This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

Neural network accelerator design with resistive crossbars: Opportunities and challenges

S. Jain, A. Ankit, I. Chakraborty, T. Gokmen, M. Rasch, W. Haensch, K. Roy, A. Raghunathan

Deep neural networks (DNNs) achieve best-known accuracies in many machine learning tasks involved in image, voice, and natural language processing, and are being used in an ever-increasing range of applications. However, their algorithmic benefits are accompanied by extremely high computation and storage costs, sparking intense efforts in optimizing the design of computing platforms for DNNs. Today, GPUs and specialized digital CMOS accelerators represent the state-of-the-art in DNN hardware, with near-term efforts focusing on approximate computing through reduced precision. However, the ever-increasing complexities of DNNs and the data they process have fueled an active interest in alternative hardware fabrics that can deliver the next leap in efficiency. Resistive crossbars designed using emerging non-volatile memory technologies have emerged as a promising candidate building block for future DNN hardware fabrics, since they can natively execute massively parallel vector-matrix multiplications (the dominant compute kernel in DNNs) in the analog domain within the memory arrays. Leveraging in-memory computing and dense storage, resistive-crossbar-based systems cater to both the high computation and storage demands of complex DNNs, and promise energy efficiency beyond current DNN accelerators by mitigating data transfer and memory bottlenecks. However, several design challenges need to be addressed to enable their adoption. For example, the overheads of peripheral circuits (ADCs and DACs) and other components (scratchpad memories and on-chip interconnect) may significantly diminish the efficiency benefits at the system level. Additionally, the analog crossbar computations are intrinsically subject to noise due to a range of device and circuit level non-idealities, potentially leading to lower accuracy at the application level. In this paper, we highlight the prospects for designing hardware accelerators for neural networks using resistive crossbars. We also underscore the key open challenges and some possible approaches to address them.

1. Introduction

The field of artificial intelligence (AI) has witnessed tremendous growth in recent years with the advent of deep neural networks (DNNs) that surpass humans in a variety of cognitive tasks. Consequently, many real-world products and services such as speech recognition, image analysis, natural language processing, and search engines use DNNs [1]. The algorithmic performance of DNNs comes at extremely high computation and memory costs that pose significant challenges to the hardware platforms executing them. Currently, GPUs [2] and specialized digital CMOS accelerators such as Google's TPU [3], Microsoft's Brainwave [4] and Intel's Nervana Neural Network Processor [5] are the state-of-the-art in DNN hardware. However, the ever-increasing complexity of DNNs and the data they process have led to a quest for the next quantum improvement in processing efficiency.

Resistive crossbars, in particular, have attracted significant interest due to their ability to natively perform vector-matrix multiplication, *i.e.*, the dominant computational kernel in DNNs, highly efficiently and compactly. They can be designed using emerging non-volatile memory (NVM) technologies including PCM [6], ReRAM [7], [8], spintronics [9], [10], and Ferroelectric FETs [11], [12]. These devices possess several highly desirable characteristics such as high density, low voltage operation, low leakage, and non-volatility. Consequently, there have been many research efforts that focus on resistive crossbars at the device, circuit, architecture and algorithmic levels [13], [14], [15], [16], [17], [18], [19].

Resistive crossbar systems uniquely cater to both the high computation and the high storage demands of large-scale DNNs. First, multi-bit NVMs enable highly dense memory arrays that can achieve at least an order of magnitude higher storage density than SRAMs (6T cells) [20]. Second, they enable execution of massively parallel vector-matrix multiplications (in the analog domain) within the crossbar array itself. For example, a vector of input voltages applied at crossbar rows could be multiplied by a 2D matrix of weights stored as conductances of the crossbar elements to yield a vector of output currents at the crossbar columns. The in-memory computing ability of resistive crossbars mitigates data transfer and memory bottlenecks for the data stored in the crossbar, and has the potential to achieve efficiency well beyond multi-cores, GPUs and current DNN accelerators. In comparison to state-of-the-art DNN accelerators, resistive crossbar based accelerators are projected to achieve large improvements in compute density and energy efficiency [9], [16], [17], [20]. However, the system-level improvements are highly sensitive to design choices at (i) the algorithmic level (network topology, precisions of weights and activations), (ii) the architecture level (micro-architecture of the crossbar-based processing element, memory hierarchy, interconnect network, and mapping and scheduling), and (iii) the circuit and device levels (*e.g.*, crossbar dimensions, precision of each crossbar element, minimum-to-maximum

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

conductance ratio, variations).

Resistive crossbar based neural fabrics are still a nascent technology with several design challenges that need to be addressed. For example, a key challenge with resistive crossbars is that they suffer from various device and circuit-level non-idealities that manifest as errors in the realized vector-matrix multiplications. These computational errors can accumulate and propagate across a deep network, degrading the application-level accuracy [21]. Although DNNs do possess some intrinsic error resilience, it is still essential to model, evaluate, and mitigate the impact of crossbar non-idealities. In this paper, we highlight design challenges in realizing inference and training systems for DNNs using resistive crossbars along with some approaches to address them.

The rest of the paper is organized as follows. In Section 2, we provide a brief background on resistive crossbar arrays designed for executing vector-matrix multiplications. Section 3 discusses the architecture and system-level challenges and design choices for resistive crossbar based neural fabrics. Section 4 presents modeling frameworks to evaluate and train large-scale DNNs on resistive crossbar systems. It also highlights the challenge posed by crossbar non-idealities to the accuracy of large-scale DNNs and presents a few compensation schemes to address the issue. Finally, Section 5 concludes the paper.

2. Designing resistive crossbar arrays for DNNs

Figure 1 illustrates a resistive crossbar array (RCA) designed for executing DNN inference and training functions during forward-pass, backward-pass, and weight update operations. As shown, an RCA consists of synaptic elements arranged into 'M' rows and 'N' columns, Digital-to-Analog (DAC), and Analog-to-Digital (ADC) converters, and write circuitry. The synaptic elements may be realized using non-volatile memory (NVM) technologies such as PCM [22], [23], ReRAM [24], [25], [26], Spintronics [27], Ferroelectric FETs [11], [12], and embedded Flash [28], [29], [30]. Further, the synaptic elements in the same row share a wordline (WL), and those in the same column share a bitline (BL). RCA supports three main operations (i) Serial programming, (ii) vector-matrix multiplications, and (iii) parallel updates. The programming operations, *i.e.*, write operations on synaptic elements, are performed row-wise, wherein, the write circuitry applies the necessary current and set them to the desired conductance. In contrast, vector-matrix multiplications are executed by simultaneously driving all WLs (BLs) using DACs and sensing the resulting current flowing through each BL (WL) using ADCs. In a single array access, RCAs can realize massively parallel vector-matrix multiplications, *i.e.*, the primitive compute kernel in DNNs during the forward and the backward pass. The realized vector-matrix multiplications during forward- and backward-pass are represented by $I_{out} = V_{in} \times G$ and $I_{out} = V_{in} \times G^T$, respectively, where Vin represents the input voltages, Iout the output currents, and G denotes the programmed synaptic conductances. For example, the vector-matrix multiplications during the forward-pass are

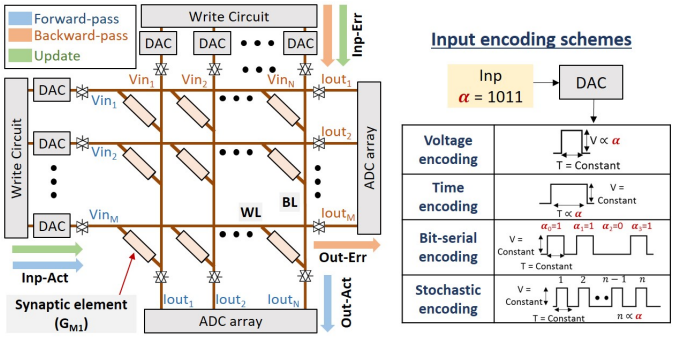

Figure 1 Resistive crossbar array designed for executing DNN primitives. During the forward-pass, inputs (Inp-act) are applied using the DACs on the left, and the outputs (Out-act) are received at the bottom. During the backward-pass, inputs (Inp-Err) are applied using the DACs on the top, and the outputs (Out-err) are obtained on the right. Updates are achieved by applying both inputs (Inp-act, Inp-Err) simultaneously using DACs on the left and the top.

executed by applying the input activations (Inp-Act) on WLs as voltages (V_{in}) to obtain currents (I_{out}) on BLs. Subsequently, the currents are converted to digital output activations (Out-Act). Finally, the parallel update operation is performed by driving DACs along both BLs and WLs. The errors (Inp-Err) and activations (Inp-Act) are applied on BLs and WLs, respectively, resulting in local updates in all synaptic elements.

2.1. Design considerations for vector-matrix multiplications

Vector-matrix multiplications (VMMs) can be executed on RCAs using a variety of input and weight encoding schemes. We briefly discuss these choices in turn below.

Input encoding. Inputs can be applied to RCAs using various encoding schemes including voltage, time, bit-serial and stochastic encodings [14], [15], [31], [32], as shown in Figure 1. In voltage and time encoding schemes, the voltage magnitude (V) and the pulse width (T) are modulated, respectively, in proportion to the digital inputs. In contrast, the bit-serial and stochastic encoding schemes utilize multiple voltage pulses, each of fixed time and width, to convert digital inputs. In bit-serial and stochastic encoding, a vector-matrix multiplication operation is executed for each voltage pulse, leading to multiple partial outputs. These partial outputs are combined either using shift-and-add logic for the bit-serial scheme or by simply adding them for the stochastic scheme [15]. The stochastic and bit-serial encodings differ based on the method used for converting digital values to voltage pulses. The pulses are generated deterministically in bit-serial encoding using standard parallel-to-serial conversion. In contrast, the pulses are generated randomly in stochastic encoding such that the total number of pulses in the pulse-train is on average proportional to the magnitude of the encoded digital number. The input encoding schemes present interesting design trade-offs. For example, the bit-serial encoding simplifies the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

DAC design substantially, leading to energy and area efficiency. However, it requires multiple VMM operations and thereby incurs higher latency in comparison to the voltage and time encoding schemes.

Weight encoding. DNNs have both positive and negative weights. However, RCAs consist of only positive conductances. To encode signed weights in RCAs two schemes have been proposed. (i) A range shifting approach, wherein half of the synaptic device's dynamic range is reserved for representing positive values and the other half for negative values [32], [31]. In this scheme, an offset is generated to partition the dynamic range and to determine the numerical '0'. (ii) Using separate (two) devices to store positive and negative weights [16]. Both these schemes have unique advantages; the second scheme provides higher precision, while the first offers superior area efficiency [32]. Another challenge posed by RCAs is that the practical synaptic device precision is limited to 2-6 bits [33]. Therefore, to realize higher precision (8-16 bits) vector-matrix multiplications, the weights are split based on bit significance and mapped to multiple crossbar columns or multiple crossbars. For example, mapping an 8-bit weight to RCAs with 2-bit synaptic elements requires four RCA columns, and the final output is obtained by shifting and adding the individual column outputs based on bit significance. Programming a synaptic element to the desired conductance level is also quite challenging, and often requires program-read-verify or closed-loop programming, wherein a read operation is performed after every write operation to verify the success of the write operation. Subsequently, we repeat the write-read cycle in a loop to precisely program the synaptic device to the desired conductance state. To mitigate such expensive writes (that may require 10s-100s of pulses), parallel write techniques that perform smaller updates on the entire crossbar in a single cycle have been demonstrated [15], [32], [31]. However, parallel writes are imprecise and have been shown to work only for simple tasks (MNIST). Therefore, the impact of such fast but imprecise technique on large-scale DNNs that perform complex tasks (e.g., ImageNet classification using ResNets) remains an open question. Further, device-to-device variations in the synaptic devices also pose functional challenges, as they impact the application-level accuracy of DNNs on crossbar based hardware.

2.2. Design considerations for weight update

Implementing the weight update on a 2D crossbar array of resistive devices in parallel is challenging. It requires calculating a vector-vector outer product which consists of a multiplication operation and an incremental weight update to be performed locally at each synaptic element. There are at least three different schemes proposed to perform a fully parallel update on crossbar arrays that rely on some sort of a coincidence detection at the cross-point and generation of either deterministic [34], [35] or stochastic pulses [36], [14] from the periphery. All three schemes are shown to work for a small fully-connected network using the MNIST dataset,

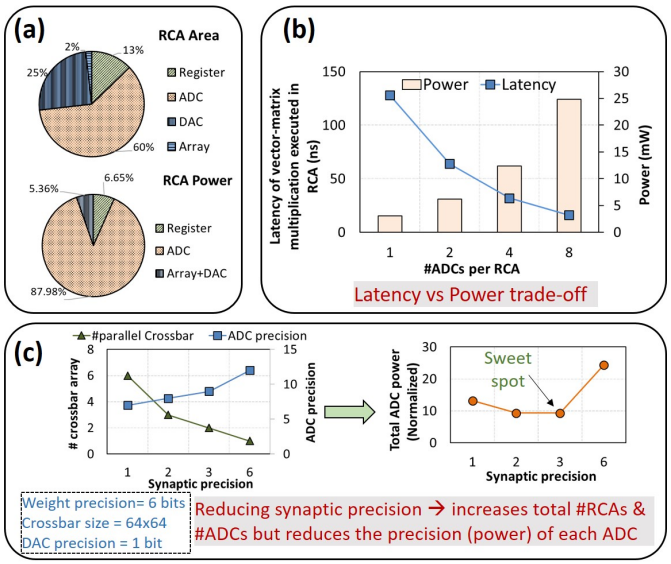

Figure 2 (a) Area and power breakdown of a 64x64 RCA with 10-bit ADCs [39] and 6-bit DACs [40], (b) Latency-power trade-off in a 128x128 RCA with 8-bit ADCs [39], and (c) Power optimization by exploiting the relationship between synaptic precision and ADC power.

and the stochastic pulsing scheme has been shown to work for medium scale networks [15], [37]. It is important that these update schemes be tested on large-scale networks to determine their viability.

Material and device design. In order to achieve a successful training there is also a set of characteristics that has to be realized by the crossbar elements performing the update. A key requirement is that these crossbar elements must change conductance symmetrically when subjected to positive or negative pulse stimuli [14]. Indeed, this requirement differs significantly from standard memory and therefore requires a systematic search for new physical mechanisms, materials and device designs to realize a desirable resistive element for DNN training. It is also appreciated that accomplishing such symmetrically switching analog devices is a difficult task. Besides material engineering, circuit assisted solutions [38] combined with algorithmic modifications might, conceivably, relax the device requirements and hence possibly enable the realization of crossbar-based DNN training accelerators in the near future. We note that the switching symmetry is not an important parameter for inference-only accelerators as the weight initialization is done only once using closed loop programming.

2.3. Peripheral design

Next, we discuss the impact of peripherals such as ADCs and DACs on the efficiency (power, area, and latency) of RCAs. Figure 2(a) shows the area and the power breakdown of an example RCA design that assumes the crossbar dimension to be 64x64, and ADC and DAC precisions to be 10-bit and 6-bit, respectively. The ADC and DAC circuits utilized in this design are proposed in [39], [40]. For this

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

specific design, ∼65% and ∼85% of the total RCA area and power, respectively, are consumed by the ADCs. Therefore, ADCs are one of the most critical design components in crossbar based hardware. Various trade-offs may be explored to reduce the total ADC power/area and increase the efficiency of DNNs on crossbar-based systems. We highlight some of these trade-offs in Figure 2 (b) and (c). Sharing ADCs across RCA columns is the most popular approach to reduce ADC power/area. However, sharing ADCs can increase the latency of vector-matrix multiplication operations. Figure 2(b) shows the latency-power trade-off of a 128x128 RCA. Using one ADC per crossbar (i.e., 1 ADC per 128 columns) in a time-multiplexed manner requires the least power, and many recently proposed crossbar based accelerators [17], [20] favor this design choice. Further, there also exists a trade-off between synaptic precision and ADC power, as shown in Figure 2(c). For a crossbar of fixed size, reducing synaptic precision (i) increases the number of crossbars required (as we need to slice the weights and map it to proportionally more synaptic elements), but (ii) lowers the precision requirement of each ADC linearly. A linear decrease in ADC precision causes a superlinear reduction in ADC power, creating opportunities for optimizing power by changing the synaptic precision. Exploring such area-power-latency trade-offs is key to the design of energy and area efficient RCAs.

3. Crossbar-based DNN architectures

3.1. Inference accelerators

The key features of a resistive crossbar that drive the design of an inference accelerator are (i) high storage density, and (ii) high write cost (energy and latency). First, the compact cell structure ($4F^2$ cell area for the in-line two terminal selector configuration) [32] and the multi-bit storage device (2-6 bits) [33] enable resistive crossbars to achieve order-of-magnitude higher on-chip storage density compared to SRAM. A large on-chip memory can enable efficient data reuse to optimize the inference efficiency. Second, the high write voltage [32] and multiple programming cycles (program-verify approach [41]) result in much higher write cost (energy and latency) than SRAM. Expensive memory writes limit the applicability of a time-multiplexed architecture, where the crossbars are reused across layers by re-programming weight matrices and executing the corresponding VMM operations. Consequently, a spatial architecture where the DNN is partitioned such that the weights (stationary data) are pinned to crossbars located across multiple cores is more efficient as it leverages the benefits of high storage density while alleviating the costly writes. Note that the benefits of spatial architectures rely on the premise of pinning of weights under the assumption that the weights do not change during a DNN inference. However, crossbars written once during the configuration time, may be subject to resistance drifting over time [42]. This might require repeating write cycles for restoration of weights in order to ensure accurate functionality.

Recall that, crossbars typically perform VMM operations with stationary matrices, i.e., DNN weights (Section 2). However, DNNs require several other operations such as vector additions, vector multiplications, scalar and non-linear operations. Additionally, other non-stationary data such as inputs and partial sums need to be stored and moved. Consequently, designing a computation core requires augmenting the resistive crossbar with CMOS based digital logic units as well as memory units to provide the required generality for DNN workloads. A naive approach towards integrating the CMOS and resistive technologies is not viable because of the huge disparity in compute and storage density between the two technologies. CMOS digital logic (multiply-and-accumulate) has an order of magnitude higher area requirement than a crossbar of equal output width (∼20×) [17]. Moreover, a crossbar's storage density (2-bit cells) is $160MB/mm^2$, which is at least an order of magnitude higher than SRAM (6T, 1-bit cell) [20].

Resistive crossbars have been used to build special-purpose ML accelerators such as SPINDLE [9], RENO [43], PRIME [16], ISAAC [20] for Convolutional Neural Networks (CNN) and Multi Layer Perceptrons (MLP). Past research has also proposed using memristive crossbars for Spiking Neural Networks [44], [45], Boltzmann Machine [46] and BSB [47]. These efforts have shown substantial improvements compared to CMOS-based general-propose architectures (CPU, GPU) and ASICs, thereby demonstrating the potential of resistive crossbar based systems. However, there are important challenges to be addressed for enabling the adoption of resistive crossbar based systems for general-purpose ML workloads. Each design supports limited (one or two) types of neural networks, where layers are encoded (hardwired) as state machines. The increasing pervasiveness of ML workloads in different application domains has given rise to large varieties of DNNs owing to their task-specific nature. Consequently, there is a need to think in terms of instruction set architecture (ISA) programmability as the hardwired approach can lead to increased decoding overhead and complexity. Further, supporting flexible data movement and control operations to capture the variety of access and reuse patterns in different DNNs is important. Since crossbars have high write latency, they typically store constant data while variable inputs are routed between them in a spatial architecture. This data movement can amount to a significant energy consumption which calls for flexible operations to optimize the data movement.

PUMA [17], a recently proposed inference accelerator addresses the needs by proposing a general-purpose and ISA-programmable accelerator built with resistive crossbars. It is a spatial architecture organized in three hierarchies: cores, tiles and nodes, and features a microarchitecture, ISA and compiler co-designed to optimize data movement and maximize energy and area efficiency. Figure 3 shows a PUMA core, which consists of an instruction execution pipeline, functional units and analog crossbars. The instruction execution pipeline and specialized ISA enable compact representation of ML workloads with low decoder complexity. This is based on the observation that despite the large variety of ML workloads, these workloads share many

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

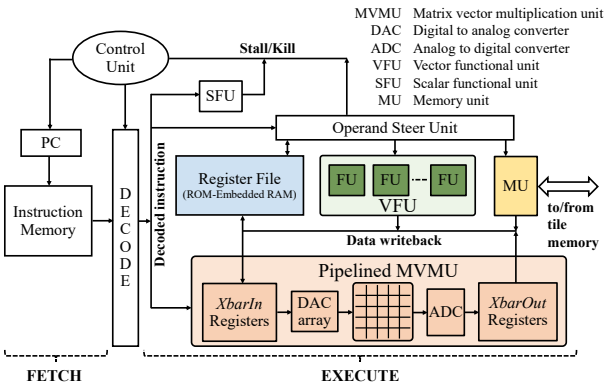

Figure 3 Core architecture based on hybrid CMOS-memristive technology. The VFU is 4-lane wide. The MVMU is comprised of eight 128×128 RCAs with each crossbar cell representing 2-bits. The RCA receives inputs from 1-bit DAC array and sinks its output to a 8-bit ADC, shared across all 128 columns. [17]

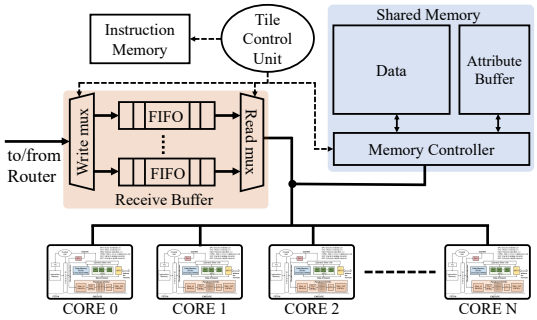

Figure 4 PUMA Tile Architecture [17]

low-level operations (instructions). PUMA cores employ temporal SIMD (single instruction multiple data) based VFUs for linear and non-linear vector operations, where a narrow-width VFU executes wide vector operations over multiple cycles. Since ML workloads have high data parallelism, they execute wide vector operations, which motivates having wide vector instructions. On the other hand, hardware considerations motivate having narrow VFU vector width to avoid offsetting the area efficiency of crossbars (discussed in Section 2.1). Therefore, temporal SIMD balances the tradeoff between workloads favoring wide vector width and hardware favoring narrow vector width. The VMM operations are executed using the analog crossbars present in the MVMU (shown in Figure 3). DAC reuse across crossbars that share the inputs and ADC reuse across a crossbar's columns reduce the area overhead from peripherals to enable an efficient MVMU design. PUMA cores use a special register file designed with ROM-Embedded RAM [48], which provides regular operand buffering (RAM mode) as well as an in-memory primitive to realize transcendental (e.g., sine, logs, tanh, etc.) functions (ROM mode). In summary, a PUMA core colocates different kinds of execution units: VFU, MVMU and Register File (ROM-Embedded RAM) to leverage near-memory processing for reducing the inference cost and employs techniques to appropriately size the execution units for preserving the high storage density from crossbars.

Figure 4 illustrates the architecture of a PUMA tile, which consists of multiple cores connected together with a shared memory. The shared memory consists of the attribute buffer which tracks the producer-consumer relationships between cores to enable inter-core synchronization. Subsequently, optimizing the DNN mapping to reduce the communication between cores reduces the data movement cost. The receive buffer enables data to be received through the network irrespective of the underlying receive-instruction ordering in the program. This independence is important because receive instructions are executed in program order in a blocking manner for hardware simplicity. It is worth noting that the data movement in a spatial architecture can be seen as analogous to the conventional data movement within the memory hierarchy in temporal architectures (CPU, GPU), where the cost increases from intra-core communication to inter-tile communication. In summary, PUMA tiles enable efficient inter-core communication to improve the inference efficiency by reducing data movement costs.

Multiple tiles are connected together using an on-chip network to form a PUMA node. Further, multiple PUMA nodes can be connected using suitable chip-to-chip interconnect, such as CCIX, Gen-Z or OpenCAPI to execute DNNs larger than the node's storage capacity. Figures 5 and 6 show the inference energy and inference latency (at equal area) respectively for PUMA compared to a CPU (Intel Skylake) and a GPU (NVIDIA Pascal). The PUMA node used in evaluations consists of 138 tiles in a $\simeq 90mm^2$ area, and can store upto $\simeq 69MB$ of weight data. Note that Pascal obtains lower energy and latency than Skylake for all benchmarks. It can be seen that across all benchmarks PUMA achieves significant energy reductions at lower or comparable latency than Pascal. CNNs show the least energy reductions (upto $13\times$) owing to the abundant weight and input reuse which is leveraged well by CMOS systems (CPU, GPU) to amortize the expensive memory accesses. MLPs show upto $80.1\times$ energy reductions, while LSTMs show the highest energy reductions of upto $2446\times$. While both LSTMs and MLPs have little to no weight reuse, MLPs show lower savings than LSTMs owing to their smaller model sizes. A small model (few million parameters) that can be fit on the on-chip memory (last-level cache) on CPUs or GPUs does not expose the high off-chip memory energy consumption. For similar reasoning as energy comparisons, PUMA's inference latency reductions are lower for CNNs than LSTMs. However, PUMA's latency is higher than GPU for MLPs. This is because the small model size of MLPs does not expose the high off-chip memory latency. Compared to the Google TPU (CMOS ASIC), PUMA demonstrates peak area and power efficiency improvements of more than $9.7\times$ and $1.8\times$ respectively across all ML workloads. While PUMA's peak efficiency does not get affected with data-batching, TPU's efficiency is almost an order lower for lower data-batching (batch-size=1) due to its inability to amortize the weight data movement. Note that

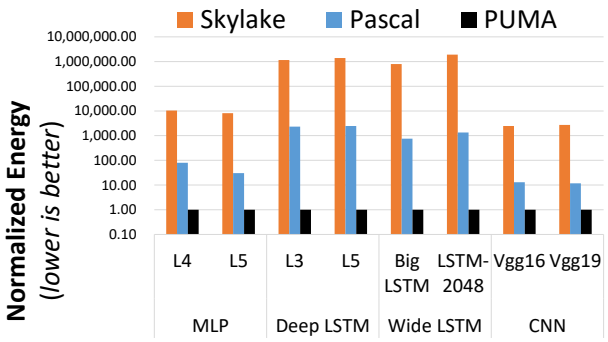

Figure 5 PUMA inference energy comparison with respect to CPU (Skylake) and GPU (Pascal) for batch size 1 [17]

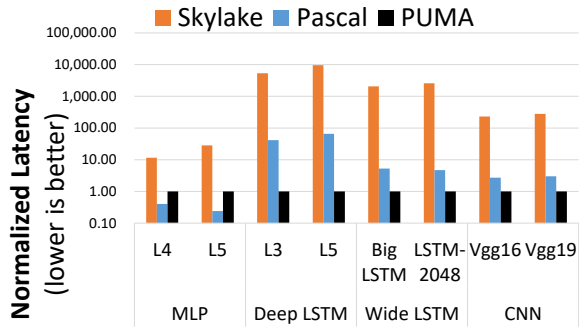

Figure 6 PUMA inference latency comparison with respect to CPU (Skylake) and GPU (Pascal) for batch size 1 [17]

the common use-case of inference applications has little to no data batching [49].

3.2. Training accelerators

The system-level considerations for training differ considerably from inference. For instance, during inference, the computed activations can be discarded once they are fed to the following layers. However, during training, these computed activations should be stored, as they are also required during the weight update operations for computing the gradients. Therefore, a system for DNN training has much higher storage requirements than a system for DNN inference. For example, a relatively simple ResNet-50 DNN model operating on the ImageNet dataset generates ∼50 million activations during the forward pass, resulting in a 100 Mbytes (2 bytes/activation) storage requirement per image. The storage requirement further increases for larger networks, bigger datasets, higher resolution inputs, and concurrent processing of multiple images due to pipeline parallelism. For even larger DNNs, the storage requirements can be multiple gigabytes. Thus, the on-chip memory of crossbar-based inference accelerators is often insufficient, and these accelerators have to be accompanied with high bandwidth and high capacity external memory to enable a training system that is capable of handling large-scale DNNs. Consequently, a training system also requires the traditional memory hierarchy (with multiple levels), to efficiently hide the main memory latency. Lastly, it is also challenging to achieve data parallelism in crossbar-based training accelerators due to weight synchronization issues [50]. In data parallelism, weights are replicated across multiple crossbars, and after every forward and backward pass, the stored weight copies are updated by the same values. However, errors during the update operations cause different crossbars to obtain different updates, leading to convergence issues. Alternatively, crossbar-based training accelerators can utilize pipeline parallelism [51], [52] that also naturally fits weight stationary data-flows. In pipeline parallelism, the whole network is partitioned into a set of pipeline stages, each of which is responsible for only a small part of the network. Activations flow through the pipeline stages, while the weights are kept stationary.

4. Modeling and compensation of non-idealities

4.1. Crossbar non-idealities: Overview

One of the key challenges with resistive crossbars is that the computed function differs from the desired vector-matrix multiplication operation ($I_{\text{out}} = V_{\text{in}} G_{\text{ideal}}$) due to a range of circuit and device level non-idealities, *viz.*, wire resistances, non-linearities in peripheral circuits (DACs and ADCs), sensing resistance, driver resistance, sneak paths, imperfect write operations, non-linearities in synaptic conductances, and process variations, as shown in Figure 7. The cumulative effect of all these crossbar non-idealities manifests as errors in the VMM operations computed using RCAs. These errors in VMMs accumulate and propagate through the DNN, causing significant degradation in application-level accuracy. Therefore, it is essential to model, evaluate, and compensate for the impact of crossbar non-idealities to enable the adoption of resistive crossbar systems.

4.2. Modeling and simulating crossbar non-idealities

To evaluate the impact of crossbar non-idealities on large-scale DNNs a fast as well as accurate simulation framework is required. Device and circuit simulation models are very accurate but extremely slow and hence not feasible for large-scale network evaluation. On the other hand, fast architectural models [53], [54], [55] that target design space exploration use highly simplified error models that are inadequate for evaluating DNN accuracy on resistive crossbar systems.

A promising approach for evaluating and training DNNs on crossbar-based hardware is to utilize existing machine learning frameworks (*e.g.*, Caffe/2, PyTorch, TensorFlow) and enhance them with fast crossbar models. To simulate DNN inference, we can adopt a weight-transformation based approach, wherein the weight matrices are first transformed to non-ideal weight matrices by abstracting the effect of all crossbar non-idealities. Next, the original weight matrices are replaced with these non-ideal weight matrices. Subsequently, the VMMs are evaluated using the non-ideal weight matrices and peripheral models (DACs and ADCs) that are invoked as pre- and post-processing steps on input and output activations, respectively. To simulate DNN

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

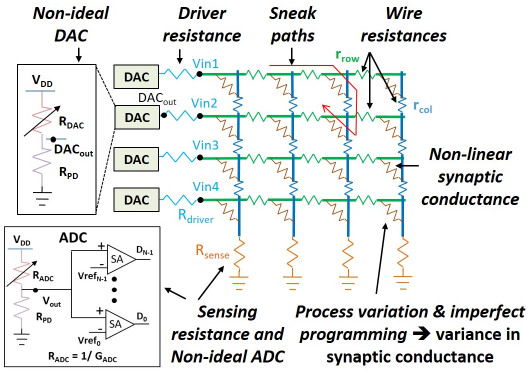

Figure 7 Crossbar non-idealities

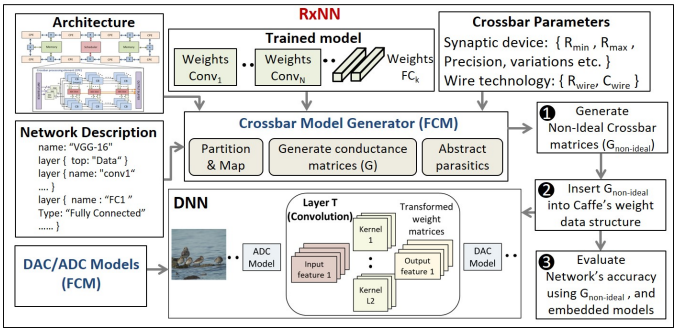

Figure 8 RxNN framework for modeling the impact of crossbar non-idealities on large-scale DNNs [21]

training, a more complex approach is required wherein the VMM operations are replaced by a simulated crossbar model that captures the non-idealities of the RCA including analog-digital conversions. Although this approach involves re-implementing the convolution and fully-connected operations, it is closer to the actual hardware and can capture static as well as dynamic non-idealities that include cycle-to-cycle variations, non-ideal switching characteristics, generation of stochastic pulse trains for updating, and temporal dependencies of the device. Next, we introduce recent examples of the two approaches in turn below.

Simulation of DNN inference. RxNN [21] is a recently proposed software framework that addresses the need for a fast and accurate functional simulator for evaluating large-scale DNNs on crossbar systems. RxNN is obtained by modifying the Caffe [56] deep learning framework. Like Caffe, RxNN models the convolution and fully-connected layers of DNNs as matrix-matrix and vector-matrix multiplications. It maps these operations to RCAs and emulates the non-ideal vector-matrix multiplications by transforming weight matrices into "non-ideal" weight matrices that reflect all non-idealities. It takes a resistive crossbar system description, RCA parameters, and a trained DNN model as inputs, and computes the DNN accuracy using embedded Fast Crossbar Models (FCMs) [21]. RxNN focuses on DNN inference and can model the impact of device and circuit-level non-idealities on the vector-matrix multiplications executed during the forward-propagation. The device-level non-idealities modeled include process-variations, non-linear synaptic devices, and imperfect write operations. The circuit-level non-idealities modeled are wire resistances, sensing and driver resistances, non-linear ADCs and DACs, and sneak paths. Since RxNN targets DNN inference, it assumes that weight programming is performed one-time. Therefore, it abstracts the impact of all device-level non-idealities into a single programming phase. However, RxNN can also handle temporal variations (e.g., drift) in synaptic devices during inference operations by periodically repeating the steps involving the programming phase. RxNN's models are five orders of magnitude faster than SPICE models and accurate to within 0.26%. RxNN also enables DNN re-training to improve inference accuracy.

The RxNN flow, shown in Figure 8, comprises of 3 major steps. First, RxNN maps the DNN to the target architecture. It reads weights from the trained model and virtually programs them into RCA instances. Subsequently, the ideal conductance matrices (G) are transformed to non-ideal matrices ($G_{non-ideal}$) by abstracting crossbar non-idealities using RxNN's crossbar model generator. Next, the $G_{non-ideal}$ matrices associated with the convolution (Conv) and fully-connected (FC) DNN layers are incorporated back into the Caffe's original weight data structure. The key to RxNN's performance and scalability is its ability to transparently utilize Caffe's underlying data structures and optimized BLAS libraries. Finally, RxNN evaluates DNNs using embedded $G_{non-ideal}$ matrices and invoking peripheral (ADC and DAC) models as pre- and post-processing steps on the inputs/outputs of each Conv and FC layer.

Using the RxNN framework, we evaluate the impact of limited precision and device and circuit-level non-idealities on inference accuracy for simple and complex DNNs (shown in Figure 9) [21]. In the figure, FP32 is a floating-point implementation, Cross-Ideal6 is an "ideal" crossbar implementation that only considers limited precision (6 bits), and Cross-NI6-64x64 is an implementation with 64x64 crossbars that considers all crossbar non-idealities. In our evaluation, we observe the accuracy degradation to be minimal for simple networks (LeNet and ConvNet) with smaller and fewer layers. However, we find the accuracy degradation to be drastic (19.8%-57.8%) for large-scale ImageNet DNNs such as VGG-16, GoogleNet, and ResNet-50. Therefore, it is essential to address the challenges posed by crossbar non-idealities to enable future adoption of resistive crossbar systems.

Simulation of DNN inference and training. In recent years, some of the present authors have developed a fast C++ library that simulates the forward, backward, and update operations on RCAs in modular fashion, i.e. different RCA hardware designs and material properties can be used with the same user-interface [14], [15], [37]. The simulation comprises of abstract models of hardware realistic components, such as generating stochastic update pulses, implementing device specific weight ranges and saturation, various types of noise, and update non-linearities and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

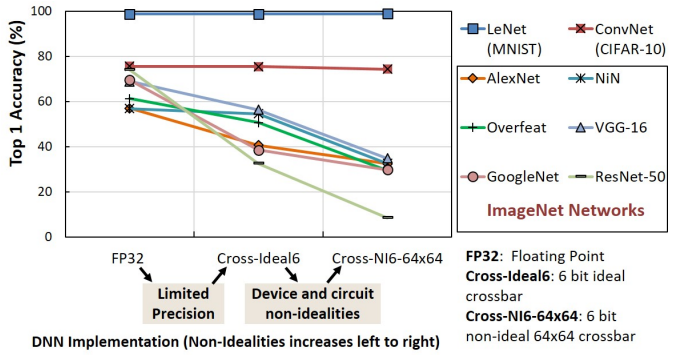

Figure 9 Accuracy degradation due to crossbar non-idealities during DNN inference evaluated using RxNN. In this evaluation, we use 64x64 crossbar arrays designed using 6-bit synaptic devices with $R_{min} = 200\text{K}\Omega$ and $R_{max} = 1.4\text{M}\Omega$, and 6-bit DACs and ADCs. The modeling details are provided in [21].

asymmetries, as well as incorporating compensatory measures, such as noise management schemes ([15]; see also below). More recently, the C++ library was wrapped into the Caffe2 machine learning framework by re-implementing Conv and FC layers. We also added highly optimized GPU kernels for speeding up the run time of the simulations on large scale networks, and now typical large-scale DNNs using RCA simulations run only 2-3x longer than native floating point training of the DNN with Caffe2.

In this new framework, currently under development, any convolution or fully-connected layer, or any downloaded pre-trained Caffe2 model, can be flexibly replaced layer-wise with a simulated hardware crossbar array with user-defined device characteristics. Moreover, the weight update is computed in-place, as it would be done in real hardware, and thus the weight gradient is not available explicitly, as it would be otherwise in all ML frameworks. Thus, the user is forced to adhere to hardware-realistic gradient decent and regularization schemes, and not use optimized software techniques developed in the ML community that need explicit access to the gradient values, but could not be used when training RCAs in hardware.

Since the framework also includes the ability for data and model parallelism, it is feasible to study hardware-realistic training and inference on state-of-the-art networks used in the machine learning community today and drive the development of algorithmic modifications to compensate for non-idealities (e.g. see [57]).

4.3. Compensation techniques for DNN inference

In recent years, many efforts propose methods to mitigate and compensate device and circuit-level non-idealities to improve the performance of DNN inference on resistive crossbar systems. These efforts can be broadly grouped into (i) methods that use re-training to compensate for errors [58], [21], [59], [60], (ii) mapping to reduce computational errors [58], [61], [62], [59], [63], and (iii) closed-loop programming methods to overcome impacts of IR drop, drift in synaptic conductances, and imperfect

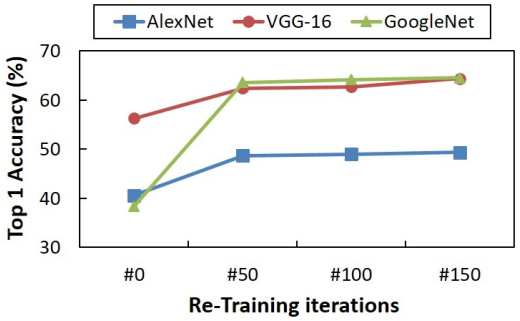

Figure 10 Compensating accuracy degradation due to limited precision for large-scale DNNs [21]

programming [64]. We detail these techniques in turn below.

Software models of resistive crossbars can be used during training and re-training to improve inference accuracy by overcoming the impact of crossbar non-idealities. The (re-)training approach can be broadly visualized as updating the weights of DNNs based on information of crossbar non-idealities using back-propagation. The accuracy degradation due to conversion of full-precision models to synaptic conductances with limited precision can be compensated with limited re-training [21]. Figure 10 illustrates the effectiveness of this re-training algorithm [21], wherein we achieve considerable (8%-26%) accuracy improvement for AlexNet, VGG-16, and GoogleNet DNNs with only 150 re-training iterations. Further, a defect-map based re-training procedure that judiciously maps crucial weights to defect-free zones can minimize the effects of stuck-at-faults on classification accuracy [58]. Similarly, variation distribution aware re-training avoids highly variation sensitive cells [59]. Other non-ideal device characteristics such as non-linearity and asymmetry can be accounted for by novel programming schemes [61], [62]. Although re-training compensates the effect of non-idealities to some extent, the training procedure can be further sophisticated. To that effect, a technology-aware back-propagation algorithm has been proposed which account for crossbar non-idealities [60] both in the forward and backward pass. Moreover, in this approach, the gradient descent algorithm is modified with functional crossbar models to compensate for the effects of non-idealities.

Other techniques that do not involve re-training of DNNs have also been explored. For example, effective mapping of weight matrices to synaptic conductances can account for computational errors due to device physics and circuit effects [63]. Gradient search has also been used to alleviate circuit-level non-idealities by obtaining weight matrices that resemble the ideal weight matrices [65]. Another popular approach to mitigate crossbar non-idealities involves closed-loop programming, wherein we perform a read operation after every write operation to verify the success of the write operation. Subsequently, we repeat the write-read cycle in a loop to precisely program the synaptic devices to the desired conductance states. This approach has been extended to compensate for synaptic drifts in real-time [64].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1147/JRD.2019.2947011, IBM Journal of Research and Development

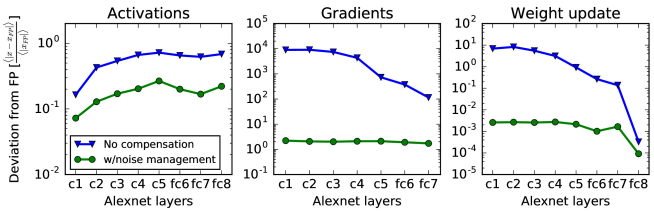

Figure 11 Effect of noise management on AlexNet training. The panels show the average relative deviation between the perfect (floating point) and hardware-aware simulated model of activations, gradients, and weight update that occurred during the training of one mini-batch. If no compensatory measure is taken, deviations due to imprecision in the analog compute, accumulate dramatically with layer depth, in particular for the backward pass. Noise management reduces the magnitude of the deviations drastically, however, a certain level of inaccuracies remain.

Error correction circuits can further eliminate the erroneous crossbar outputs. A scaling factor based approach for correcting RCA outputs has also been proposed [66].

Despite considerable progress in mitigating the undesirable characteristics of memristive crossbars, there is still a need for further research in cross-layer error mitigation and compensation techniques to enable future adoption of resistive crossbar based hardware.

4.4. Compensation techniques for DNN Training

In principle, training a DNN directly on the available analog hardware automatically solves many of the problems faced by an analog inference engine, namely the discrepancy between the architecture used for training the weights (floating point) and inference. In fact, when using the same architecture for both training and inference, many non-idealities of the RCAs as described above can be compensated by the DNN training itself. This is possible, because, by training in-situ on the RCA, an adapted solution in the weight space of the DNN is found, that takes the non-idealities into account. Indeed, simulations show that the neural network training process accounts for corrupt devices, device-to-device variations of the saturation levels and other RCA non-linearities, and can generate a more noise-robust DNN when trained on RCAs. However, simulations also show that not all hardware constraints are readily compensated without some algorithmic improvements. For instance, the limited dynamic range of input and outputs, as well as limited ranges of the resistive values in the RCA, require algorithmic adaptions during the training process. It turns out that normalization of the activations, as well as scaling the limited weight ranges properly, proves crucial for successful training with limited weight and input/output ranges of the RCA [57]. Moreover, the input vectors to the RCA during the forward pass (previous layer activations) are typically very different in magnitude when compared to the inputs during the backward pass (the back-propagated error vectors). If the magnitude of the inputs to the RCA are too small, outputs are buried in the noise floor and the DNN cannot be trained, as further propagated signals do not contain any useful information. This is illustrated in Figure 11 (blue curves) in case of the AlexNet DNN [67]. With increasing depth, activations or gradients become vastly different from the correct values, if no noise reduction technique is used. One way to account for the vastly different amplitudes of activations and error signals using the same dynamic range of the hardware design is to divide every input vector element by the (absolute) maximum of the input values and re-scale the output to recover the original magnitude (noise management, see [15]). This normalization also has the advantage of increasing amplitudes above the analog noise floor. Figure 11 (green curves) shows that this technique is very effective in reducing the overall amount of noise in the DNN trained on RCAs in comparison to the floating point reference.

These results show that algorithmic modifications and compensatory measures are essential for training DNNs on RCAs. Note that a straightforward way to implement such algorithmic modifications is to leverage the digital domain before and after the RCA digital-analog conversions. We believe that a successful design of an analog accelerator for DNN training should have such algorithmic compensations seamlessly embedded into its digital fabric and therefore introduces additional design challenges.

Finally, the switching characteristics of the device materials need to be balanced in the up and down directions and device materials need to show a sufficient number of resistive states for the stochastic gradient descent to work (as discussed in [14]). We expect that together with material and devices, significant algorithmic modifications or compensatory measures need to still be developed, to successfully realize DNN training accelerators using RCAs.

5. Discussion

Resistive crossbars have garnered great interest as a hardware building block for the next generation of neural network hardware. The native in-memory evaluation of vector-matrix multiplications performed by resistive crossbars has the potential to provide considerable benefits in area and energy efficiency. In this paper, we provide a system-level perspective on resistive crossbar based systems designed for executing DNN inference and training operations. We present a variety of design considerations, opportunities, and challenges associated with resistive crossbar based systems. These challenges can be broadly summarized into five key directions:

- **Preserving the efficiency of crossbar-based computation at the system level**: Peripheral circuits such as ADCs and DACs, and other components such as the memory hierarchy and interconnect network, required for a complete system, diminish the benefits of crossbar-based computation. New circuit and architectural techniques (*e.g.*, low-cost peripheral circuits and time-multiplexing of non-crossbar components) may be required to ensure that the intrinsic efficiency of resistive crossbars is preserved at the system level.

- **Acceleration of sparse DNNs**: Sparsifying DNNs has emerged as a powerful approach to obtain optimized DNNs particularly for resource-constrained systems [68]. However, sparse DNNs can lead to inefficient crossbar realizations as a pruned connection in a DNN merely translates to an unused crosspoint in the crossbar, leading to energy and area inefficiency [69]. Unlike a CMOS system, where sparsity can be leveraged by adding indirection logic between the storage and compute units, resistive crossbar based systems lose this flexibility due to the in-memory nature of VMM. Recent research has explored techniques to exploit sparsity at row, column and crossbar granularities to improve the execution of sparse DNNs on crossbars [69], [70], [71]. Further research to achieve energy savings commensurate to the algorithmic sparsity will improve the inference efficiency for sparse DNNs on crossbar based systems.
- **Supporting algorithmic optimizations:** Several algorithmic optimizations such as model compression, lower-complexity convolutions (FFT and Winograd convolution), *etc*. have been explored to reduce the computation and storage requirements of DNNs. However, it is challenging to realize them on crossbar-based hardware.
- **Programming frameworks:** The success of deep neural networks has been facilitated by the availability of easy-to-use open-source programming frameworks such as PyTorch, TensorFlow and Keras. For broad adoption, crossbar-based hardware must be just as easy to program as current GPUs and digital CMOS accelerators.
- **Maintaining accuracy in the presence of crossbar non-idealities:** Device and circuit-level non-idealities cause errors in crossbar operations during DNN inference and training, leading to accuracy degradation. Most efforts addressing this challenge have focused on smaller networks and simpler problems such as MNIST, where the degradation in accuracy is very minimal. However, accuracy evaluations on large-scale networks tell a very different story, indicating drastic accuracy degradation. We believe that cross-layer error mitigation and compensation techniques are needed to bridge the accuracy gap due to crossbar non-idealities.
- **DNN training:** Training with resistive crossbars poses two primary challenges. First, unlike inference which is done with fixed-point arithmetic, training requires floating-point arithmetic to achieve convergence. While past work has explored fixed-point arithmetic for training CNNs on simpler datasets [72], it is not clear how this would scale to other models and more complex datasets [73]. This is because the magnitudes of updates in DNN training are much smaller than what can be captured by fixed-point representations. However, crossbar based systems are naturally suited for fixed-point computations. Second, writes to resistive crossbars are highly expensive (discussed in Section 3.1). Since DNN training requires frequent writes, the energy benefits of efficient VMM can be outweighed by the cost increase from crossbar writes. Research directions encompassing technology, circuits and systems towards enabling floating-point arithmetic and efficient crossbar writes can pave the way for the adoption of resistive crossbars for DNN training.

In summary, a comprehensive approach involving device, circuit, architecture, and algorithm co-design may be required to realize the potential of crossbar based hardware fabrics.

References

1. R. Parloff, "The AI Revolution: Why Deep Learning Is Suddenly Changing Your Life. http://fortune.com/ai-artificial-intelligence-deep-machine-learning ," Online. Accessed Sept. 17, 2017. [Online]. Available: http://fortune.com/ai-artificial-intelligence-deep-machine-learning/

2. NVIDIA Turing architecture based GPUs, "https://www.nvidia.com/en-us/design-visualization/technologies/turing-architecture/ ," Online. Accessed Sept. 23, 2019. [Online]. Available: https://www.nvidia.com/en-us/design-visualization/technologies/turing-architecture/

3. N. P. Jouppi et al. , "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246

4. J. Fowers et al., "A Configurable Cloud-scale DNN Processor for Real-time AI," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 1–14. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00012

5. Intel Nervana Neural Network Processors, "https://www.intel.ai/nervana-nnp/ ," Online. Accessed Sept. 23, 2019. [Online]. Available: https://www.intel.ai/nervana-nnp/

6. H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

7. S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

8. H. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. Chen, and M. Tsai, "Metal-oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 6 2012.

9. S. G. Ramasubramanian, R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "Spindle: Spintronic deep learning engine for large-scale neuromorphic computing," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 15–20.

10. A. Sengupta and K. Roy, "Encoding neural and synaptic functionalities in electron spin: A pathway to efficient neuromorphic computing," *Applied Physics Reviews*, vol. 4, no. 4, p. 041105, 2017.

11. M. Jerry, P. Chen, J. Zhang, P. Sharma, K. Ni, S. Yu, and S. Datta, "Ferroelectric fet analog synapse for acceleration of deep neural network training," in *2017 IEEE International Electron Devices Meeting (IEDM)*, Dec 2017, pp. 6.2.1–6.2.4.

12. M. Jerry, S. Dutta, A. Kazemi, K. Ni, J. Zhang, P.-Y. Chen, P. Sharma, S. Yu, X. S. Hu, M. Niemier, and S. Datta, "A ferroelectric field effect transistor based synaptic weight cell," *Journal of Physics D: Applied Physics*, vol. 51, no. 43, p. 434001, aug 2018. [Online]. Available: https://doi.org/10.1088%2F1361-6463%2Faad6f8

13. W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 108–122, Jan 2019.

14. T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," *Frontiers in Neuroscience*, vol. 10, p. 333, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2016.00333

15. T. Gokmen, O. M. Onen, and W. Haensch, "Training deep convolutional neural networks with resistive cross-point devices," *Frontiers in Neuroscience*, vol. 11, p. 538, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2017.00538

16. 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," in *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 2016, pp. 27–39.

17. A. Ankit, I. El Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. Hwu, J. P. Strachan, K. Roy, and D. Milojicic, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

18. X. Sun, X. Peng, P. Chen, R. Liu, J. Seo, and S. Yu, "Fully parallel RRAM synaptic array for implementing binary neural network with (+1, -1) weights and (+1, 0) neurons," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2018, pp. 574–579.

19. A. Ranjan, S. Jain, J. R. Stevens, D. Das, B. Kaul, and A. Raghunathan, "X-mann: A crossbar based architecture for memory augmented neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: ACM, 2019, pp. 130:1–130:6. [Online]. Available: http://doi.acm.org/10.1145/3316781.3317935

20. A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 2016.

21. S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "Rx-Caffe: Framework for evaluating and training Deep Neural Networks on Resistive Crossbars," *CoRR*, vol. abs/1809.00072, 2018. [Online]. Available: http://arxiv.org/abs/1809.00072

22. G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

23. G. Burr, P. Narayanan, R. Shelby, S. Sidler, I. Boybat, C. di Nolfo, and Y. Leblebici, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *Electron Devices Meeting (IEDM), 2015 IEEE International*. IEEE, 2015, pp. 4–4.

24. C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, vol. 1, no. 1, p. 52, 2018.

25. M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, p. 61, 2015.

26. S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, p. 60, 2018.

27. A. Sengupta, Y. Shim, and K. Roy, "Proposal for an all-spin artificial neural network: Emulating neural and synaptic functionalities through domain wall motion in ferromagnets," *IEEE transactions on biomedical circuits and systems*, vol. 10, no. 6, pp. 1152–1160, 2016.

28. P. Wang, F. Xu, B. Wang, B. Gao, H. Wu, H. Qian, and S. Yu, "Three-dimensional nand flash for vector–matrix multiplication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 4, pp. 988–991, April 2019.

29. D. F. et al., "Analog computation in flash memory for datacenter-scale ai inference in a small chip," Online. Accessed Aug. 21, 2018. [Online]. Available: https://www.hotchips.org/hc30/2conf/2.05_Mythic_Mythic_Hot_Chips_2018_V5.pdf

30. X. Guo, F. M. Bayat, M. Bavandpour, M. Klachko, M. R. Mahmoodi, M. Prezioso, K. K. Likharev, and D. B. Strukov, "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded nor flash memory technology," in *2017 IEEE International Electron Devices Meeting (IEDM)*, Dec 2017, pp. 6.5.1–6.5.4.

31. S. Agarwal, S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James, and M. J. Marinella, "Resistive memory device requirements for a neural algorithm accelerator," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 929–938.

32. M. J. Marinella, S. Agarwal, A. Hsia, I. Richter, R. Jacobs-Gedrim, J. Niroula, S. J. Plimpton, E. Ipek, and C. D. James, "Multiscale co-design analysis of energy, latency, area, and accuracy of a reram analog neural training accelerator," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 86–101, 2018.

33. M. Hu, C. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, 2018.

34. G. W. Burr, P. Narayanan, R. M. Shelby, S. Sidler, I. Boybat, C. di Nolfo, and Y. Leblebici, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 4.4.1–4.4.4.

35. Z. Xu, A. Mohanty, P.-Y. Chen, D. Kadetotad, B. Lin, J. Ye, S. Vrudhula, S. Yu, J.-s. Seo, and Y. Cao, "Parallel programming of resistive cross-point array for synaptic plasticity," *Procedia Computer Science*, vol. 41, 12 2014.

36. S. Kim, T. Gokmen, H. Lee, and W. E. Haensch, "Analog cmos-based resistive processing unit for deep neural network training," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2017, pp. 422–425.

37. T. Gokmen, M. J. Rasch, and W. Haensch, "Training lstm networks with resistive cross-point devices," *Frontiers in Neuroscience*, vol. 12, p. 745, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00745

38. S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, Y. Jaoudi, and G. Burr, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, 06 2018.

39. L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1mW 8b 1.2GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32nm digital SOI CMOS," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Feb 2013, pp. 468–469.

40. J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6t sram array," in *VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on*. IEEE, 2016, pp. 1–2.

41. E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, "Repeatable, accurate, and high speed multi-level programming of memristor 1t1r arrays for power efficient analog computing applications," *Nanotechnology*, vol. 27, no. 36, p. 365202, 2016.

42. D. Ielmini, S. Lavizzari, D. Sharma, and A. L. Lacaita, "Physical interpretation, modeling and impact on phase change memory (pcm) reliability of resistance drift due to chalcogenide structural relaxation," in *2007 IEEE International Electron Devices Meeting*. IEEE, 2007, pp. 939–942.

43. X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu *et al.*, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

44. A. Ankit, A. Sengupta, P. Panda, and K. Roy, "Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 27.

45. Y. Kim, Y. Zhang, and P. Li, "A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing," *J. Emerg. Technol. Comput. Syst.*, vol. 11, no. 4, pp. 38:1–38:25, Apr. 2015.

46. M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1–13.

47. M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of bsb recall function using memristor crossbar arrays," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 498–503.

48. D. Lee and K. Roy, "Area efficient rom-embedded sram cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 9, pp. 1583–1595, 2013.

49. E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, 2018.

50. M. J. Rasch, T. Gokmen, M. Rigotti, and W. Haensch, "Efficient convnets for analog arrays," *CoRR*, vol. abs/1807.01356, 2018. [Online]. Available: http://arxiv.org/abs/1807.01356

51. A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, and P. B. Gibbons, "Pipedream: Fast and efficient pipeline parallel DNN training," *CoRR*, vol. abs/1806.03377, 2018. [Online]. Available: http://arxiv.org/abs/1806.03377

52. Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *CoRR*, vol. abs/1811.06965, 2018. [Online]. Available: http://arxiv.org/abs/1811.06965

53. L. Xia, B. Li, T. Tang, P. Gu, X. Yin, W. Huangfu, P. Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "MNSIM: Simulation platform for memristor-based neuromorphic computing system," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 469–474.

54. P. Gu, B. Li, T. Tang, S. Yu, Y. Cao, Y. Wang, and H. Yang, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," in *The 20th Asia and South Pacific Design Automation Conference*, Jan 2015, pp. 106–111.

55. P. Chen, X. Peng, and S. Yu, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2018.

56. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

57. M. J. Rasch, T. Gokmen, and W. Haensch, "Training large-scale ANNs on simulated resistive crossbar arrays," *arXiv preprint arXiv:1906.02698*, 2019.

58. C. Liu, M. Hu, J. P. Strachan, and H. H. Li, "Rescuing

memristor-based neuromorphic design with high defects," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM Press, 2017. [Online]. Available: https://doi.org/10.1145%2F3061639.3062310

59. L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, mar 2017. [Online]. Available: https://doi.org/10.23919%2Fdate.2017.7926952

60. I. Chakraborty, D. Roy, and K. Roy, "Technology aware training in memristive neuromorphic systems for nonideal synaptic crossbars," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 335–344, 2018.

61. P.-Y. Chen, B. Lin, I.-T. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J. sun Seo, Y. Cao, and S. Yu, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, nov 2015. [Online]. Available: https://doi.org/10.1109%2Ficcad.2015.7372570

62. I. Kataeva, F. Merrikh-Bayat, E. Zamanidoost, and D. Strukov, "Efficient training algorithms for neural networks based on memristive crossbar circuits," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2015. [Online]. Available: https://doi.org/10.1109%2Fijcnn.2015.7280785

63. M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Proceedings of the 53rd annual design automation conference*. ACM, 2016, p. 19.

64. B. Yan, J. Yang, Q. Wu, Y. Chen, and H. Li, "A closed-loop design to enhance weight stability of memristor based neural network chips," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, nov 2017. [Online]. Available: https://doi.org/10.1109%2Ficcad.2017.8203824

65. B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell, "Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, nov 2014. [Online]. Available: https://doi.org/10.1109%2Ficcad.2014.7001330

66. Y. Jeong, M. A. Zidan, and W. D. Lu, "Parasitic effect analysis in memristor-array-based neuromorphic systems," *IEEE Transactions on Nanotechnology*, vol. 17, no. 1, pp. 184–193, jan 2018. [Online]. Available: https://doi.org/10.1109%2Ftnano.2017.2784364

67. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105.

68. S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

69. A. Ankit, A. Sengupta, and K. Roy, "Trannsformer: Neural network transformation for memristive crossbar based neuromorphic system design," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 533–540.

70. Y. Wang, W. Wen, B. Liu, D. Chiarulli, and H. H. Li, "Group scissor: Scaling neuromorphic computing design to large neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 85.

71. L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Neural network pruning for crossbar architecture," *IEEE Access*, pp. 1–1, 2018.

72. S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1737–1746.

73. P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

Shubham Jain School of Electrical and Computer Engineering, Purdue University (jain130@purdue.edu)

Aayush Ankit School of Electrical and Computer Engineering, Purdue University (aankit@purdue.edu)

Indranil Chakraborty School of Electrical and Computer Engineering, Purdue University (ichakra@purdue.edu)

Tayfun Gokmen IBM T.J. Watson Research Center, Yorktown Heights, NY (tgokmen@us.ibm.com)

Malte Rasch IBM T.J. Watson Research Center, Yorktown Heights, NY (malte.rasch@ibm.com)

Wilfried Haensch IBM T.J. Watson Research Center, Yorktown Heights, NY (whaensch@us.ibm.com)

Kaushik Roy School of Electrical and Computer Engineering, Purdue University (kaushik@purdue.edu)

Anand Raghunathan School of Electrical and Computer Engineering, Purdue University (raghunathan@purdue.edu)