Memristor Crossbar Deep Network Implementation Based on a Convolutional Neural Network

Chris Yakopcic, Md Zahangir Alom, and Tarek M. Taha
Department of Electrical and Computer Engineering
University of Dayton, Dayton, OH
{cyakopcic1, alomm1, tarek.taha}@udayton.edu

***Abstract*—This paper presents a simulated memristor crossbar implementation of a deep Convolutional Neural Network (CNN). In the past few years deep neural networks implemented on GPU clusters have become the state of the art in image classification. They provide excellent classification ability at the cost of a more complex data manipulation process. However once these systems are trained, we show that the analog crossbar circuits in this paper can highly parallelize the recognition phase of a CNN algorithm. One of the drawbacks of using memristors to carry out computations is that the data stored will likely have less precision when compared to typical 32-bit floating point memory. However, we show the proposed system is capable of operating with zero loss in classification accuracy if the memristors utilized are able to store at least 16 unique values (essentially acting as 4-bit devices). To the best of our knowledge, this is the first paper that presents a memristor based circuit for implementing CNN recognition. This is also the first paper that provides a circuit for precise memristor based analog convolution.**

Keywords—memristor; convolution; CNN; neural network; deep network

I. Introduction

The main obstacles for continued performance improvement in future computing systems are reliability and power consumption. Embedded neuromorphic processing systems have significant advantages to offer, such as the ability to solve complex problems while consuming very little power and area [1,2]. These neural network accelerators can be used for a broad number of applications [3]. In the past few years deep neural networks implemented on GPU clusters have become the state of the art in image classification [4]. Additionally, Chen et al. [5] have shown that Recognition, Mining, and Synthesis (RMS) applications (described by Intel as the key application drivers of future [3]) can be performed using neural networks.

The memristor [6] has received significant attention as a potential building block for neuromorphic systems. The physical realization of the memristor [7] yields a nanoscale non-volatile device with a large programmable resistance range. Voltage pulses can be applied to memristors to alter their conductivity and tune them to a specific resistance state. Physical memristors can be laid out in a high density grid known as a crossbar [8]. Using this layout, memristors have the potential to be fabricated with a synaptic density greater than that of brain tissue [9]. Systems based on these circuits will produce high density, extreme low-power, neuromorphic hardware that is capable of performing many multiply-add operations in parallel in the analog domain.

Both memristor-based [10-24] and non-memristor based [25-27] wafer-scale hardware implementations of neural networks have been proposed. Related works [24] have proposed methods for developing a memristor based Multilayer Perceptron (MLP), and a Restricted Boltzmann Machine (RBM) [12,23]. Some of these systems are capable of performing in-situ training [16,17], meaning that the learning algorithm is contained in the hardware, and software is not required to predetermine the weight values for the memristor crossbar. Other systems are based on ex-situ training where a leaning algorithm generates the weights in software, and then these values are directly programmed into the crossbar [12,16].

In the system we are proposing, the memristor conductance values are predetermined using an ex-situ training process. The key benefit of ex-situ training is that any training algorithm can be used to learn the input data set. If a core is to be reprogrammed multiple times for different applications, ex-situ training is useful since any existing set of weights can be directly downloaded. In the in-situ training process each chip may need a lengthy training process, and only the training algorithm on the chip can be used. We use a feedback process to program these crossbars to ensure that memristor devices (which commonly exhibit some degree of stochastic behavior when switching [28]) are correctly programmed.

In this paper we have built on previous memristor based neuromorphic hardware [11,12] to show that memristor crossbar circuits are capable of implementing an ex-situ Convolutional Neural Network (CNN). The circuits used in this paper are based on an improvement of our previous designs [12], and in addition to mapping weight matrices to crossbars, we are also using them to perform multiple parallel analog convolution operations. To the best of our knowledge, no other paper has presented a detailed circuit that is capable of performing analog convolution using a memristor crossbar.

Using memristors to both process and store data in this system provides a significant throughput advantage, and in the following sections we show that a memristor crossbar is capable of performing $N \times M$ convolution operations in parallel

978-1-5090-0620-5/16/$31.00 ©2016 IEEE

(where N is equal to the number input maps and M is equal to output maps in a given layer in the CNN system).

In this paper we utilize a weight programming scheme [12] that uses a D-to-A for programming, and we also study the minimum required bit-width and precision of the D-to-A circuits. Thus we not only have a way of determining the optimal memristor conductance values, we also have a method for programming these values into a crossbar. While related works [14,24] have used summing amplifier techniques to carry out analog multiply-adds in a memristor crossbar, this paper uses a method for weight conversion that allows us to obtain the expected result precisely and reliably. Others have proposed alternative transformation strategies for memristor crossbars [21], but not to achieve versatile neural network functionality based on accurate dot products and convolutions. In general, this paper presents a complete, ex-situ, memristor based neuromorphic system that is capable of implementing the recognition phase of a CNN deep learning algorithm.

II. Convolutional Neural Networks

The CNN network structure was first proposed by Fukushima in 1980 [29]. However, it has not been widely used because the training algorithm was difficult to implement. In the 1990s, LeCun *et al.* applied a gradient-based learning algorithm to a CNN and obtained successful results [30]. After that, researchers further improved the CNN and reported strong results in the field of image recognition. Recently, Cireşan *et al.* presented multi-column CNNs to recognize digits, alpha-numerals, Chinese characters, and traffic signs [4,31]. CNNs have also been designed to imitate human visual processing [32], and have highly optimized structures to process 2D images [32]. Furthermore, a CNN can effectively learn the extraction and abstraction of 2D features [33]. In addition to the common advantages of deep neural networks, the CNN has additional desirable properties. Composed of sparse connections with tied weights, the CNN has significantly fewer parameters than a fully connected network of a similar size. Furthermore, a CNN is trainable with a gradient-based learning algorithm, and suffers less from the diminishing gradient problem [30]. Given that a gradient-based algorithm trains the whole network to minimize an error criterion directly, a CNN can produce highly optimized weights.

Fig. 1 shows that the overall architecture of the CNN consists of two main parts, the feature extractor and classifier. In the feature extraction layers, each layer of the network receives an input from the immediate previous layer. The CNN architecture consists of a combination of three types of layers: convolution, subsampling, and classification. The convolution and subsampling layers alternate, and are found in the feature extraction section of the network. The outputs of the convolution and subsampling layers are organized into multiple 2D planes known as feature maps. Convolution layers extract the features from the input images using convolution operations, and the subsampling layers abstract the feature maps through an averaging filter.

As the features propagate through the network, the size of the features is reduced (in terms of pixels) depending on the size of the convolutional and subsampling kernels respectively. However, the number of feature maps is also increased so the network can determine the most suitable features of the input images for better classification accuracy. The outputs of the last layer of the CNN are then input to a fully connected network, which is called the classification layer. A feed-forward neural network is used as a classifier in this work because it has been shown to provide the best performance compared to other options [34,35]. In the classification layer, the output node with the highest value designates the class assigned to a given input image.

III. Memristor Convolution in a CNN

This section describes how a single column within a memristor crossbar can be used to perform a convolution operation. Contrary to commonly studied convolution operations, the convolution used in this CNN network differs in a few ways. First this CNN does not require the input signal (x) to be zero-padded before it is applied to the convolution kernel (k). Therefore, the length of the resulting signal is equal to the length of x, minus the length of k plus 1. In MATLAB this could be achieved using the '`valid`' flag within a convolution function. In the memristor crossbar designs this is achieved by simply not zero-padding the input signal.

Additionally, each output sample generated using the convolution function is squashed using a sigmoid activation function (after accounting for an additive bias). The circuit in Fig. 2 was designed to account for this after it performs the convolution. Fig. 2 (a) shows the circuit diagram in detail and

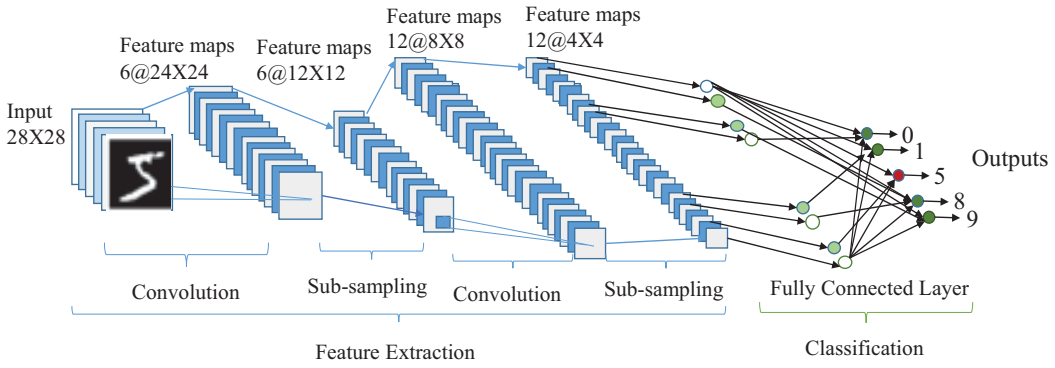

Fig. 1. CNN Block Diagram

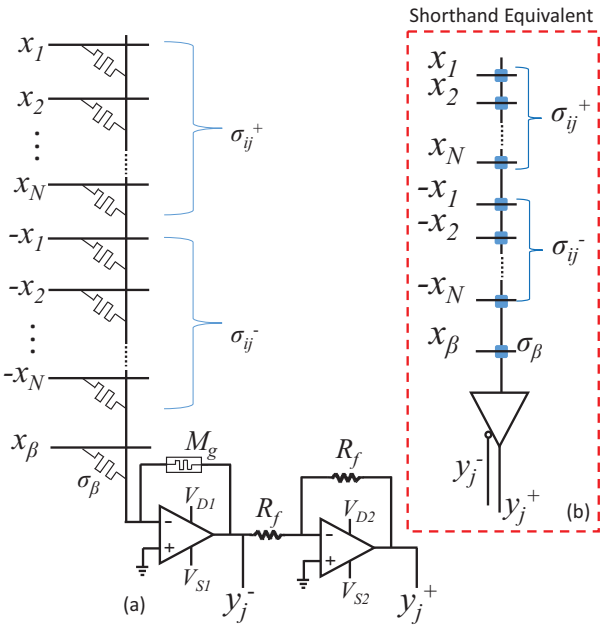

Fig. 2. (a) Circuit diagram displaying a crossbar column circuit that is capable of performing convolution, and (b) the equivalent shorthand depiction for this circuit that is used throughout the paper. In (a), V_{S1}=-1V, V_{D1}=-0V, V_{S2}=0V, V_{D2}=1V, M_g is a memristor used to control feedback gain based on its resistance R_g, σ_β is the conductance of the memristor M_β used to implement an additive bias along with the input x_β, R_f is a resistance used to implement unity gain, the terms σ_{ij}^+ and σ_{ij}^- represent the arrays of conductance values described in equations (3) and (4), and y_j^+ and y_j^- are the pair of positive and negative outputs generated by each convolution circuit in a memristor crossbar.

Fig. 2 (b) displays a more compact shorthand depiction that will be used throughout this paper.

The circuit in Fig. 2 assumes that the convolution kernels required for this system have already been determined in software using an ex-situ training process. Also, some extra manipulation must be completed so that these 2D kernels can be stored as memristor resistance values. For example, if you were to store the 3×3 kernel in equation (1) in a crossbar column, it would have to be arranged so that each input value was aligned with the correct kernel value. Furthermore, the kernel array is converted into two arrays so that the memristor crossbar can easily account for kernels that have both positive and negative values. This conversion process is also shown in equation (1). Equation (2) shows how an image segment must be converted into two arrays of identical values with reversed sign. Finally, equations (3) and (4) show how the kernel arrays must be converted to conductivity values that fall within the bounded range of a memristor crossbar. This conversion (in conjunction with the op-amp circuit at the output of each crossbar column) makes a convolution operation possible that produces the exact same result (± some analog circuit error) that would be produced by a digital software equivalent.

The op-amp circuit at the crossbar column output in Fig. 2 is used to both scale the output voltage (thus returning the output to a numerical value after the MAC operations are completed in the conductivity domain) and implement the sigmoid activation function. This is an alternative approach to the systems that require a sigmoid activation circuit following an op-amp output such as [14,24]. To accomplish this, the amplifier at the output of the circuit is used to create a pseudo sigmoid function as shown in Fig. 3. A linear amplifier transfer function (with bounded upper and lower voltage rails) matches the sigmoid relatively closely (see eq. (5)), and the simulation results show that this is an effective alternative. To obtain the optimal linear fit of the sigmoid function m = 1/4 and b = 1/2.

$$k_{ex} = \begin{bmatrix} 0.1 & -0.2 & 0.3 \\ -0.4 & 0.5 & -0.6 \\ 0.7 & -0.8 & 0.9 \end{bmatrix} \rightarrow \begin{bmatrix} 0.9 \\ -0.6 \\ 0.3 \\ -0.8 \\ 0.5 \\ -0.2 \\ 0.7 \\ -0.4 \\ 0.1 \end{bmatrix} \rightarrow \begin{matrix} k_{ex}^+ \\ \begin{bmatrix} 0.9 \\ 0 \\ 0.3 \\ 0 \\ 0.5 \\ 0 \\ 0.7 \\ 0 \\ 0.1 \end{bmatrix} \end{matrix} \quad \begin{matrix} k_{ex}^- \\ \begin{bmatrix} 0 \\ 0.6 \\ 0 \\ 0.8 \\ 0 \\ 0.2 \\ 0 \\ 0.4 \\ 0 \end{bmatrix} \end{matrix} \tag{1}$$

$$x_{ex} = \begin{bmatrix} 0 & 0.5 & 0.3 \\ 0.5 & 0.8 & 0.5 \\ 0 & 0.5 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0.5 \\ 0.3 \\ 0.5 \\ 0.8 \\ 0.5 \\ 0 \\ 0.5 \\ 0 \end{bmatrix} \rightarrow \begin{matrix} x_{ex} \\ \begin{bmatrix} 0 \\ 0.5 \\ 0.3 \\ 0.5 \\ 0.8 \\ 0.5 \\ 0 \\ 0.5 \\ 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} -x_{ex} \\ \begin{bmatrix} 0 \\ -0.5 \\ -0.3 \\ -0.5 \\ -0.8 \\ -0.5 \\ 0 \\ -0.5 \\ 0 \end{bmatrix} \end{matrix} \tag{2}$$

$$\sigma^+ = \frac{(\sigma_{\max} - \sigma_{\min})}{\max(|W|)} W^+ + \sigma_{\min} \tag{3}$$

$$\sigma^- = \frac{(\sigma_{\max} - \sigma_{\min})}{\max(|W|)} W^- + \sigma_{\min} \tag{4}$$

$$y_j^+ = \begin{cases} 1, & v > 2 \\ mv + b, & |v| \le 2 \\ 0, & v < -2 \end{cases} \approx \frac{1}{1 + e^{-v}} \tag{5}$$

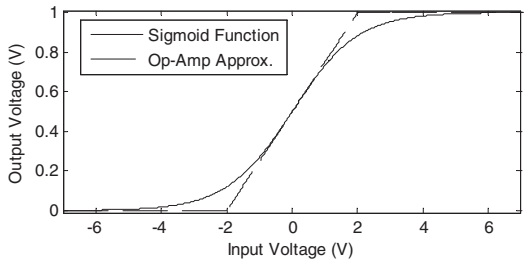

Fig. 3. Comparison of the sigmoid transfer function and the amplifier alternative (the first op-amp stage will actually produce an inverted result, but this will be the eventual outcome seen at the y_j^+ output).

The complete circuit in Fig. 2 will operate according to equations (6) through (8). The op-amp producing y_j^- output acts as a summing amplifier that also accounts for the slope and bias required of the approximate sigmoid activation

function in eq. (5). The y_j^- output is then fed into a unity gain inverting amplifier to obtain the y_j^+ output.

In eq. (6) output of the summing amplifier is determined. The voltage $x_\beta = 1$ V is used to shift the starting position of the sigmoid (b=1/2) and to control the additive bias value (β) in the CNN algorithm. The resistance R_g is the resistance of the programmable gain memristor M_g. The value σ_β is the conductance of the memristor M_β and $\sigma_\beta=(b+\beta/4)/R_g$. In eq. (7), R_g is set so that accumulation of conductance and voltage pairs is multiplied by the inverse of the scaling factor in equations (3) and (4), in addition to the slope of the activation function m. Using this method, the resulting outputs y_j^+ and y_j^- will now have a value equal to that of the convolution operation performed in the software implementation. Using this method, the outputs are already formatted for the next layer of the CNN system where y_j^+ becomes the set of inputs for σ_{ij}^+ and y_j^- becomes the set of inputs for σ_{ij}^-

$$y_j^- = -R_g\left(\sum_{i=1}^{N}\left[x_i\sigma_{ij}^+ - x_i\sigma_{ij}^-\right] + x_{N+1}\sigma_b\right) \tag{6}$$

$$R_g = m\frac{\max(W)}{(\sigma_{\max}-\sigma_{\min})} \tag{7}$$

$$y_j^+ = -\frac{R_f}{R_f}y_j^- = -y_j^- \tag{8}$$

IV. Crossbar Programming

The previous section discusses how to obtain the correct conductivity values required by a memristor crossbar. This section shows how these values can be physically programmed into a crossbar. An idealized memristor device has a continuous bounded programmable resistance range, but there appears to be a limit to how many discrete states can be programmed into a memristor [36,37]. There is also a certain amount of stochastic resistance change present when attempting to program a device to a specific resistance [23,38].

The circuit used to implement this programming method was first proposed in [12]. To program each of the weights in this memristor crossbar, only one target row input x_t (where t=1,…,N+1) is active (set to 1V) and all other inputs are set to 0V. This will reduce the summing amplifier to a simple inverting amplifier. Also, the second inverting amplifier that produces y_j^+ can be ignored during the write process. In this circuit, the feedback memristor M_g is held at a constant value σ_{MAX} (or R_{MIN}). Using this approach, when the target memristor being programmed (M_t) is set to σ_{MAX}, the output y_j^- = – 1 V. Likewise, when the target memristor M_t is set to σ_{MIN}, the output $y_j^-\approx 0$ V. This method allows for a linear mapping between sensed voltage and target memristor conductivity. In our study we use 8×10^{-9}S for σ_{min} and 8×10^{-6}S for σ_{max} according to the memristor characterization data in [39].

The schematic in Fig. 4 displays the entire programming circuit in addition to the equivalent circuit for the active part of the memristor crossbar when programming a single memristor. This circuit utilizes two D-to-A converters to implement a bounded voltage range for successful programming. It is assumed that these D-to-A converters have access to the weight values produced by the CNN algorithm in software. Also, it is assumed that the software is able to convert each floating point weight (or kernel element) to a value within a set of 2^B predefined conductivity states, where B corresponds to the bit width of the D-to-A used. This will essentially convert the weights from floating point to much lower resolution values. In the following sections, we show that a 4 bit D-to-A is sufficient for successful programming with no reduction in accuracy.

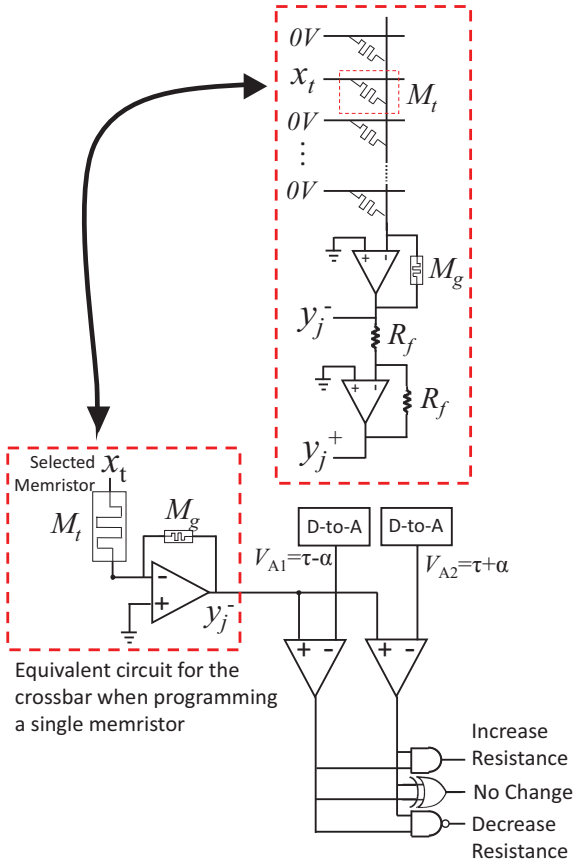

Fig. 4. Circuit used to program a single memristor to a target resistance.

For the programming process to determine that a memristor is programmed, y_j^- must be greater than τ-α and less than τ+α. A digital logic layer determines whether the memristor conductivity should be increased or decreased based the output voltage of the two comparators. If resistance is to be decreased, a positive write pulse will be applied to the memristor M_t, and if the resistance is to be increased, a negative pulse will be applied. This process will be repeated until the XOR gate in Fig. 4 has a high output, signifying a successfully programmed device. A pulse width known to be much smaller than that required to fully switch the memristor is used to program these memristors to these specific resistances. This is done to induce very small amounts of resistance change [40]. This iterative process is what drives the feedback programming system.

V. CNN Recognition System

This section describes in detail how this memristor based recognition system for the CNN operates. The recognition algorithm is organized according to the flowchart in Fig. 5. The layer numbers were kept equivalent to the original CNN algorithm [41] that was modified for the memristor system. Therefore layer 1 (l=1) holds a testing set of 500 MNIST images [41], and layer 2 (l=2) is the first convolution layer.

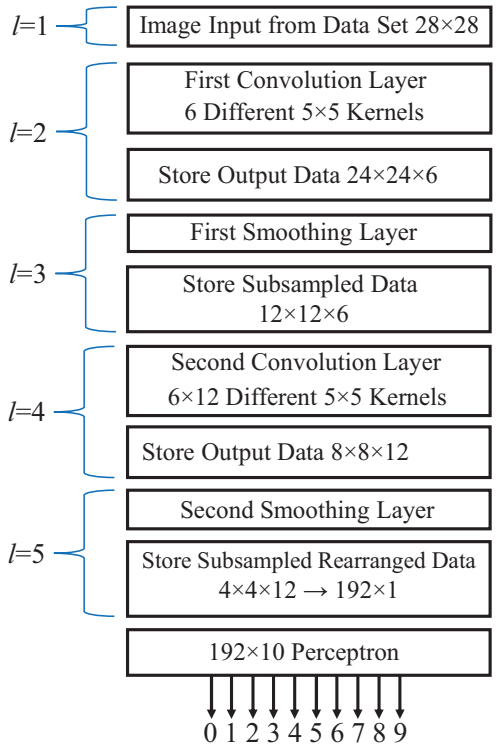

Fig. 5. Flowchart describing the CNN recognition system.

A. First convolution layer (l=2)

In this layer, an input image will be convolved with six different 5×5 kernels. Therefore the output of this step results in a data array that has a size of 24×24×6. Since we are using a memristor crossbar to perform the convolution operations, we can generate all six of these output maps in parallel. The crossbar used to perform this operation is shown in Fig. 6. Each target image will be applied at the input rows $x_1,\ldots,x_{25}$ and $-x_1,\ldots,-x_{25}$. Each image contains 784 pixels, but the image is applied 25 pixels at a time where each 25-pixel section generates a single output value. Therefore this layer requires a 51×6 memristor crossbar. As described in the previous sections, the kernel matrix obtained in software is broken into two arrays in the memristor crossbar to easily account for negative values in the kernel arrays.

B. First Smoothing Layer (l=3)

Following the first convolution layer, a smoothing (and subsampling) operation is performed on the six generated feature maps. This is just a simpler convolution operation where the single kernel applied to each feature map is set according to equation (9).

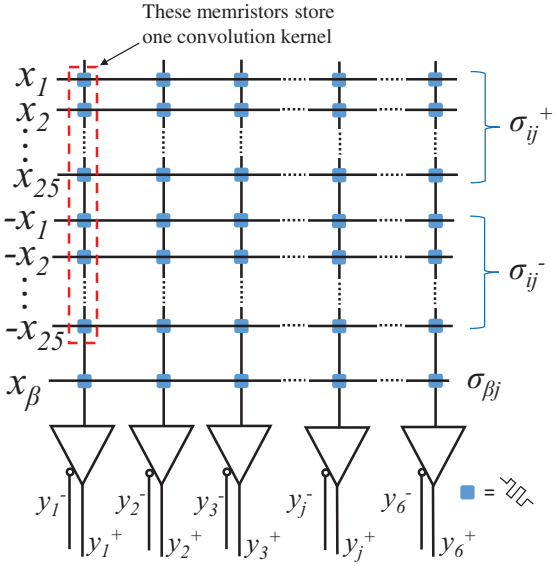

Fig. 6. Circuit used to perform the convolution operations for the second (l=2) in the CNN recognition system.

$$k = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix} \tag{9}$$

It would be desirable to process all six feature maps in parallel, but this cannot be done with a single crossbar since different input values must be applied to each column. To keep the processing throughput relatively equal between layers, we can just separate this operation into six smaller crossbars as in Fig. 7. In this specific example, the conductivities (σ_{ij}^-) corresponding to negative elements in the kernel matrix are meaningless. However, utilizing the same convolution circuit described in Fig. 1 allows for more flexibility so more complex smoothing filters could be implemented if desired.

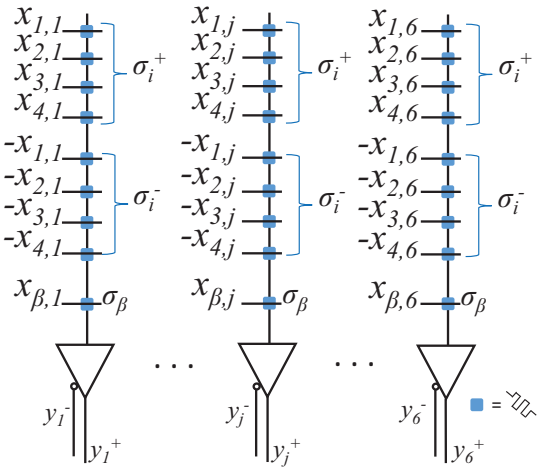

Fig. 7. The group of convolution circuits that is used to implement the smoothing operation.

Following each of the smoothing crossbars, a subsampling operation is performed that reduces the size of each feature map by a square factor of 2. This could be designed in a number of ways, but one simple approach would be to place a

single-bit counter on the memory array where the data output from the smoothing operation is stored. The memory address would only update for every other sample (whenever the counter output is high) so all unwanted data would be overwritten during the smoothing step. The resulting dataset at the end of this operation is an array of size 12×12×6, and this layer requires six 9×1 memristor crossbars.

C. Second Convolution Layer (l=4)

Following the smoothing/subsampling operation is another convolution operation that is similar to the one carried out in the second layer (l=2). However, there are now six input maps instead of just one, and there are twelve output maps (instead of six). Therefore, the data array generated at the output of this layer will have a size of 8×8×12. This layer performs 12 convolutions on each of the 6 input maps, so a total of 72 convolutions are performed (using 72 different 5×5 kernels). To obtain a final result of only 12 output maps (instead of 72) 12 groups of 6 output maps are summed together before the sigmoid activation function is applied. These computations can be completed very efficiently with a single memristor crossbar.

In this layer, the CNN software implementation is carrying out equation (10) where i=1,..,N and N is equal to the number of input maps (N=6). Likewise, j=1,..,M and M is equal to the number of output maps (M=12). Each X_i denotes an entire input array containing 25 elements, and each K_{ij} denotes an entire kernel array containing 25 elements.

$$v_j = (X_1 * K_{1j}) + (X_i * K_{ij}) + \ldots + (X_N * K_{Nj}) \qquad (10)$$

Since convolution (denoted by $*$) is a linear operation we can rearrange this equation so that the entire process can be completed in a single crossbar column as in equation (11).

$$v_j = [X_1,\ldots,X_i,\ldots,X_N] * [K_{1j},\ldots,K_{ij},\ldots,K_{Nj}] \qquad (11)$$

Each column in the crossbar will generate a different output map, where each output map is the result of summing the convolution of six different input maps with their corresponding kernels. This layer demonstrates one of the biggest advantages to the memristor based CNN design since a single crossbar can carry out M×N convolutions in Y cycles, where Y is equal to the total number of samples in the convolution output map. Fig. 8 shows the circuit that is used to complete this operation. Each column in the crossbar holds 6 kernels each stored in a group of 50 memristors (plus a single bias memristor). The 12 crossbar columns simultaneously generate the samples required to construct the 12 output maps. Thus, this crossbar must contain 301×12 memristors.

D. Second Smoothing Layer (l=5)

Following the second convolution layer, another smoothing and subsampling layer is applied (that will further reduce the size of the data array). The circuit used to perform this operation will be nearly identical to that displayed in Fig. 7, although 12 parallel single column crossbars will be required since this layer processes 12 output maps instead of 6. The resulting array size after the subsampling layer is 4×4×12. This layer requires twelve 9×1 crossbars.

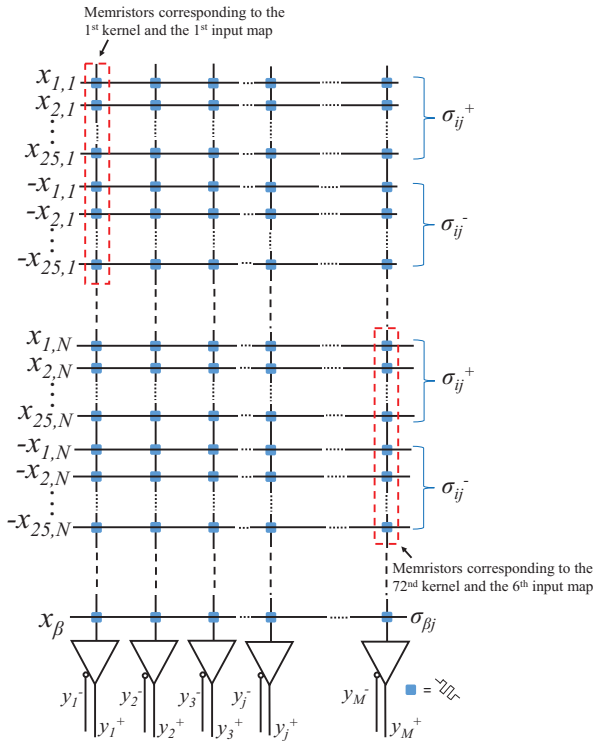

Fig. 8. The circuit that is used to implement the second convolution layer.

E. Classification Layer

Following the convolution and smoothing layers in the CNN layout, a single layer perceptron is used to classify the resulting feature maps. To complete this, the perceptron has 192 inputs (one input for each of the 16 values in each of the 12 output maps) and 10 outputs (one for each MNIST digit). The circuit used to complete this operation can be seen in Fig. 9. It's very similar to the convolution circuit, except that it's completing a series of multiply add operations commonly seen in a perceptron network. The main difference between this circuit and the one used to implement convolution is how the resistances in the memristor grid are organized. This circuit is used to store a weight matrix, and the convolution circuits are storing a set of convolution kernel arrays. The crossbar in this layer requires 193×10 memristors.

F. Digital Storage Layers

Fig. 5 shows that digital storage layers were placed at the output of each convolution. This has the disadvantage of requiring an A-to-D and D-to-A step between each crossbar, although this may also reduce the amount of analog circuit error that is transmitted between layers. A complex data controller could significantly reduce the amount of memory required. However, it is likely that we would still need some form of storage between layers to store at least the input set required for each subsequent convolution. Therefore, we chose to store an entire image between layers because any benefit gained by a systematically reduced memory size would likely be outweighed by the complexity of a data controller of this nature. Alternatively, a highly parallel system (with many crossbars in each layer) may eliminate the need for storage between layers. This will be investigated as future work.

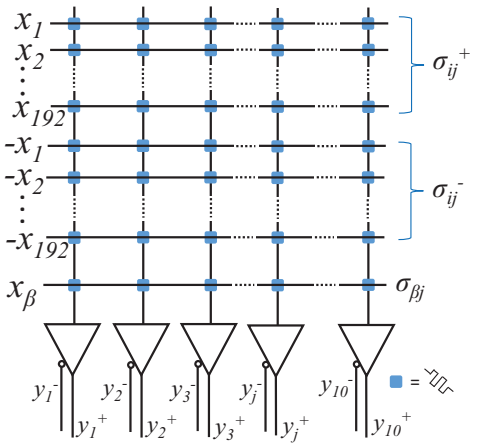

Fig. 9. Circuit that is used to implement the final perceptron layer of the CNN recognition system.

VI. Simulation Results

The first step in the simulation process is to train the CNN completely in software to generate the weights and kernel values that will be programmed into the memristor crossbar recognition system. This was completed by training with a subset of 10,000 images from the MNIST handwritten digit database for 10 epochs. Fig. 10 shows the error minimization pattern for this process. Only 10 epochs were used to train, but each epoch iterates through 200 batches of 50 images, so the error minimization shows a total of 2,000 iterations.

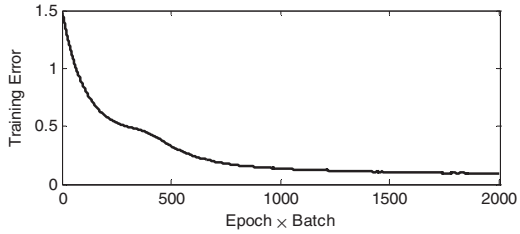

Fig. 10. Error present when training the CNN algorithm is software. This step is required to obtain the weights that will be transferred to the memristor crossbars.

Testing the CNN algorithm purely in software under these training conditions results in 92% classification accuracy. The next step in the simulation process is to test the accuracy of the memristor based CNN recognition system described in the previous section. When testing the simulated memristor crossbars, an accuracy of 91.8% was achieved.

One of the potential limitations of this system is the stability and accuracy of the memristors that store the kernel and weight values. The above testing results were completed assuming that each memristor could be set to one of 2^{12} possible values, and that they could be programmed to an accuracy within 1mV of a target (see Section IV). This means that virtually no memristor variation effects were considered when obtaining this classification accuracy, and that we are assuming the existence of extremely stable devices.

Fig. 11 shows how the CNN system is able to operate even when the reliability of the memristors in the system is reduced. This is achieved using two different methods. First the number of states that can be programmed in a memristor is reduced. Since the memristor conductivity values are first determined in software, the number of unique values that could possibly be programmed is limited by the width of the D-to-A in the programming circuit (see Fig. 4). Fig. 11 shows that this system will operate with virtually no reduction in accuracy as long as each memristor can be programmed with at least 16 unique values. This is promising since previous results show that is it possible to program at least 128 different states within a single memristor [36]. The second way simulated memristor reliability is reduced is by increasing the error between the target conductivity value to be programmed and the actual memristor conductivity. Since the feedback programming circuit in Fig. 4 tracks a voltage to determine if programming is successful, the programming precision value α is measured in volts (not conductance). Fig. 10 shows that there is no real reduction in accuracy unless the programming error measured is greater than 10mV. Furthermore, a slightly looser system is demonstrated that utilizes only 4 unique conductance states programmed to within 100mV of their target values, and it can still provide a recognition accuracy of 88%.

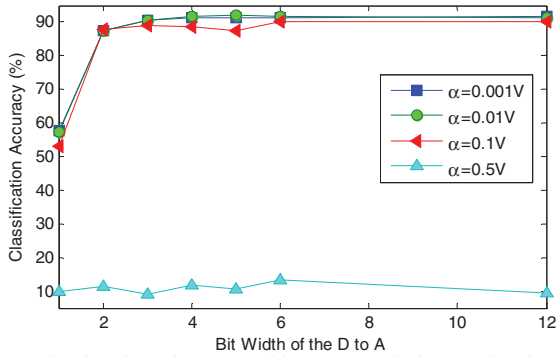

Fig. 11. Plot that shows how recognition accuracy is decreased as the number of programmable states (or the programming accuracy) in the memristors in the crossbar is reduced.

VII. Conclusion

This paper presents one of the first memristor based CNN systems, and we believe that it is the first that has demonstrated a complete circuit level design. It shows that groups of summed convolution operations can all be completed in a single crossbar which provides the ability to perform $N \times M$ convolutions in parallel. The results also show that 32-bit floating point accuracy is not a requirement when programming the memristors. This is a significant benefit to designing these systems with memristors, since existing memristor technology is prone to inter and intra-device variation.

There are several aspects about this system that can be investigated as future work. Most immediately we plan on studying the most efficient design for the digital storage layers between the memristor crossbars in the recognition system. Furthermore, we would like to see if a larger completely parallel system is feasible. We also plan on estimating the energy consumption and throughput of this system and comparing it to the competitive alternatives. In general this paper demonstrates promising results that show memristors can be used to efficiently design a compact CNN recognition architecture.

REFERENCES

[1] T. M. Taha, R. Hasan, C. Yakopcic, and M. R. McLean, "Exploring the Design Space of Specialized Multicore Neural Processors," IEEE International Joint Conference on Neural Networks (IJCNN), August 2013.

[2] B. Belhadj, A. J. L. Zheng, R. Héliot, and O. Temam. "Continuous real-world inputs can open up alternative accelerator designs," ISCA 2013.

[3] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," Technology@Intel Magazine, Feb. 2005.

[4] Dan Cireşan and J. Schmidhuber, "Multi-column Deep Neural Networks for Offline Handwritten Chinese Character classification," IDSIA Technical Report, No. IDSIA-05-13, 2013.

[5] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, O. Temam, "BenchNN: On the Broad Potential Application Scope of Hardware Neural Network Accelerators," IEEE International Symposium on Workload Characterization (IISWC), November 2012.

[6] L. O. Chua, "Memristor—The Missing Circuit Element," IEEE Transactions on Circuit Theory, 18(5), 507–519 (1971).

[7] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing Memristor found," Nature, 453, 80–83 (2008).

[8] S. H. Jo, K.-H. Kim, and W. Lu, "High-Density Crossbar Arrays Based on a Si Memristive System" Nano Letters, 9(2), 2009, pp. 870-874.

[9] G. S. Snider, "Cortical Computing with Memristive Nanodevices," SciDAC Review, (2008).

[10] C. Yakopcic, R. Hasan, T. M. Taha, M. McLean, and D. Palmer, "Memristor-based neuron circuit and method for applying a learning algorithm in SPICE," IET Electronics Letters, vol.50, no.7, pp.492,494, 2014.

[11] C. Yakopcic and T. M. Taha, "Energy efficient perceptron pattern recognition using segmented memristor crossbar arrays," in Neural Networks (IJCNN), The 2013 International Joint Conference on, pp. 1–8, IEEE, 2013.

[12] C. Yakopcic, R. Hasan, and T. M. Taha, "Memristor Based Neuromorphic Circuit for Ex-Situ Training of Multi-Layer Neural Network Algorithms," IEEE IJCNN, 2015.

[13] D. Soudry, D. D. Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training," IEEE Trans. on Neural Networks and Learning Systems, issue 99, 2015.

[14] Boxun Li; Yuzhi Wang; Yu Wang; Chen, Y.; Huazhong Yang, "Training itself: Mixed-signal training acceleration for memristor-based neural network," Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific , vol., no., 20-23 Jan. 2014.

[15] D. Chabi, W. Zhao, D. Querlioz, J.-O. Klein, "Robust Neural Logic Block (NLB) Based on Memristor Crossbar Array" IEEE/ACM International Symposium on Nanoscale Architectures, pp.137-143, 2011.

[16] F. Alibart, E. Zamanidoost, and D.B. Strukov, "Pattern classification by memristive crossbar circuits with ex-situ and in-situ training", Nature Communications, 2013.

[17] R. Hasan and T. M. Taha, "Enabling Back Propagation Training of Memristor Crossbar Neuromorphic Processors," IEEE International Joint Conference on Neural Networks, 2014.

[18] M. Hu, H. Li, Q. Wu, G.S. Rose, and Y. Chen, "Memristor crossbar based hardware realization of BSB recall function," in The 2012 International Joint Conference on Neural Networks (IJCNN), pp.1-7, 10-15 June 2012

[19] J. H. Lee and K. K. Likharev, "Defect-tolerant nanoelectronic pattern classifiers," International Journal of Circuit Theory and Applications, vol. 35, no. 3, pp. 239–264, 2007.

[20] D. Chabi, D. Querlioz, W. Zhao, and J. Klein, "Robust learning approach for neuro-inspired nanoscale crossbar architecture," JETC, vol. 10, no. 1, p. 5, 2014.

[21] 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 Trans. Neural Netw. Learning Syst., vol. 25, no. 10, pp. 1864–1878, 2014.

[22] J. Starzyk and Basawaraj, "Memristor crossbar architecture for synchronous neural networks," IEEE Trans. Circ. Syst. - I, vol. 61, pp. 2390 – 2401, 2014.

[23] A. M. Sheri, A. Rafique, W. Pedrycz, M. Jeon, "Contrastive divergence for memristor-based restricted Boltzmann machine" Engineering Applications of Artificial Intelligence 37(2015) 336-342

[24] I. Kataeva, F. Merrikh-Bayat, E. Zamanidoost and D. Strukov, "Efficient Training Algorithms for Neural Networks Based on Memristive Crossbar Circuits", IEEE International Joint Conference on Neural Networks (IJCNN), 2015.

[25] J. Schemmel, J. Fieres, K. Meier, "Wafer-Scale Integration of Analog Neural Networks", IEEE International Joint Conference on Neural Networks (IJCNN), 2008.

[26] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," IEEE Custom Integrated Circuits Conference (CICC), vol., no., pp.1-4, 19-21 Sept. 2011.

[27] J. V. Arthur, P. A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. B. D. Rubin, R. Manohar, D. S. Modha, "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," The International Joint Conference on Neural Networks (IJCNN), pp.1-8, June 2012.

[28] W. Yi, F. Perner, M. S. Qureshi, H. Abdalla, M. D. Pickett, J. J. Yang, M.-X. M. Zhang, G. Medeiros-Ribeiro, R. S. Williams, "Feedback write scheme for memristive switching devices," Appl Phys A (2011) 102: 973–982, DOI 10.1007/s00339-011-6279-2.

[29] K. Fukushima, "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Biological Cybernetics, vol. 36, no. 4, pp.193–202, 1980.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] D. C. Cireşan, U. Meier, J. Schmidhuber, "Multi-column Deep Neural Networks for Image Classification," IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2012.

[32] "Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation".DeepLearning 0.1. LISA Lab. Retrieved 31, August 2013

[33] Matt Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks" Accepted and published in ECCV, 2014.

[34] Mohamed, Abdel-rahman, George E. Dahl, and Geoffrey Hinton. "Acoustic modeling using deep belief networks,"Audio, Speech, and Language Processing, IEEE Transactions on 20.1 (2012): 14-22.

[35] V. Nair and G. Hinton, Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th International Conference on Machine Learning (ICML-10). 2010.

[36] F. Alibart, L. Gao, B. Hoskins, and D.B. Strukov, "High-precision tuning of state for memristive devices by adaptable variation-tolerant algorithm", Nanotechnology 23, art. 075201, 2012.

[37] A. S. Oblea, A. Timilsina, D. Moore, and K. A. Campbell, "Silver chalcogenide based memristor devices," in Proc. Int. J. Comp. Netw., Oct. 2010, pp. 1–3.

[38] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," Nano Letters, vol. 10, pp. 1297–1301, Mar. 2010.

[39] W. Lu, K.-H. Kim, T. Chang, S. Gaba, "Two-terminal resistive switches (memristors) for memory and logic applications," in Proc. 16th Asia and South Pacific Design Automation Conference, 2011, pp. 217-223.

[40] C. Yakopcic and T. M. Taha, "Determining optimal switching speed for memristors in a neuromorphic system," Electronics Letters, 51(21), 2015.

[41] Palm, Rasmus Berg. "Prediction as a candidate for learning deep hierarchical models of data." Technical University of Denmark (2012).