**A Case Study on Neuromorphic Computing and Memristor Based Crossbar Arrays**

Sebastian Marcel Witkowski

Northwestern University

***Abstract* – Current CMOS technology is reaching a threshold in terms of size and processing potential, thus signaling the end of the progression of Moore’s Law. In order to keep up with consumer demands for ever increasing processing power and speed while minimizing power consumption efforts are being made worldwide to find a potential alternative to current Von Neumann based CMOS hardware implementations.**

**One proposed alternative computational method is one that models and tries to recreate the functionality of a biological computational system which could potentially have major implications in multiple research areas. This paper explores this proposed alternative, neuromorphic computing, and also looks into innovative implementation methods involving the recently discovered memristor.**

I. INTRODUCTION

Neuromorphic computing, first proposed as a theory by Carver Mead in the late 1980’s, is the attempt to mimic the behavior of biological neurological systems utilizing very-large-scale-integration (VLSI). The basic premise is to model hardware after synapse networks in the brain. Individual cells in the brain form connections to other neighboring cells in the brain which are strengthened via the passing of voltage. This voltage then accumulates until it passes a certain threshold value after which the neurons which have been activated send out current to other neighboring neurons, thus repeating the cycle.

This type of processing has many potential benefits, many of which are simply not possible with classical Von Neumann architectures due to its integration of memory and processing which Von Neumann machines fundamentally cannot do. Some of these include extremely efficient parallel processing, specialized pattern recognition, and autonomous learning. In the long term, this technology also shows potential in furthering our understanding of artificial intelligence on the hardware level in hopes of someday building an architecture to mimic the human brain, assuming the technology could be scalable down to biological levels.

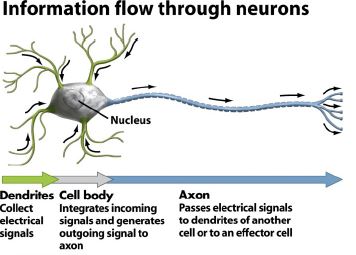


FIGURE 1: Neuron Functionality

One of the more common ways this type of processing is implemented in current times is with a crossbar array. At one edge of the crossbar array, inputs voltages are cascaded in and are processed by the array, which then send current out through the output nodes, each of which correspond to an individual neuron.

What enables this technique to truly mimic a biological system however is the fact that each intersection within the crossbar array, or “synapse”, is assigned its own weight which determines how easily voltage passes through that point which can then be adjusted.

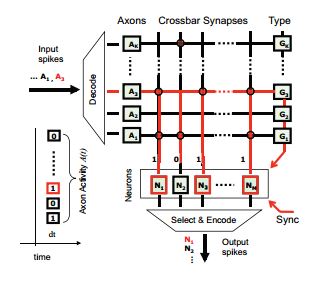


FIGURE 2: Model Neuromorphic Processor Implemented via a Crossbar Array

What this also implies however, is that the array must be trained for whatever purpose the user desires prior to final implementation.

II. CHALLENGES

There are several unique challenges that come up when considering designing a neuromorphic processor. One of these is the form of training algorithm used to adjust all the weights of the crossbar array, but more importantly is how to implement the storage of weights. Currently, one of the more common implementations is done using SRAM as was done by the research and development branch of IBM in 2011. Their SyNAPSE chip implemented 256 individual neurons with 1024 axons and 262,144 programmable synapses. While functionally it was a great success, it also had several drawbacks. Since it was implemented using an SRAM array for the synapses, the chip ended up using a very high number of transistors, an estimated 500 transistors per synapse.

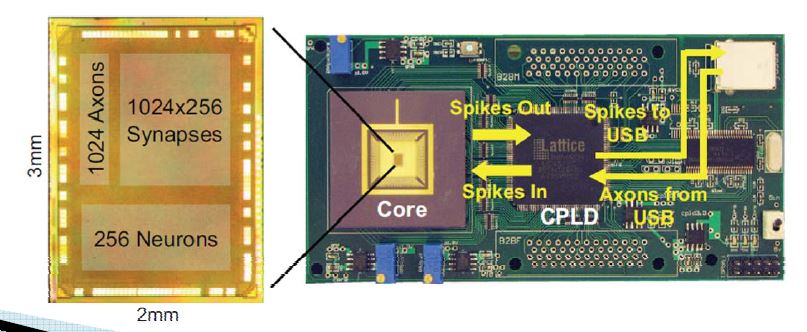


FIGURE 3: The IBM SyNAPSE

This in turn forces the chip to require a large amount of power and also have a very large layout area. This is not ideal in the long run where we would like to eventually scale down the technology to biological system sizes.

III. INTRODUCING MEMRISTORS

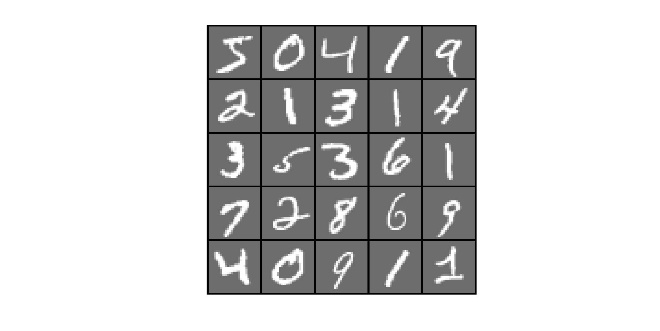
Memristors, first hypothesized in 1971 and then recently discovered in 2008 by IBM labs, are a potentially ideal solution to the scaling problem facing current implementations of neuromorphic processor designs. Due to their unique property of being able to automatically alter their effective resistance, or memristance, based off the strength of voltage applied to the memristor, they could in theory be used to completely replace SRAM in crossbar arrays.

For my project, I wished to implement a small scale neuromorphic processor implementing memristors as the junctions between input and output bars in the crossbar array.

IV. IMPLEMENTATION

I decided I would implement a simple 25x10 memristor crossbar array with differential op-amps at each of the output pins to function as neurons, only allowing current to pass if the output voltage from the crossbar array surpassed a threshold.. Since memristors are a relatively new technology, most software used in VLSI design, particularly Cadence Virtuoso, do not have the memristor included as one of their available pre-programmed components. To overcome this obstacle, I decided to use Matlab to simulate the memristor array and also the training of the array

The task I would be training the array for would be to recognize number values from scans of handwritten digits from the MNIST database. The database contains 60000 training images, of which I used 30000, and 10000 test images. I took the original 28x28 images, scaled them down to be 5x5, and then sorted all of the training images in order from all entries corresponding to the digit 0 would be group together, followed all of the 1’s, then 2’s, so on and so forth



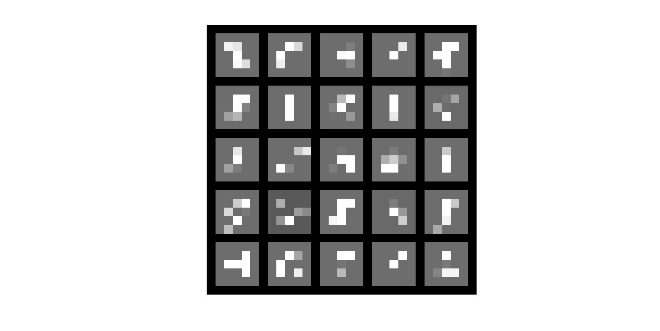
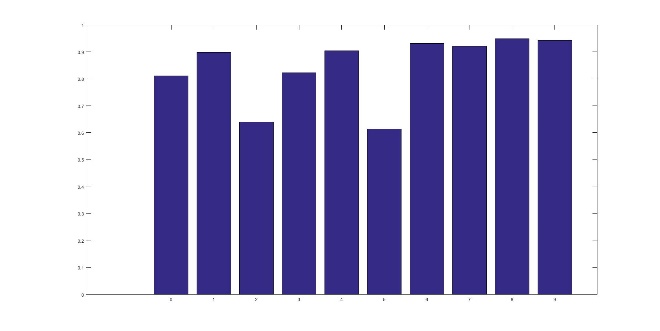


FIGURE 4: Unscaled and Scaled MNIST Images

I then multiplied the input training vectors with the weight matrix, one at a time, scaling the weights of the column above the desired output pin if the output voltage did not reach threshold voltages. After calculating the final weight matrix, I ran the test image vectors through it and compared the outputs to an arbitrary threshold value. I also built a schematic and layout of a differential op-amp with all 25 current sources being summed up and fed into the op-amp for comparison with the threshold value.

V. RESULTS



My Matlab simulation was highly successful and every simulation ran with the circuit in Virtuoso was able to successfully indicate what digits the image’s input values corresponded to.

VI. CONCLUSION

I believe any source of error in the result most likely comes from the very large image scaling and do to with the crudeness of my training algorithm.

MATLAB CODE

%Generate MNIST training and test image data for simulation

%Code borrowed from ufldl.stanford.edu

RawTrainingImages = loadMNISTImages('train-images.idx3-ubyte');

TrainingLabels = loadMNISTLabels('train-labels.idx1-ubyte');

RawTestImages = loadMNISTImages('t10k-images.idx3-ubyte');

TestLabels = loadMNISTLabels('t10k-labels.idx1-ubyte');

%resize and scale image data to be [(5x5)x #images]

ScaledTrainingImages = zeros(25,30000);

ScaledTestImages = zeros(25,10000);

RawBuffer = rand(28,28);

ScaledBuffer = rand(5,5);

%resize training images

for ii = 1:30000

RawBuffer = reshape(RawTrainingImages(:,ii),28,28);

ScaledBuffer = imresize(RawBuffer,[5,5],'nearest');

ScaledTrainingImages(:,ii)=reshape(ScaledBuffer,25,1);

end

figure(1);

title('25 Unscaled 28x28 MNIST Image Members');

display\_network(RawTrainingImages(:,1:25))

figure(2);

title('25 Scaled 5x5 MNIST Image Members');

display\_network(ScaledTrainingImages(:,1:25));

%resize test images

for ii = 1:10000

RawBuffer = reshape(RawTestImages(:,ii),28,28);

ScaledBuffer = imresize(RawBuffer,[5,5],'nearest');

ScaledTestImages(:,ii)=reshape(ScaledBuffer,25,1);

end

ScaledTrainingImages = ScaledTrainingImages';

ScaledTestImages = ScaledTestImages';

SortedScaledTestImages = zeros(10000,25);

SortedScaledTrainingImages = zeros(30000,25);

SortedTestLabels = zeros(10000,1);

SortedTrainingLabels = zeros(30000,1);

sortcount = 1;

TrainingDigitTally = zeros(10,1);

TestDigitTally = zeros(10,1);

for ii = 0:9

for jj = 1:30000

digit = int8(TrainingLabels(jj));

if digit == ii

TrainingDigitTally(ii+1) = TrainingDigitTally(ii+1)+1;

SortedScaledTrainingImages(sortcount,:) = ScaledTrainingImages(jj,:);

SortedTrainingLabels(sortcount,1) = TrainingLabels(jj);

sortcount = sortcount+1;

end

end

end

sortcount = 1;

for ii = 0:9

for jj = 1:10000

digit = int8(TestLabels(jj));

if digit == ii

TestDigitTally(ii+1) = TestDigitTally(ii+1)+1;

SortedScaledTestImages(sortcount,:) = ScaledTestImages(jj,:);

SortedTestLabels(sortcount,1) = TestLabels(jj);

sortcount = sortcount+1;

end

end

end

%initialize matrices

weights = rand(25,10);

Vout1 = zeros(1,10);

%train weights

for ii = 1:30000

Vout1 = SortedScaledTrainingImages(ii,:)\*weights;

Vout1 = Vout1/norm(Vout1,inf);

for jj = 1:10

if ((jj-1)==SortedTrainingLabels(ii)) && (Vout1(jj) < 0.75)

%if at correct output voltage pin and voltage is less than

%threshold, scale all weights corresponding to Vin pins with

%voltage > 0 up, and all others in the column down

for kk = 1:25

if SortedScaledTrainingImages(ii,kk) >=0.25

weights(kk,jj) = weights(kk,jj)\*1.3;

else

weights(kk,jj) = weights(kk,jj)\*0.875;

end

end

end

end

end

weights = weights/norm(weights,inf);

Correct = zeros(10,1);

DigitTally = zeros(10,1);

for ii = 1:10000

digit = int8(SortedTestLabels(ii));

DigitTally(digit+1) = DigitTally(digit+1)+1;

Vout1 = SortedScaledTestImages(ii,:)\*weights;

Vout1 = Vout1/norm(Vout1,inf);

if Vout1(digit+1)>0.75 %Vout1(digit+1) == Vmax;

Correct(digit+1) = Correct(digit+1)+1;

end

end

PercentCorrect = zeros(10,1);

for ii = 1:10

PercentCorrect(ii) = Correct(ii)/DigitTally(ii);

end

x = 0:9;

figure(3);

title('Proportion of Correct Number Identification');

xlabel('Image Label');

ylabel('Percent Correct');

bar(x,PercentCorrect);

REFERENCES

[1] - P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, “A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm,” 2011 IEEE Custom Integrated Circuits Conference (CICC), 2011.

[2] - R. E. Pino, H. (H. Li, Y. Chen, M. Hu, and B. Liu, “Statistical memristor modeling and case study in neuromorphic computing,” Proceedings of the 49th Annual Design Automation Conference on - DAC '12, 2012.

[3] - M. J. Sharifi and Y. M. Banadaki, “General Spice Models For Memristor And Application To Circuit Simulation Of Memristor-Based Synapses And Memory Cells,” Journal of Circuits, Systems and Computers, vol. 19, no. 02, pp. 407–424, 2010.

[4] - B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, “Training itself: Mixed-signal training acceleration for memristor-based neural network,” 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), 2014.

[5] - J. Mnatzaganian and R. Reinhardt, “Memristors and Neuromorphic Computing.” 13-Nov-2012.

[6] - I. Vourkas and G. C. Sirakoulis, “Memristor-Based Nanoelectronic Computing Circuits and Architectures,” Emergence, Complexity and Computation, pp. 9–28, 2016.

[7] - J. Gu and J. Li, “Exploration of self-healing circuits for timing resilient design using emerging memristor devices,” 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 2015.

[8] - C. Yakopcic, “MEMRISTOR DEVICE MODELING AND CIRCUIT DESIGN FOR READ OUT INTEGRATED CIRCUITS, MEMORY ARCHITECTURES, AND NEUROMORPHIC SYSTEMS,” May 2014.