A Device Non-Ideality Resilient Approach for Mapping Neural Networks to Crossbar Arrays

Arman Kazemi*, Cristobal Alessandri†, Alan C. Seabaugh†, X. Sharon Hu*, Michael Niemier*, Siddharth Joshi*
*Department of Computer Science and Engineering, University of Notre Dame
†Department of Electrical Engineering, University of Notre Dame, akazemi@nd.edu

***Abstract*—We propose a technology-independent method, referred to as adjacent connection matrix (ACM), to efficiently map signed weight matrices to non-negative crossbar arrays. When compared to same-hardware-overhead mapping methods, using ACM leads to improvements of up to 20% in training accuracy for ResNet-20 with the CIFAR-10 dataset when training with 5-bit precision crossbar arrays or lower. When compared with strategies that use two elements to represent a weight, ACM achieves comparable training accuracies, while also offering area and read energy reductions of 2.3× and 7×, respectively. ACM also has a mild regularization effect that improves inference accuracy in crossbar arrays without any retraining or costly device/variation-aware training.**

I. Introduction

Automated analysis of vast amounts of data can potentially revolutionize governance, manufacturing, medicine, and many other fields. Over the past decade, increasingly complex deep neural network (DNN) models have been proposed as a means to perform such automated analysis. The cost to train and deploy such models has grown along with model complexity, leading to a need for hardware platforms that are energy-efficient and low-latency [1].

A promising avenue for hardware research that addresses these challenges is based on analog domain computation of matrix-vector multiplication (MVM), a critical kernel in the forward pass of DNN training as well as inference. One family of accelerators uses analog crossbar (XBar) arrays which could be composed of emerging devices such as resistive random access memories (RRAM) [2], phase change memories (PCM) [3], and ferroelectric field-effect transistors (FeFET) [4], for highly parallel MVMs. A XBar array represents the input vector as analog voltages. Applying these voltages to the rows of the XBar array (where weights are stored as conductances at row/column crosspoints) induces a current along the XBar columns. The current of each column represents the dot product of the input vector and the weight vector represented by the synapse devices on the column [3].

While XBar arrays efficiently implement MVMs and offer many performance advantages in energy and latency, their use poses many practical challenges. One such challenge is inherent in representing weights as conductance values. This constrains XBar arrays to use *non-negative* conductance values to implement arbitrary *signed* MVMs. As shown in Fig. 1, two approaches have been widely adopted: **(i)** differential encoding where two elements are used to represent one weight (a **double element (DE)** approach) [5], [6] and **(ii)** a constant bias to remap values in the range $(-w, w)$ to $(0, 2w)$ (a **bias column (BC)** approach) [7], [8]. Another challenge with performing MVMs via XBar arrays arises due to the limitations of devices employed for synapse elements, e.g., RRAMs [2], PCMs [3], FeFETs [4], etc, with respect to achievable weight resolution and weight update linearity, which in turn can adversely impact training accuracy. Furthermore, methods to overcome these issues, e.g., using multiple synapse elements for a single weight [9], further reduce the energy and area savings that might otherwise be obtained from XBar arrays. Moreover, the accuracy of models deployed on XBar arrays for inference is further degraded by device variation [7], [10].

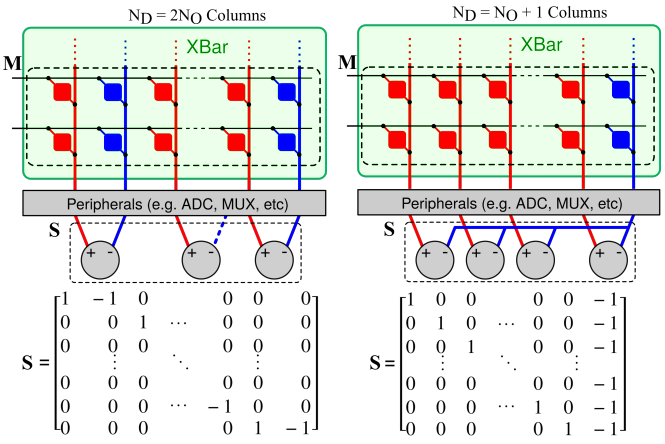
As stated in [11], to capitalize on the benefits offered by XBar arrays, breakthroughs in material development or architecture design are needed. Within this context, this paper presents a method, referred to as **adjacent connection matrix (ACM)**, for efficiently mapping signed MVMs to XBar arrays. By learning the most effective representation of a weight through a combination of a XBar array column and its immediate neighbor, ACM increases the effective dynamic range of weight representations. This nearest-neighbor coupling also introduces a mild regularization effect that improves resilience to device variation. ACM has been evaluated with the MNIST [12] and CIFAR-10 [13] datasets and results indicate that using ACM can lead to **(i)** up to 20% improvements in training accuracy when compared to other strategies under equivalent resource constraints (i.e., the BC approach), **(ii)** comparable training accuracies coupled with area reductions of 2.3× and read energy reductions of 7× when compared to the DE approach, and **(iii)** a 10% average improvement compared to both DE and BC in the inference accuracy of a VGG network trained on the CIFAR-10 dataset with 3-bit precision XBar arrays, assuming a 15% device variation.

The remainder of the paper is organized as follows: Section II reviews strategies for mapping signed MVMs to XBar arrays. Section III presents ACM and a method to evaluate and compare it with other mappings. Section IV discusses the results of our simulations and quantifies DNN accuracy improvements under resource constraints and device variation; system-level evaluations of energy, area, and delay are also presented. Section V concludes by summarizing our findings.

II. Background

There are two approaches typically used to map a MVM to a XBar array; these can be implemented in both the analog

978-1-7281-1085-1/20/$31.00 ©2020 IEEE



(a) Double Element (DE) (b) Bias Column (BC)

Fig. 1: Prior approaches to map DNN models to ReXB arrays; **(a)** Using two resistive elements to represent one weight; **(b)** using a single column of resistive elements as reference.

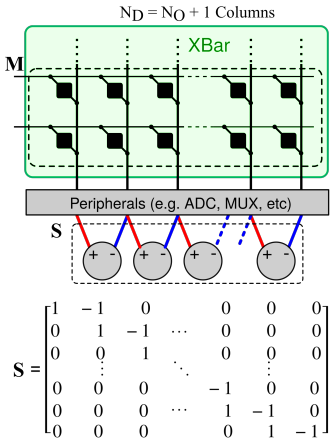


Fig. 2: ACM computes the outputs of a signed MVM as a combination of the outputs of adjacent columns with alternating signs.

and the digital domain. The first approach, i.e., DE shown in Fig. 1a, uses two XBar array columns to represent one weight column [14], [15]. With this approach, the difference between column-pairs in the XBar array represents the equivalent signed weighted sum. The second case is an input dependent bias approach, i.e., BC [7], [8]. As illustrated in Fig. 1b, a single XBar array column is used as a reference to implement the bias. The conductance of each element in this column is fixed to the middle of the conductance range. The output from this column is subtracted from the output of all other columns to compute the signed weighted sum.

In order to perform a MVM with XBar arrays with these mapping methods, the outputs of the MVM are digitized in the periphery of the XBar array [3]. The operational overhead of the mappings are the additions and subtractions performed after digitization. Since both mappings require a single subtraction per weight, this overhead is the same for both approaches. The hardware overhead due to the number of elements used for each mapping is evaluated in Section IV.

Note that, if the conductance values of the XBar array elements are limited to the range $[G_{min},\ G_{max}]$ (for simplicity, we assume $G_{min} = 0$), the weights of BC will be in the range $[-G_{max}/2,\ G_{max}/2]$, with the conductance values of the bias column elements fixed to $G_{max}/2$. For DE, the range of weights will be $[-G_{max},\ G_{max}]$, at the expense of using twice as many weight elements, while representing twice as many weight values as that of the BC. In short, DE utilizes $2\times$ synapse elements and gains a $2\times$ wider dynamic range of weight representation compared to the BC approach. However, in both the DE and BC approaches, a MVM with non-negative weights is performed in the XBar array, followed by a simple (linear) combination of the outputs from its columns to obtain an equivalent MVM with signed weights. A critical aspect of the effectiveness of these mapping solutions is the simplicity of the linear transforms implemented. These transforms consist of the addition and subtraction of values at the XBar periphery.

III. Adjacent Connection Matrix

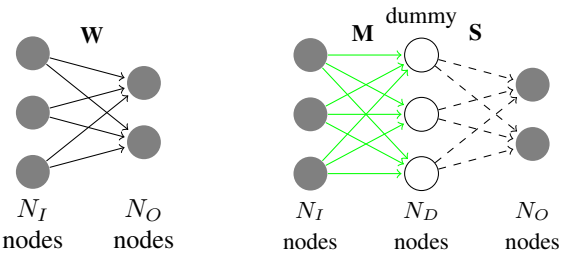
We extend the idea of using simple linear transforms in the periphery of the XBar array circuit to map a MVM onto XBar arrays and present a new method called the adjacent connection matrix (ACM). After a brief qualitative introduction of the ACM and its benefits over other mapping methods, i.e., DE and BC, we provide a formal definition of the different mapping techniques (DE, BC, and ACM) and show how a signed MVM can be realized with a non-negative matrix representing the XBar array, and a constant signed matrix representing each mapping. Finally, we show the regularization effect of ACM using its formal definition.

A. Adjacent Connection Matrix Concept

In contrast with **(i)** the DE approach that uses two elements per weight (each a reference for the other) and **(ii)** the BC approach that uses a fixed reference column per MVM, the ACM approach uses each column as a reference for its immediate neighbor. ACM computes the outputs of a signed MVM as a combination of the outputs of neighboring columns of the XBar array with alternating signs as shown in Fig. 2. Like BC, ACM also requires one additional column. This compares favorably against DE which requires two elements per weight, and therefore requires almost double the number of synapse elements in large XBar arrays. It is noteworthy that the operational overhead of ACM is the same as BC and DE, since it requires a single subtraction for each weight. Furthermore, ACM provides a mild regularization effect which increases resilience against device variation.

B. Formal Definition of Different Mappings

ACM, DE, and BC, all combine the outputs from columns of the XBar array in a fixed and predefined way. This combination of columns is comprised of additions and subtractions only and can be thought of as a matrix with its non-zero entries limited to ± 1. We refer to this matrix as the periphery matrix. The three different mappings and their corresponding unique periphery matrices are presented in Figs. 1 and 2. In order to



(a) Signed Matrix **W** of a Fully-Connected Layer (b) Equivalent Non-negative Matrix **M** and the Periphery Matrix **S**

Fig. 3: An example in the context of fully-connected layers, where the original matrix is decomposed as a sequence of a non-negative matrix **M**, followed by a periphery matrix **S**.

verify that the periphery matrix holds as a general technique to map MVMs onto XBar arrays, we first demonstrate the decomposition of a signed MVM into **(i)** a non-negative matrix that is stored on the XBar array and **(ii)** a fixed, signed matrix. We then characterize the requirements of this matrix and demonstrate that it can be implemented through addition and subtraction operations at the periphery of the XBar array.

Consider an arbitrary signed matrix **W** with dimensions $N_O \times N_I$[1]. To constrain the multiplication to non-negative weights only, matrix **W** is factored into a matrix with non-negative elements **M**, followed by a periphery matrix **S**, i.e.,

$$\boldsymbol{S}\,\boldsymbol{M} = \boldsymbol{W}, \quad \boldsymbol{M} \geq 0, \tag{1}$$

where $\mathbf{M} \geq 0$ means that all elements of **M** are non-negative. Within the context of DNN layers, **W** is the weight matrix of a fully-connected layer with N_I inputs and N_O outputs as shown in Fig. 3a. For ease of visualization, we define a *dummy layer* Y with N_D neurons. Fig. 3b shows the layer mapped to a XBar array where **M** is the non-negative matrix with dimensions $N_D \times N_I$, and **S** is a periphery matrix of dimensions $N_O \times N_D$. While we have used a fully-connected layer from a DNN in our example, all linear transforms, including convolutions, are possible through ACM.

There are two properties we desire from **S**: **(i)** a fixed **S** must guarantee that any multiplication using a signed matrix **W** can be realized using a non-negative matrix **M** and a fixed signed matrix **S** and **(ii)** **S** must be of a form that does not impose large hardware implementation costs.

C. Sufficient Conditions of a Periphery Matrix

As before, given a **W** with the dimensions $N_O \times N_I$, **S** can be assigned dimensions $N_O \times N_D$, and **M** the dimensions $N_D \times N_I$. Formulated independently for each column of **W** and **M**, this can be expressed as:

$$\boldsymbol{S}\,\boldsymbol{m_k} = \boldsymbol{w_k}, \quad \boldsymbol{m_k} \geq 0, \tag{2}$$

where $\boldsymbol{m_k}$ and $\boldsymbol{w_k}$ are the k-th columns of **M** and **W**, respectively, and $k \in \{1, 2, \ldots, N_I\}$.

[1]In this section we examine the transpose of the matrices to simplify the solution of equations and the explanations.

A necessary condition for the existence of a solution to Eq. (2) is that $\boldsymbol{w_k}$ is in the column space of **S**. This condition will be satisfied for any arbitrary $\boldsymbol{w_k}$ if and only if $\mathrm{rank}(\boldsymbol{S}) = N_O$. The sufficient condition for the existence of a non-negative solution is the existence of a vector x_h in the null space of **S** with strictly positive elements. This guarantees that any particular solution $\boldsymbol{x_p}$ to the system $\boldsymbol{S}\,\boldsymbol{m_k} = \boldsymbol{w_k}$ can be shifted as $\boldsymbol{x'_p} = \boldsymbol{x_p} + \alpha \boldsymbol{x_h}$ to be non-negative. The sufficient conditions are summarized as:

$$\begin{aligned} &1. \quad \mathrm{rank}(\boldsymbol{S}) = N_O \\ &2. \quad \exists\, \boldsymbol{x_h} > 0, \boldsymbol{x_h} \in \mathrm{R}^{N_D},\ \text{s.t. } \boldsymbol{S}\,\boldsymbol{x_h} = 0. \end{aligned} \tag{3}$$

If these conditions are met, the signed matrix **W** can be decomposed to a non-negative matrix **M** and periphery matrix **S**, such that, $\boldsymbol{W} = \boldsymbol{S}\,\boldsymbol{M}$. Equations $N_D = \mathrm{rank}(\boldsymbol{S}) + \mathrm{nullity}(\boldsymbol{S})$ and $\mathrm{nullity}(\boldsymbol{S}) \geq 1$ hold, if there is at least one element ($\boldsymbol{x_h}$) in the null space of **S**. Therefore, **M** with N_D columns has at least one more column than **W** with N_O columns. A particular case that satisfies the second condition of Eq. (3) is $x_h = \mathbf{1}$, which implies that the elements of the rows of **S** add up to 0. Thus, an **S** such that neighboring columns are subtracted from each other, introduced earlier as the ACM, satisfies both conditions, i.e., one extra column in **M** and the sum of rows of $\boldsymbol{S} = 0$. Note that **S** also has all the properties listed as desirable per Section III-B.

D. Analysis of Different Mappings

Using the periphery matrix decomposition discussed above, we can derive not only the ACM mapping but also the DE and BC mappings. We can observe in Figs. 1 and 2 that all three approaches satisfy the conditions stated in Eq. (3). In all cases, each row in the periphery matrix has two nonzero elements (1 and -1), hence the sum of the elements of the rows add up to 0. Furthermore, $N_D \geq 1$ is true for all three cases. However, DE has $N_D = 2N_O$ columns, whereas BC and ACM have the minimum number of columns, $N_D = N_O + 1$. Therefore, BC and ACM require minimal additional hardware resources. Furthermore, we assume that the elements of **M** have a conductance range of $[G_{min}, G_{max}]$ (again, for simplicity, we assume $G_{min} = 0$). Thus, by using the ACM approach, addition and subtraction of neighboring elements can result in the representation of weights over the range $[-G_{max}, G_{max}]$ while using the same hardware resources as BC. That said, while DE can always demonstrate the full range in weights, ACM is limited by having to balance DNN accuracy and weight range, as neighboring columns are not guaranteed to have a large disparity in weights. Section IV quantifies the effect of these different mappings on system-level DNN training and inference accuracy in the presence of non-ideal XBar array synapse devices.

E. Regularization Effect of Adjacent Connection Matrix

An observation of the nearest-neighbor coupling induced by the ACM approach naturally leads to the question, how does the ACM approach constrain the neighboring weights and what is the consequence of such constraint? In this subsection we

examine the effects of these constraints through the lens of *regularization*. Let us denote the sum of all the elements of a column of $\mathbf{M}$ as $\boldsymbol{M_j}$, in other words $\boldsymbol{M_j} = \sum_{i=1}^{N_I} \boldsymbol{M_{ij}}$. Inserting this expansion into Eq. (1) and explicitly writing out the values in the periphery matrix $\mathbf{S}$ of ACM (Fig. 2) leads to the following expression:

$$\begin{aligned}\sum_{i=1}^{N_I}\sum_{j=1}^{N_O} \boldsymbol{W_{ij}} &= \boldsymbol{M_1} - \boldsymbol{M_2} + \boldsymbol{M_2} - \boldsymbol{M_3} + \ldots \\ &\quad + \boldsymbol{M_{N_{O-1}}} - \boldsymbol{M_{N_O}} \\ &= \boldsymbol{M_1} - \boldsymbol{M_{N_O}}. \end{aligned} \tag{4}$$

Eq. (4) demonstrates that any non-negative weight matrix $\mathbf{M}$ trained with the ACM approach must satisfy the constraints on the first and last columns in the matrix $\mathbf{M}$. For a quantized matrix where the elements of matrix $\mathbf{M}$ have bit precision B, each element can only be assigned one of 2^B values. Thus, for any index j, the column $\boldsymbol{M_j}$ can be assigned 1 of 2^B values per element, leading to $N_I \times 2^B$ different values. Consequently, for quantized matrices, Eq. (4) constrains $\sum_{i=1}^{N_I}\sum_{j=1}^{N_O} \boldsymbol{W_{ij}}$ to $2 \times N_I \times 2^B - 1$ values. When each element in $\mathbf{M}$ has a small set of possible values (i.e., when 2^B is smaller), this constraint is tighter. Thus, leading to a regularization effect when training with ACM. As the results in Section IV-B indicate, the nearest-neighbor coupling of ACM increases device variation resilience in reverse relation with B (the smaller B, the higher the resilience). However, ACM based training is not meant to replace standard regularization methods, e.g. L-2, dropout, etc, which have a much stronger regularization effect.

IV. Evaluation and Results

In this section, we first evaluate the training accuracy of the ACM method and compare it with alternative mappings. We follow this with an evaluation of inference accuracy on a pre-trained network with different mappings when the weights are subject to device variation. We have developed a model for neural network training using TensorFlow [16] that incorporates the non-idealities of the synapse devices. While training, matrix $\mathbf{M}$ is constrained to be non-negative and is followed by a periphery matrix that is defined as a fixed layer with values in $\{-1, +1, 0\}$ as depicted in Figs. 1 and 2. In our studies, we consider two non-ideal device characteristics that virtually exist in all physical synapse devices used for XBar arrays and impact the accuracy of DNNs trained on XBar arrays: **(i)** limited weight precision (i.e., the number of representable states) and **(ii)** non-linear weight update of XBar array synapse devices. For the former, we quantize the weights similar to [17], and for the latter, we present results for devices with symmetric increase/decrease steps [4], [18] (Fig. 4a) to isolate the effect of the nonlinear weight update on ACM from the effect of the nonlinearity on the learning rule. Since ACM is a linear transform, it is also compatible with learning rules tailored for devices with asymmetric weight update nonlinearity [19]. We present results for activations quantized to 8 bits of resolution; results on 6-bit quantized activations followed the same trend and were omitted for brevity.

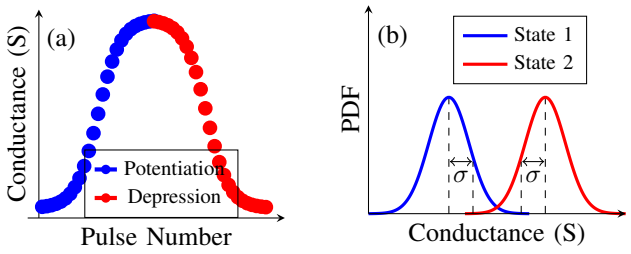


Fig. 4: **(a)** Up/down symmetric non-linearity observed in [4], [18] and assumed in training with non-linear weight update; **(b)** Gaussian distribution used to model device variation for a 1-bit (2 state) device, observed in [10].

To evaluate DNN training accuracy when using ACM, we train a variant of LeNet [20] with the MNIST dataset; we also train a VGG-9 [21] network (with 6 convolutional layers and 3 fully-connected layers) and a ResNet20 network [22] with the CIFAR-10 dataset using a vanilla stochastic gradient descent. We train four types of models for the above networks: **(i)** a baseline model, i.e., the original network trained with signed weights, **(ii)** DE, i.e., a network trained using non-negative weights and the periphery matrix in Fig. 1a, **(iii)** BC, i.e., a network trained using non-negative weights and the periphery matrix in Fig. 1b, and **(iv)** ACM, i.e., the network trained using non-negative weights and the periphery matrix in Fig. 2. We also evaluate the impact of device variation on the inference accuracy of the VGG-9 network trained with the CIFAR-10 dataset when different mappings are used. Variation is modeled as a zero-mean, normal distribution [10] as depicted in Fig. 4b, and this is added to the desired conductance value. Finally, we investigate the system-level characteristics of a XBar-based accelerator that uses different mappings with the NeuroSim+ [7] tool in terms of area, delay, and energy.

A. Neural Network Training Accuracy

Figs. 5a and 5e show training and test accuracies for the LeNet and ResNet20 networks with single-precision floating-point (FP32) weights, respectively. The three mapping schemes achieve results equivalent to the baseline model. Furthermore, the training and test errors follow similar trajectories as a function of the number of epochs. This is consistently observed for different networks and training conditions for the two datasets. These observations validate our analysis in Section III, experimentally showing that the three decompositions (DE, BC, ACM) can achieve similar accuracy in DNN training when there are no constraints on weight precision during training. Note that while ACM provides identical test accuracy at FP32 weights, the training accuracy is lower than that of DE and BC in Fig. 5e. This is due to the mild regularization effect of ACM that is discussed in Section III-E.

Figs. 5b, 5c, and 5d show the effect of limited XBar array weight precision on training DNNs on MNIST and CIFAR-10 tasks. The test errors are shown as a function of the number of bits for weight precision. Since there exists no *array-level*

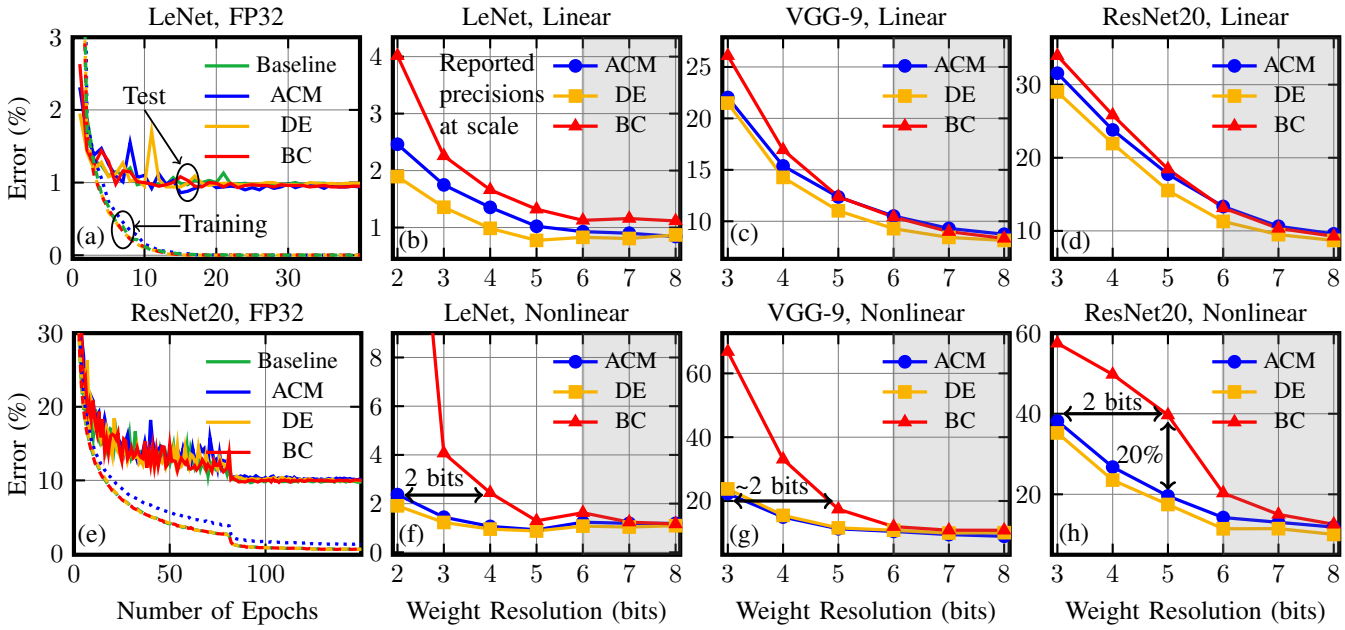


Fig. 5: This figure presents training results with MNIST and CIFAR-10 datasets. **(a)** and **(e)** are results from training with FP32 weights and activations and the solid and dashed lines are test accuracies and training accuracies, respectively. **(b)**, **(c)**, and **(d)** are results from training with limited weight precision with linear weight update and 8-bit activations. **(f)**, **(g)**, and **(h)** show results for training with limited weight precision with non-linear weight update and 8-bit activations. The grey area illustrates the bit precisions that have not been demonstrated experimentally at the array scale for synapse devices.

experiments demonstrating XBar array synapse devices with weight precision higher than 5 bits, we focus our study on weights from 2-6 bits. For precision lower than 6 bits, it can be observed that the error of DE is lower than that of the other mappings. As one would expect, this is because DE uses twice the number of elements as BC and ACM and consequently has twice the range in weight representation. When using ACM, some of the resolution lost through BC is recovered, placing its accuracy between BC and DE. The ACM encoding distributes the weight value over two columns to provide better tolerance to limited resolution compared to the BC approach. Thus, at *resource parity*, ACM provides a resolution advantage over the BC approach. At precision higher than 5 bits, for DNNs trained on the MNIST dataset, training accuracy saturates to values achieved through FP32 training. However, results on the CIFAR-10 dataset (Figs. 5c and 5d) start to diverge from the FP32 results for precision lower than 8 bits due to the higher complexity of the task.

When training with non-linear weight update (Figs. 5f, 5g, and 5h), the differences between DE and BC approaches become even more apparent. As before, the error when training with ACM is lower than the error obtained when training using BC, approaching the error of DE. However, the accuracy improvement is much more dramatic due to the disparate accuracy impact of the nonlinear weight update. These results show that ACM consistently improves upon the accuracy of BC while using the same hardware resources. The VGG-9 network is overparameterized and better offsets the non-linearity as compared to the ResNet20 network. Therefore, the accuracy decline starts at 5 bits in Fig. 5g rather than 6 bits in Fig. 5h. The largest gain is obtained for non-linear weights when training with 5-bit precision XBar arrays or lower: e.g. for ResNet20, effectively 2 bits in weight resolution are recovered, leading to a 20% improvement in accuracy.

B. Effects of Device Variation on Neural Network Inference

We evaluate the inference accuracy of a VGG-9 network trained on CIFAR-10 under the conditions of device variation when operating with different weight precision. After training, variation is added to the trained model weights and the inference accuracy is evaluated without any fine-tuning. Fig. 6 shows the results averaged over 25 samples per data point for 4 different precision values. There is a disparity in DNN inference accuracy even when not subjected to any device variation (e.g., see Fig. 5e for the difference in quantized DNNs trained on CIFAR-10 with DE, BC, and ACM). Results on inference with added variation show that this initial disparity is dramatically increased and BC consistently performs worse than the other two mappings regardless of the bit precision. Due to limited space, we only show results for 1-bit, 3-bit, 4-bit, and-6 bit weights; the 2-bit and 5-bit weights follow the expected trends. ACM performs better than DE and BC for 1, 2, and 3-bit weights and DE outperforms the other mapping methods for the 4, 5, and 6-bit weights. The improvement in inference accuracy when using ACM at lower bit precision may appear counterintuitive. However, this can be explained by considering the regularization effect of ACM discussed in Section III-E. At lower bit precision, the constraint is tighter,

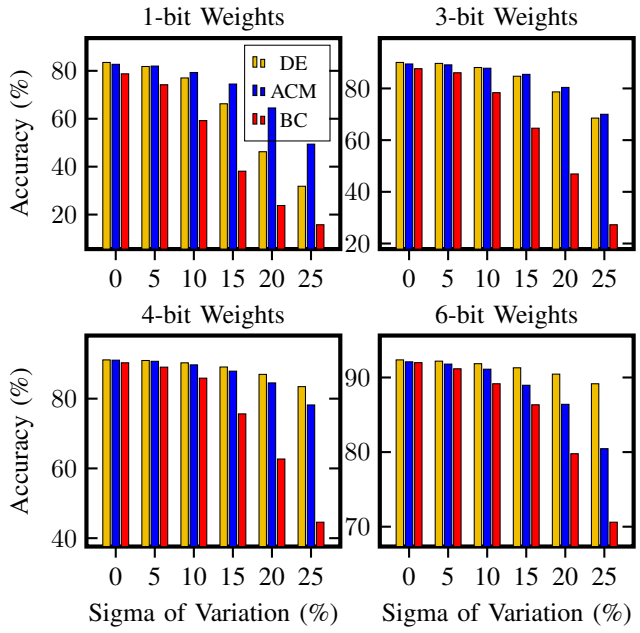


Fig. 6: Effects of device variation on the inference accuracy of a VGG-like network trained with different mapping approaches and bit precision on the CIFAR-10 dataset

while it is relaxed at higher bit precision. This constraint strengthens the network against device variation.

C. System-Level Evaluation

Table I shows system-level results for the three mapping approaches generated using the NeuroSim+ [7] tool with the default parameters in the 14nm tech node. The assumed peripherals include MUX, ADC, word-line decoder, bit-line and select-line switch matrices, adders, and shift registers. The read energy and latency values are for one epoch of training a two-layered multi-layer perceptron (MLP) network with a XBar-based hardware accelerator. Read energy, area, and read delay values for BC and ACM approaches are exactly the same, as there is practically no difference in their hardware resource utilization. The read energy of DE is 7× more than that of the ACM due to the longer wires for rows of the XBar array. DE uses 2.3× XBar area compared to the ACM, as it requires twice as many elements. The peripherals are also larger and require more area. Furthermore, DE has a 1.33× higher read delay due to the additional columns that need to be multiplexed for the associated peripherals.

V. Conclusion

We introduced the ACM mapping method to mitigate the effects of limited weight resolution and non-linearity on neural network training while incurring minimal hardware overhead. We demonstrated, both mathematically and by simulation, that ACM is a general approach and represents the same MVMs as previous mapping approaches. Neural network training evaluations with limited resolution and non-linearity show that ACM consistently improved upon the accuracy of BC while using the same hardware resources. The largest gain was obtained for non-linear weights when training with 5-bit precision XBar arrays or lower: effectively 2 bits in weight resolution were recovered, leading to a 20% accuracy improvement. Compared to DE, ACM can achieve comparable training accuracies while reducing the read energy consumption by 7× and area by 2.3×. Furthermore, the regularization effect of ACM makes it resilient to device variation. Assuming a 15% device variation, ACM improves the inference accuracy of a VGG network trained on the CIFAR-10 dataset with 3-bit precision XBar arrays by an average of 10% compared to other mappings.

TABLE I: System-level results of the three mapping approaches for training a two-layered MLP on XBar arrays.

Mapping	BC	DE	ACM
XBar Area (μm^2)	914	2088	914
Periphery Area (μm^2)	157	246	157
Read Energy (μJ)	2.402	14.408	2.402
Read Delay (ms)	0.240	0.318	0.240

Acknowledgment

This work was supported in part by ASCENT, one of six centers in JUMP, a SRC program sponsored by DARPA under task ID 2776.043.

References

[1] S. Han *et al.*, "Eie: efficient inference engine on compressed deep neural network," in *ISCA*, 2016.
[2] H. P. Wong *et al.*, "Metal–oxide rram," *Proceedings of the IEEE*, 2012.
[3] G. W. Burr *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, 2017.
[4] M. Jerry *et al.*, "A ferroelectric field effect transistor based synaptic weight cell," *Journal of Physics D: Applied Physics*, 2018.
[5] S. Ambrogio *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, 2018.
[6] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices," *Frontiers in Neuroscience*, 2016.
[7] P.-Y. Chen *et al.*, "Neurosim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *IEDM*, 2016.
[8] C. C. Chang *et al.*, "Mitigating asymmetric nonlinear weight update effects in hardware neural network based on analog resistive synapse," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, 2017.
[9] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, 2016.
[10] Y. Lin *et al.*, "Demonstration of generative adversarial network by intrinsic random noises of analog rram devices," in *IEDM*, 2018.
[11] W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing," *Proceedings of the IEEE*, 2018.
[12] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.
[13] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.
[14] G. W. Burr *et al.*, "Experimental demonstration," *IEEE T-ED*.
[15] P. Narayanan *et al.*, "Reducing circuit design complexity for neuromorphic systems based on non-volatile memory," in *ISCAS*, 2017.
[16] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX OSDI*, 2016.
[17] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv*, 2016.
[18] J. Woo *et al.*, "Resistive memory-based analog synapse: The pursuit for linear and symmetric weight update," *IEEE Nanotechnol. Mag.*, 2018.
[19] M. Fouda *et al.*, "Independent component analysis using rrams," *IEEE Transactions on Nanotechnology*, 2018.
[20] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," 2015.
[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.
[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.