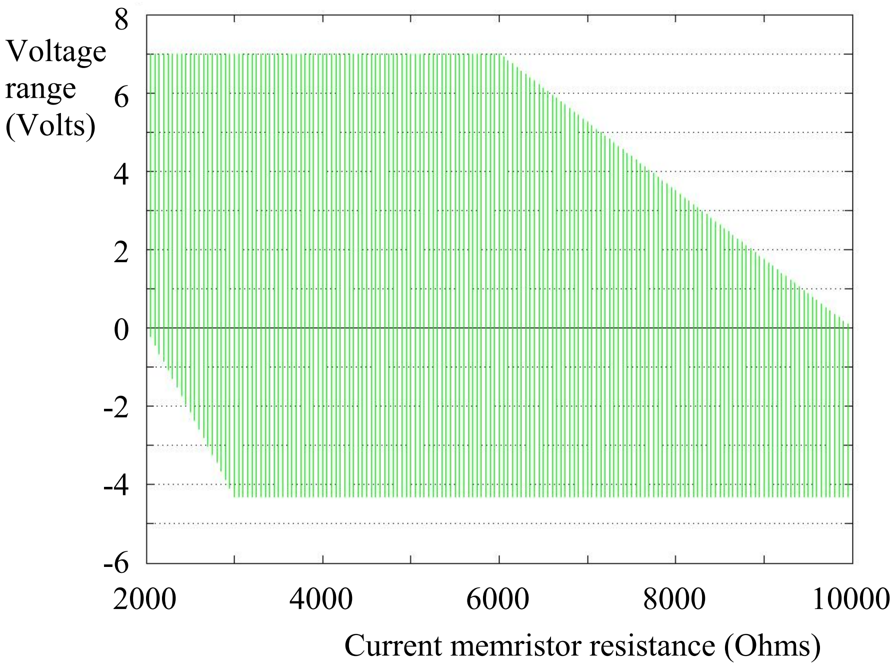


Figure 16: Random pulse distribution

For this document, the experimental apparatus was applied to a specific memristor and 9.6 million samples were collected in somewhat over 24 hours, yielding a 300+ megabyte file. We did a manual inspection of the data file and concluded that the memristor became around 20% less responsive over the course of the testing, which we attribute to wear out. The test protocol has been applied to other memristors, including memristors from different fabs (but this paper discusses results from only one memristor).

Results

Let us first discuss how Figure 14 is incompatible with the idea that a memristor's state is entirely captured by resistance. Figure 14 plots the change in resistance due to a pulse in the range of -3v…-3.1v (we will subsequently call this a -3v pulse), selected from a long sequence of random voltage pulses. Each dot represents a test. Memristors had been subjected to a random voltage sequence, with the sequence ending in a random voltage generated in accordance with Figure 16. The resistance was then measured and designated R_x. If a memristor's state is indeed captured entirely by resistance, the history of the how the memristor came to state R_x would be irrelevant; more on this point below. The memristor was the subjected to a -3v pulse and measured again and the resistance designated R_y. A dot was placed at (x, y) coordinates corresponding to (R_x, R_y).

The plot shows two regions, that can be identified subjectively as a "galaxy" viewed edge on and a "wisp."

1. The galaxy structure represents tests where the resistance was stable, which we will designate the "**-eager**" state (meaning the memristor was not eager to change resistance). A brief look at the galaxy structure reveals it is a noisy rendition of the identity function; the change is resistance is about zero on average.

2. The wisp structure shows a memristor in a state that is more easily programmed by the -3v pulse, which we will designate the "**eager**" state. The -3v pulse will increase resistance by 50% or so.

Each dot is drawn in a color that represents something about the history of how the memristor got into the R_x state. The color is specifically the voltage of the random pulse applied to the memristor before the test, thus representing the immediate history but not the long-term history. If the memristor's state were entirely represented by resistance, the plot should have a nearly uniform color. (Dots are colored by a software random number generator programmed to the distribution in Figure 16, which the human eye will blend into a nearly uniform color.) However, the wisp structure has a plainly evident blue-green color corresponding to a random pulse of 2v…4v.

The broad conclusion from Figure 14 is that a memristor can be in a given resistance state with different internal physical configurations, although a discussion of the physical nature of a memristor's state is out of scope of this document by the choice of the authors. These configurations reveal themselves in this experiment by applying a -3v "probe" pulse and seeing how much the resistance changes. The probe pulse is a "destructive read" in at least some instances.

We can describe some of the behavior of the extra state in a memristor based on Figure 14: A memristor enters the **eager** state when it drops sharply to a low resistance. A subsequent negative pulse will be cause a large rise in resistance. However, if the memristor wanders up and down between resistances in a narrow range, it will be in the more stable **–eager** state.

Emulation

We used the data to emulate a memristor in a non-parametric manner. One option is to find the nearest neighbor in *(R_1, 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 20kΩ in 100Ω increments. We distributed each sample to its four nearest bins in *(R_1, V)* space with bilinear weighting. For each sample we calculated $\Delta = R_2 - R_1$, and for each bin summed weight, $weight \times \Delta$ and $weight \times \Delta^2$. From these we calculated the mean and standard deviation of Δ for each bin.

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 ±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 measurement board (Figure 15). 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 17A shows the resulting table.

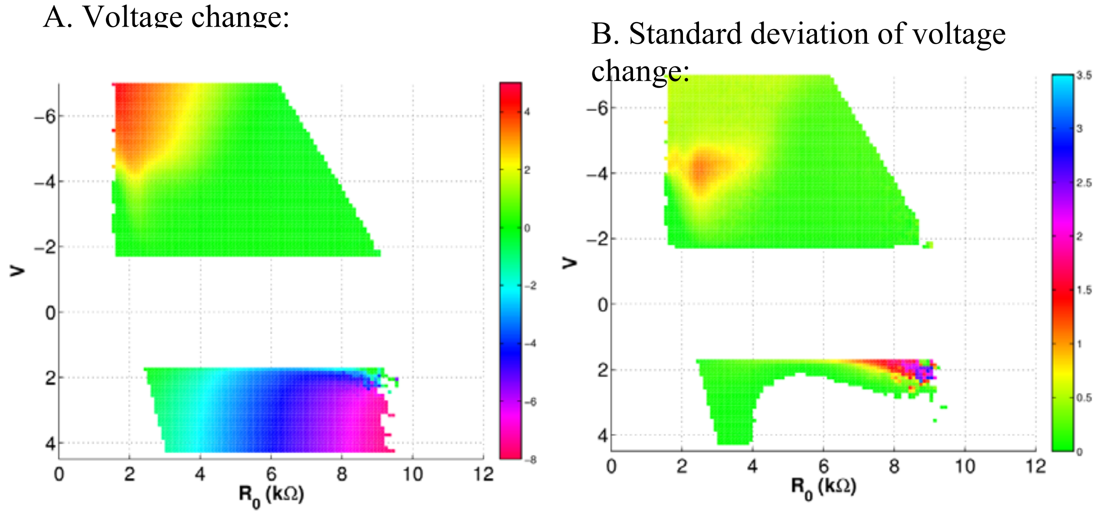


Figure 17: Measured memristor voltage change and standard deviation

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 17B shows the resulting table. Note that these two tables ($\mu\Delta$, $\sigma\Delta$) do not fully capture the behavior of the device, because they treat the hidden state described above as noise. However, this is sufficient for the experiments described next.

The Δ (resistance change) values showed some noteworthy patterns. Unlike the typical model of a memristor as a symmetric pair of exponentials, we found significant asymmetry. Figure 18 plots a row of the $\mu\Delta$ table at 4kΩ, 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 and cause an temperature increase due to E^2/R heating. A rising temperature increases the rate of resistance change, creating a feedback loop that drives resistance down even faster.

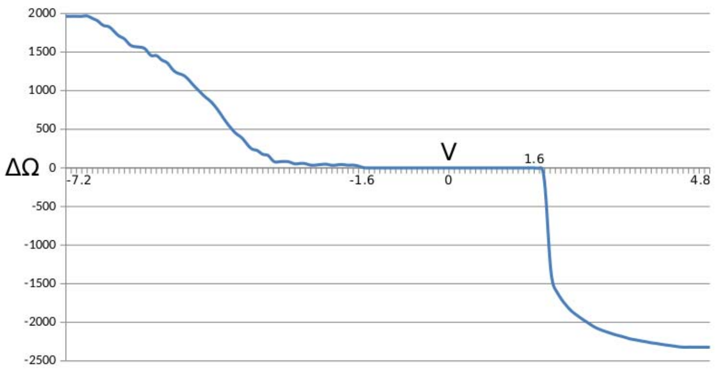


Figure 18: Real albeit non-ideal memristor resistance change with voltage

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_1. Write noise for negative pulses was about 0.5% of R_1. With the finished table, we simulated a pulse to the memristor by retrieving the four cells closest to (R_1,V) and interpolating between them. The new resistance resulting from the pulse was $R_2=R_1+\Delta+G(\sigma_W)\times N_W$, where $G(\cdot)$ is the Gaussian distribution with zero mean, σ_W is the interpolated standard deviation, and N_W is an optional scaling for write noise. Similarly, we simulated a noisy read as $R_1+G(\sigma_R)\times N_R$, where N_R is an optional scaling for read noise.

This model approximates the function $f(R_1,V)\rightarrow R_2$ with a piecewise linear (ruled) surface. In rapidly changing regions 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.

Backprop Learning

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: $\boldsymbol{O}=f(\boldsymbol{WI})$, where $\boldsymbol{I}$ is the input vector, $\boldsymbol{W}$ is the weight matrix, $\boldsymbol{O}$ is the output vector, and typically $f(x)=1/(1+e^{1-x})$ (the logistic sigmoid function). Simple function composition expresses multiple layers: $\boldsymbol{O}=f(\boldsymbol{W}_2 f(\boldsymbol{W}_1\boldsymbol{I}))$. The Backprop algorithm is an application of gradient descent to the weights $\boldsymbol{W}_i$ where the derivatives are determined by the chain rule. Specifically:

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

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 element wise multiplication. Each layer is updated using a step size α:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\, \partial\, (\tfrac{1}{2}\boldsymbol{E}^2)/\partial\, \boldsymbol{W}. \qquad (15)$$

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 memristor crossbar (abbreviated "xbar") could be part of an electronic module that does all the work of one perceptron layer. These modules could be chained together to form an MLP. A module may be implemented in several different ways. Figure 22 shows a highly abbreviated xbar. In addition to a dense 2D array of memristors, a certain amount of peripheral circuitry would be needed to drive the devices and perform the functions described by the equations above. 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.

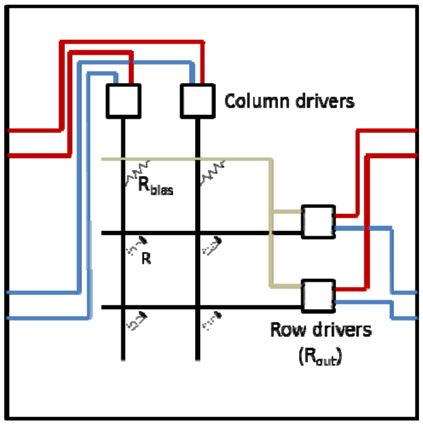


Figure 19: Xbar

The rows and columns also have driver circuits which pulse the array to program the memristors. 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:

$$w=R_{out}(1/R - 1/R_{bias}). \qquad (16)$$

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}=4kΩ, as this is roughly in the center of the cleanest data in the table. The scale factor was R_{out}=40kΩ, which gives us weights in −3.3 to 10 for resistances in 6kΩ to 2kΩ respectively. We observed that conventional MLPs trained on the problems presented here typically had weights in [−3, 3], so this configuration was sufficient.

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 digital ("float") network. In particular, they were incapable of reaching 100% accuracy on a simple Boolean mapping (one of the green blocks in Figure 2). 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 digital 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 $\alpha=0.01$, $\boldsymbol{I}=1$, and $\boldsymbol{WI}=0$ so $f(\boldsymbol{WI})=f(0)=0.25$.

Substituting these into Equation (15) gives:

$$\boldsymbol{W}\leftarrow\boldsymbol{W}+\alpha\boldsymbol{E}\circ f'(\boldsymbol{WI})\boldsymbol{I}^T$$
$$w\leftarrow w+0.01\times e\times 0.25\times 1$$
$$w\leftarrow w+0.0025\times e$$

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 [Holt 93] 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:

$$w_a - w_b = R_{out} \times (1/R_a - 1/R_b). \tag{17}$$

If we let $R_a = R_b - 100\Omega$ and $R_b = 4\text{k}\Omega$, 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 $0.0025 \times 0.01 \times 100\Omega / 0.244 \approx 0.01\Omega$. 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. We observed $\sigma_R \approx 60\Omega$, which degraded the performance in our simulations by various degrees, but often produced tolerable results.

LOTTO

In order to train the Boolean circuits in a Turon we must achieve 100% accuracy, that is, full convergence. Backprop on memristors is problematic, so we devised a random-walk learning method, whimsically named LOTTO (Lazy Optimization Through Targeted Oversampling).

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 20).

A random set of weights (exploration step) can be created in a manner similar to the random-pulse sampling method described above. 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.

In outline form, the algorithm is:

```
repeat
    {Alternate explore and exploit steps}
    if explore then
        Random (low cost) initialization
    else
        Set to best point, using -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
```

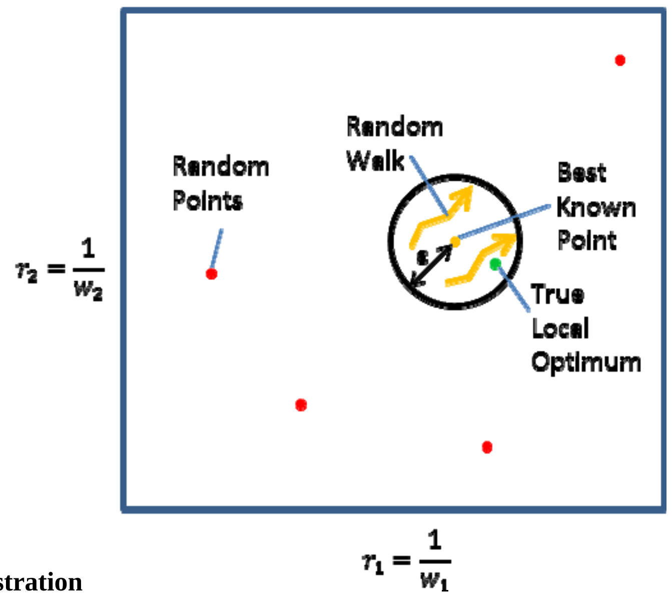


Figure 20: LOTTO illustration

Conclusions

In this document, we specified the Turon computer architecture that blends properties of Turing machine and neural networks.

In the language of computational complexity, Turon can simulate a von Neumann computer. Turon's memory can be used essentially to store gate-level descriptions of computers circuits and then execute them like an FPGA. Thus, Turon can load a von Neumann CPU (microprocessor) into a portion of its memory and use another portion of its memory as the von Neumann computer's memory. Turon can then simulate the von Neumann computer, and hence a finite Turing machine. Furthermore, multiple von Neumann computers could be loaded into different portions of Turon's memory, letting Turon simulate a parallel von Neumann computer – like a current supercomputer. This latter simulation would be entirely parallel, meaning the full speedup of the parallel von Neumann computer would be realized in its simulation by Turon.

On the other hand, Turon could load its intermixed memory/neural network resources with a pattern recognition neural network of the type used in Deep Learning. However, this form of neural network was only discussed in this document for perceptron-style neural networks with back propagation learning. This portion of the resource could be used to learn and recognize "cat faces" in Youtube videos [Le 13] (the famous example).

The document discussed the potential physical efficiency of a Turon computer based on two exemplary implementations. According to the theories of computation put forth by von Neumann and Landauer, the minimum energy for computation of this type should be multiples of kT. These theories are normally assumed to apply to digital computation, but this is not really clear. When we estimate the minimum energy for Turon, the minimum energy is a multiple of kT for both analog and digital forms – although the expressions are different in constant factors and they scale differently.

A memristor-based analog implementation was analyzed for both function and theoretical limits. The document used an idealization of memristor behavior widespread in the literature. A functional circuit for both learning and performance was presented. The document then included simulation tests of less idealized memristors learning the digital circuits that are a novel advance or Turon. The non-idealized memristors worked much less well. A theoretical limits analysis of memristor-based circuits revealed that the idealized memristor circuit has theoretical limits that can be compared to Landauer's "limit" of O(kT) per device or the Shannon-inspired digital limit of around 100 kT/operation. However, the analog memristor limit has a different scale dependence than digital circuits (more on this later).

A digital implementation based on 3D scaling of memory was discussed, yet in reference to other published works by the author. The physical limits based on the same Landauer or Shannon argument from the paragraph above were analyzed.

A comparison of maximum performance for the analog and digital approaches were made and were instructive.

An analog approach could potentially outperform the digital approach in small neural networks. However, digital neural networks scale better. This is important to the ability of Turon to simulate digital circuits. The digital circuits would be constructed from large numbers of small neural networks, each simulating a dozen or so logic gates. These neural networks would be in the size range where the efficiency of the analog implementation would dominate its poor scalability.

The digital approach has its merits as well. There are important classes of neural network (e. g. brains) that have a large amount of memory in comparison to the throughput of the logic. Existing research prototypes tend to be chips with very low power consumption (e. g. TrueNorth has just a few milliwatts of power consumption) because most of the chip area is filled with memory, leaving little space for logic. A digital Turon approach could exploit trends in 3D integration, which are more mature for memory than logic. A Turon could have a large amount of memory in the volume of its 3D structure while having logic only on a 3D face. We discussed an computer and brain scaling, showing that this 2D/3D scaling was a fairly good match in terms of polynomial exponent.

References

[Dennis 80] Dennis, Jack Bonnell. "Data flow supercomputers." *Computer* 11 (1980): 48-56.

[Herculano-Houzel 11] Herculano-Houzel, Suzana. "Scaling of brain metabolism with a fixed energy budget per neuron: implications for neuronal activity, plasticity and evolution." PLoS One 6.3 (2011): e17514.

[Holt 93] 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.

[Hu 14] 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.

[Karakiewicz 12] Karakiewicz, Rafal, Roman Genov, and Gert Cauwenberghs. "1.1 TMACS/mW fine-grained stochastic resonant charge-recycling array processor." *Sensors Journal, IEEE* 12.4 (2012): 785-792.

[Kadetotad 15] D. Kadetotad, X. Zihan, A. Mohanty, C. Pai-Yu, L. Binbin, Y. Jieping, et al., "Parallel Architecture With Resistive Crosspoint Array for Dictionary Learning Acceleration," Emerging and Selected Topics in Circuits and Systems, IEEE Journal on, vol. 5, pp. 194-204, 2015.

[Killian 93] Kilian, Joe, and Hava T. Siegelmann. "On the power of sigmoid neural networks." *Proceedings of the sixth annual conference on Computational learning theory*. ACM, 1993.

[Kung 79] Kung, H., and Charles E. Leiserson. "Systolic arrays (for VLSI)." *Society for Industrial & Applied* (1979): 256.

[Landauer 61] Landauer, Rolf. "Irreversibility and heat generation in the computing process." *IBM journal of research and development* 5.3 (1961): 183-191.

[Le 13] Le, Quoc V. "Building high-level features using large scale unsupervised learning." *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013.

[Merkel 14] Cory Merkel, Dhireesha Kudithipudi, "A stochastic learning algorithm for neuromemristive systems", System-on-Chip Conference (SOCC), 2014 27th IEEE International, pp. 359-364, Sep. 2014.

[Nikonov 13] Nikonov, Dmitri E., and Ian Young. "Overview of beyond-CMOS devices and a uniform methodology for their benchmarking." *Proceedings of the IEEE* 101.12 (2013): 2498-2533.

[Shannon 48] Shannon, Claude Elwood. "A mathematical theory of communication." *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001): 3-55.

[Shinn 13] Shinn, T., and Tadao Takaoka. "Efficient graph algorithms for network analysis." *First International Conference on Resource Efficiency in Interorganizational Networks-ResEff 2013*. 2013.

[Snider 08] Greg S. Snider. "Spike-Timing-Dependent Learning in Memristive Nanodevices." 2008 IEEE/ACM International Symposium on Nanoscale Architectures. pp. 85-92.

[Walter 05] Chip Walter, Kryder's Law, Scientific American (August 2005), 293, 32-33

[Wing yy] Example inspired by http://wing.comp.nus.edu.sg/pris/ArtificialNeuralNetworks/Perceptrons.html

[Wikipedia SR] https://en.wikipedia.org/wiki/Semiring

[Wikipedia YY] http://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons

Appendix I: Energy limit of a sparse memristor network

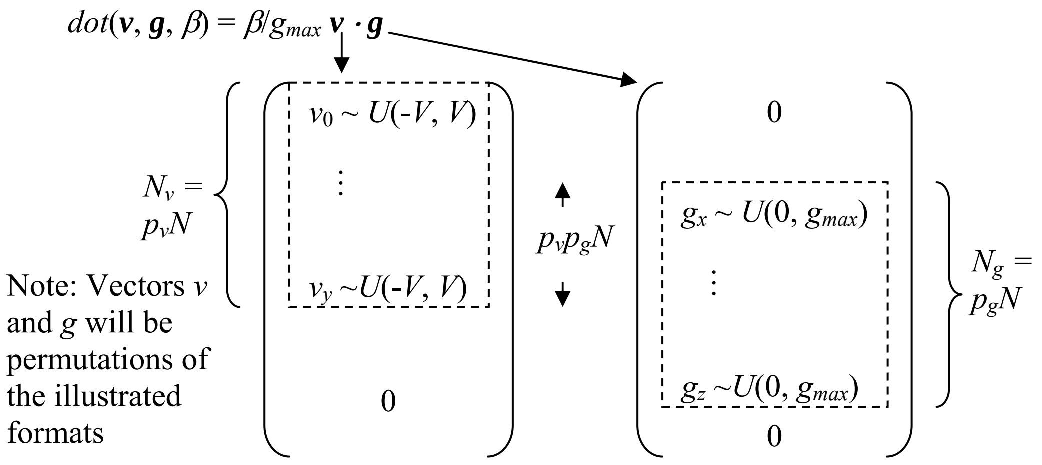
We analyze a scaled N-element dot product of two sparse vectors $\boldsymbol{v}$ and $\boldsymbol{g}$, based on a sparsity model discussed below. By scaling, we mean the result of the mathematical dot product multiplied by a scaling factor β. M such dot products become equivalent to an $N\times M$ sparse vector-matrix multiply. The voltage on each row and the synapse values are both defined to be L-level analog signals. In preparation for comparison with digital approaches, we want the answer to be correct with probability $1\text{-}p_{error}$. The strategy is to multiply using Ohm's law for the memristors and add with Kirchhoff's current law for the column conductor.

It will turn out that the energy per operation will be in units of kT, k being Boltzmann's constant and T being the absolute temperature. While this is the same form as the minimum energy for a digital computation as identified by von Neumann [cite], Shannon [cite], Landauer [cite], and others, the scaling will be different from a digital circuit. As a preview, the energy for a vector-matrix multiply will turn out to be

$$\overline{E_{vmm}}^{(B)} = \ln(1/p_{error})/24\ \beta L^2 N^2 M\,\mathrm{kT}, \qquad (18)$$

for a $N\times M$ matrix of L-level signals, and a gain factor of β. The quadratic exponents on N and L are different from digital computers, which allow interesting comparisons.

Voltages v_i driving each row are assumed to be uniformly distributed in the range $[-V, V]$ and comprise vector $\boldsymbol{v}$, as illustrated in Figure 21. (The notation "$v_i \sim U(a, b)$" means v_i is a random variable uniformly distributed between a and b.) The other vector $\boldsymbol{g}$ is defined by the state of memristors in a column and comprises conductances g_i uniformly distributed in the range $[0, g_{max}]$. (The V's will cancel algebraically in the energy consumption equation, but the fact that the V's are distributed symmetrically around zero will contribute to constant factors. The value g_{max} will cancel similarly, but becomes the definition of the numerical value 1 for the elements of $\boldsymbol{g}$.)



$dot(\mathbf{v}, \boldsymbol{g}, \beta) = \beta/g_{max}\ \mathbf{v} \cdot \boldsymbol{g}$
$N_v = p_vN$
$v_0 \sim U(-V, V)$
$v_y \sim U(-V, V)$
0
p_vp_gN
0
$g_x \sim U(0, g_{max})$
$g_z \sim U(0, g_{max})$
0
$N_g = p_gN$
Note: Vectors v and g will be permutations of the illustrated formats

Figure 21: Definition of the dot function as β/g times the dot product of v and g

The vectors have N elements and may be sparse, as illustrated in Figure 21. Exactly p_vN elements of $\boldsymbol{v}$ and p_gN elements of $\boldsymbol{g}$ are nonzero, with p_vp_gN pairs of corresponding elements both being nonzero (meaning the multiplication step in the dot product will yield a nonzero result in exactly p_vp_gN cases). While p_v and p_g may serve as probabilities in practice, in this analysis we interpret them as exact ratios.

Figure 22 includes the p_gN non-infinite resistors. Of these, p_vp_gN are driven by a voltage v_j. The remainder shown in red are modeled as connected to ground, as they represent sparse vector elements that would have zero numerical value and hence zero voltage. Elements of $\boldsymbol{v}$ in the sparse region paired with elements of $\boldsymbol{g}$ elements in the nonsparse region load the circuit electrically but should not change the result.

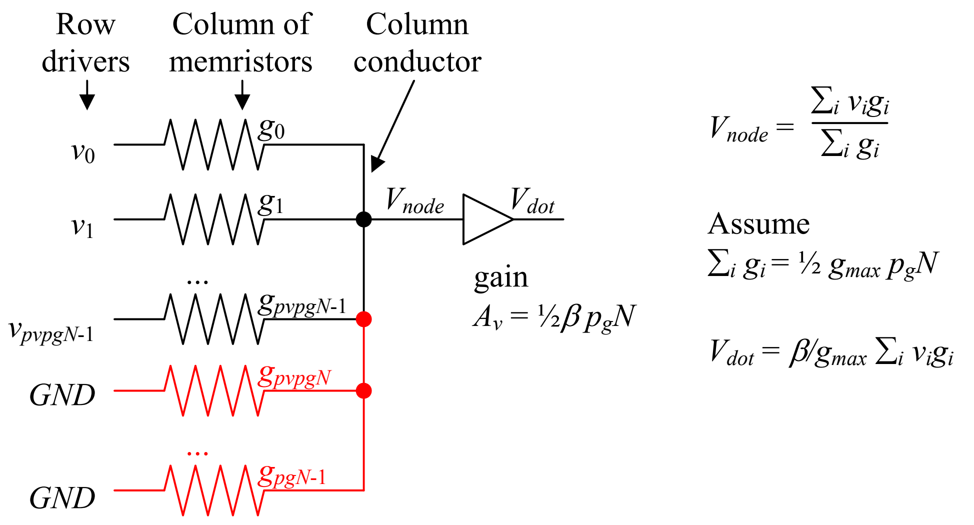


Figure 22: Circuit for analysis of sparse dot product

The network in Figure 22 through point V_{node} forms the weighted average of the input voltages instead of the dot product. However, the equations show a weighted average is mathematically equivalent to a dot product divided by the sum of the weights. We can recover the dot product by amplifying the voltage, but the gain has to be chosen carefully.

Since multi-level neural circuits feed the output of one vector-matrix multiply to the input of the next, it would be best for the output and input signals have the same physical representation and units. We use voltage. However, when the dot product multiplies voltage v_i by conductance g_i, the result is a current. Our remedy is to interpret g_i's in the range $[0, g_{max}]$ as weights in the range [0, 1].

If we assume the average memristor conductance is $½\, g_{max}$ (which is the statistical midpoint but strictly speaking implies normalization on each step), the sum of the weights will be $½\, g_{max}\, p_gN$. If the amplifier's gain is set to $A_v = ½\beta N$, the circuit will compute $V_{dot} = \beta/g_{max}\, \boldsymbol{v} \cdot \boldsymbol{g}$, which is close enough to dot product for the circumstances.

The $A_v = \frac{1}{2}\beta p_g N$ is controversial among reviewers. In subjective terms, a signal entering one input will leak backwards through the N-1 other inputs. As N grows, the amplifier must counteract with more and more gain. However, the amplifier also amplifies noise and creates other deleterious effects.

Transients will be due to Johnson-Nyquist noise in the limiting case, with the time available for transients to die out determined by system speed or clock rate. For the time being, let us assume the circuitry in Figure 7 is bandlimited to frequency f. The noise power according to the Johnson-Nyquist noise theorem will be 4kT f at the input to the amplifier. (F will cancel algebraically in the energy consumption equation.) Equating noise power to voltage^2 divided by resistance, which is equivalent to voltage^2 times the average conductance $\frac{1}{2}\, g_{max}\, p_g N$,

$$\overline{P_{noise}} = 4\ \mathrm{kT}\, f = \overline{V_{noise}}^{\,2}\ \tfrac{1}{2}\, g_{max}\, p_g N \qquad (19)$$

Which yields

$$\overline{V_{noise}} = \left[\frac{8\,kT}{p_g N} \frac{f}{g_{max}} \right]^{1/2} \qquad (20)$$

In accordance with previous discussion, the noise will be amplified before appearing on the output. However, we are interested in the peak noise transients that occur with probability p_{error}, which requires multiplying the average noise by $\ln(1/\sqrt{p_{error}})$.

$$\overline{V_{peaknoise}} = \overline{V_{noise}}\, A_v \ln(1/\sqrt{p_{error}}) = \overline{V_{noise}}\ \tfrac{1}{2}\, \beta\, p_g N \ln(1/\sqrt{p_{error}}) \qquad (21)$$

The number of resolution levels L will be the output range $2V$ divided by the peak noise voltage $V_{peaknoise}$.

$$L = \frac{2V}{V_{peaknoise}} = \left[\frac{p_g N}{8\,kT} \frac{g_{max}}{f} \right]^{1/2} \frac{4V}{\beta p_g N \ln(1/p_{error})} \qquad (22)$$

Squaring (22) yields

$$L^2 = \frac{2\,p_g N}{kT} \frac{g_{max}}{f} \frac{V^2}{\beta^2\, p_g^{\,2} N^2 \exp(2/\mathrm{p_{error}})} \qquad (23)$$

Rearranging (23) to a form that has units of energy and will be useful later

$$\frac{V^2 g_{max}}{f} = \tfrac{1}{2}\, \beta^2 L^2\, p_g N \ln(1/p_{error})\ \mathrm{kT} \qquad (24)$$

The power consumption of the circuit is addressed now. Only the energy turned into heat in a resistor is irrecoverable in this situation, so we will analyze the average power dissipated by all the resistors. We will express the power as a base value plus small-scale correction.

V_{node} moves asymptotically to zero as N increases, with the base value for power assuming $V_{node} = 0$. If V_{node} is grounded, there will be $p_v p_g N$ uniformly distributed voltages $[-V, V]$ across resistors with average conductance $g_{max}/2$. This yields the base power of $1/6\ V^2\ g_{max}$ per resistor and total power

$$\overline{P_{neuron}}^{(B)} = 1/6\ V^2\ g_{max}\ p_v p_g N \qquad \textbf{(25)}$$

We will designate the base power with the superscript $^{(B)}$ and use it in subsequent discussion of higher level functions. However, we have numerically computed the small-scale correction. Given that the v's and g's in Figure 21 have well-defined distributions, the average heat produced by the resistors in Figure 7 can be computed as

$$\overline{P_{neuron}} = P_{simulate}(p_v p_g N,\ (p_v - p_v p_g)N)\ V^2 g_{max}\ \ p_v p_g N, \qquad \textbf{(26)}$$

where $P_{simulate}(M, Z)$ is the result of a numerical computation. The authors wrote a computer program that rolls uniformly distributed random numbers in the range [-1, 1] for v's and [0, 1] for g's and computes the power dissipation as a function of the number of uniformly-distributed drive voltages $M = p_v p_g N$ and additional zero voltages due to sparseness $Z = (p_g - p_v p_g)N$ and given $V = g_{max} = 1$. A graph of this function is illustrated in Figure 6.

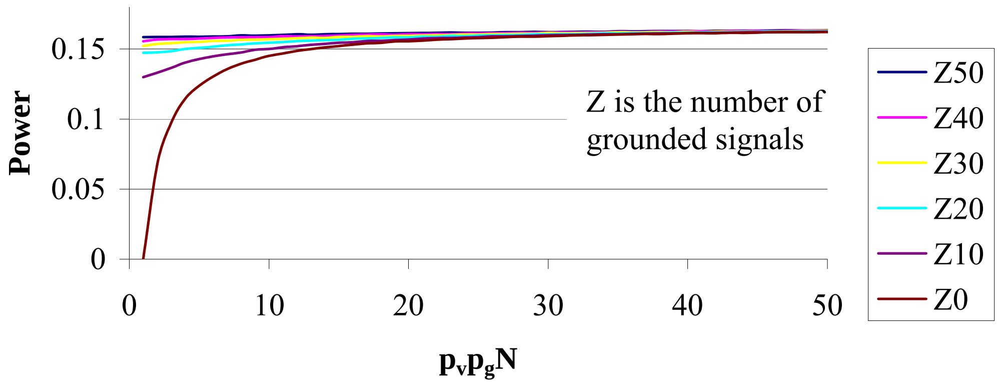


Figure 23: Computation of average power $P_{simulate}$

The curve in Figure 23 is asymptotic to 1/6, with the interesting behavior on the left. The lowest curve labeled Z0 is the power when all the applied voltages are uniformly distributed in the range $[-V, V]$ and there are no additional grounded signals applied due to sparsity. This curve shows less average power due to V_{node} shifting away from zero towards the applied voltages and reducing power. The other curves labeled ZNN include the addition of NN grounded signals applied due to sparse values in the voltage vector. Tying V_{node} to additional grounds would be expected to reduce fluctuation in V_{node} and cause the curve to approach the asymptotic value more quickly – which is what is observed.

We must now establish a connection between operating speed and the Johnson-Nyquist noise. We had previously assumed the circuitry would be limited to frequencies below *f*, but *f* has so far been just an algebraic symbol. We are now free to choose a specific value for *f* for lowest energy per operation. The Nyquist sampling theorem states that the maximum rate at which voltages could be applied to the rows would be 2*f*. This would imply the limiting case of a clock period of 1/(2*f*). In this limiting case, the energy to evaluate a neuron would be the power P_{neuron} multiplied by the clock period.

The equation below is a rearrangement of terms from (25) above, divided 2*f*.

$$\overline{E_{neuron}}^{(B)} = \frac{\overline{P_{neuron}}^{(B)}}{2f} = \frac{V^2 g_{max}}{f} \frac{p_v p_g N}{12} \qquad \textbf{(27)}$$

Substituting (24)

$$\overline{E_{neuron}}^{(B)} = ½\, \beta^2 L^2\, p_g N \ln(1/p_{error})\ \frac{p_v p_g N}{12} = 1/24\, \beta^2 L^2\, p_g{}^2 p_v N^2 \ln(1/p_{error}) \qquad \textbf{(28)}$$

The equation above is notable because V, g_{max}, and f cancel out algebraically. The equation is thus an implementation-independent representation of the minimum energy $E_{neuron}{}^{(B)}$ as a function of the nature of the problem being solved and the thermodynamic quantity kT.

There is redefinition of terms that may yield insight. Expressing $E_{neuron}{}^{(B)}$ from (28) in terms of $N_v = p_v N$ and $N_g = p_g N$ the nonzero signals in the vectors,

$$\overline{E_{neuron}}^{(B)} = \frac{\beta^2\, L^2\, N_v{}^2\, N_g \ln(1/\sqrt{p_{error}})\ \mathrm{kT}}{24\, N} \qquad \textbf{(29)}$$

In conventional computer terminology, the system will perform $p_v p_g N$ multiply operations. The energy per operation will be $E_{neuron}{}^{(B)}$ divided by $p_v p_g N$.

$$\text{Energy/op} = 1/24\ \beta^2\, L^2\, N_g \ln(1/p_{error})\ \mathrm{kT} \qquad \textbf{(30)}$$

Which tells us the energy per equivalent multiply operation is proportional to the number of nonzero elements in a column of the weight matrix. In subjective terms, this means the cost of a multiply depends on how many similarly computed products might be added up afterwards.

Further note the implication to software-based Artificial Neural Networks (ANNs), such as Deep Learning. As mentioned above, Deep Learning typically does not assume any sort of sparse coding or gain β. This would imply $\beta = p_v = p_g = 1$ and the energy per neural evaluation would be

$$\overline{E_{neuron\text{-}Deep\text{-}Learning\text{-}equivalent}}^{(B)} = 1/24\ \ L^2\, N^2 \ln(1/p_{error})\ \mathrm{kT} \qquad \textbf{(31)}$$

The above expressions are for a dot product. If we multiply by *M,* which is the number of output neurons, we get the energy of an $N \times M$ vector-matrix multiply, as may occur in software-based methods such as Deep Learning.

$$\overline{E_{vmm}}^{(B)} = 1/24\, \beta L^2 N^2 M \ln(1/p_{error})\, \mathrm{kT}, \qquad (32)$$

where E_{vmm} is the energy of a vector-matrix multiply.

Appendix II: Memristor data

While using the same 9.6 million sample data set, the software portion of the experiment was run for pulse sizes of 1.5v, 1.8v, 2v, 2.5v, 3v, 3.5v, 4v, 4.5v. The plots are include below.

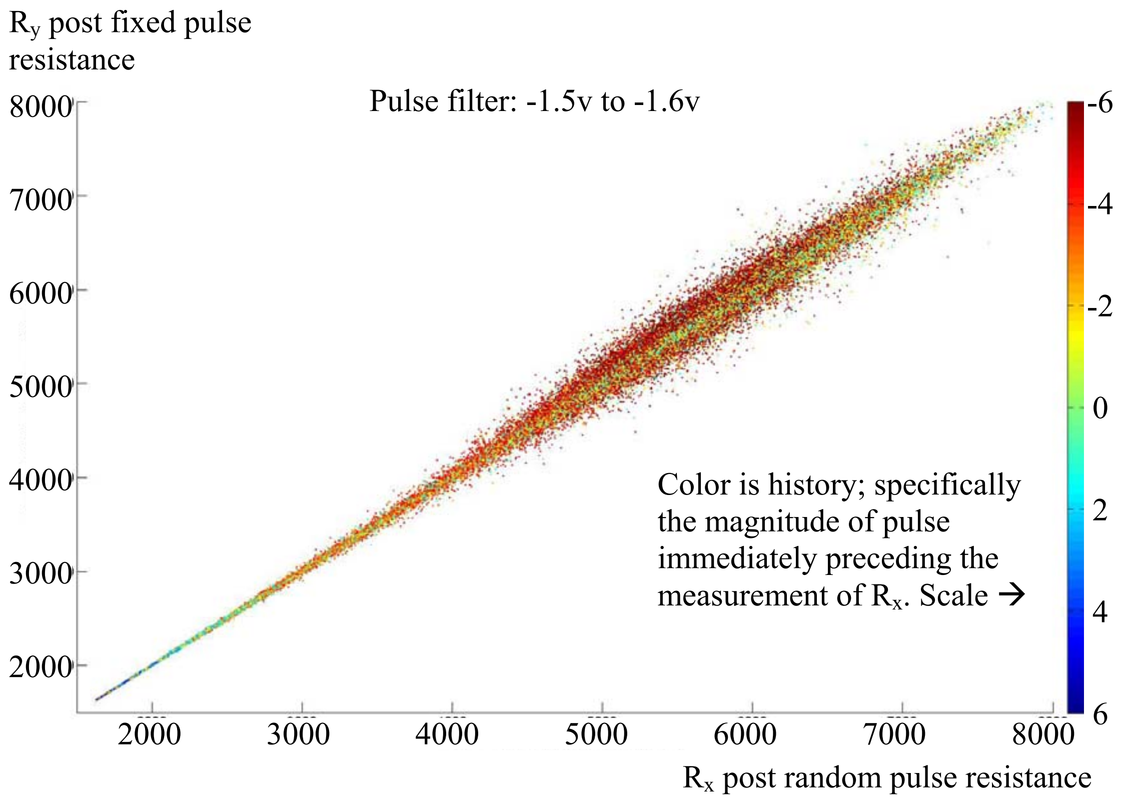


Figure 24: Memristor resistance change test due to a -1.5v pulse

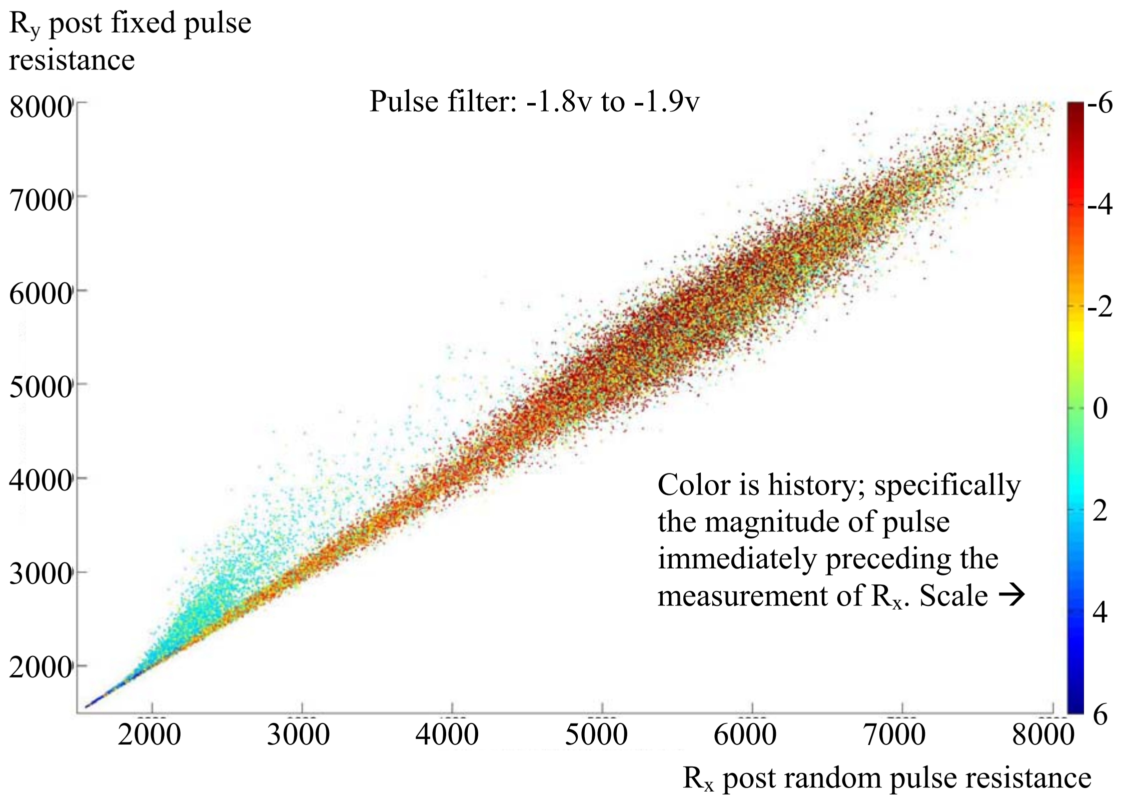


Figure 25: Memristor resistance change test due to a -1.8v pulse

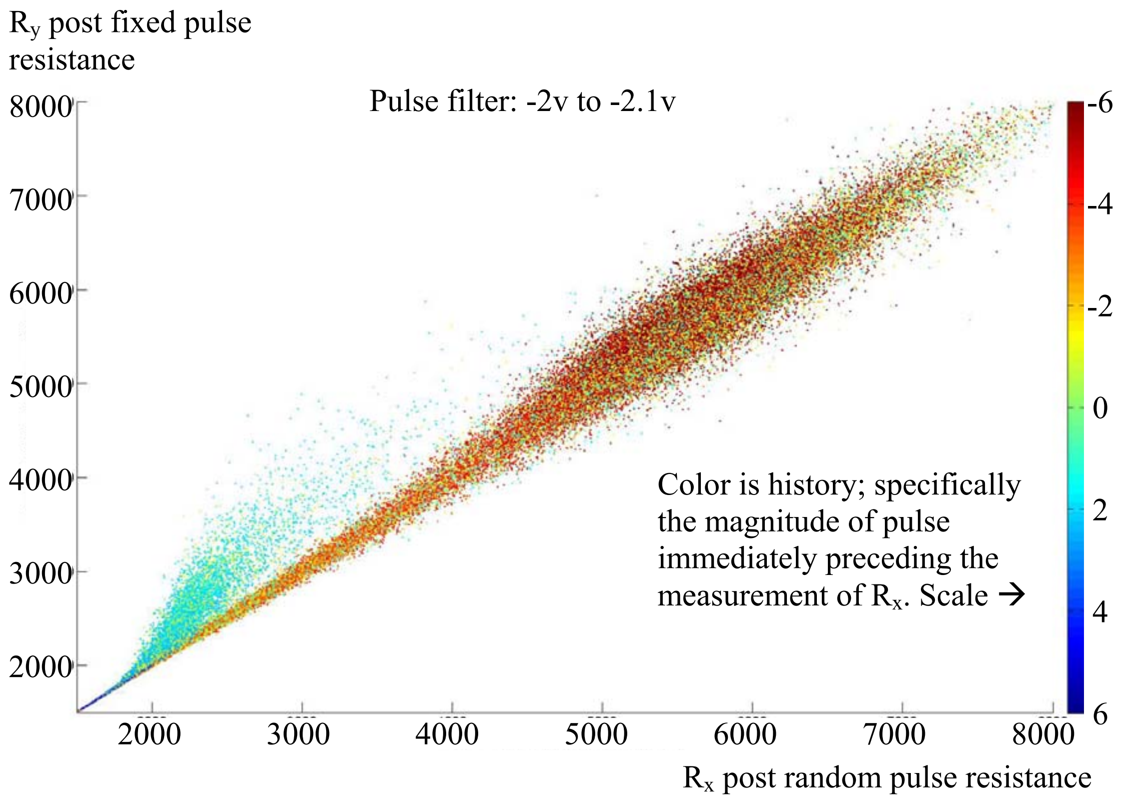


Figure 26: Memristor resistance change test due to a -2v pulse

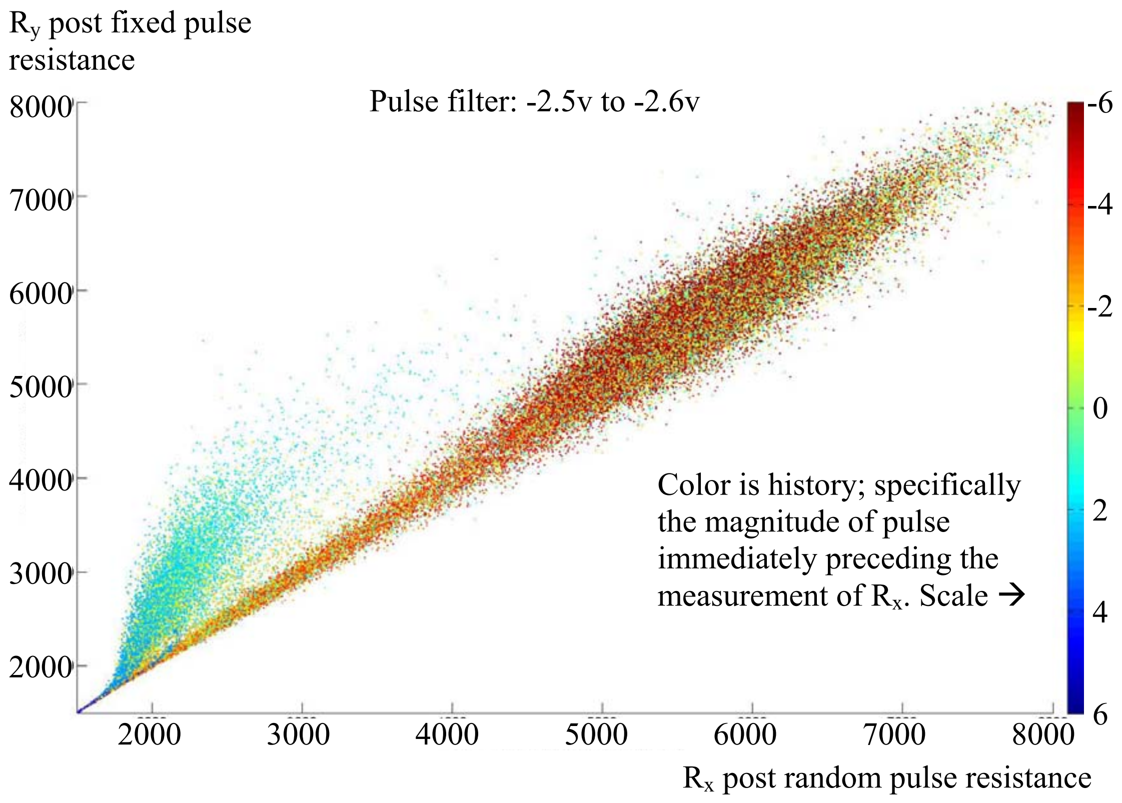


Figure 27: Memristor resistance change test due to a -2.5v pulse

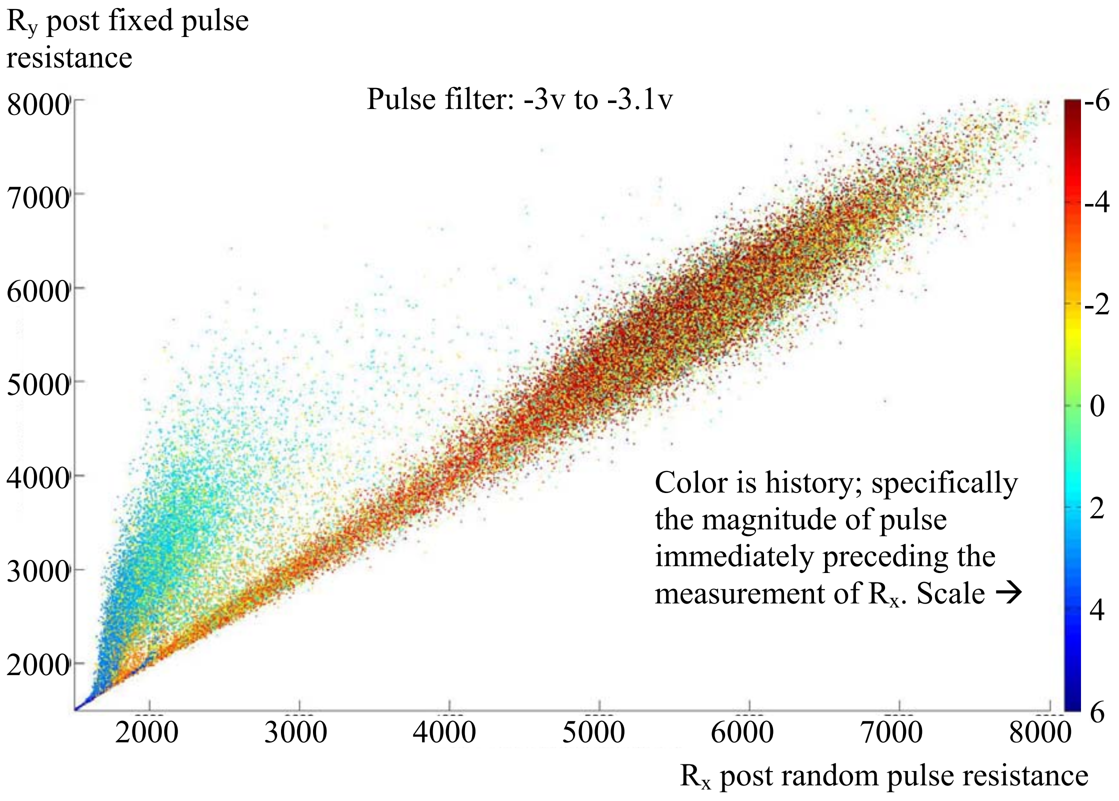


Figure 28: Memristor resistance change test due to a -3v pulse

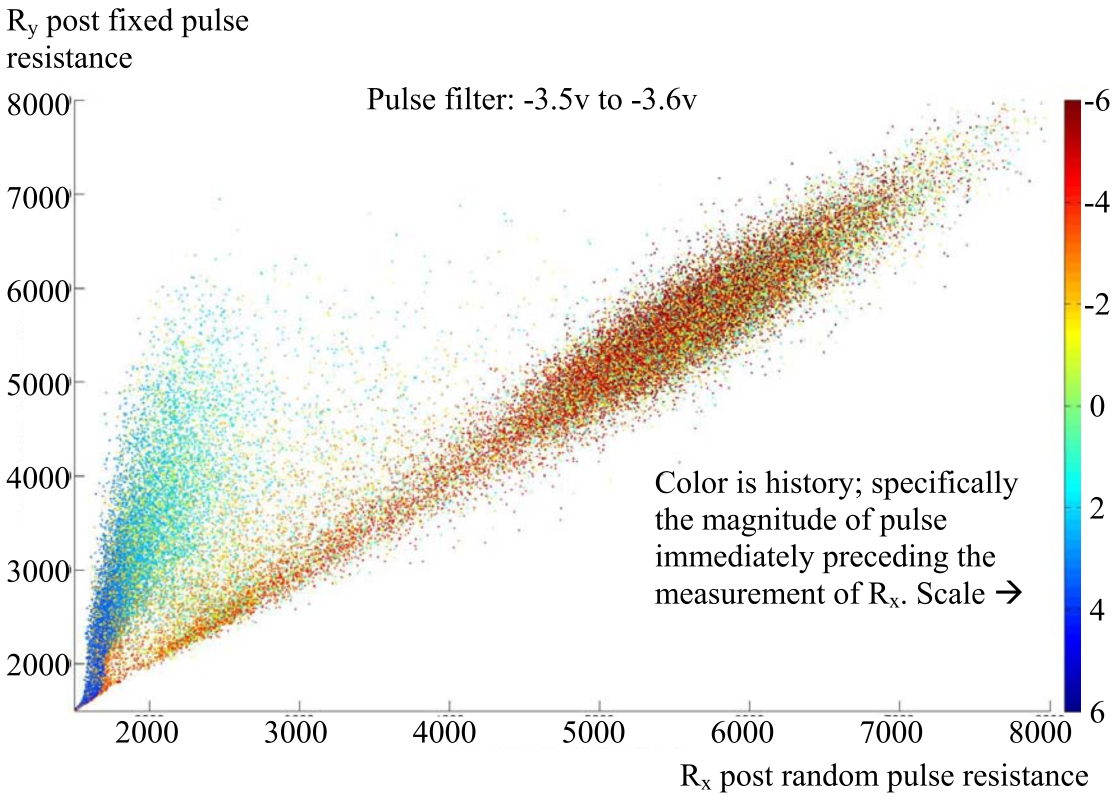


Figure 29: Memristor resistance change test due to a -3.5v pulse

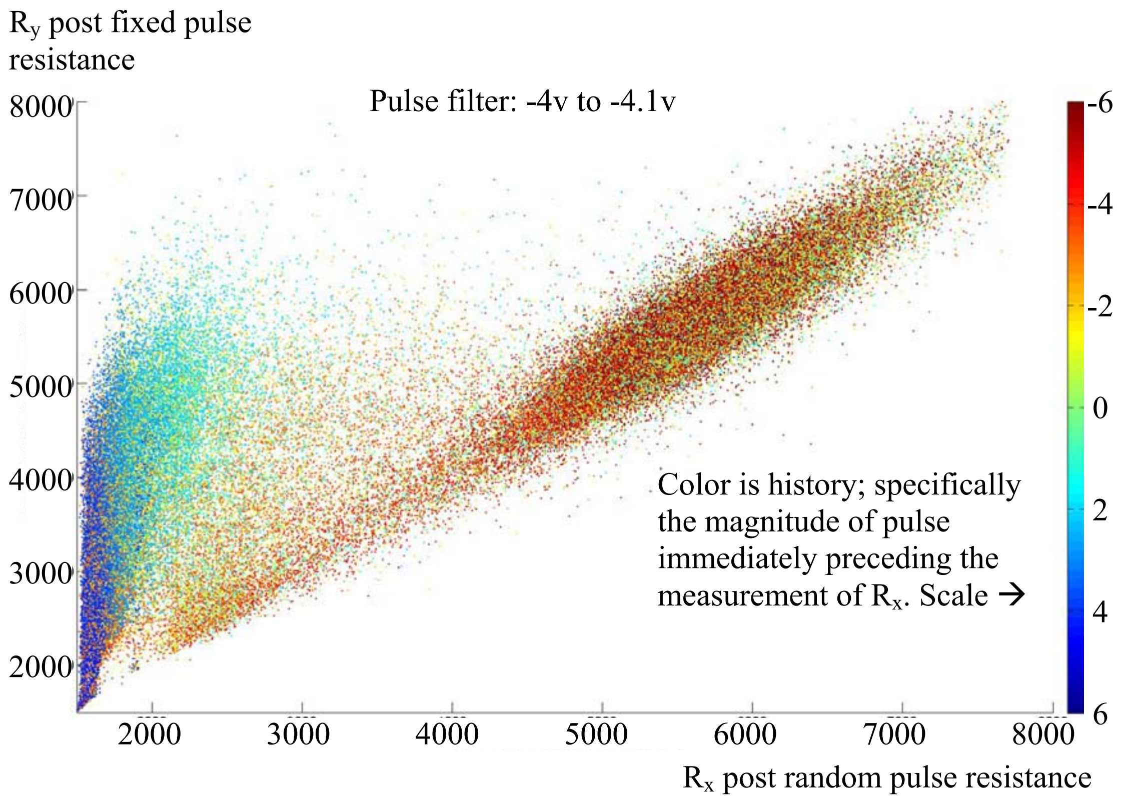


Figure 30: Memristor resistance change test due to a -4v pulse

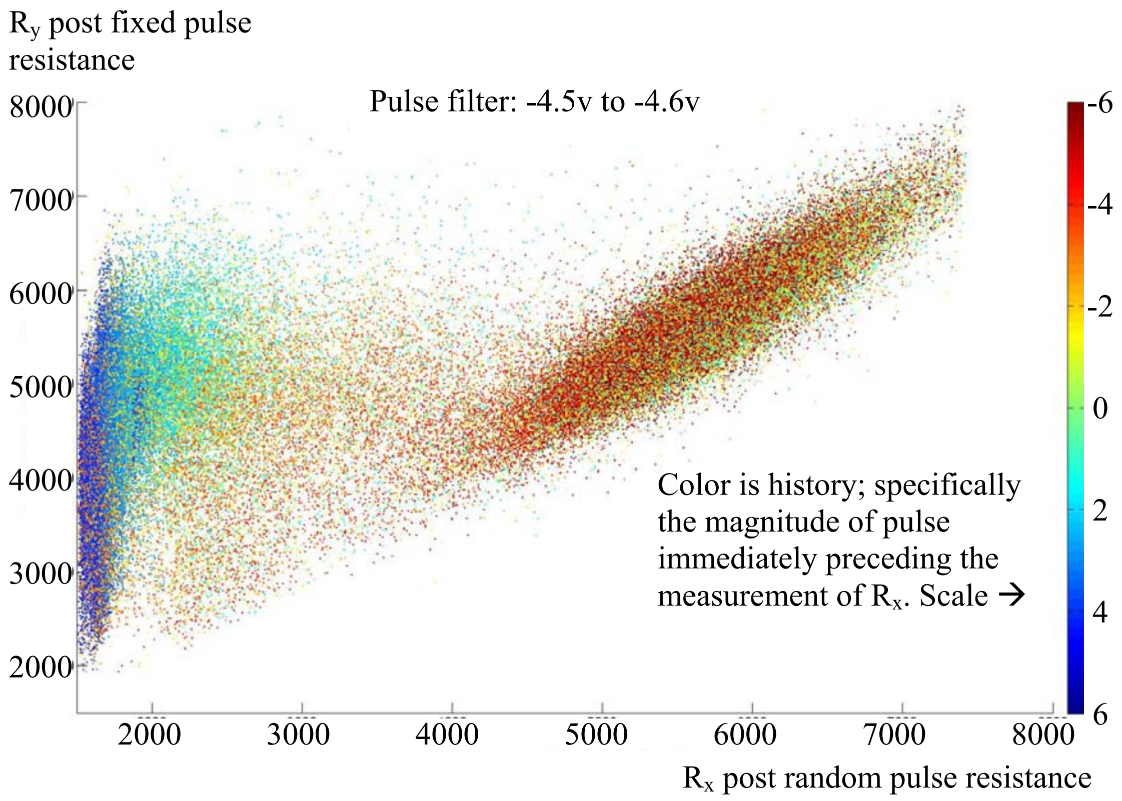


Figure 31: Memristor resistance change test due to a -4.5v pulse

Appendix III: Analog delta learning rule

Many learning methods have been proposed for memristor and other analog synapse types, but idealized devices are sufficient for this document.

We use natural memristor behavior to implement the delta learning rule [cite] in analog and on an entire array at a time. The delta learning rule is significant because it is the core of several neural learning algorithms, including back propagation. The delta rule essentially performs in-place addition of the outer product of two vectors to a weight matrix. Specifically, for matrix W, two vectors r and c, and a rate constant α, the delta rule computes $\exists_{j,k}\ W_{jk} \mathrel{+}= \alpha\ r_j\ c_k$. We developed a system where the weight matrix W is defined by the memristor conductances in an array such as shown in Figure 3B. The delta rule update uses four applications of voltages simultaneously to all the rows and columns of the matrix. Each application scales and shifts the voltages differently.

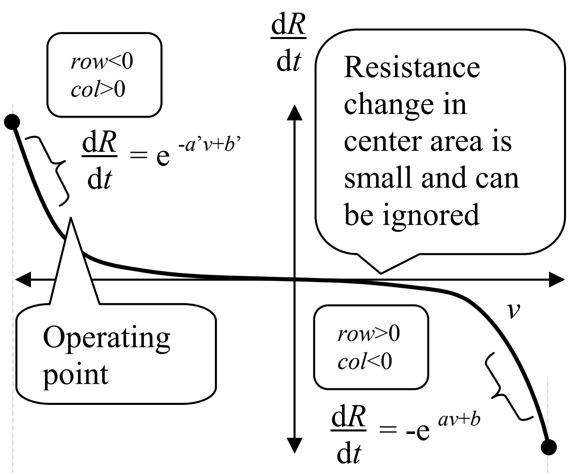
The initial insight is to use the memristor property of exponential resistance change as a function of voltage as a way to change multiplication into addition. The general rule of thumb is that $\mathrm{d}W_{jk}/\mathrm{d}t$ of a memristor is exponential in the applied voltage. Furthermore, the voltage applied to a memristor is the difference in voltage between the row and column wires. An array such as in Figure 3B thus generally performs an analog voltage subtraction between the voltage on the row wire and the voltage on the column wire (or vice versa), applying the difference in voltage to the device at the cross point. If the columns are driven with log r_j and the columns with -log c_k, the voltage applied to each memristor (in parallel) will be log r_j + log c_k. The resistance rate of change will be the exponential of this voltage $\mathrm{d}W_{jk}/\mathrm{d}t = \exp(\log r_j + \log c_k) = r_j c_k$. The insight above is a good start, but there are a number of issues before we can claim complete execution of $\exists_{j,k}\ W_{jk} \mathrel{+}= \alpha\, r_j\, c_k$.

While the memristor learning rate may be exponential in voltage, it is not an unadorned exponential function. This description assumes only a locally exponential behavior for positive and negative applied voltage, but that the specific exponential functions are shifted and scaled differently on the positive and negative sides. We assume $\mathrm{d}R/\mathrm{d}t = -\mathrm{e}^{av+b}$ for positive applied voltages and $\mathrm{d}R/\mathrm{d}t = \mathrm{e}^{-a'v+b'}$ for negative applied voltages, as illustrated in Figure 3B. We further assume the system will be engineered to have a maximum positive or negative rate of change, and that these maxima are the places for which the exponential approximation is defined.

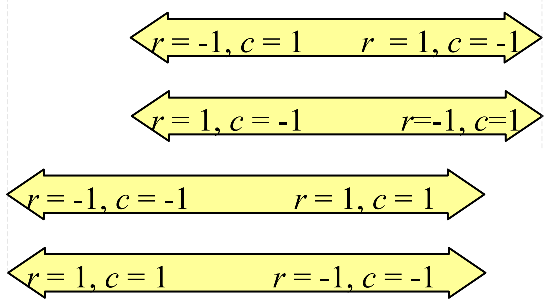
Memristor behavior:

Let v = voltage applied to memristor with standard polarity (+ voltage causes resistance to decline).

Let *row*, *col* = be the voltages on the array terminals, such that $v = row - col$



Four-step protocol:
Also illustration of short range condition (see text)



$r = -1, c = 1$ $r = 1, c = -1$
$r = 1, c = -1$ $r=-1, c=1$
$r = -1, c = -1$ $r = 1, c = 1$
$r = 1, c = 1$ $r = -1, c = -1$

Dynamic behavior, rising resistance……………….falling resistance:

($row<0$; $col > 0$; assume $r>0$; $c>0$)

$$\frac{dR}{dt} = e^{-a'v+b'} = e^{a'(-row) + a'\,col + b'}$$

Now let $row = -\log(r)/a'$
$col = (B' + \log c)/a'$

$$= e^{\log r + \log c + B' + b'} = rc\, e^{B'+b'}$$

($row>0$; $col < 0$; assume $r>0$; $c<0$)

$$\frac{dR}{dt} = -e^{av+b} = -e^{a\,row + a(-col) + b}$$

Now let $row = \log(r)/a$
$col = -(B + \log(-c))/a$

$$= -e^{\log r + \log(-c) + B + b} = rc\, e^{B+b}$$

Figure 32: Delta learning rule

As illustrated by the algebra on the bottom of Figure 32, it is possible to scale voltages so resistance will rise or fall based on the analog multiplication of signals on the rows and columns. However, it is necessary to scale the voltages to match the *a* or *a*' parameter of the memristor. The circuit actually computes $(rc)^x$ (multiplication and then raising the product to a power x), but proper scaling of voltage causes x=1 and the result to be a simple multiplication.

The engineer should choose offset voltages *B* and *B*' so the circuit meets requirements for maximum positive and negative rate of resistance change. The engineer will know memristor parameters *b* and *b*' and the maximum engineered values of *r* and *c*, so *B* and *B*' can be found.

The voltages described in Figure 32 can be applied to all rows and columns at once with a useful result, although this point depends on a property we will call here the "short range condition." Figure 32 contains two descriptions of dynamic behavior (lower left and lower right) which apply to mutually exclusive conditions $r>0$, $c>0$ and $r>0$, $c<0$. The short range condition implies that no resistance change occurs when one of these voltage scenarios is applied without the conditions being met. The inputs to the delta learning rule (*r* and *c*) will each have a range defined. The procedure described linearly transforms *r* and *c* and applies their difference to each memristor, which means the voltage applied to each memristor will also have a range that can be computed. The paragraph above further specifies that the engineer should pick *B* and *B*' so one end of the range meets the engineered point for maximum resistance change. If the short range condition is met, the memristors will see a voltage range like the ones illustrated by the yellow arrows in the middle of Figure 32. The condition is that one end of each range meets an engineered maximum voltage but the range is short enough that the other end falls short of a voltage where the memristor starts to program in the opposite direction. If the short range condition is met, the portion of the voltage range that does not meet the conditions ($c>0$, $r>0$, etc.) will fall onto a region of the memristor curve that does not cause significant resistance change. If this is true, it will be possible to create a protocol involving application of multiple voltages where the steps in the protocol do not interfere with each other.

The short range condition illustrated by the yellow arrows of Figure 32 seems to apply to all memristors in our experience, but we cannot guarantee it is always true. If the memristor d*R*/d*t* curve is "steeper than an exponential" in voltage, the yellow arrows will tend to be shorter. It would seem prudent for readers to revisit this question upon encountering memristors with new behavior.

The equations at the bottom of Figure 32 only have two ($r>0$, $c>0$ and $r>0$, $c<0$) of the required four voltage applications, with $r<0$, $c<0$ and $r<0$, $c>0$ missing. It is left as an exercise to the reader to develop equations like the ones on the bottom of Figure 32 but with signs flipped to meet new conditions. However, the author will point out here that these additional conditions could be met by setting $r \leftarrow -r$ and $c \leftarrow -c$ and then applying the same equations (presuming *r* and *c* have a defined range symmetric about the origin.)

The complete protocol is in Table 3.

Table 3: Delta rule protocol.

Phase	Condition addressed	Row voltage	Column voltage
1	r>0; c<0	LOG r /a	-(B + LOG (-c))/a
2	r<0; c>0	LOG(-r)/a	-(B + LOG c)/a
3	r>0; c>0	-LOG r/a’	(B’ + LOG c)/a’
4	r<0;c<0	-LOG(-r)/a’	(B’ + LOG (-c))/a’
Note: LOG(x) function is equivalent to log(x) in an operating range (of say 1-5v) but zero elsewhere, including zero for all negative values.			

There are several other issues or approximations that have not been dealt with in the discussion above. However, the discussion is just to justify the architecture. The issues are:

- Memristors have resistance change noise.
- No memristor curves are truly exponential, and it is unknown if the exponential approximation is good enough.
- The memristor programming rate dR/dt depends on R. However, as R is restricted to a narrower and narrower range, the dependence of R drops.

Appendix IV: Non-interrogatable logic

This appendix describes a new implementation of tamper-resistant logic circuits that can perform functions but which are resistant to revealing their function by interrogation of the device.

An encryptor is an example application for such a circuit. A non-interrogatable encryptor would readily encrypt by using key built into its circuit but would resist revealing the specific encryption algorithm and hence the key by interrogation of the device.

The method used is based on a novel use of artificial neural network technology. While high-level cognition from artificial neural networks is a research topic, artificial neural networks have little difficulty being programmed into systems with the sophistication of a half-dozen logic gates. However, figuring out which half-dozen logic gates a specific artificial neural network is implementing is difficult by physical analysis.

If an encryptor (for example) were implemented as a network of these small non-interrogatable circuits, reverse engineering of the encryptor would just show it to be a circuit of many gates. The reverse engineer would not know which gates were AND, OR, or some other gate type and hence would have too little information to figure out the encryption function and hence the key.

The method is illustrated in this paper with TiO_2 memristors. In these devices, a large electrical current will change the thickness of a tunnel barrier by a few angstroms. This small change varies the resistance of the device to a smaller sensing current and can represent the synapse weight in an artificial neural network. However, the synapse weights would be difficult to image microscopically or read it out in other ways.

The method should be scalable to more cognitively sophisticated functions. This may be a future growth path, but is not essential to basic use of the function. It is well known today how to make artificial neural networks learn behavior equivalent to less than some number of logic gates. This maximum number is expected to rise with advancing technology. At some point, the circuits would become "cognitive" as traditionally defined. This document does not address the issue of cognitive function, but should be able to make a secure implementation of cognitive functions if they are created by other means.

Non-interrogatable functions

The basis of non-interrogatability is illustrated in Figure 33. A logic designer will recognize the general pattern of the circuit on the left as an Arithmetic and Logic Unit (ALU), but to actually figure out whether the circuit will add, subtract, perform logical AND or OR requires analyzing the exact wiring pattern and gate types.

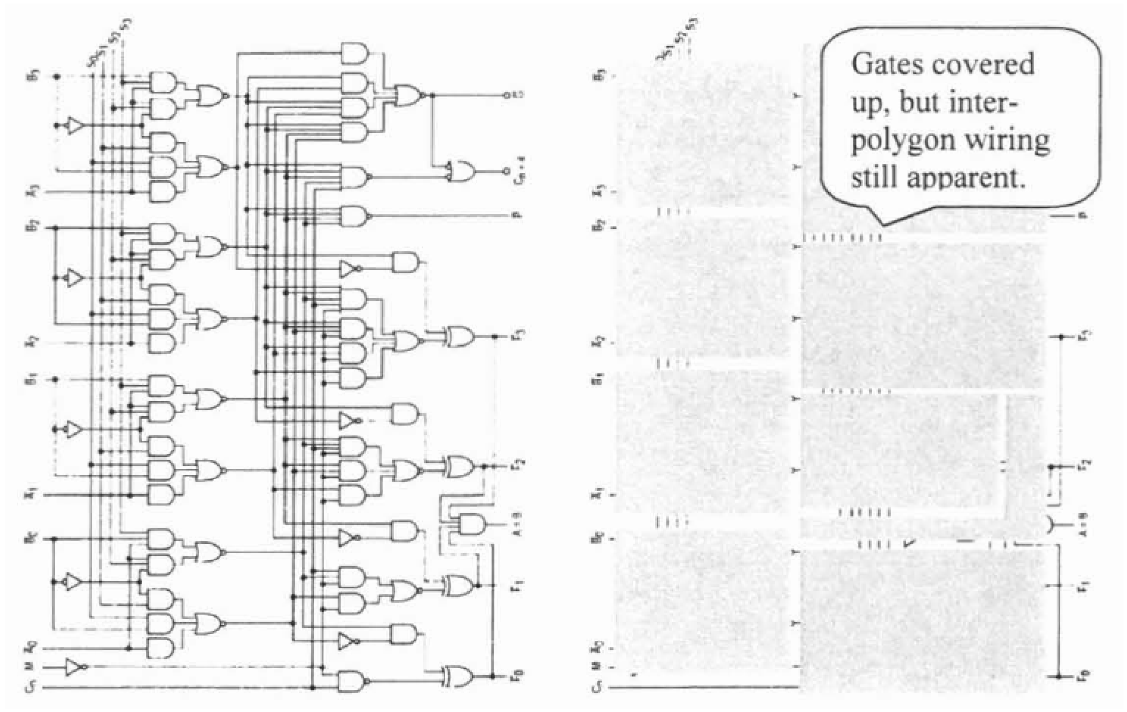


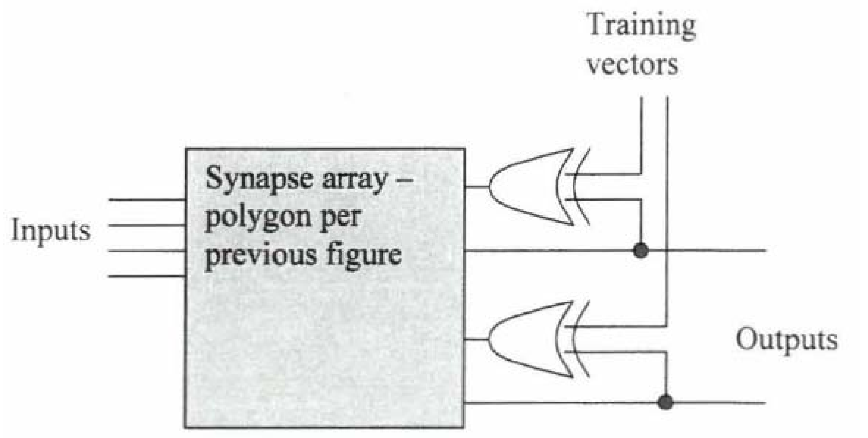
Figure 33: Basis of non-interrogatability

The right of Figure 33 shows a way to make the ALU non-interrogatable. A gray polygon has been placed over groups of up to 8 gates. Imagine that each gray polygon contains an artificial neural network that implements a function equivalent to the gates on the left. Provided that we can create a neural network that emulates 8 gates, we can obviously create a tiling of any logic circuit.

The function of the right part of Figure 33 is harder to identify. While a logic designer could trace the interconnections between the polygons and might identify the right of Figure 33 as an ALU, identifying whether the ALU is adding or subtracting at a given time would require information from within the polygons. As will be further described below, the function of an artificial neural network is difficult to discern by observation.

Creation of the circuit on the right of Figure 33 must be considered as well. The problem can be seen by analogy to Field Programmable Gate Arrays (FPGAs). An FPGA chains the equivalent of the polygons (which are called Look Up Tables or LUTs by analogy) together into a configuration shift register. An external memory feeds the register and loads the circuit identity into each LUT. The FPGA shift register can be used to unload the function as well, providing a means of discerning the function. The remedy in for a non-interrogatable circuit is to give each polygon its circuit identity with a "write only" capability and where there is simply no electrical circuitry provided that can read out the circuit identity.

One implementation is illustrated in Figure 34, where each polygon obtains its circuit identity by an automatic learning process. The idea here is that the synapses in each polygon are initially unprogrammed. The polygon is sent training vectors during a programming phase and is sets the synapses to learn the proper behavior. Once this phase is complete, the circuit identity is in synapse weights.



Training vectors are never stored in the synapse array. Instead, the difference between the learned output and the training vectors is computed with XOR gates and serves as a stimulus to change synapse weights. Therefore, the path from the circuit output to the training vectors would require going backwards thorough a gate

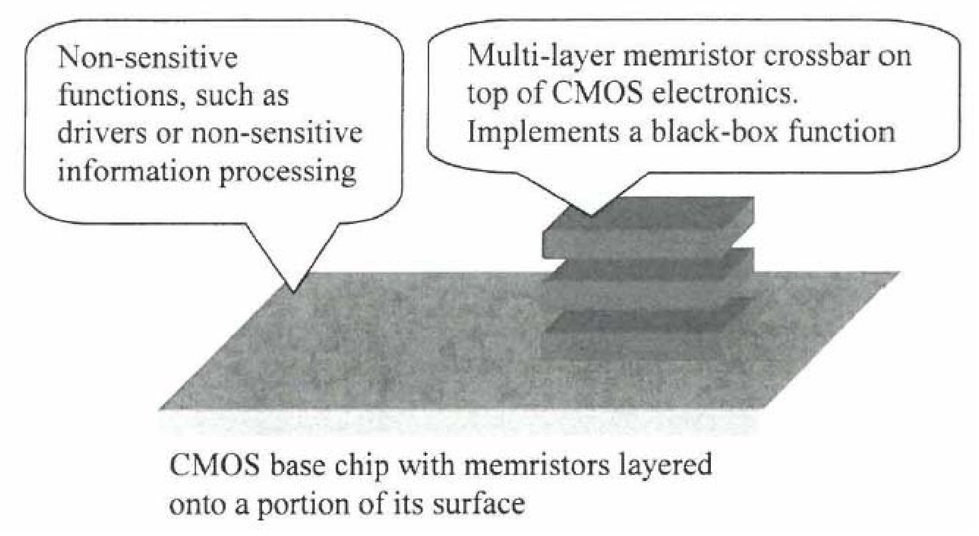
Figure 34: Programming a synapse array without a readout path.

The obvious way to identify the function programmed into Figure 34 would be to unload the training vectors. However, the training vectors are applied at the factory and they are transient voltage signals. Once the programming phase completes, the training vectors will only be present indirectly through synapse weights.

A second way to get circuit function would be to monitor the outputs during operation and deduce the operation of the circuit from the learned behavior. After all, each output shares an XOR gate with a wire going to the source of training vectors. The XOR gate is pointed in the wrong direction, forming a basis for security.

Application-level security

We propose a security paradigm based on protected "black-box" functions, and illustrated in Figure 35. A black box in the general English usage is defined as a function that can be evaluated and accessed through its inputs and outputs, but whose internal structure and operation are hidden. We propose a black-box function whose function is determined by the synapse weights in an artificial neural network like Figure 34 and occupying a portion of a CMOS chip. A collection of neural networks could duplicate any logic function like Figure 33, but it would be difficult for reverse engineering to figure out the function based on analysis of the synapses However, it would still be possible to deduce the function by analysis of input/output behavior. We will give two examples of black-box functions for illustration. One is a black box that encrypts with a specific key (there is an obvious but incorrect interpretation of this statement, so this will be clarified in the next paragraph) and the other is a complex circuit for dynamic control of a robot.


Non-sensitive functions, such as drivers or non-sensitive information processing
Multi-layer memristor crossbar on top of CMOS electronics. Implements a black-box function
CMOS base chip with memristors layered onto a portion of its surface

Memristors are layered onto a portion of a CMOS chip. The memristors implement a function that can be used but is difficult to reverse engineer through physical access.

Figure 35: Security and implementation paradigm.

There is an obvious but incorrect interpretation of the encryption example, so we will explain. Typically, an encryption function is defined as *ciphertext = encrypt(key, plaintext)*. While it would be possible to make a black box implementation of the encrypt function, the sensitive information is really the key not the algorithm. If the key is outside the black box, it is not protected. So, we are suggesting black-box implementation of a key-specific encryption algorithm defined as *ciphertext* = $encrypt^{key}$*(plaintext)*. The $encrypt^{key}$ function is a special function that encrypts with only a specific key. It incorporates the key into the implementation of the encryption algorithm by replacing the variable key with constants and optimizing the implementation. This essentially encodes each bit of the key into a gate by selecting the gate's function to be AND or OR. If $encrypt^{key}$ is in a protected black box, there is no key stored separately to find. We do not propose protection of $encrypt^{key}$ from reverse engineering by observing its external behavior; encryption algorithms are specifically designed to be secure against that type of attack.

The second example is the protection of a complex system from reverse engineering. The idea here is that the substantial resources are invested in electronic designs. While the basic algorithms are not nearly as sensitive as an encryption key, original designers would like copying to be as difficult as possible. In this example, we propose that the expensive intellectual property investment be put in the synapse weights and protected as a black-box function. Reverse engineering will not be able to simply "read out" the function but would instead have to reverse engineer by observing the behavior of the system. However, reverse engineering by observing the behavior is nearly as expensive as developing the function from scratch.

Memristors and physical security

Memristors are an emerging new device that can be added to CMOS chips in order to bolster both function and security. Industry is in a race to try and commercialize hybrid memristor-CMOS chips as new "redox memories" for memory markets like flash drives and Solid State Disks (SSDs). There has also been research on the use of memristors for logic, such as Field Programmable Gate Array (FPGA) replacements and artificial neural network circuits.

It is hard to determine memristor device state physically, particularly in 3-D layered structures. Reverse engineering programmable memristor logic would thus require accurate determination of analog state from memristors. We discuss in this document a sequence of security features that create a "black box" function using memristors, or a function that could be used just like logic on a chip but where it would be very difficult to determine the design of the function.

Overview of security features

The basic sequence of the exposition in this paper is as follows:

1. Memristors encode 0s, 1s, or analog values in their changeable resistance, but they are encoded in a very subtle physical form. The state of a memristor is so subtle physically that is troublesome to research scientists trying to get the devices to work in the first place and may pose a significant problem for reverse engineering.

2. Memristor-based programmable logic is feasible; it will have its function encoded in the memristor values, rather than the structure of the chip. This offers flexibility like FPGAs. This also means reverse engineering process must determine the state of the memristors otherwise they will end up with just the equivalent of the schematic of an unprogrammed FPGA.

3. Since memristors do not require any crystalline materials (as transistors do), they can be grown in multiple layers on a chip. This bolsters density, but determining the values programmed into the memristors will require characterizing and removing the outer layers without changing the programming in the middle layers. This raises difficulty.

4. Memristor logic chips are likely to be manufactured with flaws, just like today's memory chips. In programming memristor logic chips, slightly flawed chips will be programmed to avoid the unique pattern of flaws on each chip. This means every chip will be different. This will bolster manufacturing yield, but it also means reverse engineering will have to be performed on just one instance of a chip. Flaws can also serve as the basis of a trust anchor; more below.

5. Memristors also have an irreversible short circuit state. Besides the items above, a memristor could be coupled to an active tamper detection system that shorts some or all or the memristors when intrusion is detected. Memristor shorts seem to have

a substantial advantage over e. g. erasing cells in a flash memory. To erase a flash memory requires repeatedly writing zeros and ones to the same cell, a process that takes considerably longer than just writing a value one time. In contrast, shorting a memristor requires no exceptional voltages and a write time perhaps just nanoseconds longer than just writing data. In addition to the shorted state destroying that memristor's state, the shorted memristors will make the values of other memristors more difficult to reverse engineer due to the interconnected nature of memristors on the crossbar. Shorts can also serve as the basis of a trust anchor; more in the next item.

6. Both flawed and shorted memristors can be the basis of forming a unique ID for a chip that cannot be unprogrammed, also called a trust anchor. The multilayer nature of memristor chips could make the unique ID difficult to extract, further bolstering its security.

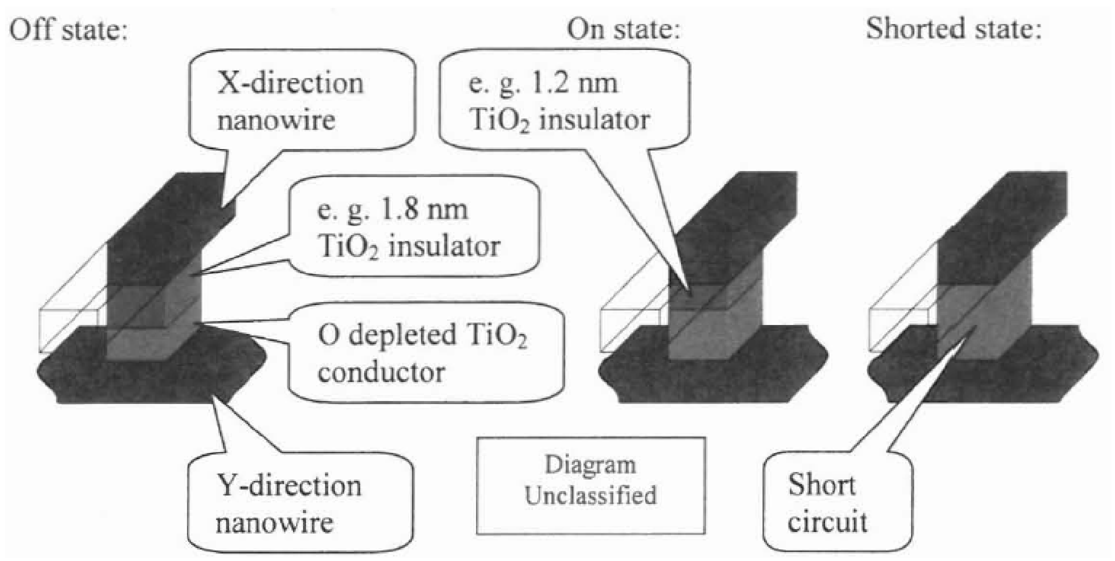
7. Memristors are always willing to learn, and this could become the basis of a security feature. A black box can be characterized by observing the output when all possible stimuli are applied (a method that does not require physical tampering). An example is cracking a password by applying all possible passwords and seeing which one works. However, memristor circuits will learn during operation whether we want them to or not. It is possible to create a circuit where applying huge numbers of test vectors as is done in an exhaustive search would cause the circuit to learn the tests and forget the intended behavior.

8. Some proposed memristor logic chips (specifically the artificial neural networks), use memristors as resistors programmable to a continuum of values rather than just on and off. This can be a security feature. Reverse engineering becomes more difficult as the resistance values need to be obtained to tighter variances, like 1% or .1 %. The technical basis of the security feature is as follows: Neural network chips depend on ratios of resistances rather than absolute resistances. This makes neural chips tolerant of process variations, temperature variations, etc., which is a benefit. However, the reverse engineering process will need to characterize resistance ratios to a specific overall degree of precision otherwise the function will not be identified correctly. The security feature is to design and build the memristor crossbars so that the trauma inherent in reverse engineering will alter resistance values so much that the function cannot be identified.

The design freedom offered by the new memristor component permits the series of tricks outlined above. While the tricks may not be new or unique, we believe the collective value of these tricks could be quite large.

Memristor operation

The relevant operation of a memristor device is shown in Figure 36. The devices are grown at the crosspoints of a nanowire array for all the architectures under consideration. For applied voltages under around 1 volt, each memristor is characterized by a resistance in the range of 1M-1K ohm. Memristors can be programmed by applying more than around 1 volt; a voltage in one direction increases the resistance and in the other direction the resistance decreases.



The memristor shown in read and green is in the junction of two nanowires and nominally comprises oxygen-depleted TiO_2, which is a conductor. A layer comprised of insulating TiO_2 of e. g. 1.2-1.8 nanometers is thin enough for tunneling and produces effective resistances of 1M ohms (left) to 1K ohms (center) in typical circuits. By applying 1-3 volts, a voltage gradient of a gigavolts per meter is created across the insulating material, which causes the boundary to move (left to center or back). If the insulator thickness is reduced to the pointwhere it disappears, the junction shorts. When the insulator disappears, there is no longer a layer to create the large field to move the oxygen vacancies, so a short is irreversible

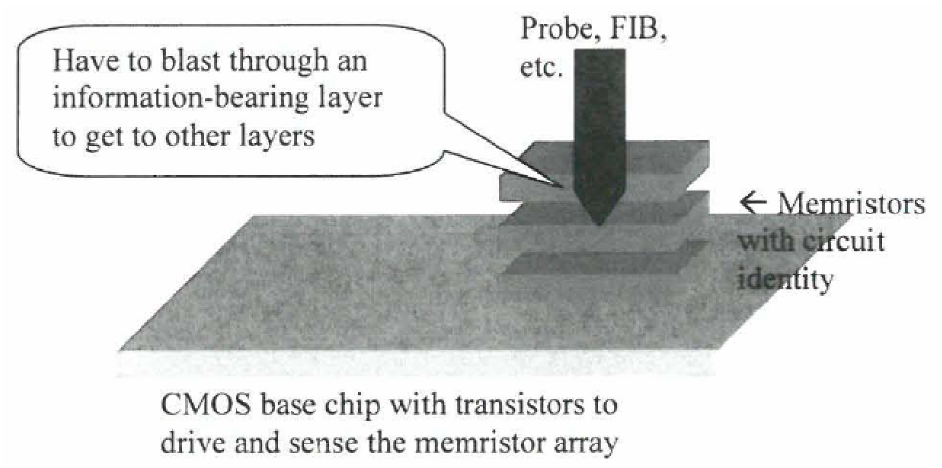
Figure 36: Memristor operation for TiO_2.

The basic nature of the device has security implications for attempts to read out the state physically. Other memory technologies (e. g. phase change memory) have a cell where the state of all the material is different (e. g. crystalline or amorphous) between the 0 and 1 states. However, the change in the memristor cell is an e. g. 6 angstrom change in the thickness of the non-depleted layer (Picken, 2009), which is merely a stoichiometry change caused by movement of just a handful of atoms. The fact that states are differentiated by such a small physical difference confounds the physical science researchers that are trying to engineer the systems and would confound reverse engineering (Guanglei Cheng, 2010).

Notably, the memristors can be put into an irreversible shorted configuration. A shorted memristor has resistance of e. g. 200 ohms, which is substantially below the lowest on state resistance. The shorted state could have security value, as will be explained later.

Naturally booby-trapped 3-D layered structure

The use of 3-D layering has obvious advantages in terms of density, but it will also make it more difficult to read memristor states physically. Due to the fact that memristor layers can be applied through Molecular Beam Epitaxy (MBE) and other surface deposition methods rather than requiring a crystalline layer, it is straightforward to make a handful of layers. See Figure 37. If reverse engineering of memristor chips is to be successful, it will need information from all the layers. It is of course possible to remove layers, cut holes in layers to access lower ones, etc. However, the layers that have to be removed will have information as well. Memristor outer layers are thus a booby trapped protection of the inner layers because destroying the protection destroys some of the information needed.


Have to blast through an information-bearing layer to get to other layers
Probe, FIB, etc.
← Memristors with circuit identity
CMOS base chip with transistors to drive and sense the memristor array

The information needed for effectively duplicating function is in multiple layers. To get to inside layers will require blasting through outer layers. However, the outer layers contain information also relevant to reverse engineering.

Figure 37: Multi-layer memristor security advantage.

The multi-layer structure would be particularly effective against reverse engineering where only one copy of a chip is available, and we will see below how to enhance the probability of that occurring. With only one copy of a chip, the reverse engineer would need to fully characterize and remove layers one at a time for access to the ones below. This would have to be done gently enough that the information in lower layers would not be corrupted.

Manufacturing variability and security

Programmable logic has well-known advantages for manufacturing yield, but this type of logic could yield security advantages as well. The best known use of soft programmable logic is to increase the manufacturing yield for memories. Most large memories are constructed with spare rows and columns. During the manufacturing test process, a fully functional memory is created by using fuses (or a related technology) to substitute spare rows or columns for others that had manufacturing defects. As a result, the layout of the actual data in memory chips in PCs, notebooks, etc. is different on a chip-to-chip basis. This is a factor to consider when using memory imprints to identify the past contents of memories.

Soft-logic memristor circuits could use a similar concept for a security advantage. It is a great convenience for reverse engineering to have a supply of identical systems to work on. This permits destructive processes to be used, because the destroyed chip can be replaced with another. If every circuit is different, the reverse engineer will be limited to gentle processes. This leverages the protection illustrated in Figure 37.

Analog properties and security

Artificial neural networks typically have analog synapses, which would be implemented by memristors used in an analog programmable resistor. The fact that the memristors are analog would further confound the reverse engineering process.

Neural networks are fault tolerant – to an extent, and we can exploit both the fault tolerance and the limits of the fault tolerance. Neural networks rely on forming weighted sums of neuron outputs to form the inputs to the next layer of neurons. In that model, the weights are analog values representable with e. g. 100 levels of coupling strength. The number of levels is controllable to some extent during the engineering of the circuit. During engineering of the overall neural network circuit, the system can be over provisioned so that a few missing synapses will not make the circuit stop working. The circuit can also be made tolerant of systematic shifts in neurons, such as might be caused by a change in temperature. These are advantages to a neural network that will serve to bolster reliability and manufacturing yield.

However, a given neural network system will require that ratios of neuron resistances stay fixed within some specific tolerance. If a reverse engineering process cannot meet the required level of fidelity on a large scale, a replacement circuit will not function. This consideration invokes the issues in Figure 37 again: In chopping apart the circuit in Figure 37, we not only have to avoid destroying the binary values of memristors in the inner layers, but we have to avoid changing their analog resistance by e.g. 1%.

Bibliography

DeHon, A. L. (2005). Hybrid CMOS/Nanoelectronic Digital Circuits: devices, architectures, and design automation. Proceedings of the 2005 IEEE/ACM International Conference on Computer Aided Design, 375-382.

Guanglei Cheng, F. M.-R. (2010). Microwave frequency capacitance measurement of memristive devices. Palo Alto, CA.

Pickett, M. D. (2009). Switching dynamics in titanium dioxide memristive devices. Journal of Applied Physics 106,074508.

Williams, R. S. (2007). Nano/CMOS architectures using a field-programmable nanowire interconnect. Nanotechnology 18 , 035204.

Distribution

1	MS0899	Technical Library	9536 (electronic copy)
1	MS0359	D. Chavez, LDRD Office	1911

For CRADA reports add:

1	MS0115	OFA/NFE Agreements	10012

Sandia National Laboratories