Extremely Parallel Memristor Crossbar Architecture for Convolutional Neural Network Implementation

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 based Convolutional Neural Network (CNN). Deep networks implemented on GPU clusters have become the state of the art in providing excellent classification ability, at the cost of a more complex data manipulation process. In this work we show that once deep networks are trained, the analog crossbar circuits in this paper can parallelize the recognition phase of a CNN algorithm. With the massively parallel structure proposed in this work, we are able to generate multiple output feature maps in a single processing cycle. We show the proposed system is capable of operating with a negligible loss in classification accuracy if the memristors utilized are able to store at least 16 unique values (essentially acting as 4-bit devices). Furthermore, we quantify the amount of classification accuracy lost due to analog amplification in the signal path. To the best of our knowledge, this is the first paper that presents a memristor based circuit that is able to completely parallelize the CNN recognition process.**

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 [10-19] and non-memristor based [20,21] wafer-scale hardware implementations of neural architectures and deep networks have been proposed. Of these, few have studied the impact of using memristors to implement convolution [11,22] or develop a Convolutional Neural Network (CNN) [11,13,14]. Work in [13] proposes a mixed-signal memristor CNN that combines analog arithmetic with digital shift registers to increase output precision while keeping required memristor precision at a reasonable level. Work in [14] proposes using resistive memory in a neural accelerator both as high density storage, and to perform high throughput matrix multiplication. This design requires buffers to store the large amount of output data generated by the matrix multiplies and also utilizes significant digital routing.

Alternatively, our work aims to take full advantage of the parallel analog processing capabilities of memristors for maximum throughput. We show a method for kernel-to-crossbar mapping that allows for multiple output feature maps to be generated in a single processing cycle. For example, if a convolutional layer within the CNN architecture requires P input maps and generates Q output maps, the proposed architecture can be used to perform all $P\times Q$ convolutions on all pixels in each input map at the same time.

The obvious drawbacks to an architecture such as the one proposed are the number of memristors required to achieve the desired throughput, and the accuracy of the analog amplifiers between each layer in the CNN. Therefore, we present detailed studies that quantify how precisely memristors must be programmed, and we determine how much error can be present in the analog signal path before classification accuracy degrades.

In the system we are proposing, the memristor conductance values are predetermined using an ex-situ training process [12,16] (as opposed to in-situ, or on-chip learning [16,23]). 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. During programming, we use a feedback process to ensure that memristor devices (which commonly exhibit some degree of stochastic behavior when switching [24]) are correctly programmed.

This work builds on our previous memristor based neuromorphic hardware [10-12] and our prior work developing memristor based CNNs [11]. Our previous design [11] requires digital storage between each analog computation layer to hold output feature maps before the next analog layer is executed. In this work our goal was to study the impacts of bypassing digital hardware components (as used in [11,13,14]) to demonstrate use of the extremely parallel mathematical capabilities inherent in memristor crossbars.

II. The Convolutional Neural Network

A. Background

The CNN network structure was first proposed by Fukushima in 1980 [25]. 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 [26]. CNNs have been designed to imitate human visual processing [27], and their structures have been highly optimized to process 2D images [27]. 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 [26].

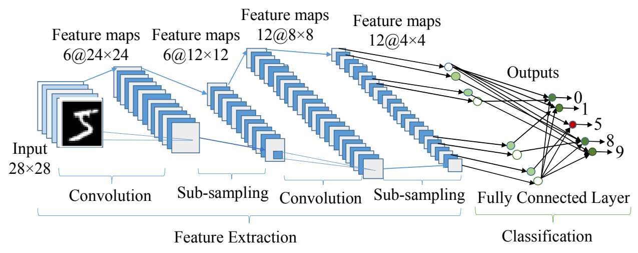


Fig. 1. CNN block diagram.

B. Network Architecture

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 classification layer. In the classification layer, the output node with the highest value designates the class assigned to a given input image.

C. Training

To acquire a baseline for testing the memristor implementations in the following sections, the CNN was first trained and tested completely in software. Training was completed using the entire training set of 60,000 images from the MNIST handwritten digit database for 100 epochs. Fig. 2 shows the error minimization pattern for this process. Only 100 epochs were used to train, but each epoch iterates through 1200 batches of 50 images, so the error minimization shows a total of 120,000 iterations. This training step generated the weights and kernel values that will be programmed into the memristor crossbar recognition system. Testing the CNN algorithm purely in software under these training conditions results in 98.92% classification accuracy. The following sections in this work describe how the memristor crossbar CNN architectures were implemented, and compare resulting classification accuracy to that of the software approach.

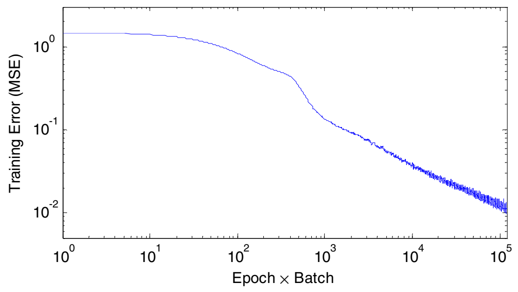


Fig. 2. Error present when training the CNN algorithm is software. This step is required to obtain the weights that will be transferred to the memristor crossbars.

III. Parallel Memristor Convolution in a CNN

Previous work suggests cycling an input image through a small memristor crossbar in segments to eventually obtain a complete output feature map [11] (based on the kernel stored in the crossbar). Alternatively, this paper describes a method for arranging a kernel in a memristor crossbar so that the entire output feature map can be obtained in one processing cycle. This eliminates the requirement of a data storage layer between each convolution layer because the system no longer needs to wait for an entire output feature map to be completed before the next layer can be executed.

Contrary to commonly studied convolution operations, the convolution used in this CNN network differs slightly. First, this CNN does not require the input signal (x) to be zero-padded before it is applied to the convolution kernel (k). In MATLAB this could be achieved using the `valid` flag within a convolution function. In our designs this is achieved by simply not zero-padding the input signal applied to the crossbar rows. Additionally, output samples generated using the convolution function are squashed using a sigmoid activation function (after accounting for an additive bias).

The circuit in Fig. 3 assumes that the convolution kernels have already been determined in software, but they must be arranged so that an entire feature map can be determined from a matrix multiply operation. To do this, the convolution kernels are expanded into large sparse matrices. For example, the matrix k_{ex} represents a 3×3 convolution kernel. Equation (1) displays how k_{ex} can be converted into k_{exp}^{+} and k_{exp}^{-} using the proposed method for parallelizing memristor based convolution. The kernel is converted into two matrices so that the memristor crossbar can easily account for kernels that have both positive and negative values [11]. For this small scale example, assume that the input x_{ex} has dimensions of 4×4 and represents an entire input feature. Since the image x_{ex} has 16 elements, each kernel matrix k_{exp}^{+} and k_{exp}^{-} must have 16 rows. Equation (2) shows that x_{ex} can be converted into x_{exp} and $-x_{\text{exp}}$, which are vector versions of the input image that are to be multiplied by k_{exp}^{+} and k_{exp}^{-} respectively. Since k_{ex} has dimensions of 3×3 and x_{ex} has dimensions of 4×4, the resulting output feature will have dimensions of 2×2. Thus, the kernel matrices must have 4 columns, one for each output value. 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. To implement the example convolution kernel k_{ex} in the crossbar Fig. 3 (b), the row count N should be set to 16 and the column count M should be set to 4.

In the work presented in the following sections, the input images are considerably larger, but this example is meant to show how these parallel convolution kernels can be visualized on a smaller scale. For instance, the first convolution layer in the proposed CNN system needs to convert each 28×28 input image into six different 24×24 output features. Therefore, this convolution layer requires six crossbars each containing 1569×576 memristors to complete this operation in one cycle. This is a significant load if a single crossbar is to generate each output feature, but in this case 96.8% of the memristors in the crossbar can be set to a minimum conductivity value (representing a zero in the matrix multiplication). Since these crossbars are so sparse, reliability issues should be minimized. At the cost of flexibility, the number of devices in the crossbar could be greatly reduced to provide a more robust system (by eliminating devices at certain wire intersections).

The op-amp circuit at the crossbar column output in Fig. 3 (a) 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 [15,19]. To accomplish this, the amplifier at the output of the circuit is used to create a pseudo sigmoid function as shown in Fig. 4. 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.

$$k_{ex} = \begin{bmatrix} 0.1 & -0.2 & 0.3 \\ -0.4 & 0.5 & -0.6 \\ 0.7 & -0.8 & 0.9 \end{bmatrix} \rightarrow \begin{array}{cc} k_{\text{exp}}^{+} & k_{\text{exp}}^{-} \\ \begin{pmatrix} 0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0.3 & 0 & 0 & 0 \\ 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0.9 & 0 \\ 0.5 & 0 & 0 & 0.9 \\ 0 & 0.5 & 0.3 & 0 \\ 0 & 0 & 0 & 0.3 \\ 0.7 & 0 & 0 & 0 \\ 0 & 0.7 & 0.5 & 0 \\ 0.1 & 0 & 0 & 0.5 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0.7 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0.6 & 0 & 0 & 0 \\ 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.8 & 0 & 0 & 0 \\ 0 & 0.8 & 0.6 & 0 \\ 0.2 & 0 & 0 & 0.6 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.8 & 0 \\ 0.4 & 0 & 0 & 0.8 \\ 0 & 0.4 & 0.2 & 0 \\ 0 & 0 & 0 & 0.2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0.4 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{1}$$

$$x_{ex} = \begin{pmatrix} x_1 & \cdots & x_4 \\ \vdots & \ddots & \vdots \\ x_{13} & \cdots & x_{16} \end{pmatrix} \rightarrow \begin{array}{cc} x_{\text{exp}} & -x_{\text{exp}} \\ \begin{pmatrix} x_1 \\ \vdots \\ x_{16} \end{pmatrix} & \begin{pmatrix} -x_1 \\ \vdots \\ -x_{16} \end{pmatrix} \end{array} \tag{2}$$

$$\sigma^{+} = \frac{(\sigma_{\max} - \sigma_{\min})}{\max(|k_{ex}|)} k_{\text{exp}}^{+} + \sigma_{\min} \tag{3}$$

$$\sigma^{-} = \frac{(\sigma_{\max} - \sigma_{\min})}{\max(|k_{ex}|)} k_{\text{exp}}^{-} + \sigma_{\min} \tag{4}$$

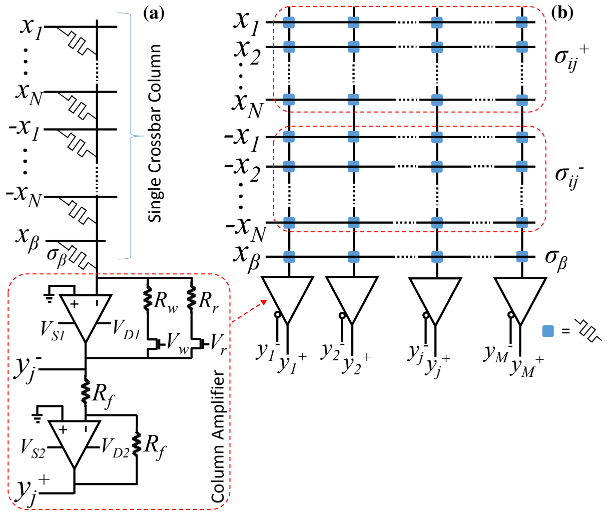


Fig. 3. (a) Circuit diagram displaying a crossbar column circuit that is capable of producing a single convolution output, and (b) the crossbar circuit that is capable realizing the kernel matrices k_{exp}^{+} and k_{exp}^{-}. In (a), V_{S1}=-1V, V_{D1}=0V, V_{S2}=0V, V_{D2}=1V, σ_β is the conductance 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 pairs of positive and negative outputs generated by each column in a memristor crossbar.

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

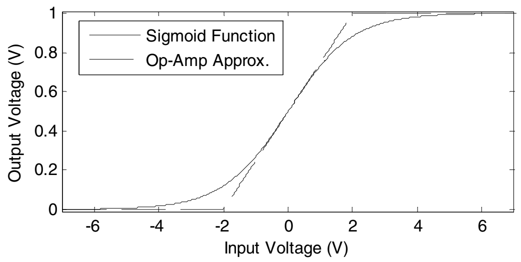


Fig. 4. Comparison of the sigmoid transfer function and the amplifier alternative (the first op-amp stage will actually produce an inverted result, but this will be the eventual outcome seen at the y_j^+ output).

The complete circuit in Fig. 3 will operate according to equations (6) through (8). The op-amp producing the 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 and 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 value σ_β is the conductance of the memristor M_β and $\sigma_\beta=(b+\beta/4)/R_r$. The first amplifier stage has two gain options, one for a programming mode (R_w), and one for a read or evaluation mode (R_r). This gain could have been programmed using a memristor [11], but the chosen approach was decided to be more stable and economical since only two different gain options are required. In eq. (7), R_r 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 the next kernel set σ_{ij}^+ and y_j^- becomes the set of inputs for the other new kernel set σ_{ij}^-.

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

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

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

IV. Crossbar Programming

This section describes how the memristor crossbar conductivity values obtained in the previous section can be physically programmed into a crossbar. An idealized memristor device has a continuous bounded programmable resistance range, but in reality there appears to be a limit to how many discrete states can be programmed into a memristor [28]. There is also a certain amount of stochastic resistance change present when attempting to program a device to a specific resistance [18].

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 reduces the summing amplifier to a simple inverting amplifier. Also, the second inverting amplifier that produces y_j^+ can be ignored during the programming process. In this circuit, the feedback gain will be programmed using the resistor R_w which should be set to σ_{MAX} (or R_{MIN}), the maximum conductivity of the memristor devices used in the CNN architecture. 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 $\sigma_{\min}$ and 8×10^{-6} S for $\sigma_{\max}$ according to the memristor characterization data in [29].

The schematic in Fig. 5 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.

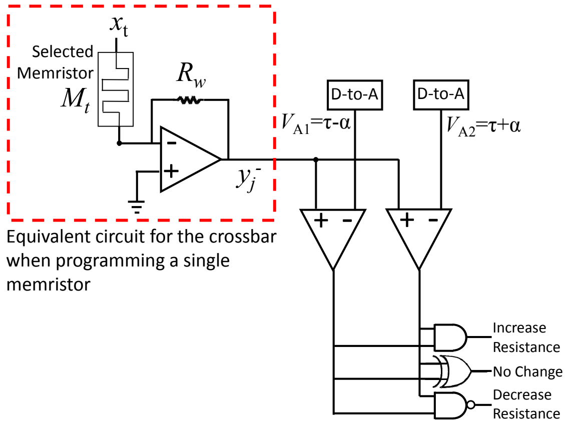


Fig. 5. Circuit used to program a single memristor to a target resistance.

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 feedback process will be repeated until the XOR gate in Fig. 5 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 devices, which will induce very small amounts of resistance change [30] until the target resistance value is reached.

V. Memristor Crossbar Implementation

A. Network Layout

Fig. 6 demonstrates the flow and hardware requirements of the highly parallel CNN recognition system that utilizes each of the components described thus far in this work. The number of outputs required in each layer is displayed, along with the number of memristors each layer must possess to produce these outputs.

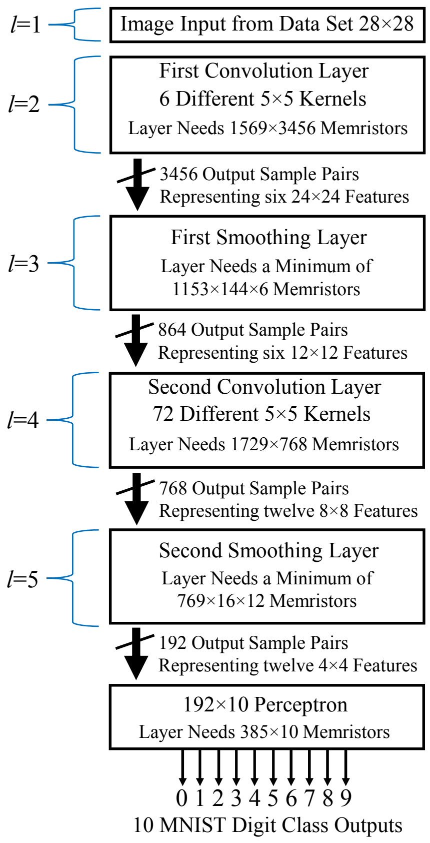


Fig. 6. Flowchart describing the CNN recognition system where no digital storage is required between layers.

The first operation in this network layout is a convolution layer that creates six output features from a single input image (l=2). Following that, a smoothing and sub-sampling layer (l=3) reduces the size of the six features so that they have dimensions of 12×12. Next, layers 4 and 5 repeat this processes, producing an even greater number of smaller features. Finally, the last layer is a fully connected classification layer that makes a final decision based on each of the twelve 4×4 output features. A pictorial demonstration layers 3 and 4 is provided in the following subsections so that the architecture can be better understood.

B. Subsampling Layer Example (l=3)

To perform the smoothing operation in the first subsampling layer entirely in parallel, six processing blocks are required each containing 1153×144 memristors. The method in Fig. 7 displays a topology that requires six separate crossbars that each contain the same smoothing kernel k_s. The requirement of six separate crossbars only exists to ensure the entire smoothing and subsampling processes can be done in one cycle. Using this approach, the subsampling operation becomes trivial, as the convolution matrix will be constructed to only produce samples that would exist in the subsampled output image. This is easily done by removing all columns from the convolution matrix that that would produce an output sample that would ultimately be down-sampled.

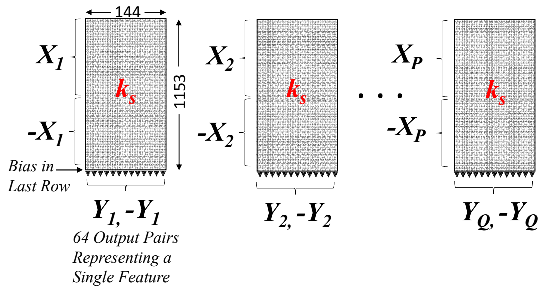


Fig. 7. Topology for implementing the first subsampling layer (l=3) based on the expanded convolution matrix.

C. Convolution Layer Example (l=4)

Fig. 8 shows the topology that is used to execute the second convolution layer. This is the more interesting of the two convolution layers, as each output map is constructed based on a combination of six different input maps. Twelve different convolution kernels are applied to each of the six input feature maps in this layer for a total of 72 convolutions. Each convolution requires a 288×64 memristor crossbar, so a total of 1729×768 memristors are required in this layer. Fig. 8 shows how the 72 convolution kernels can be arranged within a single grid (where P=6 and Q=12). Since each of the output features in this layer is based on a summation of six input features, it is possible to perform P×Q convolutions in a single crossbar [11] (where P is equal to the number of input feature maps and Q is equal to the number of output feature maps). Furthermore, this parallel system allows for these P×Q convolutions to be completed in a single processing cycle, something not capable based on designs in existing literature [11,13,14].

VI. Simulation Results

A. Accuracy Based on Memristor Programmability

The result in Fig. 9 shows how classification accuracy is affected by reliability and noise in memristor programming. 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. 5). Fig. 9 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 [28]. The impact of programming noise is studied by increasing the error between the target conductivity value to be programmed and the actual memristor conductivity. Since the feedback programming circuit in Fig. 5 tracks a voltage to determine if programming is successful, the programming precision value α is measured in volts (not conductance). Fig. 9 shows that there is no real reduction in accuracy unless the programming error measured is greater than 10mV. A slightly looser system is demonstrated that utilizes only 4 unique conductance states programmed to within 10mV of their target values, and it can still provide a recognition accuracy of 94%. However, classification accuracy is degraded significantly if α becomes greater than 100 mV. From this result, it appears that the optimal conditions for programmability are when α=0.01 V and a D-to-A bit width of 4 is used. Therefore, these are the programming settings that will be used for further analysis.

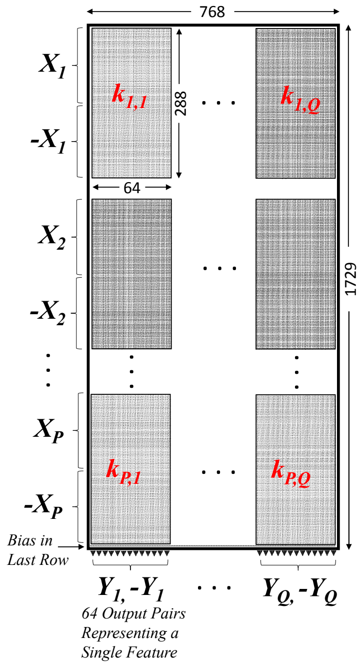


Fig. 8. Crossbar topology for performing the second convolution layer (l=4) entirely in parallel.

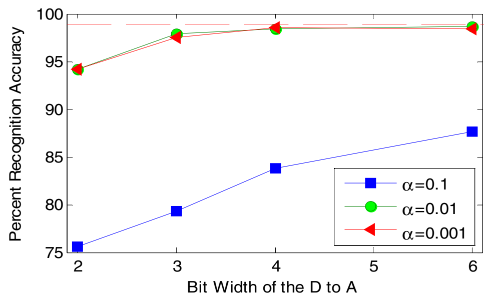


Fig. 9. Plot that shows how recognition accuracy is decreased as the number of programmable states (or the programming accuracy) in the memristors in the crossbar is reduced. Red dashed line shows the accuracy of the CNN algorithm performed entirely in software.

B. Amplifier Error Analysis

The analog circuitry at each memristor column may also impact classification accuracy. One design parameter that must be considered in an analog feedthrough network such as this one is the amount of error produced by the amplifier circuits. To do this we assume that each amplifier in the circuit (two amplifiers for every CNN layer) has an associated gain error as well as a voltage input offset error.

Fig. 10 displays the crossbar column amplifier circuit with each source of error highlighted. Each of the four plots in Fig. 10 display what impact each source of error has on the amplifier transfer functions. Each source of error is applied as a normal random variable where σ_1 is the standard deviation for input offset errors and σ_2 is the standard deviation for gain errors). Thus, the magnitude of these errors will be quite different between each memristor crossbar column. In each plot the values used to determine each error bound line are $\sigma_1 = 10$ mV and $\sigma_2 = 0.1$. One thing to note is that the sources of error between the first and second stage amplifiers are not mirror images. This is because the input swing is not bounded at the input of the first stage, but is bounded between 0 and -1 V at the input of the second stage.

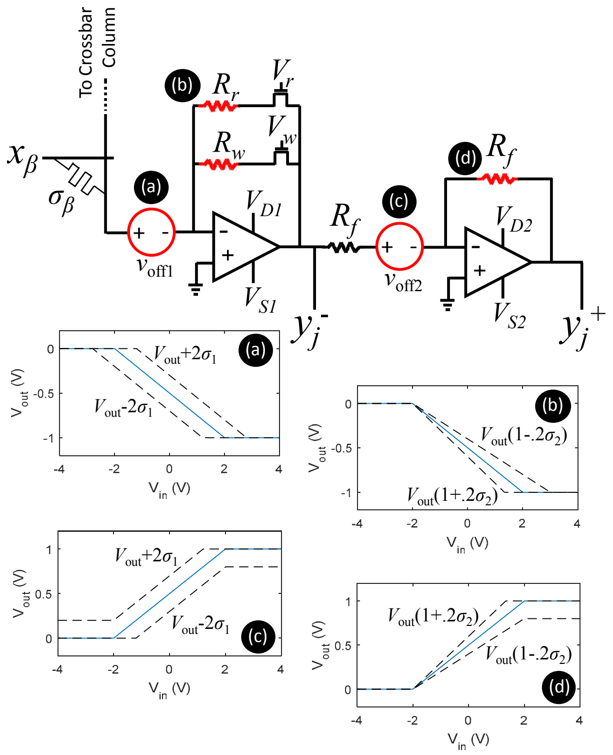


Fig. 10. Crossbar column amplifier circuit with four sources of error including (a) input offset at the first amplifier stage, (b) gain error at the first amplifier stage, (c) input offset at the second amplifier stage, and (d) gain error at the second amplifier stage. The four plots display impact of each source of error will have on the desired voltage relationship where $\sigma_1 = 10$ mV and $\sigma_2 = 0.1$.

In this study, no assumption is made that memristors are naturally initialized at a high resistance state. Physical devices are not yet developed with this kind of accuracy. Therefore, we assume that each device must be programmed to its minimum conductivity state when necessary. This has a significant impact on the large sparse crossbars in this design, since an offset voltage may cause an error in programming each memristor device meant to represent a zero. However, the results in Fig. 11 show that a significant offset voltage can be present before classification accuracy degrades.

When combining both sources of error, Fig. 11 shows that when the standard deviation of input offset error is 5 mV and the standard deviation of the gain error is 6%, the system still achieves 97.05% classification accuracy. This is a positive result considering a signal must travel through a total of 10 gain stages that all possess this error before a classification output is realized.

An advantage of this amplifier circuit is that the gain is very low. The op-amps are mainly used for their precise programmability as opposed to their high gain potential. The gain in the first stage is different for each crossbar column, but the value hovers around 1/3, and the gain all second stage amplifiers is designed to be unity. This helps to reduce the impact in input offset error.

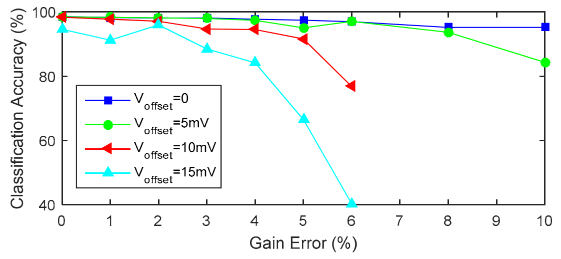


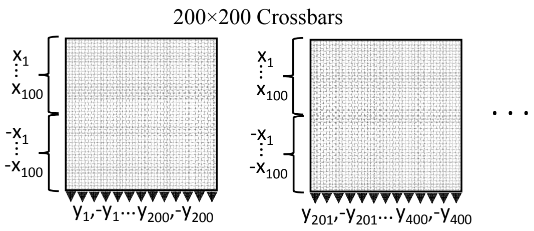
Fig. 11. Plots that displays classification accuracy as greater sources of error are introduced into the amplifier circuit.

VII. Discussion

This parallel design bypasses any digital memory requirements between CNN layers [11], and takes advantage of the true analog processing capabilities of memristors. However, a large number of memristors are required to accomplish this. An advantage for this application is that a sparse diagonally arranged weight matrix is going to have less problems due to the resistance grid, than a very dense one. When providing 784 inputs to a kernel with only 25 values, the crossbar layout happens to be a very strong one for limiting unwanted current paths.

However, in some cases a single crossbar may possess too much error to execute the entire requirements of a large convolution layer. The maximum crossbar size will depend on memristor, transistor, or input driver resistance [31], in addition to other parameters such as wire resistance and matrix sparsity. If developing large crossbars becomes challenging, these crossbars can be broken down using the following technique.

Crossbars may have to be broken down in either the horizontal or vertical direction. Breaking the crossbar down horizontally is fairly simple, since inputs can be applied to multiple crossbars at once, where each crossbar is generating a different set of output values. Fig. 12 demonstrates how this could be done assuming, in one embodiment, there were a maximum crossbar size of 200×200.



200×200 Crossbars
x_1
x_{100}
$-x_1$
$-x_{100}$
$y_1,-y_1 \cdots y_{200},-y_{200}$
$y_{201},-y_{201} \cdots y_{400},-y_{400}$

Fig. 12. Method for horizontally splitting large crossbars to obtain a larger number of outputs in a single layer.

Alternatively, when the number of data inputs spans larger than the maximum number of attainable crossbar rows, the splitting method is a bit more challenging. However, the flexibility of the crossbar circuit described in Fig. 3 makes it possible to vertically split crossbars in this architecture. In this design, subsampling crossbars do not implement the approximate sigmoid function, but the convolution layer does. The sigmoid function can be removed from the crossbar circuit by setting $m = 1$ and $b = 0$ in equations (5) through (7). Therefore, the circuit in the subsampling crossbar could be thought of as implementing equation (9) and the convolution crossbar could be thought of implementing equation (10) where Θ denotes the approximate sigmoid function in equation (5).

$$y = \sum_{i=1}^{N} x_i \sigma_i + b \tag{9}$$

$$y = \Theta\left(\sum_{i=1}^{N} x_i \sigma_i + b \right) \tag{10}$$

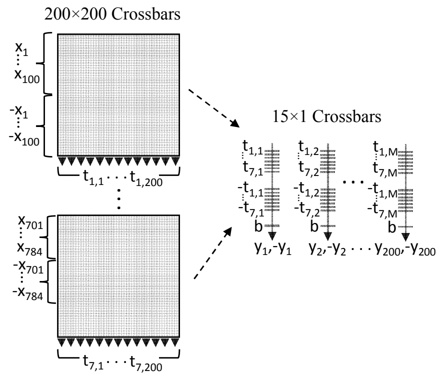


Fig. 13. Approach for splitting large input problems into several smaller, more manageable crossbars.

It is possible to sum the outputs of several smaller crossbars (that do not implement the sigmoid function) with one final crossbar that implements the sigmoid function. Fig. 13 shows a circuit depiction of how this can be implemented using a series of smaller, more manageable crossbars. In this example we again assume that the maximum size for a memristor crossbar is 200×200 memristors. Each temporary output t is generated from subset of the input feature x. Then these temporary outputs are summed, and the output values y are generated after the activation function is applied. Theoretically, the output values y obtained should be equivalent to those obtained using a larger crossbar. However, this method adds an extra amplification stage to the signal path, which may reduce reliability. This will be quantified in future work.

VIII. Conclusion

This paper presents one of the first memristor based CNN systems, and we believe that it is the first that has demonstrated a completely parallel dataflow. It shows that groups of convolution operations can be completed in parallel, producing multiple output feature maps in a single processing cycle. 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. In general, this paper also demonstrates a strong application for the highly parallel mathematical capabilities inherent in memristor crossbars.

There are several aspects about this system that can be investigated as future work. Most immediately we plan on using accurate device and layout properties to determine how large of a crossbar we can reliably construct. Then we plan on taking advantage of the diagonal structure of the proposed kernel matrices to see if unwanted current paths can further be eliminated. We also plan on estimating the energy consumption and throughput of this system and comparing it to the competitive alternatives.

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, 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," Electronics Lett., vol.50, no.7, pp.492,494, 2014.

[11] C. Yakopcic, M. Z. Alom, T. M. Taha, “Memristor Crossbar Deep Network Implementation Based on a Convolutional Neural Network” IEEE/INNS International Joint Conference on Neural Networks, pp. 963 – 970, July, 2016.

[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] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, V. Srikumar, “ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars,” ACM/IEEE 43rd Annual International Symposium on Computer Architecture, pp. 14-26, June, 2016.

[14] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y, Xie, “PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory,” ACM/IEEE 43rd Annual International Symposium on Computer Architecture, pp. 27-39, June, 2016.

[15] 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.

[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] 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.

[18] 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

[19] 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.

[20] 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.

[21] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, L. Zhong, “RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision,” ACM/IEEE 43rd Annual International Symposium on Computer Architecture, pp. 255-266, June, 2016.

[22] Y. Shim, A. Sengupta, K. Roy, “Low-Power Approximate Convolution Computing Unit with Domain-Wall Motion Based “Spin-Memristor” for Image Processing Applications,” 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), August, 2016.

[23] R. Hasan and T. M. Taha, “Enabling Back Propagation Training of Memristor Crossbar Neuromorphic Processors,” IEEE/INNS International Joint Conference on Neural Networks, 2014.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] C. Yakopcic, T. M. Taha, “Determining optimal switching speed for memristors in a neuromorphic system,” Electronics Lett., 51(21), 2015.

[31] C. Yakopcic, T. M. Taha, “Model for maximum crossbar size based on input driver impedance,” Electronics Lett., vol. 52 no. 1, pp. 25-27, 2016.