2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

CSrram: Area-Efficient Low-Power Ex-Situ Training Framework for Memristive Neuromorphic Circuits Based on Clustered Sparsity

Arash Fayyazi ‡, Souvik Kundu ‡, Shahin Nazarian, Peter A. Beerel, Massoud Pedram
Ming Hsieh Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, California 90089, USA
{fayyazi, souvikku, shahin.nazarian, pabeerel, pedram}@usc.edu

***Abstract*—Artificial Neural Networks (ANNs) play a key role in many machine learning (ML) applications but pose arduous challenges in terms of storage and computation of network parameters. Memristive crossbar arrays (MCAs) are capable of both computation and storage, making them promising for in-memory computing enabled neural network accelerators. At the same time, the presence of a significant amount of zero weights in ANNs has motivated research in a variety of parameter reduction techniques. However, for crossbar based architectures, the study of efficient methods to take advantage of network sparsity is still in the early stage. This paper presents CSrram, an efficient ex-situ training framework for hybrid CMOS-memristive neuromorphic circuits. CSrram includes a pre-defined block diagonal clustered (BDC) sparsity algorithm to significantly reduce area and power consumption. The proposed framework is verified on a wide range of datasets including MNIST handwritten recognition, fashion MNIST, breast cancer prediction (BCW), IRIS, and mobile health monitoring. Compared to state of the art fully connected memristive neuromorphic circuits, our CSrram with only 25% density of weights in the first junction, provides a power and area efficiency of 1.5x and 2.6x (averaged over five datasets), respectively, without any significant test accuracy loss.**

***Index Terms*—Clustered sparsity, memristive neuromorphic circuits, ex-situ training, Artificial neural networks (ANNs), low power circuits**

I. Introduction and Motivation

Artificial Neural Networks (ANNs) are helpful in driving a variety of technologies including image processing, pattern recognition, and speech recognition. LeCun et al. classified handwritten digits with less than 1M parameters in 1998 [1], and Krizhevsky et al. won the ImageNet competition with 60M parameters in 2012 [2]. In 2013, Coates et al. proposed a network with 10 billion parameters enabled by high performance computing (HPC) systems [3]. The large number of parameters have made ANNs very power hungry and computationally intensive. In addition, for traditional CMOS memory hierarchies, the requirement of storing and accessing the parameters increases the probability of a cache miss, which further affects performance, both in terms of speed and energy. Numerous research efforts have therefore focused on reducing the computation and storage costs of ANNs and mitigating these challenging resource demands.

In-memory computing through memristor [4] based resistive random access memory (RRAM) has emerged as a promising alternative paradigm for large scale computations, partly because it reduces the overhead associated with data movement. First physically realized in 2008 [5], memristors are particularly well suited for the design of ANNs because of their ultra low power computation of matrix multiplication [6]. In particular, memristors can be fabricated densely [7] in the form of memristive crossbar arrays (MCAs) which can thus realize the synapses of an ANN, performing a weighted sum of their inputs (i.e., the equivalent of a weight lookup and matrix multiply in digital ANN alternatives).

Several Neural Network (NN) research efforts have attempted to address the issue of (i) unnecessarily high dimensional decision boundaries for image classification, known as the overfitting problem, and (ii) getting rid of zero weights [8], [9], [10]. However, as mentioned in [11], a naive mapping from a pruned or compressed weight matrix to an MCA is usually wasteful and sometimes not feasible. After the emergence of structured pruned algorithms [12], several works related to RRAM-based structured compression [13] and sparse neural networks [14] have been reported. However, these works incur significant complexity associated with decoding sparse data formats that increases training time and inference overheads [14].

In this paper, we propose CSrram, a pre-defined clustered sparsity (CS) based ex-situ training framework. Here, by sparse we mean the weight matrices have a large fraction of zero weights and by pre-defined we mean the choice of zero weights is set before training begins, thus discarding the need for a repetitive training approach and avoiding the complex compression technique overhead. Recently [15], [16] showed that pre-defined sparse networks can be trained without any significant loss in network fidelity. However, for MCA-based NN accelerators, the study of efficient methods to utilize sparsity is still in its early stages. Our proposed clustering algorithm ensures that the reduced sparsely connected network can be area-efficiently mapped onto multiple small dense

‡Authors have equal contribution.
This work is partly supported by NSF, including grant 1763747.

978-1-7281-3391-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ISVLSI.2019.00090

MCA clusters. Our algorithm maintains a path from any input node to any output and our framework works as a solution to mitigate overfitting due to reduced number of parameters during training. The major contributions of this paper are summarized below:

- We propose a novel pre-defined block diagonal clustered (BDC) sparsity-based ANN training algorithm. Simulation results show that our trained NNs perform no worse than the random pre-defined sparsity-based counterpart provided we maintain the last junction as fully connected. In fact, classification accuracy loss in our proposed method is negligible compared to corresponding fully connected ANNs.
- Previous research [17] proposed an ANN training algorithm that included a non-linear mapping of weights to MCA conductance values to enhance inference accuracy. We extend the approach to support clustered sparse ANN structures by incorporating this non-linear mapping into the proposed training algorithm and provide an end-to-end ex-situ training framework (Fig. 1). We evaluated this enhanced framework using detailed HSPICE simulations and show that our trained MCA-based ANNs yield no significant loss in test accuracy compared to their fully connected counterparts. Also, our intelligent clustering technique helps providing an area efficient mapping of weights to smaller MCAs.
- We also evaluate the performance of our proposed framework under the constraints of process variation and limited write precision of memristors.

Section II presents CSrram, our proposed pre-defined sparsity-based ex-situ memristive training framework. Section III describes our experimental setup and results on several benchmark datasets (IRIS [18], BCW [19], MHEALTH [20], MNIST [1], and Fashion MNIST [21]). Lastly, the paper concludes in Section IV.

II. CSrram Framework

In Sections II-A, II-B and II-C we describe our block diagonal clustered (BDC) sparsity algorithm, memristor training equations to support BDC sparsity, and the associated modified training framework, respectively. Apart from the one we used in this paper, it is noteworthy that our proposed training algorithm is applicable to a variety of underlying memristive circuits such as [22]–[24].

A. Clustered Pre-defined Sparsity Characterization

An ANN with fully connected (FC) junctions has all neurons of the $(k-1)^{th}$ layer connected to all neurons of the k^{th} layer and, thus requires a large memory to store the weights of each junction. Our CSrram framework for BDC pre-defined sparse neural networks helps mitigate this issue. For a network with L layers of neurons and thus, $J = (L-1)$ junctions, the junction between layer k and $k+1$ has $N_{k+1} \times N_k$ weights resulting a total of $\sum_{k=1}^{L-1}(N_{k+1} \times N_k)$ for an FC network;

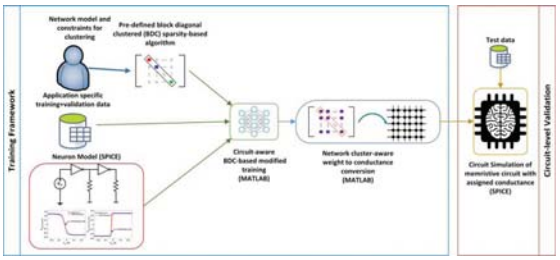

Fig. 1. Proposed ex-situ training and validation framework for CSrram.

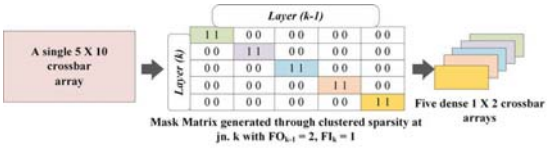

Fig. 2. An example of block-diagonal dense clustering with 20% connection at junction k, giving an area saving of 5x.

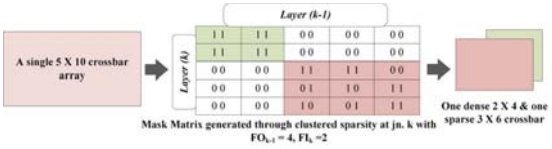

Fig. 3. An example of a block-diagonal dense-sparse clustering with 40% connection at junction k, giving an area saving of 1.5625x.

where N_k is the number of neurons in layer k. We use the definition of connection density as [16]

$$D_k = \frac{C_k}{N_k \times N_{k+1}} \tag{1}$$

Here, $C_k = N_k \times FO_k = N_{k+1} \times FI_{k+1}$, is the total number of connections in the k^{th} junction. FO_k and FI_{k+1} are the fan-out (FO) and fan-in (FI) from each of the preceeding and succeeding neurons, respectively, which we keep constant within a junction. The number of possible options of these integer FO and FI values depends on the *greatest common divisor of* (N_{k+1}, N_k) [25].

For the pre-defined sparsity in our work, we remove the edges between nodes of two successive layers using Algorithm 1 and train the network with the remaining connections. The removed weights never appear during training or inference. To address the issue mentioned in [11] and ensure area-efficient sparsity, the algorithm performs block diagonal clustering of the non-zero weights to create a notion of clustered connectivity. In particular, the algorithm first formulates NC, the required number of clusters and based on the value of $(\frac{1}{D_k})$, creates either (i) NC dense clusters or (ii) $NC-1$ dense clusters and one sparse cluster. Fig. 2 and 3 show scenario (i) and (ii), respectively. Function *SparseAssign* in Algorithm 1 (line 14) generates the sparse cluster by assigning 1 sparsely to elements in the submatrix $[(N_k - c_c \times FI_{k+1} + 1) : N_k, (N_{k+1} - c_c \times FO_k + 1) : N_{k+1}]$ such that sum over

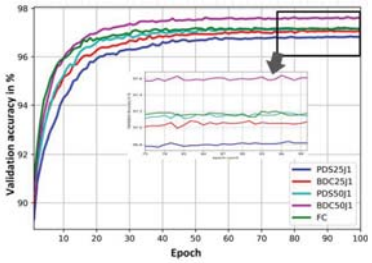

Fig. 4. Validation accuracy study using the MNIST dataset for the proposed block diagonal clustered fixed fan-in fan-out network with 25% (BDC25J1), 50% (BDC50J1) connectivity at 1^{st} junction with 2^{nd} junction fully connected, and pre-defined sparse fixed fan-in fan-out network with similar weight density [25% (PDS25J1) and 50% (PDS50J1) at 1^{st} junction] and their performance compared to a fully connected (FC) network. A 784-100-10 network structure is used. No MCA mapping is considered.

each row and column are FO_k and FI_{k+1}, respectively. We keep the bias connections outside the scope of sparsity and hence they are not addressed in the algorithm. Also, the final junction is kept FC (Algorithm 1 line 2) to ensure there is a dedicated path from any input feature node to any output node, thus guaranteeing that the decision process is aware of every input feature. Hence, the generated mask matrices are suitable for mapping any large ANN junction to mostly dense smaller MCAs with no requirement for iterative training or inference overhead for weight compression and reduction. Fig. 4 shows software simulation (without MCA weight mapping) of validation accuracy for MNIST dataset on pre-defined sparse (PDS) neural network with and without clustering. For both networks with 25% and 50% density of weights at junction 1 (2^{nd} junction remains FC) our approach has no significant accuracy drop compared to a fully connected NN and provides marginally better classification accuracy (0.09% and 0.41% respectively) than PDS neural network without clustering.

B. Sparsity Aware MCA Circuit Equations for Training

The FC junction MCA described in [17] and illustrated in Fig. 5 proposes to use inverters to model neurons because their VTC (Voltage Transfer Characteristic) is similar to the scaled *tanh* function. This structure leads to better performance, lower power consumption, higher energy efficiency, and smaller area compared to the op-amp-based memristive neuromorphic circuits proposed in [23]. This motivated us to choose a similar MCA structure. In particular, to provide differential inputs for the next layer nodes (except the last layer), each current layer output goes through a pair of inverters to generate both inverted (V_{i_n}) and non-inverted (V_{i_p}) differential signals (Fig. 5). Fig. 6(a) shows an example of an MCA model for a NN junction with BDC sparsity of 50% compared to its FC counterpart. Fig. 6(b) shows an illustration of how in our training framework, an input to output path is ensured.

The Kirchhoffs current law (KCL) at the input of the j^{th}

Algorithm 1: Pseudocode for generating the BDC mask matrices

Input: FO_k, FI_{k+1}, N_{k+1} for $k = 1, 2, ..., L-2$
Output: Mask matrices, M_k

1 Initialize M_k mask matrices each of size $N_{k+1} \times N_k$ all with 0, where $k = 1, 2, , L-2$.
2 // based on equation (1)
3 Compute D_k
4 // fill the mask matrix of the last junction with all 1
5 $M_{L-1} = 1$
6 **for** k from 1 to $L-2$ **do**
7 // NC represents number of clusters.
8 $NC = \left\lfloor \frac{1}{D_k} \right\rfloor$
9 **for** c_c from 1 to $NC-1$ **do**
10 $[(c_c-1) \times FI_{k+1} + 1 : c_c \times FI_{k+1}\ ,\ (c_c-1) \times FO_k + 1 : c_c \times FO_k] = 1$
11 **if** $1/D_k$ *is integer* **then**
12 $[N_k - c_c \times FI_{k+1} + 1 : N_k\ ,\ N_{k+1} - c_c \times FO_k + 1 : N_{k+1}] = 1$
13 **else**
14 SparseAssign(M_k, $[(N_k - c_c \times FI_{k+1} + 1) : N_k, (N_{k+1} - c_c \times FO_k + 1) : N_{k+1}]$, FO_k, FI_{k+1})
15 **return** M_k

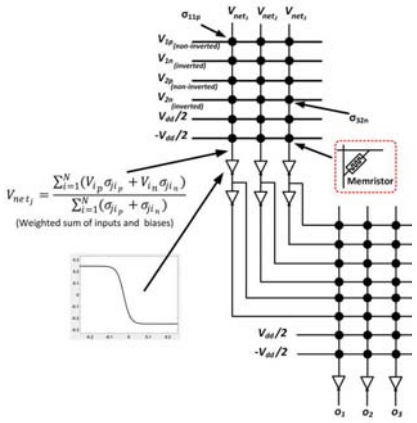

Fig. 5. Circuit structure of the FC memristive neuromorphic circuit [17] used as a baseline for this work.

inverter (Fig. 5) may be written as

$$\sum_{i=1}^{L-1} a_{ji}((V_{i_p} - V_{net_j})\sigma_{ji_p} + (V_{i_n} - V_{net_j})\sigma_{ji_n}) = 0 \quad (2)$$

Here, integer a_{ji} is introduced to represent the presence or absence of a connection between the j^{th} neuron of the current layer to i^{th} neuron of the previous layer and $a_{ji} \in \{0, 1\}$. Also, V_{net_j} is the voltage of node net_j (the input of the inverter of column *j*), and σ_{ji_p} (σ_{ji_n}) is the conductance of the memristor located in the non-inverted (inverted) line position

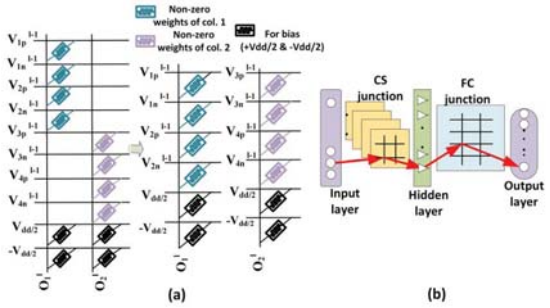

Fig. 6. (a) Clustered sparse version of a 10×2 MCA with 50% connectivity and smaller MCA (2 of 6×1) during inference. (b) An ANN structure with a single hidden layer where the red arrow shows the presence of a path from any input to any output node

(*i*, *j*) *. So, the input voltage of the j^{th} inverter, V_{net_j} can be obtained from

$$V_{net_j} = \frac{\sum_{i=1}^{L-1} a_{ji}(V_{i_p} \times \sigma_{ji_p} + V_{i_n} \times \sigma_{ji_n})}{\sum_{i=1}^{L-1} a_{ji}(\sigma_{ji_p} + \sigma_{ji_n})} \tag{3}$$

Also, in order to mimic the multiply accumulate (MAC) unit of an ANN, the input to the j^{th} neuron in terms of weights (w_{ji_p} and w_{ji_n}) and V_i is defined as

$$z_j = \sum_{i=1}^{L-1} a_{ji}(V_{i_p} \times w_{ji_p} + V_{i_n} \times w_{ji_n}) \tag{4}$$

Thus for the MCA-based structure to hold its ANN functionality with sparse training, equations (3) and (4) must be equal. Hence, the weights (w_{ji_p} and w_{ji_n}) can be modeled as

$$w_{ji_x} = \frac{a_{ji} \times \sigma_{ji_x}}{\sum_{m=1}^{L-1} a_{jm}(\sigma_{jm_p} + \sigma_{jm_n})}; \tag{5}$$

Here *x* can be either *n* or *p*. To ensure a non-trivial solution we apply the following constraint [17] modified to capture sparsity,

$$\sum_{i=1}^{L-1} a_{ji}(w_{ji_p} + w_{ji_p}) = 1 \tag{6}$$

Note that in equations (2)-(4) we have kept the bias terms out of the scope of sparsity. Rather, the two bias lines for each neuron are maintained at $\frac{V_{dd}}{2}$ and $-\frac{V_{dd}}{2}$. Our work maps all the weights to conductance values between $[\sigma_{min}, \sigma_{max}]$, where the specific bounds, σ_{min} = 0.12μS and σ_{max} = 7.9μS, are taken from [26].

C. Training of Network with Pre-defined Sparsity

Here, we propose our enhancement to [17] to support BDC sparsity. To keep the weights positive and meet the conditions of equation (6), two mapping functions g_1 [17] and g_2 were used,

$$g_1(\theta_{ji_x}) = \sigma_{min} + \frac{(\sigma_{max} - \sigma_{min})}{1 + e^{(-\theta_{ji_x})}} \tag{7a}$$

$$g_2(\sigma_{ji_x}) = \frac{\sigma_{ji_x}}{\sum_{m=1}^{L-1} a_{jm}(\sigma_{jm_p} + \sigma_{jm_n})} \tag{7b}$$

g_1 maps unconstrained training weights (θ_{ji_x}) to conductance through a scaled *sigmoid* function that adheres to the specified upper and lower bounds on the conductance. g_2 maps σ_{ji_x} to constrained function mapped weights in order to satisfy equation (6). Because the denominator of g_2 is never zero, it is guaranteed to be differentiable. Combining these mapping functions, the function mapped weights can be expressed as

$$w_{ji_x} = g_2(\sigma_{ji_x}) = g_2(g_1(\theta ji_x)) \tag{8}$$

Training for CSrram is described in Algorithm 2, where we used the following weight update equation,

$$\theta_{ji_x}^{t+1} = \theta_{ji_x}^{t} - \eta \times \frac{\partial C}{\partial \theta_{ji_x}^{t}} \tag{9}$$

where

$$\frac{\partial C}{\partial \theta_{ji_x}^{t}} = \frac{\partial C}{\partial z_j^t} \times \frac{\partial z_j^t}{\partial w_{ji_x}^t} \times \frac{\partial w_{ji_x}^t}{\partial \sigma_{ji_x}^t} \times \frac{\partial \sigma_{ji_x}^t}{\partial \theta_{ji_x}^t} \tag{10a}$$

$$\frac{\partial z_j^t}{\partial w_{ji_x}^t} = a_{ji} \times V_{i_x} \tag{10b}$$

In equations 9-10, current and next values of the parameters are represented through *superscript* t and $t+1$, respectively. η is learning rate, and *C* is the *cross-entropy* based network cost function. Our training terminates when network reaches target cost function for training accuracy of 98% or 500 epochs (1 epoch signifies training with one full train dataset).

Algorithm 2: Pseudocode for proposed training algorithm

Input: Patterns, Mask matrices
Output: Trained Weights
Initialize all $thetas$ with small random numbers
do
  for *every pattern in the training set* **do**
    Present the pattern to the network
    Calculate the $function\ mapped\ weights$ using $g1$ and $g2$
    Multiply it by corresponding mask matrix, element by element Propagated the input forward through the network
    Calculate the Cost function C
    Propagate the errors backward through the network including the multiplication of derivative of weight mapping function by its corresponding mask matrix element by element
    Update $thetas$
while *(maximum iteration of epochs is reached)* or *(cost function C is greater than specified)*
return $thetas$

*Here, *i* and *j* are row and column values, respectively.

III. Results

In this section, we first describe the simulation setup before reporting the performance of CSrram in terms of classification accuracy (with and without the constraints of process variation and limited write precision of memristors), area, and power on several benchmark datasets. All the evaluations were performed through HSPICE simulations for a TSMC 90nm technology model for transistors, using the generalized memristor model proposed in [27] for the memristor devices of [26].

A. Simulation Setup

The simulation setup for CSrram is shown in Fig. 1. Our MATLAB script accepts user inputs to evaluate the cluster structures based on FO and FI values. The script includes a mathematical model of the neurons (i.e., fitted voltage transfer characteristics of the inverters, extracted from SPICE simulation). We assumed the memristor write threshold voltage and supply voltage to be 4V and 0.5V, respectively, and to ensure 99% write accuracy in presence of device variation and stochastic write we used the scheme of [28] (the write operation is performed memristor-by-memristor). The methodology proposed in [17] is used to map the extracted conductance values to the corresponding state parameters of the memristor for SPICE simulations. It is to be noted that, we trained the networks without any data augmentation or parameter optimization techniques useful for increasing accuracy and speeding-up convergence. All these techniques along with convolutional layers can be added to our proposed framework to further improve classification performance. The important design parameters used for SPICE simulation are mentioned in Table I. Here, the size of the inverters was the reported S_{inv} times of a minimum sized inverter.

TABLE I
Design Parameters

Technology	TSMC 90nm - Typical
R_{on}	$125K\Omega$
R_{off}	$8.3M\Omega$
S_{inv}	5
Supply Voltage	$0.5V$

B. Classification Accuracy with Clustered Sparsity, Process Variation, and Limited Write Precision

We used BCW, IRIS, MHEALTH, MNIST, and fashion MNIST datasets (Table II) for our performance evaluation. Training and testing datasets were split based on a 80:20 ratio. Fig. 7(a) provides their accuracies under different levels of pre-defined BDC sparsity. It is clear from the results that with sparse connectivity as low as 25% in junction 1, the network's classification accuracy hardly degrades compared to its FC counterpart. The terms BDC50J1 and BDC25J1 represent neural network models with 50% (*NC* is 2) and 25% (*NC* is 4) connectivity at junction 1, respectively, keeping later

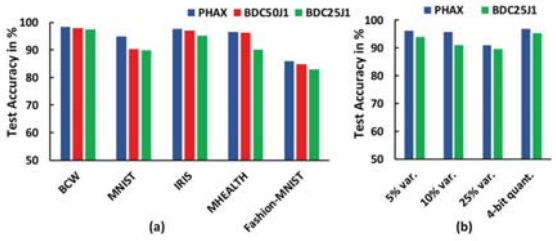

Fig. 7. (a) Test accuracy of the proposed framework compared to fully connected networks (PHAX), (b) IRIS dataset classification accuracy in PHAX and BDC25J1 under process variations and 4-bit quantization.

junction(s) as FC [†] . To evaluate performance of our proposed method under compressed input with reduced dimensions, we used a 14×14 input for MNIST instead of full size of 28×28.

To assess the performance of CSrram under process variation and limited write precision, a random Gaussian noise with $p\%$ ($p \in [5, 10, 25]$) variance was added to conductance values of memristors. Additionally, in order to examine the effect of limited write precision of memristors, we also quantized the conductance with 4-bit precision. For the sake of space, we show the variance-accuracy (and limited write precision) plots only for IRIS in Fig. 7(b). Even for model BDC25J1, worst case accuracy loss across all datasets, due to variance (quantization) is within 5%. Environmental variations, interconnect IR drop [29], impact of process variation (PV) on the inverter (especially when it operates near threshold) as well as other non-ideal characteristics, including the memristor's unstable intermediate states, may also need to be considered. Fortunately, these non-idealities can be compensated through heuristics for the variability of both memristor and CMOS components [29], [30].

TABLE II
Network structure for different datasets and corresponding power consumption

Dataset	Network Structure	Power Consumption (μW)	
		Fully Connected	*BDC25J1*
BCW	$10-8-2$	13.4	8.87
IRIS	$4-4-3$	7.67	7.45
MHEALTH	$23-92-60-13$	703.2	639
MNIST	$196^{*}-100-10$	1221	527
Fashion MNIST	$784-100-10-10$	4309	2879

*A compressed version (14×14) of actual 28×28 input image is used.

C. Power Efficiency and Area Reduction

Power and area improvements of the proposed memristive neuromorphic circuit with respect to state of the art ANN implementation [17] are compared in Fig. 8(a) and (b), respectively. Our evaluation of power consumption is based on HSPICE simulations. Table II shows the power consumption of

[†]For MHEALTH, we also simulated making junction 2 sparse and obtained similar accuracy as that with sparsity at junction 1. For fashion MNIST we did not consider junction 2 sparsity as the number of weights in the 2^{nd} junction is much lower than that in the 1^{st} junction.

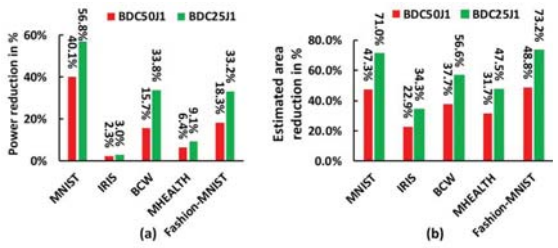

Fig. 8. (a) Power reduction, and (b) area reduction (best case) through reduction of MCA size for pre-defined BDC sparse neuromorphic circuits over fully connected networks.

FC and clustered sparse (BDC25J1) networks for five datasets. Interestingly, the inverters consume most of the reported power. Fig. 8(a) shows that our framework provides more power savings for larger networks. Additionally, we compared our proposed designs with fully combinational digital ASIC implementations (TSMC 90nm) of FC ANNs where the neurons were made using 256×8-bit LUTs. For MNIST, the power consumption for the digital ASIC (evaluated through Synopsys DC) is three orders of magnitude larger compared to our BDC25J1 model. Also, compared to FC neuromorphic circuits, BDC sparsity offers smaller mostly dense clusters of MCA structures to replace a large MCA, and thus improves area. We have considered sparsity at the junction between input and first hidden layer (J1) because it generally has much more weights than other junctions. Fig. 8(b) shows the best case area improvement where all MCA clusters in a junction are dense. We used equation (11) to compute area improvement.

$$f = [1 - \frac{\sum_{i=1}^{L-1} Ar_{j_i}^{BDC}}{\sum_{i=1}^{L-1} Ar_{j_i}^{FC}}] \times 100 \qquad (11)$$

Here, for i^{th} junction, the area corresponding to FC and BDC sparsity are $Ar_{j_i}^{FC}$ and $Ar_{j_i}^{BDC}$, respectively, and they are computed as the number of memristors in that junction. For our evaluations, $Ar_{j_i}^{BDC} = Ar_{j_i}^{FC}$ if $i \neq 1$.

IV. Conclusions

In this paper, we propose a novel pre-defined BDC sparsity-based NN training algorithm and incorporate this into to a sparsity-aware ex-situ training framework called CSrram. We analyzed its impact on classification accuracy, power, area and process variations. The results show that the added constraint of BDC sparsity improves the average power and area by $1.5\times$ and $2.6\times$, respectively, without significant accuracy drop. These results motivate further research to efficiently identify the minimum cluster size of the initial junctions given a lower bound on accuracy. Lastly, incorporating convolutional layers to this framework to boost classification accuracy and showing it on larger datasets like CIFAR-10 and Imagenet is also promising future work.

References

[1] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[2] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] A. Coates *et al.*, "Deep learning with cots hpc systems," in *International conference on machine learning*, 2013, pp. 1337–1345.

[4] W. Wang *et al.*, "Fabrication, characterization, and modeling of memristor devices," in *NAECON 2014-IEEE National Aerospace and Electronics Conference*. IEEE, 2014, pp. 259–262.

[5] D. B. Strukov *et al.*, "The missing memristor found," *nature*, vol. 453, no. 7191, p. 80, 2008.

[6] Z. Du *et al.*, "Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches," in *2015 48th Annual IEEE/ACM MICRO*. IEEE, 2015, pp. 494–507.

[7] S. H. Jo *et al.*, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[8] P. Molchanov *et al.*, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[9] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[10] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[11] J. Cui and Q. Qiu, "Towards memristor based accelerator for sparse matrix vector multiplication," in *2016 ISCAS*. IEEE, 2016, pp. 121–124.

[12] J. Yu *et al.*, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2. ACM, 2017, pp. 548–560.

[13] H. Ji *et al.*, "Recom: An efficient resistive accelerator for compressed deep neural networks," in *2018 IEEE DATE*. IEEE, 2018, pp. 237–240.

[14] P. Wang *et al.*, "Snrram: an efficient sparse neural network computation architecture based on resistive random-access memory," in *Proceedings of the 55th IEEE/ACM DAC*. ACM, 2018, p. 106.

[15] S. Dey *et al.*, "Characterizing sparse connectivity patterns in neural networks," in *2018 ITA*. IEEE, 2018, pp. 1–9.

[16] S. Dey, D. Chen *et al.*, "A highly parallel fpga implementation of sparse neural network training," *arXiv preprint arXiv:1806.01087*, 2018.

[17] M. Ansari *et al.*, "Phax: Physical characteristics aware ex-situ training framework for inverter-based memristive neuromorphic circuits," *IEEE TCAD*, vol. 37, no. 8, pp. 1602–1613, 2018.

[18] S. B. Kotsiantis, "Logitboost of simple bayesian classifier," *Informatica*, vol. 29, no. 1, 2005.

[19] O. L. Mangasarian *et al.*, "Breast cancer diagnosis and prognosis via linear programming," *Operations Research*, vol. 43, no. 4, pp. 570–577, 1995.

[20] O. Banos *et al.*, "mhealthdroid: a novel framework for agile development of mobile health applications," in *International Workshop on Ambient Assisted Living*. Springer, 2014, pp. 91–98.

[21] H. Xiao *et al.*, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[22] A. Fayyazi *et al.*, "An ultra low-power memristive neuromorphic circuit for internet of things smart sensors," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1011–1022, 2018.

[23] R. Hasan *et al.*, "Ex-situ training of dense memristor crossbar for neuromorphic applications," in *Proceedings of the 2015 IEEE/ACM NANOARCH*. IEEE, 2015, pp. 75–81.

[24] S. P. Adhikari *et al.*, "A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses," *IEEE TCAS I: Regular Papers*, vol. 62, no. 1, pp. 215–223, 2015.

[25] S. Dey *et al.*, "Pre-defined sparse neural networks with hardware acceleration," *arXiv preprint arXiv:1812.01164*, 2018.

[26] W. Lu *et al.*, "Two-terminal resistive switches (memristors) for memory and logic applications," in *Proceedings of the 16th ASP-DAC*. IEEE Press, 2011, pp. 217–223.

[27] C. Yakopcic *et al.*, "Generalized memristive device spice model and its application in circuit design," *IEEE TCAD*, vol. 32, no. 8, pp. 1201–1214, 2013.

[28] F. Alibart *et al.*, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[29] B. Liu *et al.*, "Vortex: variation-aware training for memristor x-bar," in *Proceedings of the 52nd IEEE/ACM DAC*. ACM, 2015, p. 15.

[30] A. BanaGozar *et al.*, "Robust neuromorphic computing in the presence of process variation," in *IEEE DATE*. IEEE, 2017, pp. 440–445.