Analog architectures for neural network acceleration based on non-volatile memory

Cite as: Appl. Phys. Rev. **7**, 031301 (2020); https://doi.org/10.1063/1.5143815
Submitted: 06 January 2020 . Accepted: 10 June 2020 . Published Online: 09 July 2020

T. Patrick Xiao, Christopher H. Bennett, Ben Feinberg, Sapan Agarwal, and Matthew J. Marinella

Analog architectures for neural network acceleration based on non-volatile memory

Cite as: Appl. Phys. Rev. **7**, 031301 (2020); doi: 10.1063/1.5143815
Submitted: 6 January 2020 · Accepted: 10 June 2020 ·
Published Online: 9 July 2020

T. Patrick Xiao,[a] Christopher H. Bennett, Ben Feinberg, Sapan Agarwal, and Matthew J. Marinella[b]

AFFILIATIONS

Sandia National Laboratories, Albuquerque, New Mexico 87185-1084, USA

Note: This paper is part of the special collection on Brain Inspired Electronics.
[a]**Author to whom correspondence should be addressed:** txiao@sandia.gov
[b]**Electronic mail:** mmarine@sandia.gov

ABSTRACT

Analog hardware accelerators, which perform computation within a dense memory array, have the potential to overcome the major bottlenecks faced by digital hardware for data-heavy workloads such as deep learning. Exploiting the intrinsic computational advantages of memory arrays, however, has proven to be challenging principally due to the overhead imposed by the peripheral circuitry and due to the non-ideal properties of memory devices that play the role of the synapse. We review the existing implementations of these accelerators for deep supervised learning, organizing our discussion around the different levels of the accelerator design hierarchy, with an emphasis on circuits and architecture. We explore and consolidate the various approaches that have been proposed to address the critical challenges faced by analog accelerators, for both neural network inference and training, and highlight the key design trade-offs underlying these techniques.

TABLE OF CONTENTS

I. INTRODUCTION

For decades, processor performance has advanced at an inexorable pace by riding on continued increases in transistor density, enabled by Dennard scaling, and, more recently, by running many processor cores in parallel. In the last decade, this performance has come to a plateau, as power constraints have slowed the scaling of transistors and as Amdahl's law has diminished the returns from multi-core computation using general-purpose processors. Significant future improvements in computation—measured as the raw performance (operations per second), performance per area, and performance per Watt—will increasingly come from innovative domain-specific architectures that are specialized toward a narrower set of tasks.[1]

At the same time, the fields of neuromorphic computing and machine learning—which have applications in image and speech recognition, natural language processing, predictive analytics, and many other domains—have recently gained considerable scientific and commercial interest. Modern neural networks have grown so quickly in scale that training and deploying them now often require computational resources that lie outside the reach of most researchers and individual edge devices.[2] Fortuitously, neural networks are well suited to acceleration by domain-specific hardware, as most neuromorphic algorithms require only a small set of critical computational kernels. There is already a proliferation of new architectures specifically aimed at accelerating neural networks, many of which are custom digital chips.[3,4] For modern machine learning workloads involving large datasets, however, these systems are critically constrained by the need to transfer data between memory and the processor.[5]

The so-called memory wall, also called the von-Neumann bottleneck, presents an opportunity for neuromorphic accelerators that can perform computations directly inside the memory array where the network's parameters are stored. Analog processing inside such an array can also inherently parallelize the matrix algebra computational primitives that underlie many machine learning algorithms. These intrinsic advantages can potentially translate into dramatic improvements in both the computational capacity and energy efficiency of neuromorphic hardware.

The basic processing element in such a system is an array (or crossbar) of memory devices. Many of the candidate devices for this application are emerging memory technologies, though it is also possible—and often advantageous—to leverage mature technologies such as flash memory. In this review, we discuss the practical considerations, unique challenges, and the diversity of possible implementations of a neuromorphic accelerator that is constructed from these memory arrays. We will focus primarily on considerations at the level of circuits and system architectures, as several reviews have recently covered these accelerators from a more device-oriented point of view.[6–9] We also narrow our attention to accelerators for deep supervised learning, which is the most well-researched and deployable set of neural architectures, rather than systems specialized for unsupervised learning algorithms or spiking neural networks. Resistive crossbars have notably also been explored for accelerating several related computational kernels, such as solving linear systems[10,11] and combinatorial optimization,[12,13] which will not be the focus of this review.

Somewhat differently from recent surveys[14–16] of neural network accelerators based on emerging devices, we organize this review around the basic components and ideas that make crossbar-based architectures work. These ideas are partially but not entirely agnostic of the specific choice of memory device. We begin with an overview of deep learning and the needs of a deep learning hardware accelerator (Sec. II) and then briefly discuss digital architectures specialized for this application (Sec. III). In Sec. IV, we introduce the analog, crossbar-based neuromorphic accelerator and briefly survey the field of candidate memory devices. Section V discusses the peripheral circuits, analog and digital, which support crossbar computations—if not carefully designed, these circuits can greatly offset the basic energy, latency, and area advantages of the approach. In Secs. VI and VII, we discuss the architecture of analog accelerators for neural network inference and training, respectively, with an emphasis on system-level design trade-offs. Finally, in Sec. VIII, we survey some known approaches to combat device- and array-level non-ideal effects using architectural and algorithmic techniques.

II. COMPUTATIONAL PRIMITIVES IN DEEP LEARNING

In this section, we provide a brief introduction to artificial neural networks trained by the widely used backpropagation algorithm for supervised learning. We focus our attention on the main linear-algebra computational primitives that underlie these algorithms, the acceleration of which has motivated much of the recent work on analog neuromorphic hardware. For a more complete pedagogical introduction to deep learning and other machine learning algorithms, we refer the reader to Refs. 17 and 18.

A. Inference and training of deep neural networks

The topology of a typical artificial neural network with a single hidden layer is shown in Fig. 1(a). Input data to be processed (e.g., pixels of an image or an audio sequence, cast as a 1D vector in this case) are forward-propagated through the network from left to right by a sequence of linear and nonlinear operations. The input neuron activations $\mathbf{x}^{(l)}$ to layer l are converted to the next layer's activations $\mathbf{x}^{(l+1)}$ by the transformation

$$\mathbf{x}^{(l+1)} = f\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)}\right), \tag{1}$$

where $\mathbf{W}^{(l)}$ is an $N_l \times N_{l+1}$ matrix of weights (or synapses) connecting layer l with N_l neurons to layer $l+1$ with N_{l+1} neurons. f is a nonlinear activation function that is applied element-wise to its argument, which is the product of a vector–matrix-multiplication (VMM). A bias vector $\mathbf{b}^{(l)}$ is also added to the argument of f; for brevity, we have chosen not to include it explicitly here, as it can be absorbed into $\mathbf{W}^{(l)}$.

The layer-to-layer transformation in Eq. (1) maps one representation of the input, encoded in the neurons $\mathbf{x}$, to another. For example, the pixel-wise representation of an image may be converted to a hidden-layer representation in terms of a more abstract set of features; these features are propagated forward, being mapped to higher-level features, until one reaches the final-layer neurons, whose values represent the desired output from the neural network (e.g., classification labels). The purpose of having one or more hidden layers in a deep

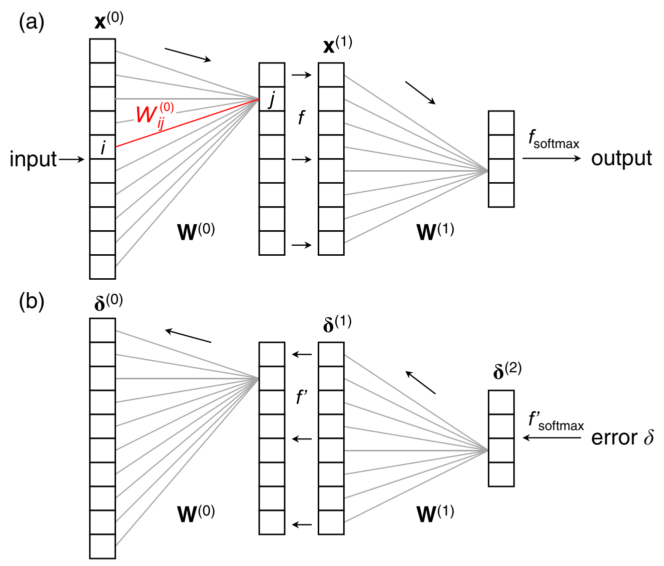

FIG. 1. A multi-layer perceptron neural network, showing a single hidden layer: (a) forward propagation through the network and (b) backpropagation of error through the network.

neural network is to convert the input into representations that are more amenable to making a correct prediction.

The nonlinear function f is essential in ensuring that the multiple layers in the network cannot be trivially collapsed into an equivalent single-layer linear network. For tasks such as computer vision, a common choice for f is the rectified linear unit (ReLU): $f(x) = \max(0, x)$. Other choices include the sigmoid function $\sigma(x)$ and the hyperbolic tangent $\tanh(x)$, which are more commonly used in manipulating sequential data, e.g., in speech recognition. A different nonlinear function, such as a softmax, is usually applied at the final layer to normalize the output of the network. The propagation of input data through the network to produce an output prediction is called the inference step. In some implementations, a batch of input examples are collected into the activations $\mathbf{x}$ so that inference can be executed as a sequence of matrix–matrix multiplications.

In supervised learning, the network is trained to make accurate predictions by iteratively updating the weight matrices $\mathbf{W}$ (and the biases $\mathbf{b}$) so that its outputs approach the provided correct outputs for a selection of input examples called the training set. Backpropagation of error, shown in Fig. 1(b), is the most widely used algorithm for training. For a given example, an inference step is first performed, and an error δ—typically a mean-squared or cross-entropy loss function—is calculated on the network's prediction based on the known correct output. By differentiating Eq. (1), the error at the output layer can be propagated backward through the network to find the error values at a layer l,

$$\boldsymbol{\delta}^{(l)} = \left(\mathbf{W}^{(l)}\right)^{\mathrm{T}} \boldsymbol{\delta}^{(l+1)} \odot f'\left(\mathbf{W}^{(l-1)} \mathbf{x}^{(l-1)}\right), \tag{2}$$

where f' is the derivative of f and $\odot$ denotes an element-wise product. Note that this expression involves the multiplication of the transpose of the weight matrix with an error vector: to differentiate it from the earlier VMM, we will refer to this operation as a matrix–vector multiplication (MVM).

From the error vectors, the derivative with respect to the prediction error δ can be found for all the weights, and these derivatives are then used to update the weights according to an optimization algorithm, which is typically a form of gradient descent. The update to be applied to a weight matrix is given by

$$\Delta \mathbf{W}^{(l)} = -\eta \delta^{(l)} \left(\mathbf{x}^{(l)}\right)^{\mathrm{T}}, \tag{3}$$

where η is a learning rate. Thus, the optimal update to the weights following a single example is an outer product of two vectors.

In stochastic gradient descent (SGD), the weight update in Eq. (3) is carried out after every training example. To speed up training and improve convergence, one can also perform forward propagation using the same weights over the full training set and then apply a sum of outer products as given by Eq. (3)—this is called batch gradient descent or minibatch gradient descent if only a subset of the training set is used before each update. Hyperparameters, such as the number and sizes of the hidden layers and the learning rate (η), are typically fixed during training but may need to be tuned externally as part of the neural network design. The ultimate goal in both training and hyperparameter tuning is not necessarily to minimize the error on the training set but rather to generalize well to a test set, which contains examples that the network has not seen. Although SGD and minibatch gradient descent tend to have noisier convergence during training than batch gradient descent and other deterministic algorithms that utilize the full gradient,[19] they have been shown to yield good generalization performance on large-scale learning problems by virtue of their stochastic nature, which has a regularization effect for some datasets.[17,20]

B. Recurrent and convolutional neural networks

We have so far considered a multi-layer perceptron (MLP)—a deep neural network consisting only of fully connected (FC) layers. Recurrent neural networks are usually employed for speech and natural language processing and can also be used for vision tasks. These networks are similar in structure to an MLP, with the critical difference that the weights are reused across inputs from multiple time steps. The widely used long short-term memory (LSTM) network, for example, processes an input vector together with a state vector for the current time step using a memory cell that combines multiple types of activation functions f.[21] For computer vision tasks, convolutional neural networks (CNNs) are the preferred choice. In convolutional layers, only nearby neurons are connected by nonzero weights, leading to a structure that is more effective in capturing spatially local patterns in visual data. MLPs, LSTMs, and CNNs represented 95% of the inference workload in Google datacenters in 2016.[22]

Figure 2 shows the structure of a convolutional layer: the input and output feature maps are three-dimensional—x, y, and an additional channel dimension—and the weights are applied in the manner of a sliding window across the input much like a spatial filter in standard image processing. Forward propagation through a convolutional layer, analogous to Eq. (1), is given by

$$O_{x,y,m} = f\left(b_m + \sum_{k=0}^{N_{ic}-1} \sum_{j=0}^{K_y-1} \sum_{i=0}^{K_x-1} I_{Sx+i,Sy+j,k} \times W_{i,j,k,m}\right), \tag{4}$$

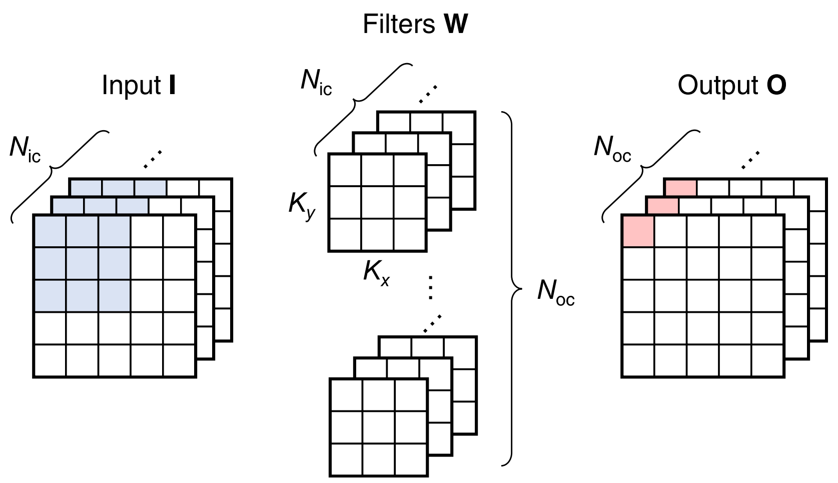

FIG. 2. A convolutional layer. The colored pixels are processed in one sliding window step over the *x-y* plane.

where **I** and **O** are the input and output feature maps and have N_{ic} and N_{oc} channels, respectively. **W** is the weight matrix that is represented here with four dimensions: the filter plane has dimensions $K_x \times K_y$ and stride S, and an independent 2D filter exists for every combination of input and output channels. The input is often zero-padded so that the output has the same size in x and y. Several convolutional layers are often followed by a max pooling layer, and a CNN generally has one or more FC layers at the output. It is important to note that due to the small filter plane, a convolutional layer reuses the same input data and a relatively small number of weights over many sequential operations. Meanwhile, a fully connected layer typically involves a much larger number of weights with no input data reuse across its VMMs. Therefore, in any hardware implementation, convolutional layers tend to be computation-bound, while fully connected layers are bounded by the memory bandwidth.[22]

C. Processing large neural networks

As the machine learning field has advanced, the size of state-of-the-art deep neural networks has grown dramatically. This is readily seen in networks developed for computer vision tasks. While LeNet-5[23]—a classic CNN trained for the MNIST hand-written digit recognition task[24]—has less than 10^5 parameters, modern CNNs optimized for the ImageNet Large Scale Visual Recognition Challenge[25] have on the order of 10^8 to 10^9 weights. Meanwhile, datacenter-targeted neural networks such as Google Brain are several orders of magnitude larger still.[26] The large size of state-of-the-art neural networks makes them difficult to train without access to large computational resources, such as distributed computing clusters, and difficult to deploy for inference applications on edge devices (such as mobile phones), which have limited storage, computational capacity, and power budget.

Some progress has been made at the algorithm level to make these neural networks more compact without greatly sacrificing their accuracy. One widely used approach is quantization:[27] reducing the bit precision of the weights and activations. Quantization is significant as it reduces not only the size of the network in memory but also the energy and area costs of memory access and arithmetic computations.[22] Good performance on image recognition tasks has been shown with neural networks that use only 1-bit (binary) weights[28] during inference and with fully binary neural networks using both 1-bit weights and 1-bit neuron activations.[29,30] BinaryNet, a state-of-the-art binary neural network, compresses AlexNet—a classic CNN designed for the ImageNet task—by a factor of 189× while suffering only a small top-1 accuracy loss from 56.6% to 51.4%.[31] These highly compressed networks still require higher-precision values during backpropagation since the errors and weight updates corresponding to individual training examples are small.[32] Besides the compression of weights and activations, there are also efforts to prune weights from the network and in general to deploy less computationally intensive CNNs such as MobileNet, which uses depthwise convolutions to greatly reduce the number of multiply accumulate (MAC) operations.[33] One caveat of these approaches is that while they reduce the network's memory footprint and computational load, they do not always produce a proportional reduction in the energy needed to process the network due to the overhead of data movement.[34]

The growing size and demand for neural networks, particularly in industrial applications, call for hardware innovations in parallel with the algorithmic ones in order to make large, high-performance neural networks available to users and researchers who are constrained by the cost of computation. Broadly speaking, large neural networks cannot be implemented efficiently on general-purpose central processing units (CPUs) in either the inference phase or the training phase. The CPU, which is specialized for executing one (or a few) potentially very complicated instruction at a time, is ill-suited for the needs of large neural networks, which are characterized by a large data volume and highly regular workload built from a small set of computational primitives.

Using graphics processing units (GPUs), which contain hundreds or thousands of co-processors that compute in parallel, significantly improves performance within the domain of neural network processing. The GPU's co-processors share access to a very fast local memory or to a global memory in a highly parallelized (coalesced) manner.[35] Many of today's state-of-the-art neural networks are trained using GPUs or distributed GPU clusters,[2,22] and GPU-accelerated libraries of the computational primitives in deep neural networks have been released, which facilitate code development for both inference and training.[36] However, in spite of the advantages over CPUs, memory transfer remains an overwhelming bottleneck in processing large neural networks on GPUs.[2,35] Since both the CPU and the GPU have limited local memory and memory bandwidths, the growing data volume in modern neural networks (as indicated by the sizes of the weights $\mathbf{W}$ and activations $\mathbf{x}$) implies that expensive off-chip memory accesses tend to dominate the energy and latency of the computation.[5]

III. DIGITAL NEUROMORPHIC ARCHITECTURES

To address the shortfalls of CPUs and GPUs for data-centric applications, there has been considerable recent interest in customized, domain-specific architectures for machine learning.[1,3,4,26] Here, we will briefly survey these digital neural network accelerators, which include both systems based on field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). We refer the reader to Refs. 3 and 4 for a more complete overview. Many of these solutions aim to accelerate neural network inference, with training still largely performed offline by GPUs.

The FPGA is a popular platform for the development of flexible hardware for various applications, including deep neural networks.[37–41] Compared to the GPU, its reconfigurability enables a greater degree of algorithm and hardware co-design, as well as the possibility of lower-precision datapaths that lead to more energy-efficient inference operations. However, the overhead needed to support reconfigurability limits the available processing resources within a chip; multiple FPGAs may be needed to implement a desired function, increasing area and power consumption. These trade-offs can be balanced by using collections of FPGAs interconnected by a high-speed reconfigurable network.[42] At the datacenter scale, Microsoft's Project Brainwave uses large pools of FPGAs to deliver inference results with low latency. Brainwave's high performance on large neural networks relies on efficiently partitioning the network across FPGAs such that each partition (or sub-graph) requires only those weights that can be pinned onto the high-bandwidth on-chip FPGA memory.[43]

Google's Tensor Processing Unit (TPU) is a custom chip that was designed for in-datacenter acceleration of neural networks.[22] The TPU v1, which is an inference accelerator, contains at its core a 256×256 block of 8-bit multiply accumulate (MAC) units that perform the individual multiplication and addition operations inside the VMM in Eq. (1). The block is operated as a systolic array, as shown in Fig. 3, where the input data are broadcast horizontally and VMM partial sums are accumulated from the top downward. On each clock cycle,

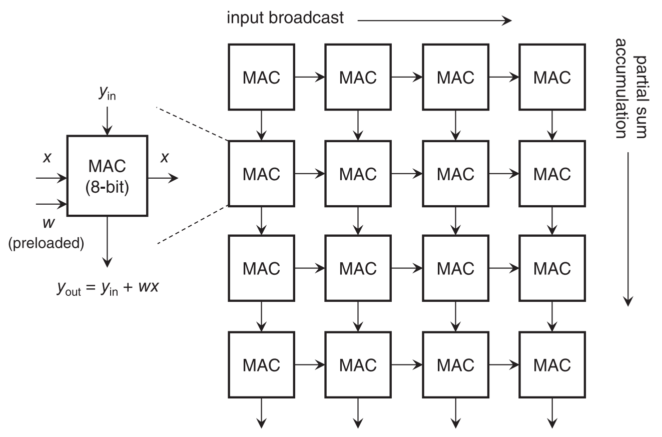

FIG. 3. The systolic array of 8-bit multiply accumulate units in the Google TPU v1.[22]

each MAC unit receives an input and partial sum from its neighbors, performs a computation, and then forwards the updated partial sum to the next element. This configuration leads to higher compute density (25× more MAC units than the larger Nvidia K80 GPU) and reduces the number of intermediate reads and writes to a buffer during a VMM.

Access to off-chip main memory is the most significant bottleneck to large neural network computations. One way to address this issue is to place optimally sized memory banks close to the processing elements for the transfer of intermediate results. DaDianNao, for example, distributes large neural network layers over tiles of processing elements and provides local embedded dynamic random access memory (eDRAM) banks within each tile.[44] 3D integration of DRAM arrays on top of a layer of digital logic elements, as in Neurocube,[45] can also effectively bring processing close to the memory.

An important approach to minimize memory access costs is to adopt a more efficient dataflow that maximally reuses the transferred data. For example, DianNao accommodates large layers, which do not fit in the local memory dedicated to its processing elements, by breaking up the VMM into tiles that minimize the transfer of input and output activations.[46] In CNNs, the heavy reuse of filter weights and feature maps provides a greater opportunity to amortize the cost of data access over many operations. Eyeriss exploits this via a careful mapping of the convolution layer to the dataflow within the chip.[47] Like the TPU, Eyeriss executes its operations on a systolic array but with a reconfigurable row-stationary dataflow that minimizes the movement of both filter weights and input data. For smaller filters, different parts of the array can be used to simultaneously process different parts of a convolution operation. Reference 3 provides a comprehensive overview of energy-efficient dataflows used by various digital architectures.

Inference accelerators have also been developed to operate on neural networks with low-precision data types and/or sparse weight matrices to save energy and area.[48,49] In the limit of quantization to a single bit, YodaNN accelerates binary weight networks,[50] the Unified Neural Processing Unit (UNPU) supports variable-precision weights (1 to 16 bits),[51] and BRein Memory accelerates networks with both binary weights and activations.[52] Specialized digital architectures have also been developed for deep neural network training—one example is ScaleDeep, which is designed as a server node architecture.[53] Cambricon, an instruction set architecture (ISA) specific to neural network accelerators, increases code density while extending support to a variety of techniques used in deep learning, for both inference and training.[54]

A distinct class of digital accelerators overcomes the memory transfer bottleneck by directly performing computation inside a modified digital memory array, exploiting the high internal bandwidth of DRAM and static random access memory (SRAM) technology. Ambit, an example of a digital processing-in-memory (PIM) architecture, implements a logically complete set of operations within a DRAM array with minimal additional area overhead.[55] It uses the simultaneous activation of three rows in a DRAM array to perform parallelized AND and OR operations on two rows and adds a row of dual-connected 2T-1C cells to implement a row-wise NOT operation, as shown in Fig. 4. To avoid overwriting the operand data, data must first be copied onto these designated rows for bitwise logic, which adds latency. DRISA (DRAM-based Reconfigurable In-Situ Accelerator) considers completing Ambit's AND/OR operations with simple CMOS logic gates (e.g., NOT) integrated with the column sense

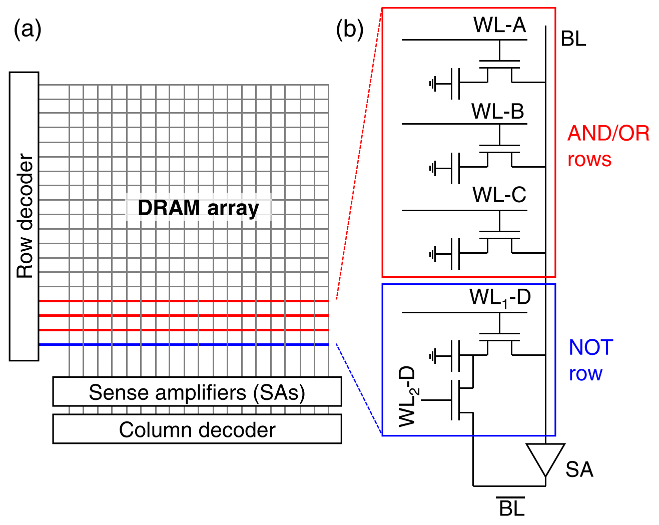

FIG. 4. (a) Layout of a DRAM array. (b) In Ambit,[55] three designated rows can be simultaneously activated to perform AND or OR operations on rows *A* and *B*, depending on the values on row *C*. A row of dual-connected 2T-1C cells can be used to perform a row-wise NOT operation. WL = world line and BL = bitline.

circuitry, as well as arrays that can perform logically complete NOR operations using 3T-1C DRAM cells.[56]

While the bitwise operations in Ambit and DRISA are logically complete, many cycles are needed to execute multi-bit arithmetic operations—hundreds of cycles, for example, to compute multiplications with three bits or more.[56] Consequently, high performance must be achieved through parallelism, not only via the simultaneous activation of multiple rows but also of multiple DRAM subarrays. More importantly, PIM architectures work most efficiently as inference accelerators for networks where either the weights or activations are binary. Deep neural networks based on bulk bitwise operations can also be implemented in SRAM arrays[57–59] and in emerging non-volatile memories such as resistive RAM, phase change memory (PCM), and spin-transfer torque (STT) magnetic RAM when operated in binary storage mode.[60]

Digital processing-in-memory architectures share some similarities and intrinsic advantages with the analog non-volatile-memory-based architectures that we consider in the remainder of this paper. We refer the reader to Ref. 61 for a detailed neuromorphic performance comparison of the two architecture types.

IV. ANALOG IN-MEMORY ACCELERATION OF NEURAL NETWORKS

By embedding neural network computations directly inside the memory elements that store the weights, analog neuromorphic accelerators based on non-volatile memory (NVM) arrays can greatly reduce the energy and latency costs associated with data movement. In this section, we will also see that computation within these arrays is inherently parallel, and this can be leveraged to accelerate the neural network primitives discussed in Sec. II—the VMM, the MVM, and the outer product update—enabling efficient architectures for both inference and training.

Figure 5 shows the basic structure of a resistive memory array, realizable with various non-volatile memory (NVM) device types,

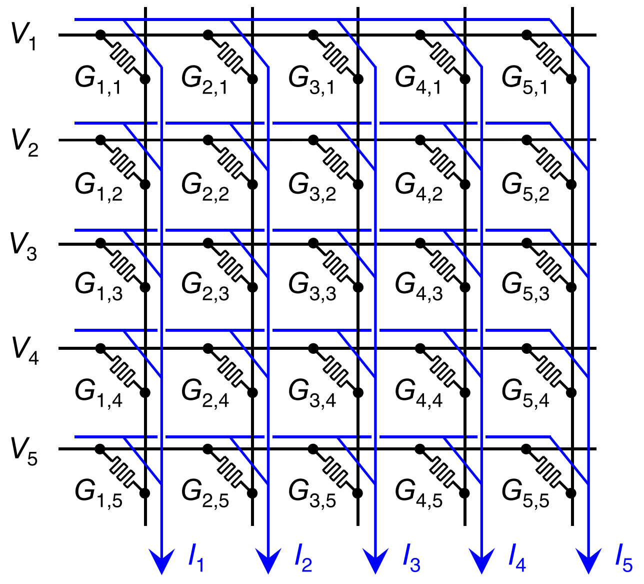

FIG. 5. The basic concept of a massively parallel analog vector–matrix multiplication within a resistive memory crossbar. The currents accumulated by the columns are given by Eq. (5).

whose conventional application is high-density data storage. To perform a VMM within the array, all the rows (or wordlines) are activated simultaneously, with a voltage V_i applied on row *i*. The total current collected by the *j*th column (or bitline), which we assume to be held at a fixed potential, is given by

$$I_j = \sum_{i=0}^{N_r-1} G_{ij} V_i \quad 0 < j < N_c - 1, \tag{5}$$

where G_{ij} is the conductance of the memory element at the array position (i, j), N_r is the number of rows, and N_c is the number of columns. The above expression implements a vector dot product, where the multiplications are realized by Ohm's law and the summation by Kirchhoff's current law. Since the currents flow through all the columns in parallel, the crossbar executes the full VMM in a single operation. The bias **b** can be added as an extra row in the array. The MVM operation, which uses the transpose of the same weight matrix, can likewise be executed by driving the columns and reading the currents on the rows. We defer discussion of the parallel outer product update to Sec. VII on training accelerators.

The physical latency of a crossbar VMM is determined by the time required to charge each row of the array. Since both the resistance and the capacitance of the row increase with the number of columns, the latency scales as $O(N^2)$ for an $N \times N$ array. Nonetheless, for metal interconnects used in the 14 nm CMOS process, this RC time is only about 0.2 ns for a very large array with $N = 1024$, which is orders of magnitude faster than the time taken to compute the same VMM (10^6 MACs) on a digital processor.[62] This RC delay is sometimes treated as constant time in practice, as it tends to be much smaller than the latency of the peripheral circuits, such as the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC).

Compared to the digital PIM accelerators discussed in Sec. III, the resistive crossbar has a more favorable energy scaling. Computing a VMM on a DRAM or SRAM array requires charging one row at a time and then one column at a time for each cell. By charging all the rows in parallel, the energy of a VMM scales as $O(N^2)$, compared to $O(N^3)$ for a digital memory array.[63] Resistive crossbars, particularly those based on ReRAM technology, can also potentially be denser than SRAM or DRAM arrays, while being capable of storing multiple bits of weight data per cell. It is often the case in analog accelerators, however, that while the in-memory matrix computations are highly efficient, the area and energy consumption is dominated not by the crossbar but by the peripheral circuitry.[62]

Comparing Eq. (5) to Eq. (1), we find that the conductance G_{ij} is simply proportional to the represented weight W_{ij}. Realistically, the mapping from the weight matrix to the array conductances must account for non-ideal effects at the device and array level, such as the tunable conductance range, I–V nonlinearity, influence of peripheral circuitry, parasitic voltage drops across the array, and device-to-device process variations. An optimal mapping can be extracted from device models or from a few measured parameters.[64] It is also possible to calibrate for the combined effects of these non-idealities by streaming a representative set of input data to the crossbar and minimizing the error in the outputs.[65] Since conductances must be positive, Eq. (5) is—strictly speaking—compatible only with positive-valued weights. Real-valued weights are typically implemented using two resistors per weight whose currents are subtracted, but various other methods have been explored. We will revisit the topic of signed computation in Sec. VI.

Similar to a fully connected layer, all the weights within a long short-term memory (LSTM) layer can be mapped onto a single crossbar. By driving the rows with the input vectors and the hidden state vectors, all matrix computations within a time step can be executed in one array read operation.[66,67] The column outputs can then be digitized and processed by digital circuits that implement the LSTM activation functions; in the next time step, these results can be passed into the same array as the hidden state input. This architecture can support other types of recurrent neural networks, such as gated recurrent units, by modifying or reprogramming the digital circuit blocks.[66]

Convolutional layers can be mapped onto resistive crossbars by unrolling the input to each sliding window computation into a vector and decomposing the convolution into VMMs. The most widely used scheme, proposed in the ISAAC architecture,[68] is shown in Fig. 6. The unrolled sliding window input is applied to the rows, and the result in each output channel is collected at the columns. While individual sliding window computations are intrinsically parallelized by the crossbar, the full convolution still requires a large sequence of sliding windows that can be processed concurrently only by replicating the weights across multiple devices.

A. Candidate memory devices

1. Device requirements for inference and training

To be useful for neuromorphic computing, non-volatile memory devices must meet a number of requirements that are considerably more stringent than those for storage-class memory,[69] particularly if these devices are to be used for training. With this being the case, many different types of memory have nonetheless been proposed as synaptic weights for neuromorphic computing. Here, we provide an overview of the device requirements and the main candidate technologies. We will keep our discussion relatively brief, as detailed device-focused reviews can be found elsewhere.[6,7,14,16]

For inference-only applications, the device must have at least two reliably distinguishable conductance states. Though not strictly necessary, memory elements with low cycle-to-cycle variability or noise in its stored weight value are desired in order to realize multiple bits of weight information in a single device. Provided that the read non-idealities, depicted in Fig. 7(a), are small, multi-bit devices can greatly enhance the accelerator's area and energy efficiency. The device must also have long retention: any drift in the stored state, caused by charge

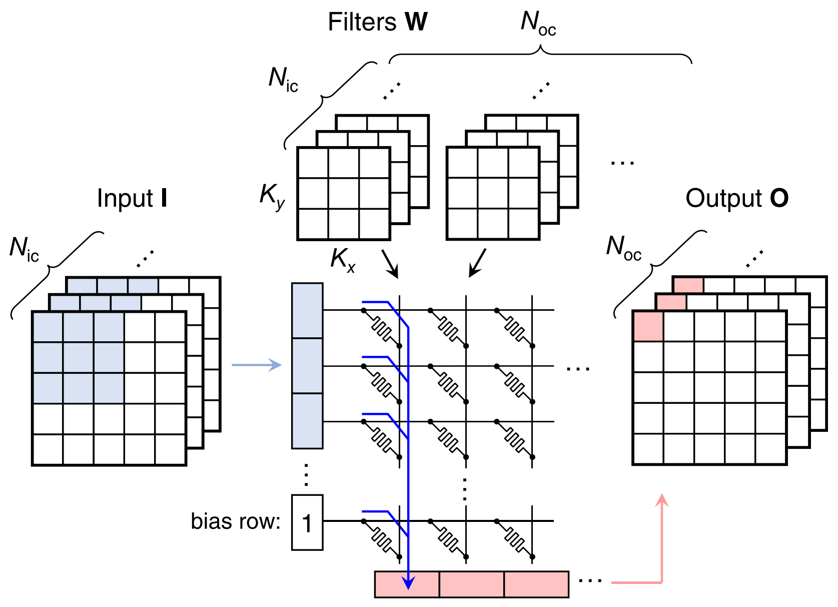

FIG. 6. Mapping a convolutional layer onto a resistive crossbar. Each column contains the full set of weights for one output channel. The size of the array is $(K_x \times K_y \times N_{ic} + 1) \times N_{oc}$.

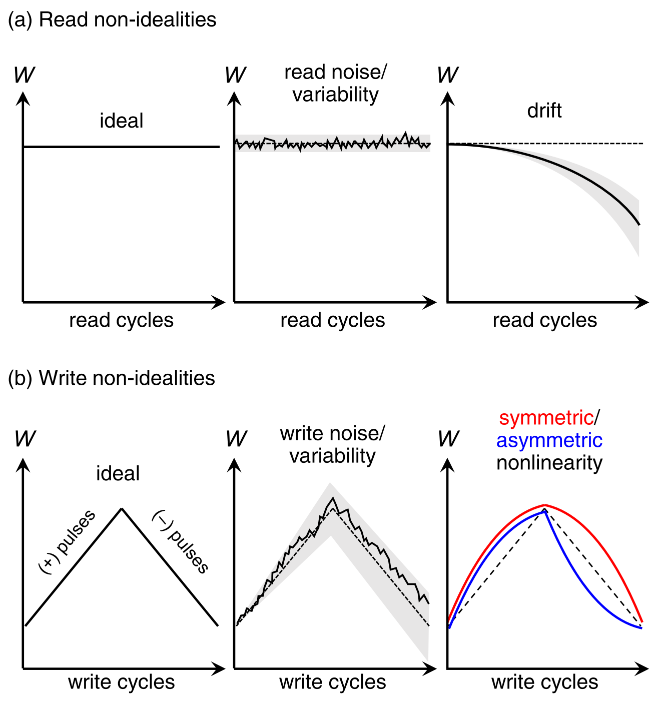

FIG. 7. (a) For reliable and accurate inference, memory devices with low read noise and small drift are desired. For reliable and accurate training, devices must have both low read and write non-idealities. (b) The response of a memory device, with and without non-idealities, to a sequence of identical positive update pulses, followed by a sequence of identical negative updates of the same magnitude.

leakage, ion migration, structural relaxation, or other mechanism, should be absent or should occur over a sufficiently long timescale. An auxiliary but nonetheless important side-effect of drift is read noise that increases over time as a result of variability in the rates of drift. Low conductance is also strongly desired to limit parasitic voltage drops along the interconnects, thus enabling larger VMMs within a crossbar.[70] Cycle-to-cycle noise, inaccurate weight programming, and drift can all accumulate to pose a serious challenge to the signal-to-noise ratio of VMM operations in large crossbar arrays. A recent resilience analysis of state-of-the-art convolutional neural networks—such as AlexNet, VGG, and ResNets—demonstrates the varying degrees to which these networks can tolerate weight and activation noise introduced by the physical hardware implementation during inference.[71]

To accelerate training, which involves more precise weight tuning, more analog conductance states are needed; this requires devices with small variability in both its read and write operations. The device should have high endurance and low write latency so that it can be trained reliably and efficiently over many examples in a large dataset and should also have low write energy. Though the memory technologies considered in Table I can each be switched with picojoule write energies, the write voltage and write current are also individually important: a low write voltage is desired to minimize the CV^2 energy needed to charge the array, and a low write current helps reduce write errors incurred due to array parasitic resistance.

To train accurately, the memory device should further have a gradual and linear response to the programming pulses used to update the weight, i.e., the same magnitude of conductance change for the same strength of the pulse, regardless of the initial state—this is a demanding but essential requirement.[6] Examples of nonlinear deviations from this behavior are shown in Fig. 7(b). In general, an asymmetric nonlinearity—a different response to positive and negative update pulses—causes a greater accuracy degradation than a symmetric nonlinearity. In the presence of asymmetric nonlinearity, an attempt to fine tune the weights via alternating positive and negative updates will tend to drive the weight value to zero.[86] The degree to which deviations from this ideal behavior affect neural network training has been explored in detail by several papers.[86–88] It is possible to compensate for some device shortcomings—particularly low precision and asymmetry—at the architecture level, as we will review in Sec. VII. Mitigation strategies at the architecture and algorithm level have also been proposed for the other non-idealities, which we will survey in Sec. VIII.

2. Emerging non-volatile memories

Table I provides a coarse comparison of the non-volatile memory technologies that are good candidates for analog synapses. We provide a brief overview of these device options below.

Resistive RAM (ReRAM) is a popular choice for crossbar accelerators, as it is generally highly scalable (with a footprint that is potentially limited only by the metal pitch[62]), back-end-of-line compatible and can have a continuously variable conductance, low write energy, low write latency, and high endurance.[6] ReRAM devices are two-terminal structures with a variable-conductance layer that is typically a metal oxide. The conductance responds to electrical pulses either through the formation and destruction of a conductive filament[89,90] or through the electric field-driven migration of oxygen vacancies through the oxide volume.[91,92] Since these changes involve atomic displacements, ReRAM devices tend to suffer from high cycle-to-cycle write noise, high device-to-device variability, relatively high read conductance, and random telegraph noise (RTN) in the stored weight value due to electron trapping, which more strongly affects the high-resistance state than the low-resistance state.[14] Several works have considered scaling dense crosspoint ReRAM arrays to the third dimension to extend the size of the neural network that can be mapped to the array[93–95] and to increase the number of synaptic bit slices to enable greater weight precision.[96]

Phase change memory (PCM) uses the energy delivered by a current pulse to cause a phase transition of a chalcogenide glass from a high-resistance amorphous phase to a low-resistance crystalline phase. Different regions of the device volume can be locally amorphous or crystalline, enabling many intermediate conductance levels. However, while gradual resistance reductions have been demonstrated (SET), increasing the resistance (RESET) requires melting and quenching the entire volume of material and is therefore abrupt[6]—this shortcoming can be mitigated at the array level, as we will discuss in Sec. VII. Compared to ReRAM, PCM devices require a larger programming current[76] and are susceptible to drift due to structural relaxation, particularly in the amorphous phase.[97]

Electrochemical devices have recently been demonstrated, which implement highly linear and symmetric synapses with a large number of finely spaced states, which are ideal properties for training.[81,98] By applying voltage pulses of the appropriate polarity to the gate of a

TABLE I. Comparison of presently available non-volatile analog memory technologies.

	Resistive RAM	Phase change memory	Floating gate/charge trap memory	Redox transistor	Ferroelectric FET
Maximum resistance	∼1 MΩ	∼1 MΩ	∼1 GΩ	∼10 MΩ [70,72]	∼1 GΩ
Device area[73]	$4F^2$	$4F^2$	4–$10F^2$	Large	$4F^2$[74]
Endurance[a]	10^{12} cycles[75]	10^{12} cycles[76]	10^5 cycles	$>10^9$ cycles[72]	10^9 cycles[74]
Programmable resolution[b]	8 bits[77]	5 bits[78]	7 bits[79,80]	9 bits[81]	5 bits[82]
Write current	1 μA [83]	100 μA [76]	1 pA[c] [84]	10 nA [72]	…
Write speed	ns	ns	ms	ms,[70] μs [72]	ns
Update stochasticity	High	High	Low	Low	Moderate
Update linearity	Poor	Poor	Moderate	Good	Poor
Symmetric update?	No	No	Variable[d]	Yes	No

[a]These values indicate the number of digital write-erase cycles between the maximum and minimum conductance states. Analog synaptic updates can be much smaller, potentially leading to higher endurance and lower write latency/energy for neuromorphic applications.
[b]The values correspond to state-of-the-art programming precision in single devices, which may require the use of iterative write-verify schemes. Achieving the same resolution for every device in an array is more challenging.
[c]Refers to programming by Fowler–Nordheim tunneling, which uses very low current but a relatively high voltage (∼10 V).
[d]The gate bias can be used to control whether the nonlinearity is asymmetric or close to symmetric.[85]

redox transistor structure, mobile ions can be injected or removed from the channel, modulating its carrier density and hence its conductance. Low channel conductance (100 nS) and sub-microsecond write times have recently been demonstrated in redox transistors integrated with a ReRAM selector to enable long state retention.[72] A remaining challenge with these devices is the difficulty of co-integration with CMOS technology due to the added fabrication complexity and the temperature incompatibility of polymer device processing with back-end-of-line forming anneal steps.[99] In spite of this, their promising electrical properties may pave the way toward modified fabrication steps to deal with these limitations or novel polymer and ionic blends that can accommodate modern CMOS process flows.

A non-volatile memory with some similarities to flash memory is the ferroelectric field-effect transistor (FeFET), which allows threshold-voltage programming by integrating a ferroelectric layer within the MOS gate stack. The degree of electric polarization of the ferroelectric layer can be controlled using the electric field, provided by short voltage pulses with very small current. While a symmetric update response with multiple conductance states has been demonstrated, training is complicated by the fact that different ferroelectric domains within the layer require different switching voltages, which necessitates a more complicated write scheme with variable-amplitude or variable-length pulses.[82,100] FeFETs are potentially more attractive as VMM engines for inference[101,102] but may be practically limited in this case by their comparatively short retention, which arises from charge leakage through the gate and the presence of a parasitic depolarization field.[103]

Magnetic tunnel junction (MTJ) based memories, including spin-transfer torque magnetic random access memory (STT-MRAM), are well suited for storage due to their high density, high endurance, low write energy, and fast write speeds. However, because STT-MRAM traditionally holds only two magnetoresistance states, it can be used only as a digital or binary synapse rather than an analog one. More recently, multi-level storage has been demonstrated in magnetic devices that use spin-transfer torque to modulate the position of a magnetic domain wall, which can then be read out electrically using an MTJ.[104] The challenge remains, however, of a low magnetoresistance on/off ratio, which reduces the effective bit resolution in the presence of noise.

3. Floating-gate and charge-trap memory

The floating-gate transistor[108] and the closely related charge-trap memory,[109] which are the basis for commercial flash memory, are also attractive options for neuromorphic computing owing to their mature fabrication technology. In these devices, the analog synaptic weight is stored as charge that resides on an electrically isolated internal electrode or within trap states in an insulator. The charge is added or removed via Fowler–Nordheim tunneling or hot-electron injection through the surrounding oxide. Their advantages lie in their low variability, multiple bits per cell, and their access to a subthreshold regime of operation with very low conductance, which enables scaling to larger crossbars. However, for *in situ* training, a significant drawback lies in the large voltage (for tunneling), large current (for hot-electron injection), and long write times incurred by the programming pulses. There are several options for integrating floating-gate devices in a dense array for VMM acceleration, which we summarize below.

Floating-gate devices can be integrated into a crossbar much like two-terminal variable resistors as shown in Fig. 8(a), with the input voltage applied across the source and drain terminals to execute a VMM through Ohm's law.[85,110,111] This configuration allows the device to be operated in various regimes of operation—subthreshold, triode, or saturation—depending on the gate voltage applied during read. In training accelerators, this scheme is also compatible with the parallel outer product update, which we describe in Sec. VII.[85]

Alternatively, by operating the transistors as programmable current mirrors as shown in Fig. 8(b), the inputs to a VMM can be applied via the gate terminal.[105,112–114] In the subthreshold regime, the weight can be implemented as a scaling factor between two currents, rather than through Ohm's law,

$$W_{ij} = \frac{I_{ij}}{I_i} = \exp\left(\kappa\,\frac{V_{\mathrm{fg},ij} - V_{\mathrm{fg,ref}}}{kT/q}\right), \tag{6}$$

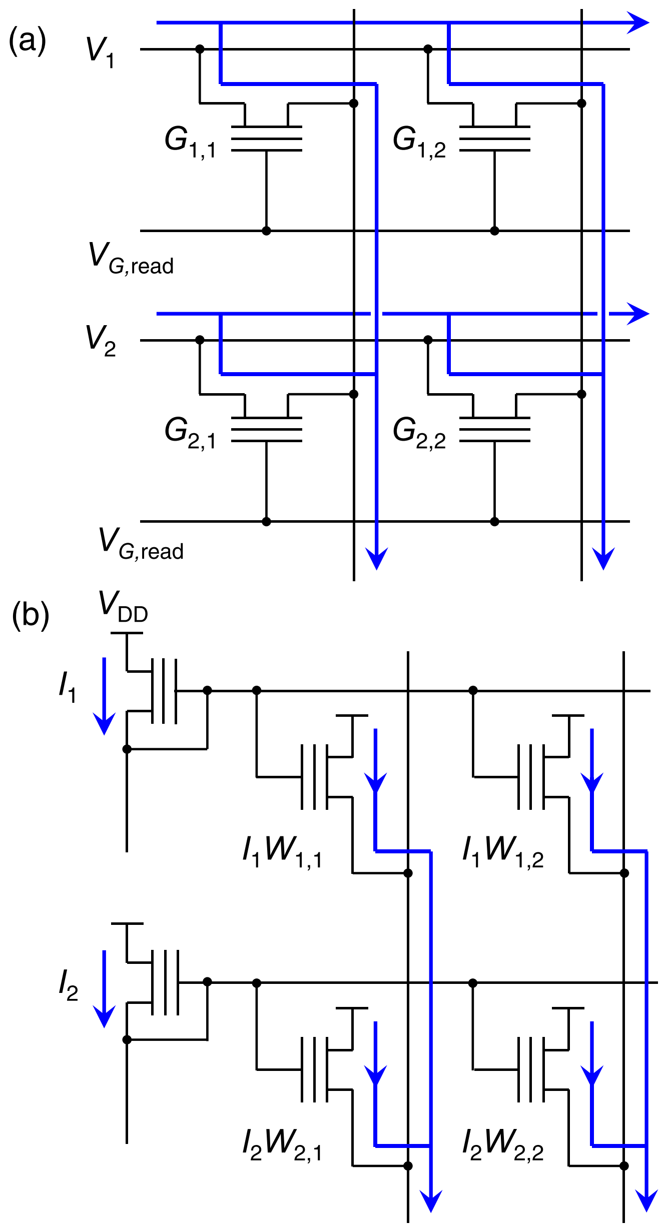

FIG. 8. VMM using a 1T array of floating-gate devices operated (a) as programmable resistors[85] and (b) as gate-coupled programmable current mirrors.[105] In the latter case, a voltage input must first be converted into a current; fully current-mode operation can also be used.[106,107] The floating gate voltage on the row drivers is set to a uniform reference, $V_{\text{fg,ref}}$.

where I_{ij} is the drain current through the floating-gate synapse, I_i is the input current on row i that is injected through a reference transistor, V_{fg} is the voltage on the floating gate, κ is the gate efficiency, T is the temperature, k is the Boltzmann constant, and q is the electron charge. The transmission of a gate voltage across the rows requires a minimal amount of input current, reducing the susceptibility of the input signal to parasitic voltage drops, but the device drain currents must still flow across both the rows and columns. The temperature dependence in Eq. (6) can be compensated using external circuitry.[105,113] The gate-coupled floating-gate configuration has been used to demonstrate moderately large crossbar inference systems for MNIST image classification.[106,115] The one-transistor-per-cell configuration has also been shown to be programmable with minimal write disturb to neighboring cells,[80] potentially enabling training in a dense array.[116]

Three-dimensional NAND flash, used for high-volume data storage, can also be used for in-memory VMM computation with large neural network models. In such an architecture, floating-gate devices are serially connected in vertical pillars, and VMMs can be executed one 2D slice at a time. To avoid undesired sneak paths, the devices in the other layers are left unselected by an appropriate voltage biasing scheme.[110,111,117] Compared to a standard 2D array, each 2D slice within a 3D array must contend with greater disturbance from parasitic resistances and capacitances.

4. VMM with volatile capacitive memories

Some early analog accelerators proposed crossbar-based parallel VMMs using volatile capacitive memories. For instance, Ref. 118 stores a 6-bit analog weight in the charge on a capacitor, whose product with a 3-bit input is computed with a multiplying digital-to-analog converter (MDAC) circuit; the currents are then summed using Kirchhoff's law. In Ref. 119, the weights are stored in binary DRAM cells. When a binary input signal is applied to a row, charge is transferred from each cell to a MOS capacitor whose gate is connected to a column line. The total amount of transferred charge among the devices in a column can be capacitively sensed as a voltage change on the column line, which is digitized by an ADC. All rows can be activated simultaneously to realize a parallel VMM. Afterwards, the charge can be returned to the DRAM cell to restore the weight. To perform higher precision computations, this work proposed bit slicing of both the inputs and the weights, as we will define in Secs. V and VI, respectively.

References 120 and 121 also store an analog weight as the charge on a trench capacitor, which in turn controls the resistance of a readout CMOS transistor operated in triode mode. The scheme requires several additional transistors per unit cell to charge and discharge the capacitor, offsetting the area efficiency of the crossbar. In general, accelerators based on volatile memories suffer from low retention and potentially require charge refresh circuits at the array crosspoints, leading to high footprint.[118] This makes them impractical or expensive for inference and greatly constrains their use as training accelerators.

Some of the more recent mixed-signal accelerators fully or partially execute the VMM in the analog domain using capacitive circuits but do not employ the concept of a computational memory crossbar. Ref. 122 uses strictly binary inputs and weights, which allows compact efficient multiplications in the digital domain but computes the sum of these binary products using an array of switched capacitors rather than a digital adder tree. In Ref. 123, switched capacitors integrated into a successive-approximation-register (SAR) ADC perform the analog VMMs, achieving a high energy efficiency of 13 fJ per multiplication using 3-bit weights and 6-bit activations.

B. Access devices

To ensure that the conductance states of the memory elements remain truly non-volatile during programming, the crossbar in Fig. 5

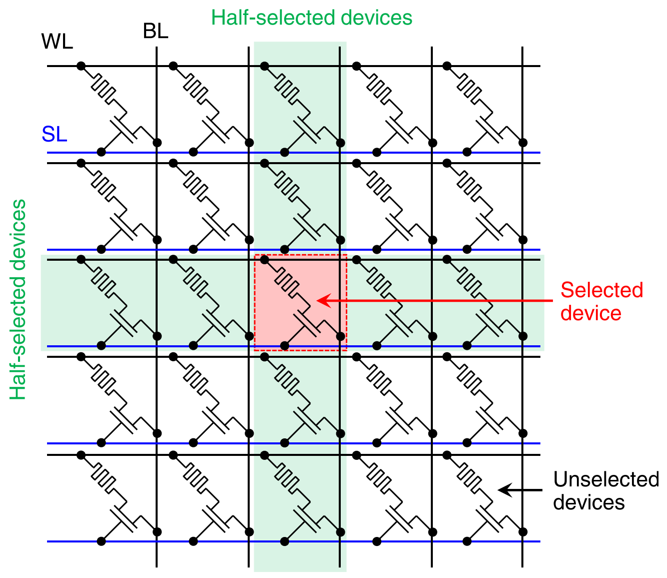

FIG. 9. Resistive crossbar with one access transistor per cell (1T1R). To select a device, its select line (SL, blue) is raised to a high voltage and its bitline (BL) is held to a low voltage.

is typically augmented with access devices (Fig. 9), as required in arrays used for storage-class memory. These devices are characterized by a highly nonlinear *I*–*V* relationship and serve as a barrier to current conduction when the applied voltage is low but are transparent when the voltage exceeds a threshold. Their primary purpose is to ensure that when both the wordline and bitline (row and column) corresponding to a selected device are activated during write, the other devices in the array see a voltage that is below the threshold of the access device and pass only a minimal current—this is particularly important for half-selected devices, which share a row or column with the selected device. Failure to suppress current flow in these devices can lead to a disturbance of the stored state. Leakage currents accumulated over the array can also dissipate significant power and cause voltage drops across the rows and columns that can lead to programming errors. The access device is typically a CMOS transistor, though highly nonlinear ReRAM devices have also been engineered to provide fast, accurate select operations.[124,125] We refer to Ref. 126 for a review of access device design considerations and available technologies.

3The access device converts the passive crossbars based on resistive two-terminal devices into 1T1R (1 transistor, 1 resistor) arrays, with a third terminal to control the access device during writes. Some configurations based on three-terminal devices, such as the floating-gate transistors discussed previously, may be less sensitive to write disturb[85] or may not need any additional access devices if the biasing scheme during programming can be optimized.[80] A passive 12 × 12 ReRAM crossbar has been experimentally demonstrated, which leverages the *I*–*V* nonlinearity of the memory device itself to avoid the use of access devices.[127] However, this approach would add to the already long list of requirements that the memory devices must meet, and the level of nonlinearity in the device is likely insufficient for crossbars of larger size.

Access transistors can also be used to more efficiently pass VMM inputs to the memory elements during inference. By applying the input as a voltage on a high-impedance access gate (i.e., on the select line in Fig. 9), the signal can remain free from distortion caused by parasitic resistance. The nonlinear characteristic of the transistor implies, however, that this voltage can practically only be binary (bias on or off). To communicate values with multiple bits of precision, the input can be transmitted one bit at a time to the array, or the input value can be encoded in the width of the input pulse, which modulates how long the access transistor remains turned on. We will review both of these approaches in Sec. V.

C. Limitations on crossbar size

For neuromorphic accelerators, an important architectural question is: what is the largest realistic memory crossbar that can be used? The numerical answer depends on the memory and access devices that make up the crossbar and is in general determined by two principal considerations:

(1) *Parasitic resistances in the crossbar*: in large crossbars, the resistive voltage drop across one full row or column can be significant. This effectively reduces both the fidelity of the input signal seen at the array element and the fidelity of the weight value seen at the output, and this effect is nonuniform across the array. It is possible to partially compensate for this effect using strategies like nonlinear weight mapping, which we will discuss in Sec. VIII. However, at some point, compensation becomes impossible without increasing the input voltage; this sets a limit on the crossbar size and strongly prioritizes nonvolatile synaptic devices with intrinsically lower conductances to enable larger arrays.
(2) *The on/off ratio of access devices*: if this ratio is insufficient, then the crossbar suffers from large power dissipation during programming and potentially also from write disturb. A reasonable limit on crossbar size is the point when the total leakage in all the unselected and half-selected devices matches the current flowing through the selected device. For example, with an On/Off ratio of 10^6, the limit on crossbar size is roughly 1000 × 1000.[126]

Other considerations may become important limiters of crossbar size by degrading the accuracy of the neural network. For example, the effects of device-to-device variability and noise on the VMM result generally increase with the number of rows or columns.[88] For emerging memory devices, yield is also a practical concern. Many of the experimental demonstrations of fabricated crossbars[127–132] have not been large enough to be severely constrained by these issues. To date, inference on the MNIST task has been demonstrated on several large-scale crossbars: a 512 × 1024 binary ReRAM array,[133] a 500 × 661 PCM array,[134] and a 785 × 128 floating-gate array.[115]

V. PERIPHERAL CIRCUITS IN ANALOG ACCELERATORS

The peripheral circuits accompanying the memory crossbar typically comprise a large share of the energy consumption and area of an analog neuromorphic accelerator, and in many cases, they also dominate the latency.[62,68] Of these components, the process of converting between analog and digital signals typically carries the largest energy overhead. Since the analog accelerator interfaces with a digital processor at its input and output, these conversion steps are always necessary at the edges of the system (unless the input is received in the analog domain directly from a sensor, as in Ref. 135). Communication between crossbars, however, can be carried out in the analog or digital domain.

A. Analog vs digital routing

Most architectures opt for digital routing between crossbars, as it allows the use of dense and efficient on-chip digital interconnection networks[138] and because analog signals are prone to degradation by noise and distortion. As a case study, HP's Dot Product Engine (128×128 crossbar) was evaluated in simulation using both analog routing—with buffers and repeaters—and digital routing, which interfaces with the crossbars via ADCs and DACs.[65] The authors found that accuracy on the MNIST task degraded more strongly with analog routing compared to digital routing with just four bits of resolution in the ADCs/DACs. In the analog approach, errors in the memory array due to noise, nonlinearity, and variability accumulate from layer to layer, whereas the quantization step in the ADC effectively limits the propagation of these errors between layers. In RENO, the authors take advantage of the short distance between crossbars ($\sim$0.5 mm) to transmit data in analog form though a network of analog sample and hold (S&H) buffers and switches.[139,140] Accounting for circuit noise and device process variations, they simulate a classification accuracy loss of several percent for a few small tasks (using one to two hidden layers) relative to a comparable system with fully digital processing and routing. The higher accuracy of the digital approach came at the cost of a lower energy efficiency and higher latency.[139]

Some papers circumvent the need for an ADC by representing data as a pulse with fixed amplitude but variable duration.[137,141] By encoding the signal in the time domain, this approach provides some immunity to noise and distortion that primarily affects the signal amplitude. We will look more closely at this scheme in Sec. V B 2.

B. Driving the crossbar inputs

One of the primary differentiators between analog architectures is the way in which they represent the input signals to a VMM operation in order to drive the rows of a memory crossbar. We review several categories of input representations below.

1. Analog voltage levels

The most conceptually direct way of implementing the VMM in Eq. (5) is to encode an element x_i of the input in the amplitude of a voltage signal V_i, as shown in Fig. 10(a).[65,129,136] For an input with B_{in} bits of precision, this method requires a B_{in}-bit DAC to supply the $2^{B_{\text{in}}}$ possible analog voltage levels to a crossbar row or a $(B_{\text{in}} - 1)$-bit DAC if one of the bits determines the sign of the voltage pulse. The advantage of this method is that the full VMM can be completed in a single crossbar read operation, and the latency does not scale with the precision of the input. However, the area and energy consumption of the DAC could scale exponentially with B_{in}. While the overhead of a single DAC scales similarly to that of an ADC, the DACs cannot be shared across the inputs (which must be driven simultaneously) in the same

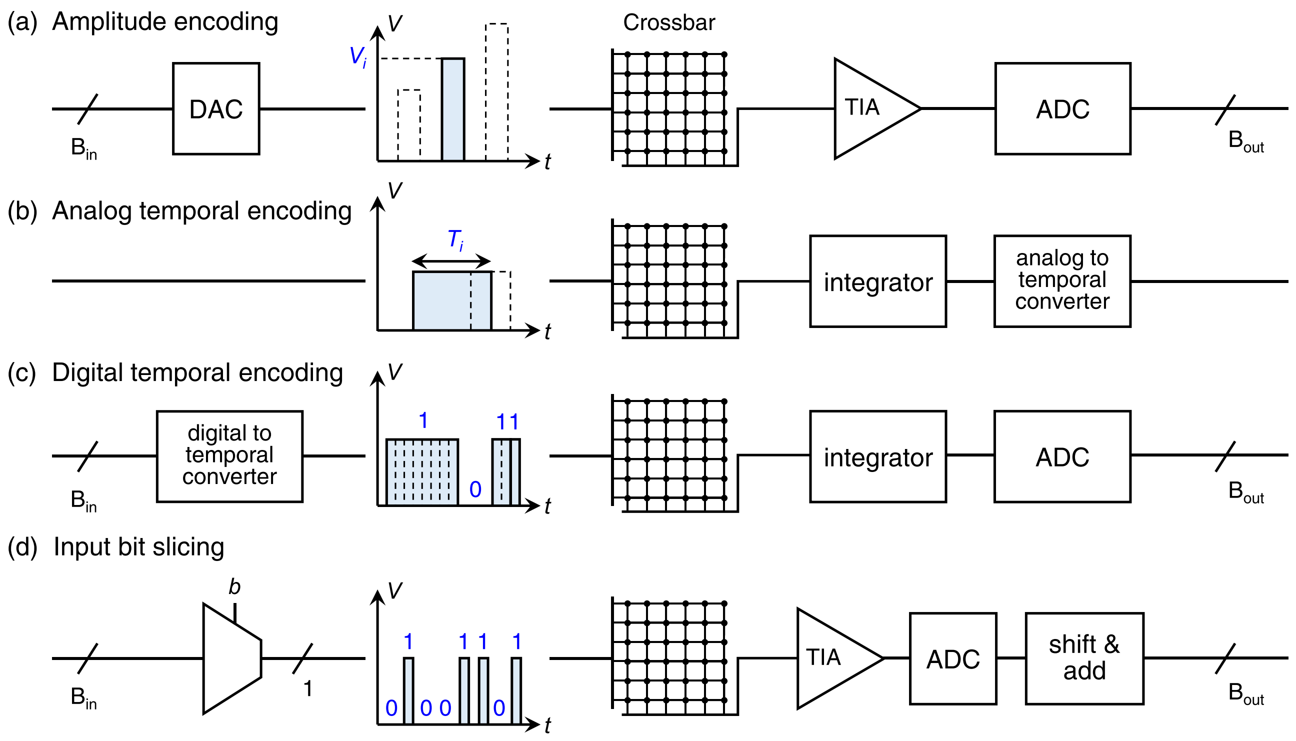

FIG. 10. Four different schemes for representing the crossbar input signal and the associated peripheral circuitry in the signal path: (a) voltage amplitude encoding,[65,136] (b) analog temporal encoding,[134,137] (c) digital temporal encoding,[62] and (d) input bit slicing.[12,68] The transimpedance amplifier (TIA) can be replaced by a different current-to-voltage converter, such as a sample-and-hold circuit.

way that a high-precision ADC can be shared or multiplexed over the outputs (as we discuss in Sec. V C). The output currents on the columns can be converted into a voltage using a transimpedance amplifier (TIA),[65] a sample-and-hold or integrator circuit, or with similar compact sensing circuitry.[142,143] The analog voltage output is then converted to a digital signal using an ADC and then transmitted to the next stage.

A significant drawback of this approach is the need for the memory elements to behave as ideal resistors with a highly linear I–V curve over the range of possible V_i values; without this, the VMM is distorted and becomes a nonlinear operation.[6] Imposing I–V linearity on the already long list of device requirements for the non-volatile memory element introduces a device/material engineering challenge, which can be circumvented by adopting one of the alternative input representations below.

2. Analog temporal encoding

To eliminate the requirement of I–V linearity, each row can be driven with a pulse having fixed amplitude ($\pm V_0$) but variable duration,[134] as shown in Fig. 10(b). The input x_i is encoded in the pulse duration T_i, converting the resistive crossbar VMM equation to

$$I_j = V_0 \sum_{i=0}^{N_r-1} S_i G_{ij} T_i, \tag{7}$$

where $S_i = \pm 1$ is the sign of the pulse. If the pulse is strictly positive, it can be applied to the gate of an access transistor in series with the synaptic memory device. To implement both positive and negative inputs, a set of switches on each row can be used to connect the unit cells to a voltage of the desired polarity.[144] The input can also be applied directly to the gates of a floating-gate synapse cell, requiring four devices per synapse to implement both real-valued inputs and real-valued weights.[105,112,137]

Aside from the relaxed requirement on device I–V linearity, a benefit of the temporal approach is that it encodes the activation data in a non-digital form that is insensitive to voltage drops incurred in communication between arrays, therefore bypassing the conventional analog-to-digital conversion step and potentially requiring no DAC at the input. However, the information remains susceptible to distortion of the pulse shape. This approach still requires precisely timed ADC-like circuitry to convert the voltage- or current-encoded VMM outputs from the crossbar into temporally coded signals. Since these circuits have finite temporal resolution, the pulse duration (and thus the VMM latency) scales exponentially with the effective number of input bits represented. If any intermediate digital processing must occur between arrays, the temporal signal must first be converted to a digital signal by an ADC.

In Refs. 141 and 145, the crossbar outputs are converted into temporal signals by continuously comparing the column voltages to a linearly ramped voltage signal: this has a similar overhead to a column-parallel ramp ADC. In Refs. 137 and 146, the conversion is performed using a two-phase approach. The summed current on each column, which represents the dot product, is first integrated onto a capacitor. In the second phase, charge is steadily added onto the capacitor over an interval T. When the capacitor voltage crosses a threshold, the output pulse is triggered, which falls at the end of this interval: the greater the charge accumulated during the first phase, the longer the pulse duration in the second phase. This functionality is somewhat reminiscent of a dual-slope ramp ADC[147] but without the overhead of a digital counter.

3. Digital temporal encoding

A variation of the temporal encoding approach maintains a digital output, allowing for digital processing and signal routing between arrays, and encodes the signal in the time domain only within the crossbar computational core. At the input of an array, a digital logic circuit can be used to convert an incoming digital signal into a voltage pulse train, with one pulse per input bit.[62] For each bit position b, the pulse has a duration proportional to 2^b, as shown in Fig. 10(c), and the entire pulse train can be made negative to implement negative inputs. The column currents produced by this pulse train are accumulated on an integrating capacitor and then digitized by an ADC. To maintain a fixed potential on the column line independent of the accumulated charge, Ref. 62 uses a current conveyor integrator that provides a virtual ground at its input. In this approach, the ADC only needs to be run once for all the input bits.

In both the temporal encoding schemes, the length of the temporal signal scales exponentially with the number of bits, making the approach prohibitively slow when using high-precision inputs. However, if the ADC delay is comparable or longer and VMM operations can be run in batch (e.g., in convolutional layers), the crossbar read latency can be hidden by pipelining the analog VMM and ADC stages.[68]

4. Input bit slicing

To avoid the use of analog voltages while maintaining a read latency that is linear with the input resolution, a digital input x_i can be passed one bit at a time into the crossbar using binary voltage pulses of fixed length. This scheme is shown in Fig. 10(d). The output of the VMM with input bit slicing is found by

$$Y_j = \sum_{b=0}^{B_{\text{in}}-1} 2^b \left(\sum_{i=0}^{N_r-1} G_{ij} V_i^{(b)} \right), \tag{8}$$

where $V_i^{(b)} \in \{0, V_0\}$ is the binary voltage pulse amplitude corresponding to the bth bit of the input x_i. This VMM operation requires B_{in} sequential crossbar read iterations (inner sum), each of which requires an ADC step. If bits are presented from the lowest to the highest significance, the exponential in Eq. (7) can be implemented by shifting the digitized crossbar output one position to the right prior to adding the output for the next bit. Thus, the exponentially weighted outer sum can be implemented using a digital shift-and-add circuit at the periphery of the crossbar.[12,68]

An advantage of this approach is that the 1-bit input signal does not require a sophisticated DAC and in some cases can simply be applied to the gate of a transistor that enables or disables current flow on the entire row.[12] Positive and negative inputs can be realized without requiring a third voltage level by using a two's complement digital representation.[68] Additionally, the binary resolution of the input reduces the required ADC resolution, as we will quantify in Sec. VI A.[12,68]

The recently proposed successive integration and rescaling (SIR) scheme, shown in Fig. 11, performs the outer sum in Eq. (7) in the analog domain, which enables input bit slicing without having to run the ADC on every iteration.[146] In SIR, the column currents for the bth input bit are first integrated onto a capacitor as an analog sum of charge; the

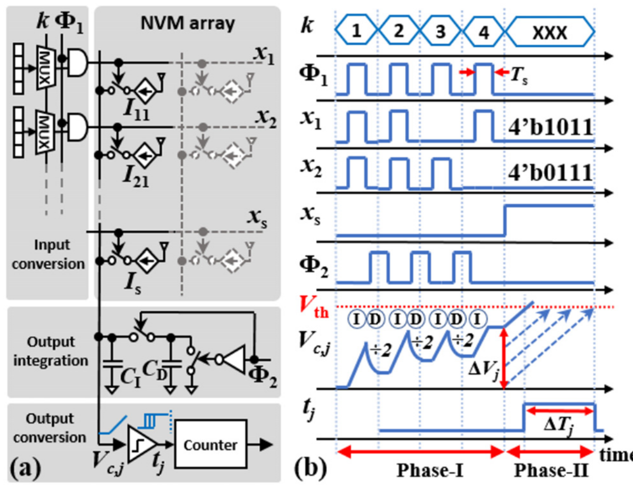

FIG. 11. The successive integration and rescaling scheme for implementing shift-and-add operations in the analog domain. (a) After the currents are integrated on capacitor C_I, it is connected to capacitor C_D, which halves the total accumulated charge on C_I. (b) Timing diagram of a 4-bit VMM. After integrating the final bit, the voltage output is encoded as a pulse length using the two-phase conversion scheme described in Sec. V B. Reproduced with permission from Bavandpour *et al.*, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **28**, 823 (2020). Copyright 2019 IEEE.[146]

change in the capacitor voltage is proportional to the dot product carried out on that input bit. After the integration is complete, a switch connects the capacitor in parallel with a second identical capacitor. The charge redistribution then reduces the voltage on the integrating capacitor by a factor of two. The capacitors are then disconnected, and the rows are driven by the next input bit. In this way, the integrating capacitors store the intermediate results of the bit-sliced VMM, and ideally, the capacitance does not need to scale with the desired bits of precision. By performing analog shift-and-add operations, the SIR technique uses the ADC only once for all the input bits. Since the column outputs now carry more precision, a higher-resolution ADC is required to maintain the same precision achieved by a sequence of digital shift-and-add steps.

C. Analog-to-digital converters

Since the crossbar executes its computation on analog signals, most accelerators require the conversion of the VMM output into a digital form that can be transmitted to the next layer of the neural network or to a host CPU. In accelerators that maintain modest to high precision in the activation values, the ADC can easily dominate the energy consumption, area, and latency of the accelerator. The ADC architecture and its resolution B must therefore be chosen carefully to preserve the intrinsic gains of using the memory crossbar for neuromorphic computations. We will discuss the optimal choice for ADC resolution in Sec. VI A. Here, we review the two types of ADCs most commonly used by crossbar accelerators: the ramp ADC, as used in Ref. 62 and PRIME,[136] and the SAR ADC, as used in ISAAC,[68] PUMA,[148] and the memristive Boltzmann machine.[12] For small resolutions, a high-speed flash ADC has also been used.[119,139] The delta-sigma ADC is another option for high precision in a relatively small area, though its slow speed means that it must be separately provided to every column.[149] A more detailed comparison of the scaling properties of ADCs, not explicitly tied to crossbar applications, can be found in Ref. 150.

In a ramp ADC, the analog signal is compared to a linearly increasing reference voltage that is produced by a ramp generator circuit.[147] The transition in the comparator output is detected and triggers the value of a digital counter to be written to a register, which stores the digital output. The long ramp time, which scales as 2^B, makes this type of ADC too slow for some applications, such as sampling a fast waveform. However, the ramp ADC is uniquely suited to the massively parallel comparisons needed for the neuromorphic crossbar. The entire array can share a single ramp generator and digital a counter, and only a separate comparator and a register need to be provided to the individual columns,[62] as shown in Fig. 12(a). Therefore, the latency scales as $O(2^B)$ and does not increase with the number of columns. The energy cost is dominated by the comparators: if their power consumption is constant during the ramp time, then the total ADC energy consumption for a VMM scales as $O(N_c 2^B)$.

A SAR ADC uses a B-bit DAC to generate the reference voltages to which the input is compared. An internal logic circuit executes a binary search algorithm to find the correct digital output within B comparisons.[151] Due to its large area,[152] a single SAR ADC is typically shared by all the crossbar columns via time multiplexing,[68,153] as shown in Fig. 12(b). The latency of this approach thus scales as $O(BN_c)$. The power consumption and area of a SAR ADC, which uses a capacitive DAC, scale exponentially with the number of bits,[154] though the area scaling might be more favorable with a floating-gate DAC.[150] The total energy consumption by the SAR ADC for a single VMM step therefore scales as $O(BN_c 2^B)$, though the constant prefactor may differ dramatically from that for the ramp ADC. In general, the desired resolution and the number of columns will determine which of the two architectures is faster and/or more energy efficient.

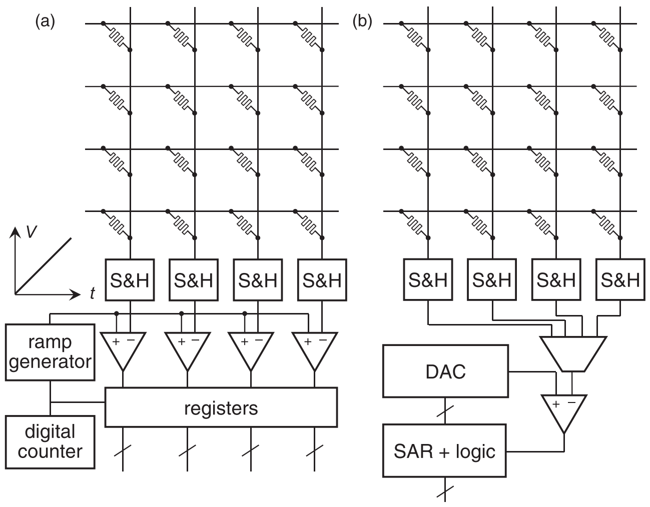

FIG. 12. Analog-to-digital conversion of the crossbar VMM outputs, using (a) a parallelized ramp ADC and (b) a time-shared SAR ADC.

D. The neuron function

The neuron activation function f is necessary for both inference and training but does not benefit directly from the energy efficiency, density, or computational parallelism of the memory crossbar. It is therefore realized separately using an analog or digital functional unit. In analog, the neuron is ideally implemented within the functionality of the already existing peripheral circuits such as the ADC—otherwise, the neuron circuit should be compact, low-energy, and fast. Likewise, in the digital domain, the neuron logic should be sized so that it neither takes up too much area nor limits the system's throughput.

Digital implementations of nonlinear functions such as the sigmoid or hyperbolic tangent typically store the function values[12] or a piecewise approximation of the function[46] in a lookup table. The larger the lookup table, which is usually implemented in SRAM, the greater its area and energy cost. For the ReLU function, typically used in CNNs, the logic block can simply check the sign bit of the input to multiplex between the input value and zero.[136,156] Since these nonlinear functions are applied identically to many input values, they can be executed in parallel using single-instruction multiple-data (SIMD) vector instructions. Wider instructions, however, come at the cost of a larger neuron functional unit. The PUMA architecture uses smaller functional units to execute the equivalent wide instruction over several cycles: this compromise offsets the parallelism of SIMD but still reduces the overhead of instruction fetch and decode.[148]

Reference 155 points out that implementing a transcendental function such as the sigmoid or tanh in the digital domain with full software-equivalent accuracy may require a higher ADC resolution than the number of output bits from the activation function. The issue can be seen in Fig. 13(a)—when a nonlinear function f is mapped onto a discretized version of the analog outputs from the crossbar, there is an inherent problem with undersampling in the steeper regions of f and oversampling in the flatter regions. Thus, for the same ADC resolution, there may be a slight advantage in precision by implementing these functions in the analog domain. However, the neural network accuracy loss resulting from the over/undersampling issue has not been quantified.

The neuron activation can be implemented using the analog peripheral circuitry of the crossbar, with the notable caveat that these methods are incompatible with input and synaptic bit slicing, since the function f is applied on the full-precision result of the VMM. The ReLU (and bounded ReLU) function can be integrated within an ADC simply by setting the appropriate upper and lower bounds on the input range.[118] A binary threshold activation function, as used in binary neural networks, can be realized at no additional overhead using column comparators and a reference voltage.[157,158]

The sigmoid and tanh functions are more complicated to accurately realize in the analog domain but can be implemented using an ADC with nonlinear quantization: equal changes in the output

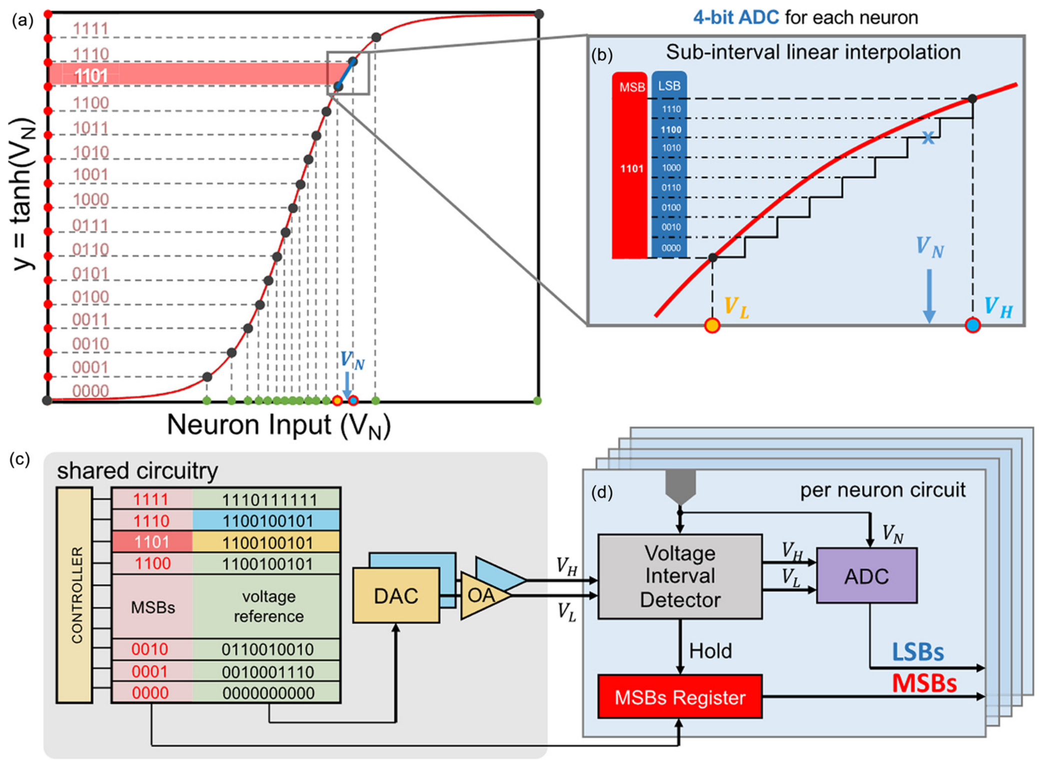

FIG. 13. Mixed-signal integration of a nonlinear activation function with an ADC. (a) The function is first coarsely mapped onto a nonlinear ADC, and then, (b) a linear ADC interpolates within the selected input voltage range of the nonlinear ADC. (c) Lookup table implementation of the nonlinear ADC, which yields the most significant bits of the output, and (d) interval detection circuitry and linear ADC, which yield the least significant bits. Reproduced with permission from Giordano *et al.*, IEEE J. Emerging Sel. Top. Circuits Syst. **9**, 367–376 (2019). Copyright 2019 IEEE.[155]

correspond to unequal spacing in the input voltages.[159] To mitigate the over/undersampling issue above, Ref. 155 splits the ADC/neuron overhead across two circuits: the neuron function is first coarsely sampled using a reconfigurable nonlinear ADC, and then, a piecewise-linear approximation of the neuron function is more finely sampled using a linear ADC. Their scheme is shown in Fig. 13. The nonlinear ADC returns the most significant bits (MSBs) of the output, and the input range corresponding to these MSBs is used as the range for the linear ADC, which returns the least significant bits (LSBs). Alternatively, dedicated analog circuits have been proposed, which approximately implement the sigmoid function using the transfer function of a low-gain comparator[160] and a more compact six-transistor circuit.[161]

VI. ARCHITECTURES FOR INFERENCE

Due to the highly demanding device and circuit requirements for accurate neural network training, the acceleration of inference is a nearer-term application for analog neuromorphic accelerators. In this use case, the neural network is first trained externally (e.g., using processors on the cloud), and the trained weights are subsequently loaded onto edge devices to perform inference on new examples. For non-volatile memory based accelerators, this means programming all the devices once, possibly followed by occasional re-programming to update the network weights or to combat component failures and drift. In this section, we review the main architectural considerations that govern the design of an analog inference accelerator.

Often, it is desirable in analog accelerators to reproduce the neural network inference accuracy that would have been achieved by an equivalent digital computation at a given level of data precision. This choice, which we will call the full precision assumption (and sometimes also called "software-equivalent" accuracy), can be realized by meeting three criteria: (1) the weights are reliably stored at full precision by the memory elements, (2) the ADC resolution matches the resolution of a digital VMM, and (3) the voltage corresponding to the least significant bit lies above the noise floor. The third requirement can be met by appropriately setting the ADC range and integrating the column currents for long enough to obtain an acceptable signal-to-noise ratio.[63] The first requirement is met by providing devices with sufficiently high programmable resolution or, if not available, by slicing the bits of a weight value across multiple memory devices—we will discuss this technique in Sec. VI A. The ADC resolution will be considered in Sec. VI B.

Some architectures forego the full precision assumption to achieve better energy efficiency or smaller area. This option may be preferable at low to moderate activation precision (e.g., eight bits or less) and a weight precision that is well matched to the effective programmable precision of the individual devices. The main advantage of this approach is the elimination of the computational and data movement overhead—in the form of energy, area, and latency—associated with synaptic and/or input bit slicing and the subsequent reconstruction of the VMM result.

A. Synaptic bit slicing

A limitation of non-volatile memory devices is the number of distinguishable conductance levels that are available to represent a synaptic weight. For inference, the required amount of precision in the weights is generally smaller than that needed for training. The Google TPU, for instance, performs large-scale inference tasks using 8-bit weights.[22] It has been shown that image recognition is possible with even deeper quantization of the weights,[27] down to a single bit,[28,29] but with a possible loss of several percent points in inference accuracy. Thus, for general-purpose inference accelerators, at least eight bits of weight precision would appear to be a good target for high accuracy. Reference 71 evaluates the resilience of a number of state-of-the-art CNNs to low-precision weights and noisy activations during inference.

By comparison, how much precision can be expected in the programmed state of a non-volatile memory device? For state-of-the-art ReRAM, the resistance has been programmed to 0.5% accuracy in a 4×4 array,[77] though the accuracy is lower when writing a 128×64 array.[129] Similarly, for floating-gate transistors, the state-of-the-art programming accuracy for the drain current is about 1% over multiple orders of magnitude.[79,80] These results correspond to 7–8 bits of weight precision and are obtained using a write-verify scheme: after every write pulse, the conductance of the device is monitored, and this is used to adjust the polarity of the next write pulse, and so on until the correct weight value is obtained to the desired precision.[64,77] To speed up programming, variable-amplitude pulses are often used to provide both coarse and fine tuning of the weight value.[77,162] While the write-verify process is potentially energy-intensive and time-consuming, its cost is amortized over the useful lifetime of the neural network or over the time between scheduled model updates, depending on the application and the retention properties of the memory elements.

Therefore, eight-bit weight precision remains at the upper limit of what can be realistically achieved today using a single non-volatile memory device. To enable full-precision computation (including floating-point precision[163]) with limited-precision memory elements, many architectures employ synaptic bit slicing. Similar to bit slicing of the input activations, as discussed in Sec. V B, the B_w bits of a synaptic weight can be segmented into N_w slices with $\tilde{B}_W = B_w/N_w$ bits each, chosen so that an analog memory element can reliably store the bits within a slice. The devices corresponding to different bit slices of the same weight are spatially partitioned onto different columns on the same row of a single crossbar[12,68,136,143,164] as shown in Fig. 14 or onto different crossbars.[153] Bit slicing allows less ideal memory devices to be used, reduces the required ADC resolution, and offers greater protection against the effects of noise and process variations. The number of slices is chosen based on these considerations and based on the area overhead, which is significant: different accelerators have implemented a single weight (of varying precision) using two,[136] eight,[68] and 32 devices.[12] When combined with bit slicing of the inputs, this has been called an "internally analog, externally digital" approach to VMM.[119,143] The VMM equation is now expressed using two digital summations and one analog summation,

$$Y_j = \sum_{b=0}^{B_{\rm in}-1} \sum_{c=0}^{N_w-1} 2^{b+c} \left(\sum_{i=0}^{N_r-1} G_{ij}^{(c)} V_i^{(b)} \right), \tag{9}$$

where b indexes the input bit and c indexes the weight bit slice. The outermost sum corresponds to summing over the input bits, which are streamed sequentially in time, and the middle sum composes the results from weight bits sliced over multiple columns or crossbars. Typically, the weight bit slices are aggregated first, followed by the input bit slices.

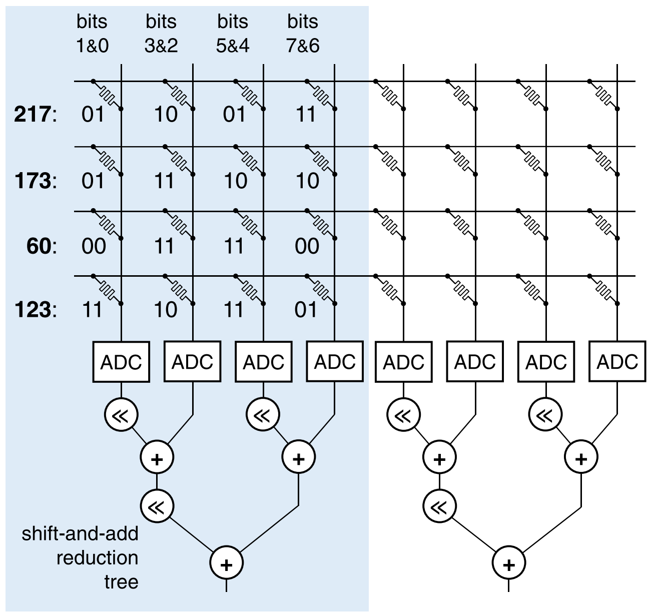

FIG. 14. Column-wise synaptic bit slicing. Here, each weight (represented as an 8-bit integer) is implemented by four 2-bit memory devices spread across four crossbar columns, and the results are aggregated using a shift-and-add reduction tree.

In the column-wise bit slicing approaches, the bits for a given weight are spread out over several columns grouped together in a stripe, as shown in Fig. 14. The sum over the bit slices is performed using a shift-and-add reduction tree, which aligns the VMM output bits collected from different columns and then computes the sum.[12,68,164] If the synaptic bits are sliced across different crossbars, the output of each array contains more partial sums but fewer bits per sum: the shift-and-add operations are applied to operands originating from different arrays and can be integrated with the local communication fabric between crossbars.[153] In both cases, to aggregate the VMM results from the input bit slices streamed over time, further shift-and-add operations are applied to these sums, as shown in Fig. 10(d).

Some energy savings are possible by modifying the way that the input and weight bits are sliced without requiring additional precision in the memory elements. Newton[153] uses Karatsuba's divide-and-conquer algorithm to reduce the number of crossbar computations needed: if both the weights and inputs are split into two slices, for example, only three crossbar reads are needed with divide-and-conquer rather than four. This results in lower ADC usage but requires a third crossbar to implement an extra set of multiplication operands, which imposes a large area overhead. The algorithm can be applied recursively, with compounding overhead. In the Newton architecture, a single application resulted in $\sim$25% greater energy efficiency.

B. Reducing ADC overhead

In full precision architectures, the required ADC resolution must account for the worst case: every input x_i uses all B_{in} bits of input precision, and every weight W_{ij} uses all $\tilde{B}_{\text{w}}$ bits of weight precision needed in a crossbar read. The maximum value of the dot product computed on a column sets the required ADC resolution,[68,119,136]

$$B_{\text{out}} = \begin{cases} B_{\text{in}} + \tilde{B}_{\text{w}} + \lceil \log_2 N_{\text{r}} \rceil & \text{if } B_{\text{in}} > 1, \tilde{B}_{\text{w}} > 1 \\ B_{\text{in}} + \tilde{B}_{\text{w}} + \lceil \log_2 N_{\text{r}} \rceil - 1 & \text{otherwise.} \end{cases} \qquad (10)$$

As we have seen, it can be costly in energy, latency, and area to add a bit of precision to the ADC. To keep the needed ADC resolution low, bit slicing of the inputs and of the weights can be used reduce the operand widths B_{in} and $\tilde{B}_{\text{w}}$, respectively. While this reduces the VMM overhead on a single bit slice, it does require repeating the same operation on all input and weight bit slices. At higher activation precision, where the exponential scaling with resolution of the ADC overhead dominates, this tends to be a favorable trade-off.

We note that where input or synaptic bit slicing is used, Eq. (10) gives the ADC resolution needed to maintain full precision in the individual bit-sliced VMMs. Not all these output bits may be needed in reconstructing the final VMM result at the desired activation precision. After combining these intermediate VMM results via shift-and-adds, several of the least and/or most significant bits of the result are discarded. The Newton architecture determines which of the output bits from the individual bit-sliced VMMs contribute only to those bits that are ultimately discarded and then uses this knowledge to separately reduce the ADC resolution on individual arrays. This is made possible by an adaptive SAR ADC whose components can be gated off when fewer comparisons are needed than the available bits in the DAC.[153] This approach reduces the energy overhead of the ADC but not the area.

ADC resolution can also be reduced at a small energy and area overhead by storing the weight values in a simple encoded manner on the crossbar. In ISAAC, a column of weights is stored in a "flipped" form ($\bar{W} = 2^{B_{\text{w}}} - 1 - W$) if the vector dot product with the maximum possible input $\mathbf{x}$ yields a 1 in the most significant bit—this can be determined statically and reduces the ADC resolution by one bit.[68] In the case that the weights are sliced over 1-bit devices, Ref. 163 uses computational inverse coding to further reduce the ADC resolution by one bit: if a column of weights has more 1's than 0's, its inverse is stored and an 'inverted' bit is used to decode the output. This method works as long as the corner case is handled in which a column contains exactly as many 1's as 0's (by, for example, using an odd number of rows).

In architectures that do not assume full precision, the resolution and range of the ADCs must be carefully optimized to have the least impact on neural network accuracy. In general, this optimization is data-dependent: as a basic example, dot products involving signed inputs and weights tend to have values that are clustered closer to zero than those involving only positive values.[63] The appropriate ADC resolution and range for each crossbar can be calibrated offline based on training data.

When not computing at full precision, a further challenge arises when composing the result of a large VMM whose MACs must be split across multiple crossbars. If a single crossbar has an insufficient number of rows, partial sum outputs from different crossbars should be added before any nonlinear function is applied. If this addition occurs after the digitization step, as is typically done, care must be taken that the crossbar outputs do not saturate the ADC: otherwise, a nonlinearity may be introduced, which was not expected during training. Alternatively, to avoid the accuracy loss, the VMM splitting and the resulting nonlinearity can be explicitly accounted for in the network topology during training (or re-training).[165]

The overhead of the ADC circuitry can also be lowered without sacrificing the full precision assumption by using networks that are trained to use binary weights[166] and/or binary neurons.[157,167–169] This strategy effectively shifts the accuracy trade-off to the algorithm level and can be beneficial in some applications since binary neural networks have been shown to reach inference accuracies that may be acceptably close to those obtained using floating-point weights, as described in Sec. II C. Using binary activations greatly reduces the ADC overhead, and the use of binary weights reduces the number of memory devices needed. Both schemes can improve tolerance to noise, low yield, and variability in the device conductance values.[167] When splitting a large VMM across multiple crossbars, several bits of ADC resolution are still needed for the accumulation of partial sums prior to applying the binary activation function.[157,168] While binary neural networks can bring substantial area and energy benefits to analog crossbar accelerators, we note that digital accelerators and PIM architectures can also effectively exploit the reduced memory requirements and the greatly simplified MAC operations (which reduce to XNOR and bit counting steps) that are involved in processing these networks.[170]

C. Signed computation

Though device conductances G_{ij} are strictly positive, several approaches exist to handle both positive and negative synaptic weights W_{ij} that either require two crossbars per weight or are compatible with a single crossbar per weight. In the most commonly used two-crossbar implementation, a real-valued weight is realized using the difference of two conductances,[62,64,128,171]

$$I_j = \sum_{i=0}^{N_r-1} \left(G_{ij}^{+} - G_{ij}^{-}\right) V_i. \tag{11}$$

The current subtraction in the above equation can easily be implemented in the analog domain by applying the same voltage input with a different polarity to the two memory elements and summing the currents using Kirchhoff's law,[62] as shown in Fig. 15. This approach is

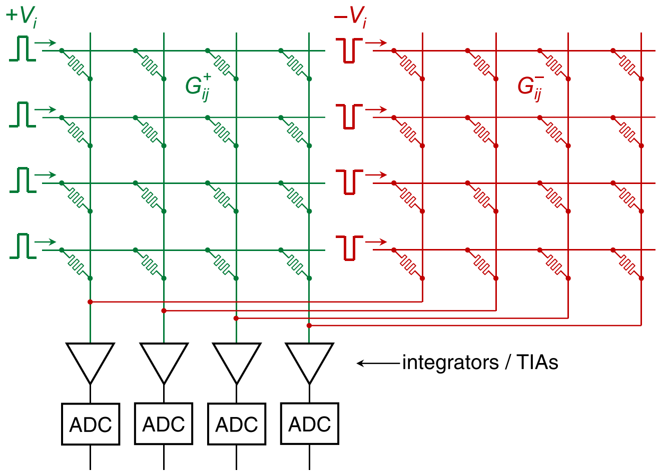

FIG. 15. A general scheme to represent positive and negative weights. When a positive input pulse is sent to the left crossbar, a negative pulse of the same magnitude is sent to the right crossbar and vice versa, and a subtraction is performed by Kirchhoff's law.

compatible with synaptic bit slicing: a separate pair of devices would be used for every bit slice.[136,143] For crossbars that utilize devices with unipolar input–output characteristics, such as floating-gate transistors, four devices per weight may be needed to perform four-quadrant multiplication.[105,112,137]

It is also possible to perform the subtraction in the digital domain, after digitizing the results from both crossbars. However, this approach requires twice the ADC usage. A further advantage of analog subtraction is that it more efficiently exploits the fact that with real-valued weights, the value of the dot product—and thus the voltage on the integrator or TIA—will on average be closer to zero. For accelerators that are not designed for full precision, this allows a significant reduction in the size of the integrating capacitor and the ADC range.[62]

A single-crossbar implementation of analog subtraction has also been proposed, which adds a column of reference bias resistors (or memristors) to the array, whose conductance G_b is effectively subtracted from the conductance of every device using an analog inverter.[172,173] While more area-efficient than a two-crossbar implementation, this approach is more susceptible to errors arising from variability, drift, or offset in the shared reference resistors and the analog inverter.[62]

Real-valued weights have been implemented within a single crossbar using two digital approaches. If the devices in the crossbar are binary, as in the memristive Boltzmann machine,[12] then a real-valued weight can be stored in a digital two's complement representation across the 1-bit memory devices. If the input is also bit sliced to a single bit, the resulting partial sums at the column outputs will be represented in two's complement format after digitization. In the case of inputs that are sliced to a single bit but multiple synaptic bits per memory device, ISAAC[68] stores the weights in a biased digital representation; after digitizing the column outputs, the bias is subtracted to obtain the real-valued dot products. Similar to the analog bias column scheme described above, this method requires an additional column with uniform weights and suffers from the same variability issue, though this is less likely to be problematic if the array elements store low-precision bit slices (two bits in ISAAC).

D. Hierarchical organization

Much like digital accelerators, analog neuromorphic accelerators that store and process large neural network layers can benefit from grouping the memory crossbars into clusters or tiles that can efficiently share a set of resources. These shared resources might include local memory to store activations (usually in the form of SRAM or embedded DRAM buffers,[12,68,148,153] but in a few cases, ReRAM buffers have been used[136,156,174]) neuron circuitry, shift-and-add reduction units, a control unit, and a local communication network. Tiles may be further grouped to perform higher-level reductions or to accommodate the mapping of large neural network layers. A hierarchy with two to four tiers is used, for example, in RENO,[139] the memristive Boltzmann machine,[12] PRIME,[136] ISAAC,[68] Newton,[153] and PUMA.[148] A few examples are shown in Fig. 16. Hierarchical groups of crossbar processing elements can operate independently and concurrently by, for example, processing different input batches, different layers, different partitions of a large VMM, or different sliding windows of a convolution.

A neural network workload is typically mapped to hardware tiles with the goal of maximizing input data reuse and thereby reducing

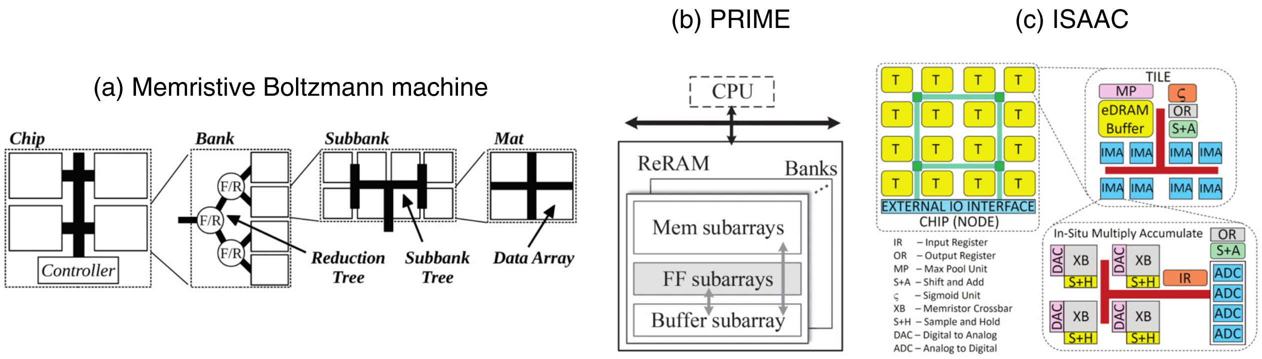

FIG. 16. Organization of crossbar-based computational units in (a) the memristive Boltzmann machine, (b) PRIME, and (c) ISAAC. Reproduced with permission from Bojnordi and Ipek, in Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA) (2016). Copyright 2016 IEEE; Chi *et al.*, in ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) (2016). Copyright 2016 IEEE; and Shafiee *et al.*, in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)* (2016). Copyright 2016 IEEE.[12,68,136]

buffer sizes and data movement: this is considered, for example, in PUMA and Newton.[148,153] In bit-sliced architectures, further area and energy savings are possible by efficiently consolidating a set of crossbars whose outputs are aggregated or reduced by a summation operation, such as the bit slice sum in Eq. (9). For example, Newton's tiles are composed of several *in situ* multiply accumulate units (IMAs), each of which is a collection of crossbars interconnected by an HTree network with integrated shift-and-add circuitry, as shown in Fig. 17. Partial sums from crossbars dedicated to adjacent bit slices are combined at a shared shift-and-add unit at a leaf of the HTree, while results from distant bit slices are aggregated closer to the center. Thus, the partial sums computed by the IMA can be fully supported by a shared communication bus that is made progressively narrower at the edges. While this structure constrains the IMA to map to a single weight matrix, leading to reduced hardware utilization in some cases, it enforces input sharing between crossbars and leads to a significant reduction in the local bus width. These benefits translate to a reduced IMA area and communication energy in comparison to more flexible systems that provide private input and output buses to individual crossbars.

The 3D-aCortex accelerator, which does not use bit slicing, is a notable example of a non-hierarchical architecture that capitalizes on the advantages of a 3D-NAND flash array.[117] Each of the 64 2D planes of the array contains a 32×16 grid of floating-gate crossbars, each of which has dimensions of 64×128. At a given time step, a single plane is selected and VMMs are executed on its crossbars, whose outputs are encoded as pulse lengths (described in Sec. V B). Partial outputs from multiple crossbars are temporally aggregated and digitized using digital counters, shared by all crossbars along a row of the grid, avoiding the communication overhead of performing these reductions across multiple levels of a hierarchy. The entire 3D array shares a global memory and a column of peripheral circuits, increasing its storage efficiency.

E. Convolutional neural network inference acceleration

Forward propagation in MLPs involves potentially large weight matrices and input activations, but data reuse is significant only when performing inference in large batches. Their processing is therefore

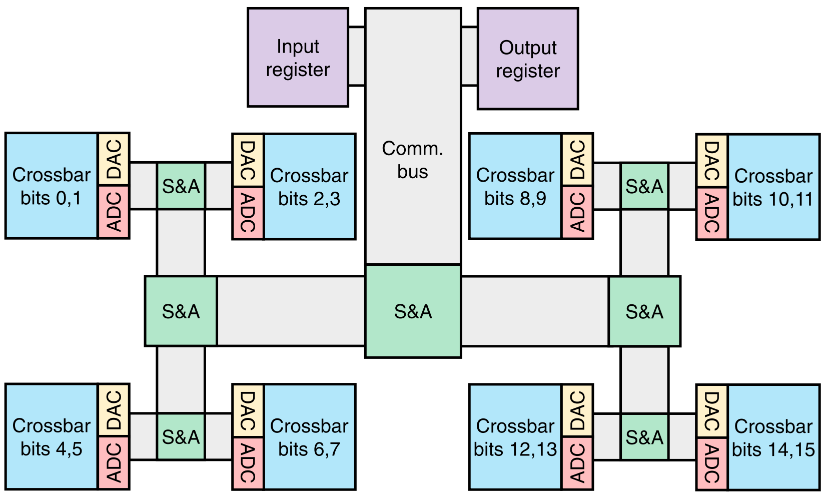

FIG. 17. The *in situ* multiply accumulate (IMA) unit in Newton, which constrains mapping to reduce the area. A 16-bit weight matrix is sliced across eight crossbars containing 2-bit memory devices, and outputs of adjacent crossbars are combined at a shared shift-and-add (S&A) unit. This allows an HTree network within the IMA that is narrower at the edges, where the crossbars reside. This diagram does not include the implementation of Karatsuba's algorithm. Adapted from Ref. 153.

limited by memory bandwidth.[22] In this case, analog computation inside memory crossbars has a large intrinsic advantage since it eliminates the movement of weight data. CNNs, on the other hand, are compute-bound due to the large number of sliding window operations involved and the significantly larger ratio of computations to weights. They make less efficient use of the fact that the VMMs are performed in memory, as digital architectures can capitalize on the extensive data reuse in these operations to amortize data access costs. For CNNs, the primary advantage of analog accelerators over their digital counterparts must come from the intrinsic parallelism of the crossbar VMM operation. Since the ADC tends to dominate array energy costs, the energy efficiency of analog computation is best capitalized by convolutional layers with larger filter sizes and/or a large number of input channels, both of which increase the number of rows in the crossbar that share the same ADC.

In terms of the number of computations, CNNs are heavily front-loaded: many more results are needed from the first layer for every result produced by a later layer. For an analog memory-based accelerator, this can lead to long delays in the early layers and an unbalanced utilization of the resources dedicated to each layer. To address this, ISAAC replicates the weights in the early layers over many physical crossbars, which execute in parallel.[68] By exploiting the fact that a convolutional layer only needs to partially finish its computation before the next layer can begin, the ISAAC pipeline increases throughput and minimizes the size of the shared tile memory. Maintaining balanced resource utilization requires enough weight replication in the earlier layers to keep the crossbars in the final convolutional layers busy. Fully connected classification layers lying at the end of a CNN have different demands than convolutional layers: since they compute only one VMM per example, they are used more sparsely in time and do not require the same buffer sizes as convolutional layers. Newton (a successor to ISAAC) exploits this by allocating different tile designs to convolutional and fully connected layers, leading to a higher energy efficiency and throughput per area than ISAAC but at some cost to workload flexibility.[153]

Similar to digital CNN accelerators, analog systems can save energy by reducing input and output data movement. To match the data access patterns of sliding window computations, shift registers[117,118] and line buffers[168] are commonly used to pass data into a convolutional layer. PUMA uses an input shuffle unit to dynamically re-route values in different registers of an input buffer to different crossbar DACs, exploiting the large input data reuse in adjacent sliding windows to reduce data movement in the buffers.[148]

While all the analog architectures for large-scale CNN acceleration have been evaluated only in simulation, a small-scale analog CNN accelerator has been realized fully in hardware.[180] The system uses eight 128×16 TaO_x/HfO_x ReRAM crossbars to implement two convolutional layers and one fully connected layer and achieves 96% accuracy on MNIST.

F. Flexibility and reconfigurability

The ability to reconfigure an inference accelerator for different machine learning workloads is a useful functionality to the end user but can introduce complexity to the instruction set, lead to additional circuits or logic units for specialized functions, and increase the needed buffer sizes and bus widths when provisioned for the worst case[153]—all of which offset the area and energy efficiency of crossbar computation. Of the analog accelerators developed to date, the PUMA architecture is notable in its highly flexible and programmable design, which provides the necessary circuit blocks to support MLP, CNN, LSTM, and various other workloads while balancing the trade-off between flexibility and its overhead on performance, area, and energy efficiency. It also provides a high-level programming interface—in the form of an ISA and a compiler—between the neural network workload and its tiled, spatial microarchitecture.[148]

Because of the large write energy and latency associated with weight programming, analog inference accelerators cannot dynamically remap different neural network layers onto its hardware during runtime—a feature that is natively supported by digital accelerators. Nonetheless, many analog architectures offer significant flexibility in the network topologies and layer sizes that can be mapped onto them. Finding an area- and energy-efficient mapping is a primary function of the compilers in PUMA[148] and PRIME.[136] In 3D-aCortex, a greedy optimization algorithm is used to map the layers of a large neural network to the fewest number of 2D planes in the 3D-NAND array.[117]

One consequence of performing computation in memory structures is that different workloads will tend to more or less efficiently utilize the hardware resources in the crossbars. The trade-off between efficiency and flexibility becomes particularly evident in the choice of the crossbar dimensions, which are usually homogeneous across the system: smaller crossbars can be more effectively tiled across weight matrices of various sizes and maintain high area utilization, while larger crossbars possess greater peak compute density and can provide greater amortization of the ADC and communication overheads across operations. The trade-off also appears in designing the local interconnection fabric between crossbars. As discussed in Sec. VI D, Newton's IMA uses a fixed reduction network to save area and energy, which is well suited to its style of bit slicing. Meanwhile, the memristive Boltzmann machine's higher-level reduction tree, which aggregates the partial sums of a large split VMM, is reconfigurable; on mapping the hardware to a workload, each node is set to the forwarding (F) or reduction (R) mode as shown in Fig. 16(a).[12] This strategy provides greater flexibility while still saving energy but not area. PRIME is unique in that it provides "full-functional" subarrays, which contain crossbars that can be operated both in computation (VMM) mode and in standard memory mode, where they serve to enlarge the buffers for activation data.[136]

At the extreme end of reconfigurability, the Field Programmable Analog Array (FPAA) provides a flexible physical infrastructure—with software support[181]—for integrating various analog functionalities and for prototyping new architectures for analog computing.[79,150] Floating-gate VMM crossbars and their peripheral circuitry,[105] along with other neuronal functional blocks such as winner-take-all circuits,[113] have been implemented in FPAAs. The Field Programmable Crossbar Array (FPCA) is a reconfigurable architecture that maps several kernels—analog VMM with binary neural networks, digital arithmetic operations, data translation, and data storage—onto its many binary ReRAM crossbars and even to virtual tiles within a crossbar.[182]

G. Performance comparison of inference accelerators

Tables II and III compare the high-level design parameters and performance (measured and simulated) of several selected digital and

TABLE II. Comparison of selected digital and mixed-signal neural network inference accelerators from industry and research.[a] TOPS: Tera-Operations per second. We have counted MACs as single operations where possible. Note that performance (TOPS) is measured at the specified level of weight and activation precision, which differs between accelerators. The results for NVIDIA T4, TPU, Goya, UNPU, and Ref. 122 are measured; others are simulated. TOPS/mm^2 values are based on the die area, where provided.

	NVIDIA T4[175]	Google TPU v1[22,b]	Habana Goya HL-1000[176]	DaDianNao[44]	UNPU[51]	Reference 122 mixed-signal[c]
Process	12 nm	28 nm	16 nm	28 nm	65 nm	28 nm
Activation resolution	8-bit int	8-bit int	16-bit int	16-bit fixed-pt.	16 bits	1 bit
Weight resolution	8-bit int	8-bit int	16-bit int	16-bit fixed-pt.	1 bit[d]	1 bit
Clock speed	2.6 GHz	700 MHz	2.1 GHz (CPU)	606 MHz	200 MHz	10 MHz
Benchmarked workload	ResNet-50[177] (batch = 128)	Mean of six MLPs, LSTMs, CNNs	ResNet-50 (batch = 10)	Peak performance	Peak performance	Co-designed binary CNN (CIFAR-10)
Throughput (TOPS)	22.2, 130 (peak)	21.4, 92 (peak)	63.1	5.58	7.37	0.478
Density (TOPS/mm^2)	0.04, 0.24 (peak)	0.06, 0.28 (peak)	…	0.08	0.46	0.10
Efficiency (TOPS/W)	0.32	2.3 (peak)	0.61	0.35	50.6	532

[a]To enable performance comparisons across a uniform application space, we did not consider accelerators for spiking neural networks.
[b]The TPU v2 and v3 chips, which use 16-bit floating point arithmetic, are commercially available for both inference and training on the cloud. MLPerf inference benchmarking results for the Cloud TPU v3 are available,[179] but power and area information is undisclosed. The TPU v1 die area is taken to be the stated upper bound of 331 mm^2; the listed TOPS/mm^2 values are therefore a lower bound.
[c]The mixed-signal accelerator in Ref. 122 performs multiplication using digital logic and summation using analog switched-capacitor circuits.
[d]The UNPU architecture flexibly supports any weight precision from 1 to 16 bits. The results are listed for 1-bit weights.

TABLE III. Comparison of selected analog neural network inference accelerators. Note that performance (TOPS) is measured at the specified level of weight and activation precision, which differs between accelerators.

	ISAAC[68]	Newton[153]	PUMA[148]	PRIME[136]	Memristive Boltzmann machine[12]	3D-aCortex[117]
Process	32 nm	32 nm	32 nm	65 nm	22 nm	55 nm
Activation resolution	16 bits 1-bit sliced	16 bits 1-bit sliced	16 bits 1-bit sliced	6 bits 3-bit sliced[a]	32 bits 1-bit sliced	4 bits temporal
Weight resolution	16 bits	16 bits	16 bits	8 bits	32 bits	4 bits
Weight storage	ReRAM 8 × 2-bit	ReRAM 8 × 2-bit	ReRAM 8 × 2-bit	ReRAM 2 × 4-bit	ReRAM 32 × 1-bit	NAND flash 1 × 4-bit
R_{high}	~2 MΩ [65]	~2 MΩ [65]	1 MΩ	20 kΩ	1.1 GΩ	…
R_{low}	~2 kΩ [65]	~2 kΩ [65]	100 kΩ	1 kΩ	315 kΩ	2.3 MΩ
Array size	128 × 128	128 × 128	128 × 128	256 × 256	512 × 512	64 × 128
ADC type	SAR (8-bit)	SAR (8-bit)	SAR	Ramp (6-bit)	SAR	Temporal to digital (4-bit)
Clock speed	1.2 GHz	1.2 GHz	1.0 GHz	3.0 GHz (host CPU)	3.2 GHz (host CPU)	1.0 GHz
Benchmarked workload	Peak performance	Peak performance	Peak performance	…	…	GNMT[178]
Throughput (TOPS)	41.3[b]	…	26.2	…	…	10.7
Density (TOPS/mm^2)	0.48	0.68	0.29	…	…	0.58
Efficiency (TOPS/W)	0.63	0.92	0.42	…	…	70.4

[a]PRIME splits the input into two 3-bit slices and uses a 3-bit DAC to convert each digital slice to one of the eight voltage levels.
[b]The ISAAC-CE design point from Ref. 68 is used.

analog inference accelerators, respectively. Of those accelerators listed, the NVIDIA T4, Google TPU, and Goya are commercially available on premise or in the cloud—the remainder are research systems. For many of the other commercial digital systems, the reported performance values are aggregated in Ref. 4. In addition, the MLPerf inference benchmarking project provides an apples-to-apples performance comparison of these accelerators for several benchmark deep learning tasks.[179] With the exception of Ref. 122, all the architectures for which performance numbers are given can support arbitrary CNN and MLP workloads; many can also natively support LSTMs. For the analog accelerators, performance details on specific benchmark models can be found in their respective papers.

The projected performance of the analog accelerators in Table III approximately reaches parity with the measured performance, compute density, and energy efficiency of the best digital accelerators when adjusted for different arithmetic resolutions. Further improvements are still likely needed before analog accelerators can become widely adopted for commercial use. A significant bottleneck for many of the analog architectures is the ADC, which consumes 49% of the total chip power in ISAAC and 41% in the memristive Boltzmann machine. Newton reduces the ADC cost by as much as ~60% over ISAAC via several previously mentioned techniques, such as adaptive ADC resolution and divide-and-conquer. Adding more rows to the crossbar can allow greater amortization of this cost across operations but would require higher ADC resolution in full-precision architectures.

VII. ARCHITECTURES FOR TRAINING

Training large neural networks within an analog neuromorphic accelerator is considered a longer-term goal for the field due to the challenge of meeting the additional device requirements enumerated in Sec. IV A—update linearity, update symmetry, high precision, low write energy and latency, and high endurance. Most of the considerations discussed in Sec. VI for inference accelerators also apply to training accelerators, which must support both forward and backward propagation. In this section, we will review the special architectural considerations for training accelerators and the inherent parallelism provided by the memory crossbar for weight update operations.

Despite the stringent device requirements, successful training of memory crossbars has been demonstrated experimentally for smaller tasks using less demanding training schemes. Most of these results use an *in situ* training approach that represents an intermediate step toward a true training accelerator: forward propagation is performed on the crossbar, the weight updates ΔW are calculated on an off-chip processor, and the updates are programmed onto the crossbar.[183] This approach, used in many demonstrations,[127,128,130,134,171,184] is suitable as a platform to test the learning capability of the memory devices. To simplify the hardware needed for programming, the Manhattan update rule—in which the weight updates are binary (positive or negative)—has commonly been used.[127,130] While this update rule reduces hardware complexity and significantly speeds up programming, Ref. 132 shows that it incurs several percentage points of accuracy loss (shown on a simple facile recognition task) compared to analog updates using write-verify schemes.

Recently, Ref. 131 integrated a memristor crossbar with CMOS circuitry that computes the weight updates and applies the appropriate electrical pulses for programming. The system successfully trained a single-layer perceptron using batch gradient descent. It also trained a small two-layer neural network using Sanger's rule for principal component analysis (PCA),[185] though unlike backpropagation, this update rule does not require the propagation of data between layers.

A. Supporting backpropagation

To provide a significant acceleration of the backpropagation algorithm, both the computation of the layer-by-layer errors $\boldsymbol{\delta}$ and the applications of the weight updates $\Delta\mathbf{W}$ should take advantage of the intrinsic parallelism provided by the crossbar. The former process is relatively simple to implement on an accelerator that has already been designed for inference. Since backpropagation by MVMs is the transpose of forward propagation by VMMs, it can be parallelized in the same manner so long as the architecture supports both the forward and backward flow of data. With additional routing around the crossbars, the same peripheral circuitry (such as DACs, integrators, TIAs, and ADCs) can be reused for an MVM computation.[186] In Ref. 62, for example, MVMs are supported with minimal overhead using additional logic to re-route the inputs to the columns and an analog multiplexer to allow the integration of currents on the rows. This reconfigurable crossbar-based neural core is shown in Fig. 18.

The backpropagation step also requires the application of the derivative of the neuron activation function f' in Eq. (2). For functions such as the sigmoid or hyperbolic tangent, this can be provided by a separate digital lookup table as described in Sec. V D. In an alternative approach, Ref. 160 computes the sigmoid derivative in the analog domain by applying analog arithmetic (with operational amplifiers) to

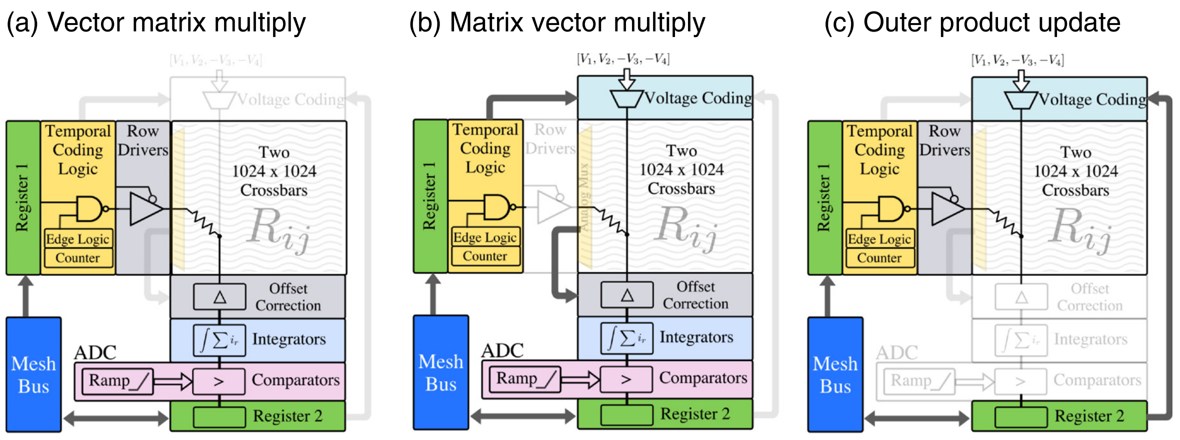

FIG. 18. Reconfigurable neural core in Ref. 62 for implementing (a) VMM, (b) MVM, and (c) outer product update. The inputs to the VMM and MVM are coded as pulse trains [see Fig. 10(d)] using the temporal coding logic. The DAC function of the voltage coding block is bypassed during a MVM but is used during an outer product update. Reproduced with permission from Marinella *et al.*, IEEE J. Emerging Sel. Top. Circuits Syst. **1**, 86–101 (2018). Copyright 2018 IEEE.[62]

the output of the sigmoid unit. The derivative of the ReLU function is typically much simpler to implement since it is a binary function that depends only on the sign of the argument. The calculation of f' can be pipelined with the VMMs during forward propagation, and the MVMs during backpropagation can be pipelined with the weight updates, as done in TIME[156] and PipeLayer.[174]

For the derivative calculation step and the weight update step, it is necessary to have on hand the values of any given layer's activation $\mathbf{x}$ and the computed values of f'. An important distinction from an inference-only accelerator is that these intermediate results generated during forward propagation must not be overwritten until the errors have been propagated back to that layer.[186] This introduces a substantial additional requirement on the size of the local memory accessible to each crossbar, which must fully store the values of $\mathbf{x}$ and f' to minimize data movement during backpropagation.

B. Parallel outer product update

By applying programming pulses simultaneously to all the rows and columns of a crossbar, the outer product weight update in Eq. (3) can be executed in parallel for the entire array. To accomplish this, the activations $\mathbf{x}$ are applied to one edge of the crossbar and the errors $\boldsymbol{\delta}$ to the other edge in such a way that a multiplication effect between the signals is seen at the crosspoints. In Ref. 62, this is done by encoding x_i as a temporal signal (pulse length or pulse train) and δ_j as a pulse amplitude using temporal coding blocks and voltage coding blocks (i.e., DACs), respectively, as shown in Fig. 18(c). The parallel outer product update is depicted in Fig. 19. The programming is performed in four phases to account for the possible sign combinations of $\mathbf{x}$ and $\boldsymbol{\delta}$ without causing write disturb. The learning rate η can be controlled by scaling the pulse lengths or the number of pulses (or pulse trains) fired.[63] Reference 160 implements a similar temporal-voltage encoding scheme

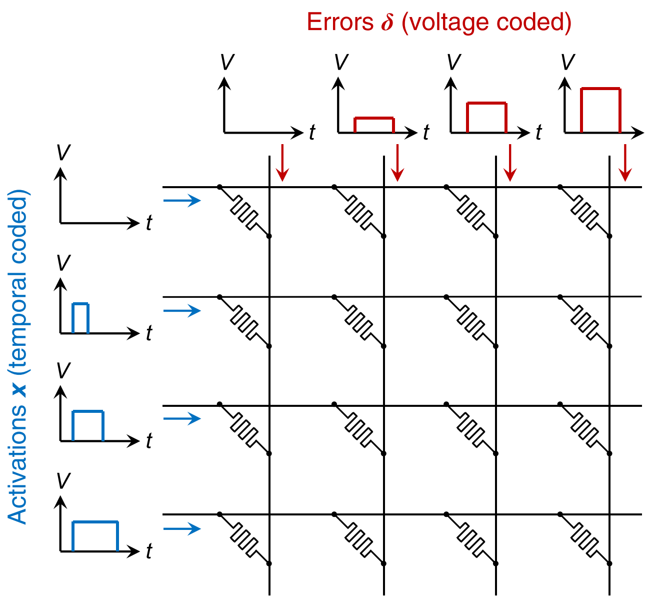
Errors δ (voltage coded)
Activations x (temporal coded)
V
t

FIG. 19. Parallel outer product update of a crossbar array, shown using temporal (pulse length or pulse train) coding for the activations and voltage amplitude coding for the errors.

but applies the time encoded signal to access transistors along a column rather than directly across the devices. The pulse length encoding is done using a circuit that takes an analog rather than digital input for $\boldsymbol{\delta}$.

Alternatively, the multiplication effect can be achieved at a constant write voltage or power by encoding one variable in the length or duty cycle and the other variable in the repetition rate of two overlapping pulse trains.[187,188] However, as with pulse length encoding for VMMs, encoding the two vectors fully in the time domain leads to a stronger exponential penalty in the latency by the desired bits of precision in $\Delta\mathbf{W}$. In the hybrid temporal-voltage coding scheme, this penalty can be efficiently shared between the write latency and the area/energy overhead of the DAC so that neither is very large.

The parallel outer product improves the weight update latency by $O(N)$ compared to a serial row-by-row programming scheme. Another advantage is that only the vectors $\mathbf{x}$ and $\boldsymbol{\delta}$ need to be stored rather than the full weight update matrix $\Delta\mathbf{W}$. One drawback is the potentially large instantaneous power that might be drawn by programming all (or one quarter) of the memory elements simultaneously. To reduce the power consumption, the parallel update can be partitioned into blockwise updates with sufficiently large blocks to avoid breaking the advantages of parallelism.[134] The amount of power consumed during the weight update is data- and task-dependent. In some applications, the fully parallel update consumes only marginally more instantaneous power than a row-by-row update.[187]

C. Training with batch size > 1

Although the parallel outer product update operation carries significant benefits to latency, energy consumption, and storage overhead, it is inherently incompatible with batch sizes greater than one, whose update can no longer be expressed as a rank-1 matrix as in Eq. (3). Being limited to stochastic gradient descent learning can be problematic as it provides a less accurate estimate of the true gradient.[17] It also precludes vectorization or pipelining of forward and backward propagation, thus increasing the training time and under-utilizing the hardware. Finally, it imposes greater endurance requirements on the devices, which need to be programmed after every training example.

Batch training can be realized efficiently, without relying on frequent serial updates, by allocating two crossbars per weight matrix.[63] Forward propagation is performed using the first crossbar, and parallel outer product updates are applied to the second crossbar, which is initialized as a zero weight matrix. At the end of a batch, the total accumulated weight update is read out from the second crossbar and serially written to the first. Since a slow serial programming step is needed only during the weight transfer process, its cost is amortized over all the examples in a batch.

The PipeLayer accelerator takes the above approach of using a second ReRAM array as a buffer to store intermediate weight updates in a batch.[174] Furthermore, it physically duplicates each weight matrix so that one is used for forward propagation and the other for backpropagation. In this way, the two processes can be pipelined within a batch without weight conflicts. At the end of the batch, the accumulated weight updates are copied from the buffer array to all crossbars containing the copies of the weight matrix. The dataflow for this pipeline is shown in Fig. 20(a) for a single training example and in Fig. 20(b) for multiple examples. The

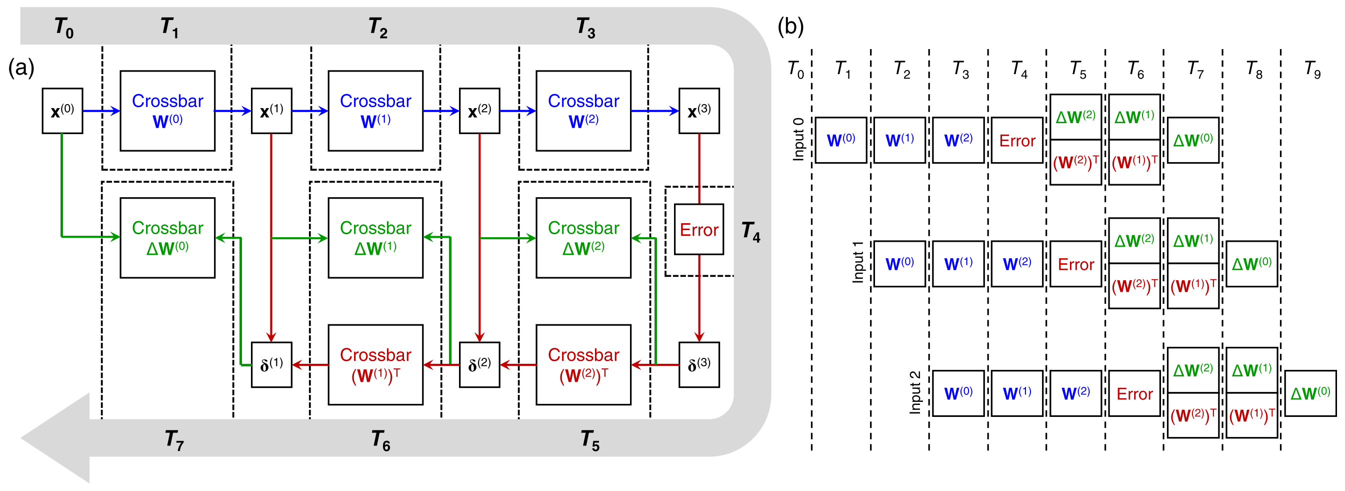

FIG. 20. The PipeLayer architecture. (a) Dataflow of a single example through a 3-layer neural network for training. Each time step may correspond to several computational cycles. The computational blocks active in each time step are labeled: forward propagation (blue), backpropagation (red), and weight update (red). (b) Pipeline for multiple training examples. The weight updates $\Delta\mathbf{W}$ are serially written to the $\mathbf{W}$ and $\mathbf{W}^T$ crossbars at the end of a batch. Adapted from Ref. 174.

intermediate activations $\mathbf{x}$ are also stored in dedicated local ReRAM buffers. The forward and backward pipelining ensures that the storage requirement for these activations is determined by the layer's depth within the network rather than by the batch size. Since these activations are accessed in a first-in first-out order, PipeLayer operates these ReRAM arrays as circular buffers.

Another approach to speed up batch training and to reduce the endurance requirement on the memory devices is to find a good rank-1 approximation to the high-rank batch update. The best such approximation can be constructed from the largest singular value of the batch update matrix and its left and right singular unit vectors. Ref. 189 uses a method based on streaming PCA to refine an estimation of these quantities as the values of $\mathbf{x}$ and $\boldsymbol{\delta}$ are found for each example in a batch. This can be done with a storage overhead that scales as $O(N)$ with the crossbar dimension. Accuracy on MNIST was shown to converge to similar levels as both stochastic and minibatch gradient descent but with about 10× fewer matrix updates as SGD for a batch size of 32.

D. Training convolutional neural networks

Training introduces additional complexities to the acceleration of CNNs. In ISAAC, which is an inference accelerator, high throughput is achieved by providing sufficient replication of the weights in the earlier layers to keep the later layers maximally busy.[68] Many cycles may nonetheless be needed to fill its pipeline. If the same hardware is used for training, the benefit of high throughput is eliminated if the deep pipeline has to be cleared at the end of every batch. The second benefit of ISAAC's pipeline, a reduced buffer size, is also eliminated since the full activation must be retained in memory for backpropagation. PipeLayer further increases the amount of weight replication in each layer to reduce the latency of a single example during training. The crucial trade-off is between the area and training throughput since requiring that early layers be computed in a single cycle results in a prohibitively large number of replicated crossbars.[174]

E. Compensation for device imprecision and asymmetry

Owing to the small parameter adjustments mandated by gradient descent, neural network training requires a higher level of precision in the weight representation than needed during inference. Since individual analog memory devices are limited to at most 7–8 bits of precision, some form of bit slicing is typically used.[96,174] For example, PipeLayer uses 4-bit ReRAM cells. During weight update, the full values of the original 16-bit weights are read out from the crossbars, the update is computed digitally, and the new weights are written back to all the devices.[174] This process incurs a substantial time and energy penalty due to the need for serial readout and programming with every update.

Bit slicing is, without modification, not fully compatible with a parallel outer product update. The periodic carry technique, shown in Fig. 21(a), mitigates this problem by allocating a portion of the conductance range of a device to store one or more carry bits.[190] Taking advantage of the fact that most weight changes are small, parallel outer product updates are carried out on the LSB devices only. Periodically, the LSB device is read, and any carry bits are propagated to the device, which stores the next higher bits. Thus, the cost of serial readout and programming can be amortized over many updates. For some devices, this technique potentially curbs the effects of device nonlinearity: as more conductance levels are dedicated to the carry while performing relatively frequent carry propagation, each device can be more effectively constrained to the most linear part of its conductance range.

Reference 141 performs a different form of bit slicing with two different types of devices: a PCM device holds the MSBs, and a capacitor connected to a read transistor is used as a memory element for the LSBs. The scheme is shown in Fig. 21(b). Charge is added or removed from the capacitor by a pair of CMOS current sources; the high linearity of these updates is used to overcome the issues of nonlinearity, endurance, and strong asymmetry in programming the PCM device. The volatile state of the capacitor is periodically written to the nonvolatile PCM device as in periodic carry. The cost of serial readout and

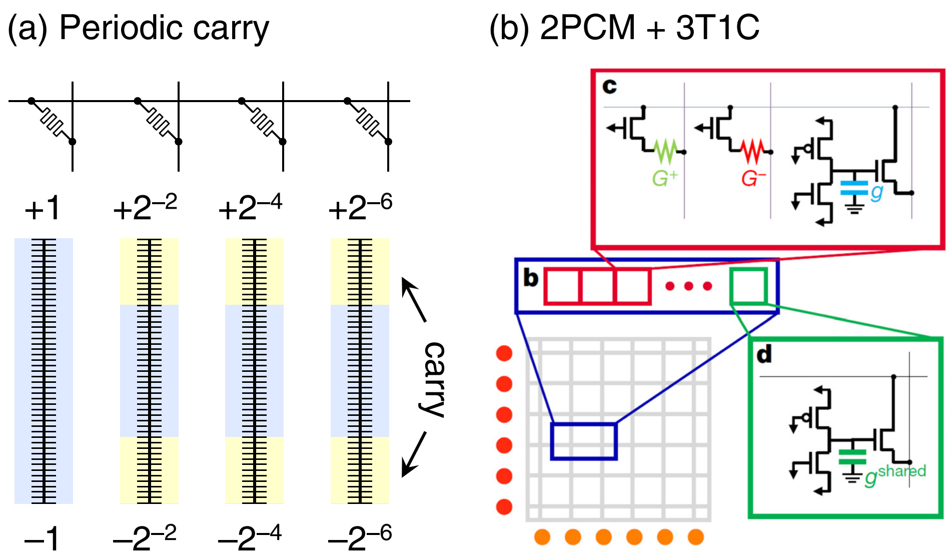

FIG. 21. (a) The periodic carry technique.[190] A portion of the device dynamic range is used to store a carry (overflow), which is periodically propagated to the higher bits. Parallel outer product update is carried out only on the device holding the LSBs. (b) Reference 141 uses a PCM device to hold the MSBs of the weight and a capacitor with charging/discharging circuitry to hold the LSBs; both are paired with a second device to implement signed weights and to overcome update asymmetry. For the capacitive memory, the second device is shared by a row, effectively subtracting a bias. Reproduced with permission from Ambrogio *et al.*, Nature **558**, 60 (2018). Copyright 2018 Springer Nature.[141]

programming is thus spread out over many updates, but the timescale of charge leakage from the capacitor must be considered. A disadvantage of this scheme is the larger size of each synaptic unit cell.

Reference 191 combats the effects of weight imprecision by computing and accumulating the weight updates over a batch in high precision in a digital co-processor. At the end of a batch, these updates are quantized to a low precision that is compatible with the programmable resolution of the PCM synaptic devices, and the residual between the high-precision (floating-point) and low-precision updates is kept in the digital co-processor as an initial value in the accumulation of the next weight update. By applying updates only once per batch to the memory elements, with the more granular per-example update computations done in the digital domain, the effects of device nonlinearity and stochasticity can be alleviated. However, in offloading a large fraction of the computational burden to a separate digital co-processor, the highly energy-efficient analog computation contributes a diminished share of the total training energy cost.

Weight update asymmetry, which is significant in PCM and some ReRAM devices, is known to considerably degrade the accuracy of neural networks.[87,192] When combined with nonlinearity, asymmetry causes weights to decay toward zero after a succession of positive and negative updates are applied.[86] One solution to asymmetry is to apply updates of only a single polarity to a pair of devices: one device to increase the weight and another to decrease the weight. This can be done without additional area overhead since the same device pair that is used to implement real-valued weights (Fig. 15) can be used for this purpose. Occasionally, after enough updates, one or both devices may saturate or enter a highly nonlinear regime; thus, the device conductances must be periodically read, reset back to an initial conductance state, and then restored to represent the correct weight.[134,193] This cost is amortized over many updates and may also be finely calibrated given prior knowledge about the average number of analog states in the asymmetric device.[194] Writing to only one device per update can also increase device resilience and decrease the number of refresh steps needed.[195] Asymmetry can also be partially compensated at the algorithm level by using different learning rates for updates of different signs[196] and by using adaptive neuron-specific learning rates.[197]

The "Tiki-Taka" method[198] mitigates the effect of asymmetric nonlinearities by splitting the weight matrix into a sum of two matrices: $\mathbf{W} = \mathbf{A} + \gamma\mathbf{C}$, each of which is mapped onto two crossbars as in Fig. 15. The crossbars for **A** are first calibrated to an operating point where $\mathbf{A} = 0$, and the devices have a symmetric update response. This is done by applying a sequence of alternating positive and negative pulses to every device until a steady-state conductance is reached. During training, parallel outer product updates are applied to **A**, and the weights in **A** are periodically transferred to **C**. Since the destructive effects of an asymmetric nonlinearity are usually accumulated over many updates, they can be largely eliminated by performing most of the updates accurately near the symmetric operating point (in **A**) while only sparsely updating the devices (in **C**) that operate in a more asymmetric regime.

VIII. MITIGATING ARRAY- AND DEVICE-LEVEL NON-IDEALITIES

The accelerators that we have considered face unique challenges compared to their digital counterparts, in part due to the analog nature of their computation and in part originating from their use of a dense array of memory elements. In spite of the intrinsic error resilience of neural networks, which are designed to generalize well to diverse and noisy real-world data, we have seen that the devices that make up these neuromorphic accelerators must meet a demanding set of requirements. This is made more challenging by the fact that most of the non-volatile memory devices that are well suited for this application are still emerging technologies. In this section, we briefly survey some techniques—in addition to those previously discussed—mainly at the architecture and algorithm level to mitigate the limitations of memory arrays and the individual memory devices on neural network performance.

A. Parasitic resistance

As discussed in Sec. IV C, line resistances along the rows and columns impose a limit on the maximum feasible size of a crossbar. This is an important limitation, as it forces a large weight matrix to be partitioned across multiple crossbars and increases the area and energy overhead of the peripheral circuits relative to that of the more efficient crossbars. The penalty for parasitic resistance, which reduces the fidelity of array reads and writes, comes mainly at the expense of neural network accuracy. At the circuit level, this issue can be partially addressed by using a 1T1R array in which the VMM inputs are applied to the gate of an access transistor as a bit-sliced or temporally encoded signal. Since the gate terminal has a large resistance, the effects of parasitic voltage drops can be minimized on the communication of the input activations but not of the partial sums.

At the technology level, large arrays can be enabled by increasing the interconnect width, which reduces the line resistance or by decreasing the conductance of the memory device, which reduces the voltage drops. The latter can be achieved using floating-gate synapses in subthreshold mode, which have conductances on the order of 1 to 10 nS[105] but require a relatively large voltage to program. Another pathway is to engineer emerging memories with low conductance. ReRAM devices with a conductance as small as 10 nS (and less than 10 μA of write current) have been demonstrated by inserting an oxide tunneling barrier into the material stack, but they also possess greater susceptibility to random telegraph noise.[199] Conductances smaller than 100 nS have also been demonstrated using redox transistors with a highly diluted conductive polymer.[72]

Several methods have been proposed to mitigate the neural network accuracy loss that arises from voltage drops across the array—this degradation effect is quantified in Ref. 200. Reference 65 calibrates for the predictable effects of the voltage drops using a nonlinear mapping of the weight matrix to the memory array, found via circuit simulations. Reference 201 adds series resistances to the periphery of the array to equalize the parasitic voltage drops seen by all parts of the crossbar. Reference 200 uses a lumped circuit model of the crossbar to show that the ideal crossbar currents (without parasitics) can be approximately recovered from the real current using an analytical expression, provided that the line resistances can be estimated or measured. Reference 202 shows that a neural network can learn around the parasitic voltage drops by modeling their effect as an injection of Gaussian noise on the VMM results during training. The mean and variance of this noise source are extracted from circuit simulations of an uncompensated crossbar that implements the desired neural network.

B. Handling sparse neural networks

Weight pruning is often used to make neural networks more compact and reduce their computational overhead. Because removing a large fraction of the neuron connections results in a relatively sparse and irregular weight matrix, the resulting network does not map in an area-efficient way onto a memory crossbar—this can be considered a limitation of the crossbar's rigid connectivity pattern. Low utilization of this connectivity implies that the energy consumption of the peripheral circuits is amortized over fewer MAC operations. As one solution, the neural network can be adapted during the pruning process to an eventual crossbar implementation by clustering the nonzero elements of a sparse matrix into blocks that can then be mapped onto multiple smaller crossbars; the zero weights no longer need to lie at the array crosspoints, improving the energy efficiency and potentially the area. Reference 203 proposes an iterative training scheme with a clustered pruning step to achieve this type of matrix compression. Reference 204 obtains a similar outcome, but starting with a sparse neural network, by re-arranging the matrix columns using *k*-means clustering, pruning elements that still remain outside of fixed-size blocks, and re-training the network. They show that at least on relatively small ReRAM crossbars (such as 32×32), their algorithm can largely retain the baseline accuracy of several large-scale neural networks and achieve a several-fold reduction in energy.

For CNNs, an alternative to weight pruning is filter pruning, where entire convolutional filters and their associated feature maps and filters in the subsequent layer are removed.[205] This method has been shown to compress large image recognition networks by more than 30% with insignificant accuracy loss and is amenable to a crossbar implementation as it implies only the removal of entire rows and columns.

C. Hardware robustness to noise, drift, and device failures

Noise and cycle-to-cycle variability in the readout currents of the individual memory elements can cause random errors in a VMM computation, while endurance failures, manufacturing defects, and programming errors can lead to persistent errors.[164] The nature and severity of these issues is technology-dependent: ReRAM devices, for example, suffer from random telegraph noise and can become stuck at high or low resistance after sufficiently many write cycles.

Reference 164 proposes an arithmetic error-correcting code for resistive crossbars. Unlike conventional error correcting codes that detect and correct individual bit flips, arithmetic codes correct for additive syndromes, which are more likely to arise in a crossbar where multiplications and summations are carried out on analog signals. Figure 22 shows the proposed error correction unit as an added component inside an ISAAC IMA.[68] The first divide/residual unit decodes the VMM output and calculates a residual that can be used to index a correction table; the overhead of this table is minimized by considering only errors that are highly probable and highly significant (i.e., affecting only devices holding the MSBs of the weight). The second divide/residual unit detects but does not correct any errors that still remain. This error correction scheme was shown to substantially reduce the misclassification rate on large machine vision tasks (5% on ImageNet using AlexNet) with only a small area, power, and latency overhead.

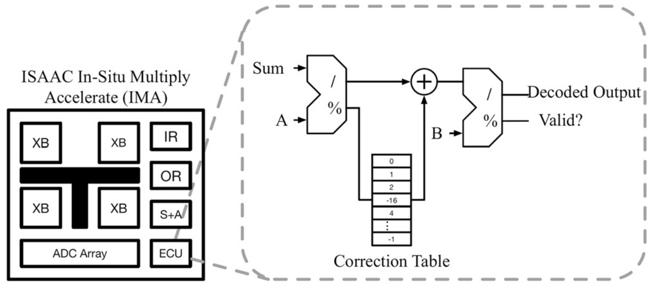

FIG. 22. Error correction unit described in Ref. 164, shown as an added component to the ISAAC accelerator.[68] Reproduced with permission from Feinberg *et al.*, in IEEE International Symposium on High Performance Computer Architecture (HPCA) (2018). Copyright 2018 IEEE.[164]

Resilience to device-level errors can also be provided through redundancy. Reference 88 considers duplicating the same weight multiple times within a crossbar to average out the effects of variability and noise at the column output. Reference 206 addresses stuck-at faults by providing redundant crossbars and redundant columns within a crossbar. Reference 207 represents a weight using the total conductance of multiple PCM devices and cycles among the devices during programming; while this only linearly increases the available number of conductance levels, it also linearly reduces the endurance requirements on each device.

To combat conductance drift, Ref. 140 monitors and periodically re-adjusts the memristor conductances; this approach introduces significant energy overhead to an inference accelerator and requires the movement of data from a host CPU that stores a redundant copy of the weights. If the drift behavior of the average device is known in advance, its effect has been shown to be partially compensated by applying an appropriate slope correction schedule to the activation functions, making them steeper over time. The effectiveness of the slope correction technique decreases as the device-to-device variability around the average case increases.[208]

Errors that are likely to arise from process variations or defects can be suppressed by an appropriate re-mapping of the neural network weights to the hardware. Reference 209 proposes a scheme to assign the rows of the weight matrix to the crossbar rows in a way that minimizes the expected deviations on the column outputs due to cycle-to-cycle variability. Then, during a re-training phase, weights that remain particularly prone to errors are frozen and reduced to zero. Reference 210 uses a similar strategy of re-assigning matrix rows to crossbar rows to mitigate errors due to stuck-at-0 and stuck-at-1 faults, whose distribution is profiled periodically during online training. Both approaches rely on an optimization algorithm to find the best mapping.

D. Device-aware neural network training

The impact of device non-idealities on inference performance can often be compensated by anticipating their effects during the off-chip training of the neural network. Reference 211 found that injection of Gaussian noise into the synaptic weights during training can compensate for a proportional degree of read stochasticity during inference. Reference 212, which focuses on charge-trap memory, showed that noise injection during training can also be employed to improve resilience to randomly variable rates of weight relaxation due to charge leakage. References 183 and 213 have meanwhile found that a device-realistic noise model considerably outperforms more standard forms of noise (such as uniform or Gaussian) when applied to the VMM outputs during training. Reference 213 further notes that noise robustness can be increased by mapping the dynamic range of the memory devices to a weight range that is smaller than that used by a given layer; weights lying outside of this range are clipped. This reduces the amount of algorithmic noise introduced by a given magnitude of physical noise but incurs an accuracy penalty due to clipping. The optimal clipping threshold can be learned for each layer during training, along with the weights.

Several works have investigated re-designing the topology of the neural network to better fit the characteristics of imperfect devices: in particular, equivalent performance may be possible with better energy efficiency using simpler networks. Reference 214 shows that in the presence of noise and device nonlinearity, replacing the first layer of learned weights with fixed, random weights (and a simpler learning rule) can achieve equivalent or superior generalization performance to MLPs trained through backpropagation. A layer of fixed random weights can potentially also be used during backpropagation, in place of the original weight matrix **W** in Eq. (2), which would reduce the number of crossbars that need to receive regular weight updates.[215] Early work suggests this may work in hardware neural network contexts,[216] but more extensive analysis will be required to test whether this approach is robust to noise and process variations.

More accurate and robust device-aware training frameworks can be built from measurements on devices that are exposed to a large number of incremental update pulses. These measurements can be collected into a device-specific lookup table that provides a probability distribution function for the update ΔG for a given initial conductance state G and queried update ΔG_{ideal}.[62] In principle, this method can be used to model effects that are not easily captured by analytic or circuit models and, in some cases, not explicitly identified at all. The effects of nonlinearity and device-to-device variability on training have been simulated using these lookup tables.[62,85,217] Open-source software tools such as CrossSim[218] and NeuroSim[219] have been developed, which allow the integration of device models to perform array-level neural network inference and training simulations. CrossSim supports the use of experimentally derived probabilistic lookup tables, while NeuroSim uses device behavioral models with parameters extracted from experiments.

IX. CONCLUSION

By performing computations locally within memory, the analog neuromorphic accelerator relieves part of the data transfer bottleneck that exists in any digital processor. However, these systems—which are generally mixed-signal rather than fully analog—are not without their own unique challenges. Without a careful design of the system architecture, the intrinsic advantages of crossbar computation can be swamped by the overhead (in power, area, and latency) incurred by the peripheral circuitry that supports it. A main goal in designing these architectures is to keep this overhead to a minimum without sacrificing performance.

The greatest energy and area penalty that is paid in an analog accelerator is the conversion between the analog and digital domains. Analog signals are needed to perform crossbar computations, while digital is needed for internal and external routing. Different architectures place the analog/digital divide at different points in the system: some remain predominantly digital by using bit-sliced inputs and weights, some are predominantly analog and preserve the digital domain solely for routing, while others are almost fully analog, with digitization only at the interface with off-chip digital processing. Some encode information in the time domain to escape the limitations of analog and avoid the overhead of a conventional ADC—even in these architectures, however, the cost of signal conversion has to be paid.

A primary trade-off that needs to be made in the design of an inference or training accelerator is that between energy efficiency and precision. Full digital (or software-equivalent) precision is possible in an analog system but comes at the cost of greater ADC and/or communication overhead due to bit slicing. Nonetheless, there are various ways to reduce this overhead, such as by encoding the weights to reduce ADC precision or by an optimal spatial arrangement of crossbars and their peripheral circuitry to exploit data locality. In accelerators that do not keep full precision, the main design challenge is to choose the correct quantization resolution while maintaining high neural network accuracy.

Training accelerators demand more precision, and extracting the requisite degree of precision while exploiting crossbar parallelism requires sophisticated methods of mapping real-valued weights to a set of limited-accuracy emerging memory devices. Handling batch training is another major challenge. In analog accelerators, it is the batch-1 update that makes the most use of crossbar-level parallelism, but as in digital hardware, it breaks the architecture-level parallelism that is enabled by pipelining. This unhappy trade-off suggests a pressing need for batch-capable learning schemes that can also use pipelined updates.

Analog accelerators also face unique challenges that arise from the crossbar geometry and from the individual memory devices that constitute the crossbar. Non-ideal physical properties of the devices can be compensated using architecture-level techniques, such as error correction, periodic carry, occasional reset, and the Tiki-Taka method. Another general approach that is gaining popularity is to learn around these non-idealities by designing the neural network training algorithm with the specific devices or array/circuit effects in mind. This powerful methodology is enabled and accelerated by open-source, physically realistic crossbar modeling tools.

Two key stumbling blocks must be overcome before the full benefits of analog acceleration can be realized: (1) peripheral circuit energy consumption, area, and latency must be brought to parity or below the costs for the crossbar cores, and (2) high neuromorphic performance must be realized with more efficient co-design or re-design of modern emerging device technologies. To a frustrating extent, one or the other of these goals are often achieved in modern academic demonstrations or accelerator designs, but not both. Thankfully, as device engineers converge toward nanoscale ideal programmable resistors, a cascading set of benefits will occur throughout the design hierarchy, fulfilling (2). Meanwhile, lower device conductance and even steeper access devices can potentially increase the maximum feasible crossbar size, which can assist with (1). Finally, a co-mingling of research efforts across the design hierarchy may suggest entirely new designs that leap over both of these barriers. By better highlighting the system-level issues that break the implementation of traditional workhorse artificial intelligence algorithms, we hope to have inspired efforts in this direction.

ACKNOWLEDGMENTS

T. P. Xiao, C. H. Bennett, B. Feinberg, S. Agarwal, and M. J. Marinella acknowledge support from Sandia's Laboratory-Directed Research and Development program. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract No. DE-NA0003525.

DATA AVAILABILITY

The data that support the findings of this study are available within the article.

REFERENCES

[1]N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," Commun. ACM **61**, 50–59 (2018).
[2]A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, and B. Catanzaro, "Deep learning with COTS HPC systems," in Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML'13) (2013), Vol. 28, pp. III-1337–III-1345.
[3]V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proc. IEEE **105**, 2295–2329 (2017).
[4]A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in IEEE High Performance Extreme Computing Conference (HPEC) (2019).
[5]W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," SIGARCH Comput. Archit. News **23**, 20–24 (1995).
[6]H. Tsai, S. Ambrogio, P. Narayanan, R. M. Shelby, and G. W. Burr, "Recent progress in analog memory-based accelerators for deep learning," J. Phys. D **51**, 283001 (2018).
[7]G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, L. L. Sanches, I. Boybat, M. L. Gallo, K. Moon, J. Woo, H. Hwang, and Y. Leblebici, "Neuromorphic computing using non-volatile memory," Adv. Phys.: X **2**, 89–124 (2017).
[8]W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing," Proc. IEEE **107**, 108–122 (2019).
[9]J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," Nat. Nanotechnol. **8**, 13 (2013).
[10]Z. Sun, G. Pedretti, E. Ambrosi, A. Bricalli, W. Wang, and D. Ielmini, "Solving matrix equations in one step with cross-point resistive arrays," Proc. Natl. Acad. Sci. **116**, 4123–4128 (2019).
[11]I. Richter, K. Pas, X. Guo, R. Patel, J. Liu, E. Ipek, and E. G. Friedman, "Memristive accelerator for extreme scale linear solvers," in Government Microcircuit Applications and Critical Technology Conference (GOMACTech) (2015).
[12]M. N. Bojnordi and E. Ipek, "Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in IEEE International Symposium on High Performance Computer Architecture (HPCA) (2016), pp. 1–13.
[13]S. Kumar, J. P. Strachan, and R. S. Williams, "Chaotic dynamics in nanoscale NbO_2 Mott memristors for analogue computing," Nature **548**, 318–321 (2017).
[14]S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," Proc. IEEE **106**, 260–285 (2018).
[15]S. Mittal, "A survey of ReRAM-based architectures for processing-in-memory and neural networks," Mach. Learn. Knowl. Extr. **1**, 75–114 (2018).
[16]Y. Zhang, Z. Wang, J. Zhu, Y. Yang, M. Rao, W. Song, Y. Zhuo, X. Zhang, M. Cui, L. Shen, R. Huang, and J. J. Yang, "Brain-inspired computing with memristors: Challenges in devices, circuits, and systems," Appl. Phys. Rev. **7**, 011308 (2020).
[17]I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).
[18]A. Ng, https://www.coursera.org/learn/machine-learning for "Machine learning;" accessed 12 August 2019.
[19]J. Nocedal and S. Wright, *Numerical Optimization* (Springer Science & Business Media, 2006).
[20]L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS'07)* (Curran Associates, Inc., 2007), pp. 161–168.
[21]S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput. **9**, 1735–1780 (1997).
[22]N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-L. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson,

B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)* (ACM, New York, 2017), pp. 1–12.

[23]Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," Proc. IEEE **86**, 2278–2324 (1998).

[24]Y. LeCun and C. Cortes, http://yann.lecun.com/exdb/mnist/ for "MNIST handwritten digit database;" accessed 7 December 2019.

[25]O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," Int. J. Comput. Vision **115**, 211–252 (2015).

[26]X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," Nat. Electron. **1**, 216–222 (2018).

[27]S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," arXiv:1510.00149 (2015).

[28]M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)* (MIT Press, Cambridge, MA, 2015), Vol. 2, pp. 3123–3131.

[29]M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," arXiv:1603.05279 (2016).

[30]M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or -1," arXiv:1602.02830 (2016).

[31]W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17)* (AAAI Press, 2017), pp. 2625–2631.

[32]S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," arXiv:1502.02551 (2015).

[33]A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv:1704.04861 (2017).

[34]Y. Chen, T.-J. Yang, J. Emer, and V. Sze, "Understanding the limitations of existing energy-efficient design approaches for deep neural networks," Energy **2**, L3 (2018); available at https://scholar.google.com/scholar?cluster=9880902826603509712&hl=en&as_sdt=0,32&sciodt=0,32.

[35]R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning* (ACM, 2009), pp. 873–880.

[36]S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient primitives for deep learning," arXiv:1410.0759 (2014).

[37]C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in International Conference on Field Programmable Logic and Applications (2009), pp. 32–37.

[38]C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in CVPR Workshops (2011), pp. 109–116.

[39]C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15)* (ACM, New York, 2015), pp. 161–170.

[40]S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)* (ACM, New York, 2010), pp. 247–257.

[41]X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in 54th ACM/EDAC/IEEE Design Automation Conference (DAC) (2017), pp. 1–6.

[42]A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," IEEE Micro **35**, 10–22 (2015).

[43]E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. E. Husseini, T. Juhasz, K. Kagi, R. K. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Y. Xiao, D. Zhang, R. Zhao, and D. Burger, "Serving DNNs in real time at datacenter scale with Project Brainwave," IEEE Micro **38**, 8–20 (2018).

[44]Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in 47th Annual IEEE/ACM International Symposium on Microarchitecture (2014), pp. 609–622.

[45]D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) (2016), pp. 380–392.

[46]T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)* (ACM, New York, 2014), pp. 269–284.

[47]Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) (2016), pp. 367–379.

[48]S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)* (IEEE Press, Piscataway, NJ, 2016), pp. 243–254.

[49]B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) (2016), pp. 267–278.

[50]R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights," in IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (2016), pp. 236–241.

[51]J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in IEEE International Solid-State Circuits Conference (ISSCC) (2018), pp. 218–220.

[52]K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," IEEE J. Solid-State Circuits **53**, 983–994 (2018).

[53]S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghunathan, "SCALEDEEP: A scalable compute architecture for learning and evaluating deep networks," in ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA) (2017), pp. 13–26.

[54]S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)* (IEEE Press, Piscataway, NJ, 2016), pp. 393–405.

[55]V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17)* (ACM, New York, 2017), pp. 273–287.

[56]S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *Proceedings of the 50th*

Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17) (ACM, New York, 2017), pp. 288–301.
[57]C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural Cache: Bit-serial in-cache acceleration of deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18)* (IEEE Press, Piscataway, NJ, 2018), pp. 383–396.
[58]J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6T SRAM array," in IEEE Symposium on VLSI Circuits (VLSI-Circuits) (2016), pp. 1–2.
[59]H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A mixed-signal binarized convolutional-neural-network accelerator integrating dense weight storage and multiplication for reduced data movement," in *IEEE Symposium on VLSI Circuits* (IEEE, 2018), pp. 141–142.
[60]S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in 53rd ACM/EDAC/IEEE Design Automation Conference (DAC) (2016), pp. 1–6.
[61]X. Yin, X. Chen, M. Niemier, and X. S. Hu, "Ferroelectric FETs-based nonvolatile logic-in-memory circuits," IEEE Trans. Very Large Scale Integration (VLSI) Syst. **27**, 159–172 (2019).
[62]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 J. Emerging Sel. Top. Circuits Syst. **8**, 86–101 (2018).
[63]S. Agarwal, T.-T. Quach, O. Parekh, A. H. Hsia, E. P. DeBenedictis, C. D. James, M. J. Marinella, and J. B. Aimone, "Energy scaling advantages of resistive memory crossbar based computation and its application to sparse coding," Front. Neurosci. **9**, 484 (2016).
[64]B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "RRAM-based analog approximate computing," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **34**, 1905–1917 (2015).
[65]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.
[66]T. Gokmen, M. J. Rasch, and W. Haensch, "Training LSTM networks with resistive cross-point devices," Front. Neurosci. **12**, 745 (2018).
[67]H. Tsai, S. Ambrogio, C. Mackin, P. Narayanan, R. Shelby, K. Rocki, A. Chen, and G. Burr, "Inference of long-short term memory networks at software-equivalent accuracy using 2.5M analog phase change memory devices," in *Symposium on VLSI Technology* (IEEE, 2019), pp. T82–T83.
[68]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 (ISCA '16)* (IEEE Press, Piscataway, NJ, 2016), pp. 14–26.
[69]G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory," IBM J. Res. Dev. **52**, 449–464 (2008).
[70]C.-S. Yang, D.-S. Shang, N. Liu, E. J. Fuller, S. Agrawal, A. A. Talin, Y.-Q. Li, B.-G. Shen, and Y. Sun, "All-solid-state synaptic transistor with ultralow conductance for neuromorphic computing," Adv. Funct. Mater. **28**, 1804170 (2018).
[71]T. Yang and V. Sze, "Design considerations for efficient deep neural networks on processing-in-memory accelerators," in IEEE International Electron Devices Meeting (IEDM) (2019), pp. 22.1.1–22.1.4.
[72]E. J. Fuller, S. T. Keene, A. Melianas, Z. Wang, S. Agarwal, Y. Li, Y. Tuchman, C. D. James, M. J. Marinella, J. J. Yang, A. Salleo, and A. A. Talin, "Parallel programming of an ionic floating-gate memory array for scalable neuromorphic computing," Science **364**, 570–574 (2019).
[73]A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," Solid-State Electron. **125**, 25–38 (2016), extended papers selected from ESSDERC 2015.
[74]Y. Long, T. Na, P. Rastogi, K. Rao, A. I. Khan, S. Yalamanchili, and S. Mukhopadhyay, "A Ferroelectric FET based power-efficient architecture for data-intensive computing," in IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2018), pp. 1–8.
[75]C. Hsu, I. Wang, C. Lo, M. Chiang, W. Jang, C. Lin, and T. Hou, "Self-rectifying bipolar TaO_x/TiO_2 RRAM with superior endurance over 10^{12} cycles for 3D high-density storage-class memory," in Symposium on VLSI Technology (2013), pp. T166–T167.
[76]S. W. Fong, C. M. Neumann, and H.-S. P. Wong, "Phase-change memory: Towards a storage-class memory," IEEE Trans. Electron Devices **64**, 4374–4385 (2017).
[77]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 **27**, 365202 (2016).
[78]A. Sebastian, M. L. Gallo, and E. Eleftheriou, "Computational phase-change memory: Beyond von Neumann computing," J. Phys. D **52**, 443002 (2019).
[79]S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R. Wunderlich, S. Nease, and S. Ramakrishnan, "A programmable and configurable mixed-mode FPAA SoC," IEEE Trans. Very Large Scale Integration (VLSI) Syst. **24**, 2253–2261 (2016).
[80]F. M. Bayat, X. Guo, H. A. Om'mani, N. Do, K. K. Likharev, and D. B. Strukov, "Redesigning commercial floating-gate memory for analog computing applications," in IEEE International Symposium on Circuits and Systems (ISCAS) (2015), pp. 1921–1924.
[81]Y. van de Burgt, E. Lubberman, E. J. Fuller, S. T. Keene, G. C. Faria, S. Agarwal, M. J. Marinella, A. A. Talin, and A. Salleo, "A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing," Nat. Mater. **16**, 414 (2017).
[82]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 IEEE International Electron Devices Meeting (IEDM) (2017), pp. 6.2.1–6.2.4.
[83]Y. Wu, B. Lee, and H. P. Wong, "Al_2O_3-based RRAM using atomic layer deposition (ALD) with 1-μA reset current," IEEE Electron Device Lett. **31**, 1449–1451 (2010).
[84]W.-S. Khwa, D. Lu, C.-M. Dou, and M.-F. Chang, "Emerging NVM circuit techniques and implementations for energy-efficient systems," in *Beyond-CMOS Technologies for Next Generation Computer Design* (Springer, 2019), pp. 85–132.
[85]S. Agarwal, D. Garland, J. Niroula, R. B. Jacobs-Gedrim, A. Hsia, M. S. Van Heukelom, E. Fuller, B. Draper, and M. J. Marinella, "Using floating-gate memory to train ideal accuracy neural networks," IEEE J. Explor. Solid-State Comput. Devices Circuits **5**, 52–57 (2019).
[86]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 International Joint Conference on Neural Networks (IJCNN) (2016), pp. 929–938.
[87]T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," Front. Neurosci. **10**, 333 (2016).
[88]S. Yu, P. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in IEEE International Electron Devices Meeting (IEDM) (2015), pp. 17.3.1–17.3.4.
[89]H. P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai, "Metal-oxide RRAM," Proc. IEEE **100**, 1951–1970 (2012).
[90]J. Woo, K. Moon, J. Song, S. Lee, M. Kwak, J. Park, and H. Hwang, "Improved synaptic behavior under identical pulses using AlO_x/HfO_2 bilayer RRAM array for neuromorphic systems," IEEE Electron Device Lett. **37**, 994–997 (2016).
[91]J. Park, M. Kwak, K. Moon, J. Woo, D. Lee, and H. Hwang, "TiO_x-based RRAM synapse with 64-levels of conductance and symmetric conductance change by adopting a hybrid pulse scheme for neuromorphic computing," IEEE Electron Device Lett. **37**, 1559–1562 (2016).
[92]K. Moon, A. Fumarola, S. Sidler, J. Jang, P. Narayanan, R. M. Shelby, G. W. Burr, and H. Hwang, "Bidirectional non-filamentary RRAM as an analog neuromorphic synapse. Part I: $Al/Mo/Pr_{0.7}Ca_{0.3}MnO_3$ material improvements and device measurements," IEEE J. Electron Devices Soc. **6**, 146–155 (2018).

[93] I.-T. Wang, C.-C. Chang, L.-W. Chiu, T. Chou, and T.-H. Hou, "3D Ta/TaO$_x$/TiO$_2$/Ti synaptic array and linearity tuning of weight update for hardware neural network applications," Nanotechnology **27**, 365204 (2016).
[94] B. Chakrabarti, M. A. Lastras-Montaño, G. Adam, M. Prezioso, B. Hoskins, M. Payvand, A. Madhavan, A. Ghofrani, L. Theogarajan, K.-T. Cheng *et al.*, "A multiply-add engine with monolithically integrated 3D memristor crossbar/CMOS hybrid circuit," Sci. Rep. **7**, 42429 (2017).
[95] G. C. Adam, B. D. Hoskins, M. Prezioso, F. Merrikh-Bayat, B. Chakrabarti, and D. B. Strukov, "3-D memristor crossbars for analog and neuromorphic computing applications," IEEE Trans. Electron Devices **64**, 312–318 (2017).
[96] Z. Li, P. Chen, H. Xu, and S. Yu, "Design of ternary neural network with 3-D vertical RRAM array," IEEE Trans. Electron Devices **64**, 2721–2727 (2017).
[97] G. W. Burr, M. J. Brightsky, A. Sebastian, H.-Y. Cheng, J.-Y. Wu, S. Kim, N. E. Sosa, N. Papandreou, H.-L. Lung, H. Pozidis *et al.*, "Recent progress in phase-change memory technology," IEEE J. Emerging Sel. Top. Circuits Syst. **6**, 146–162 (2016).
[98] E. J. Fuller, F. E. Gabaly, F. Léonard, S. Agarwal, S. J. Plimpton, R. B. Jacobs-Gedrim, C. D. James, M. J. Marinella, and A. A. Talin, "Li-ion synaptic transistor for low power analog computing," Adv. Mater. **29**, 1604310 (2017).
[99] E. J. Fuller, Y. Li, C. Bennet, S. T. Keene, A. Melianas, S. Agarwal, M. J. Marinella, A. Salleo, and A. A. Talin, "Redox transistors for neuromorphic computing," IBM J. Res. Develop. **63**(9), 1–9:9 (2019).
[100] H. Mulaosmanovic, J. Ocker, S. Müller, M. Noack, J. Müller, P. Polakowski, T. Mikolajick, and S. Slesazeck, "Novel ferroelectric FET based synapse for neuromorphic systems," in Symposium on VLSI Technology (2017), pp. T176–T177.
[101] Y. Long, D. Kim, E. Lee, P. Saha, B. A. Mudassar, X. She, A. I. Khan, and S. Mukhopadhyay, "A ferroelectric FET based processing-in-memory architecture for DNN acceleration," IEEE J. Explor. Solid-State Comput. Devices Circuits **5**, 113–111 (2019).
[102] B. Obradovic, T. Rakshit, R. Hatcher, J. Kittl, R. Sengupta, J. G. Hong, and M. S. Rodder, "A multi-bit neuromorphic weight cell using ferroelectric FETs, suitable for SoC integration," IEEE J. Electron Devices Soc. **6**, 438–448 (2018).
[103] T. P. Ma and J.-P. Han, "Why is nonvolatile ferroelectric memory field-effect transistor still elusive?," IEEE Electron Device Lett. **23**, 386–388 (2002).
[104] S. Lequeux, J. Sampaio, V. Cros, K. Yakushiji, A. Fukushima, R. Matsumoto, H. Kubota, S. Yuasa, and J. Grollier, "A magnetic synapse: Multilevel spin-torque memristor with perpendicular anisotropy," Sci. Rep. **6**, 1–7 (2016).
[105] C. R. Schlottmann and P. E. Hasler, "A highly dense, low power, programmable analog vector-matrix multiplier: The FPAA implementation," IEEE J. Emerging Sel. Top. Circuits Syst. **1**, 403–411 (2011).
[106] 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 IEEE International Electron Devices Meeting (IEDM) (2017), pp. 6.5.1–6.5.4.
[107] M. R. Mahmoodi and D. B. Strukov, "Mixed-signal POp/j computing with nonvolatile memories," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI '18)* (ACM, New York, 2018), pp. 513–513.
[108] C. Diorio, P. Hasler, A. Minch, and C. A. Mead, "A single-transistor silicon synapse," IEEE Trans. Electron Devices **43**, 1972–1980 (1996).
[109] P. C. Y. Chen, "Threshold-alterable Si-gate MOS devices," IEEE Trans. Electron Devices **24**, 584–586 (1977).
[110] Y. J. Park, H. T. Kwon, B. Kim, W. J. Lee, D. H. Wee, H. Choi, B. Park, J. Lee, and Y. Kim, "3-D stacked synapse array based on charge-trap flash memory for implementation of deep neural networks," IEEE Trans. Electron Devices **66**, 420–427 (2019).
[111] P. Wang, F. Xu, B. Wang, B. Gao, H. Wu, H. Qian, and S. Yu, "Three-dimensional NAND flash for vector-matrix multiplication," IEEE Trans. Very Large Scale Integration (VLSI) Syst. **27**, 988–991 (2019).
[112] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. Hasler, "A 531 nW/MHz, 128 × 32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity," in *Proceedings of the IEEE Custom Integrated Circuits Conference, Cat. No. 04CH37571* (IEEE, 2004), pp. 651–654.
[113] S. Ramakrishnan and J. Hasler, "Vector-matrix multiply and winner-take-all as an analog classifier," IEEE Trans. Very Large Scale Integration (VLSI) Syst. **22**, 353–361 (2014).
[114] L. Fick, D. Blaauw, D. Sylvester, S. Skrzyniarz, M. Parikh, and D. Fick, "Analog in-memory subthreshold deep neural network accelerator," in IEEE Custom Integrated Circuits Conference (CICC) (2017), pp. 1–4.
[115] F. Merrikh-Bayat, X. Guo, M. Klachko, M. Prezioso, K. K. Likharev, and D. B. Strukov, "High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays," IEEE Trans. Neural Networks Learn. Syst. **29**, 4782–4790 (2018).
[116] J. Hasler and H. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," Front. Neurosci. **7**, 118 (2013).
[117] M. Bavandpour, S. Sahay, M. R. Mahmoodi, and D. B. Strukov, "3D-aCortex: An ultra-compact energy-efficient neurocomputing platform based on commercial 3D-NAND flash memories," arXiv:1908.02472 (2019).
[118] B. E. Boser, E. Sackinger, J. Bromley, Y. L. Cun, and L. D. Jackel, "An analog neural network processor with programmable topology," IEEE J. Solid-State Circuits **26**, 2017–2025 (1991).
[119] R. Genov and G. Cauwenberghs, "Charge-mode parallel architecture for vector-matrix multiplication," IEEE Trans. Circuits Syst. II **48**, 930–936 (2001).
[120] F. J. Kub, K. K. Moon, I. A. Mack, and F. M. Long, "Programmable analog vector-matrix multipliers," IEEE J. Solid-State Circuits **25**, 207–214 (1990).
[121] S. Kim, T. Gokmen, H.-M. Lee, and W. E. Haensch, "Analog CMOS-based resistive processing unit for deep neural network training," IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS) (2017).
[122] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," IEEE J. Solid-State Circuits **54**, 158–172 (2019).
[123] E. H. Lee and S. S. Wong, "24.2A 2.5GHz 7.7TOPS/W switched-capacitor matrix multiplier with co-designed local memory in 40 nm," in IEEE International Solid-State Circuits Conference (ISSCC) (2016), pp. 418–419.
[124] Q. Luo, X. Xu, H. Liu, H. Lv, T. Gong, S. Long, Q. Liu, H. Sun, W. Banerjee, L. Li *et al.*, "Super non-linear RRAM with ultra-low power for 3D vertical nano-crossbar arrays," Nanoscale **8**, 15629–15636 (2016).
[125] R. Midya, Z. Wang, J. Zhang, S. E. Savel'ev, C. Li, M. Rao, M. H. Jang, S. Joshi, H. Jiang, P. Lin *et al.*, "Anatomy of Ag/Hafnia-based selectors with 1010 non-linearity," Adv. Mater. **29**, 1604457 (2017).
[126] G. W. Burr, R. S. Shenoy, K. Virwani, P. Narayanan, A. Padilla, B. Kurdi, and H. Hwang, "Access devices for 3D crosspoint memory," J. Vac. Sci. Technol. B **32**, 040802 (2014).
[127] 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 **521**, 61 (2015).
[128] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," Nat. Commun. **9**, 2385 (2018).
[129] M. Hu, C. E. 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," Adv. Mater. **30**, 1705914 (2018).
[130] F. M. Bayat, M. Prezioso, B. Chakrabarti, H. Nili, I. Kataeva, and D. Strukov, "Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits," Nat. Commun. **9**, 2331 (2018).
[131] F. Cai, J. M. Correll, S. H. Lee, Y. Lim, V. Bothra, Z. Zhang, M. P. Flynn, and W. D. Lu, "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," Nat. Electron. **2**, 290–299 (2019).
[132] P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, H.-S. P. Wong *et al.*, "Face classification using electronic synapses," Nat. Commun. **8**, 15199 (2017).
[133] S. Yu, Z. Li, P. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in IEEE International Electron Devices Meeting (IEDM) (2016), pp. 16.2.1–16.2.4.
[134] 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, B. N. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," IEEE Trans. Electron Devices **62**, 3498–3507 (2015).

[135]R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "RedEye: Analog convnet image sensor architecture for continuous mobile vision," in *ACM SIGARCH Computer Architecture News* (IEEE Press, 2016), Vol. 44, pp. 255–266.
[136]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 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) (2016), pp. 27–39.
[137]M. Bavandpour, M. R. Mahmoodi, and D. B. Strukov, "Energy-efficient time-domain vector-by-matrix multiplier for neurocomputing and beyond," IEEE Trans. Circuits Syst. II **66**, 1512–1516 (2019).
[138]J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in ACM International Conference on Supercomputing 25th Anniversary Volume (2006), pp. 390–401.
[139]X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, and J. Yang, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in 52nd ACM/EDAC/IEEE Design Automation Conference (DAC) (2015), pp. 1–6.
[140]X. Liu, M. Mao, B. Liu, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, J. Yang, H. Li, and Y. Chen, "Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators," IEEE Trans. Circuits Syst. I **63**, 617–628 (2016).
[141]S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," Nature **558**, 60 (2018).
[142]M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of BSB recall function using memristor crossbar arrays," in DAC Design Automation Conference (2012), pp. 498–503.
[143]M. R. Mahmoodi and D. Strukov, "An ultra-low energy internally analog, externally digital vector-matrix multiplier based on NOR flash memory technology," in 55th ACM/ESDA/IEEE Design Automation Conference (DAC) (2018), pp. 1–6.
[144]D. Soudry, D. D. Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," IEEE Trans. Neural Networks Learn. Syst. **26**, 2408–2421 (2015).
[145]P. Narayanan, L. L. Sanches, A. Fumarola, R. M. Shelby, S. Ambrogio, J. Jang, H. Hwang, Y. Leblebici, and G. W. Burr, "Reducing circuit design complexity for neuromorphic machine learning systems based on non-volatile memory arrays," in IEEE International Symposium on Circuits and Systems (ISCAS) (2017), pp. 1–4.
[146]M. Bavandpour, S. Sahay, M. R. Mahmoodi, and D. Strukov, "Efficient mixed-signal neurocomputing via successive integration and rescaling," IEEE Trans. Very Large Scale Integration (VLSI) Syst. **28**, 823–827 (2020).
[147]B. Gordon, "Linear electronic analog/digital conversion architectures, their origins, parameters, limitations, and applications," IEEE Trans. Circuits Syst. **25**, 391–418 (1978).
[148]A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-M. W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)* (ACM, New York, 2019), pp. 715–731.
[149]R. Genov and G. Cauwenberghs, "Kerneltron: Support vector "machine" in silicon," IEEE Trans. Neural Networks **14**, 1426–1434 (2003).
[150]J. Hasler, "Analog architecture complexity theory empowering ultra-low power configurable analog and mixed mode soc systems," J. Low Power Electron. Appl. **9**, 4 (2019).
[151]P. Figueiredo, "Recent advances and trends in high-performance embedded data converters," in *High-Performance AD and DA Converters, IC Design in Scaled Technologies, and Time-Domain Signal Processing* (Springer, 2015), pp. 85–142.
[152]L. Kull, D. Luu, C. Menolfi, M. Braendli, P. A. Francese, T. Morf, M. Kossel, H. Yueksel, A. Cevrero, I. Ozkaya *et al.*, "28.5 A 10b 1.5 GS/s pipelined-SAR ADC with background second-stage common-mode regulation and offset calibration in 14 nm CMOS FinFET," in *IEEE International Solid-State Circuits Conference (ISSCC)* (IEEE, 2017), pp. 474–475.
[153]A. Nag, R. Balasubramonian, V. Srikumar, R. Walker, A. Shafiee, J. Strachan, and N. Muralimanohar, "Newton: Gravitating towards the physical limits of crossbar acceleration," IEEE Micro **38**, 41–49 (2018).
[154]M. Saberi, R. Lotfi, K. Mafinezhad, and W. A. Serdijn, "Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs," IEEE Trans. Circuits Syst. I **58**, 1736–1748 (2011).
[155]M. Giordano, G. Cristiano, K. Ishibashi, S. Ambrogio, H. Tsai, G. W. Burr, and P. Narayanan, "Analog-to-digital conversion with reconfigurable function mapping for neural networks activation function acceleration," IEEE J. Emerging Sel. Top. Circuits Syst. **9**, 367–376 (2019).
[156]M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "Time: A training-in-memory architecture for memristor-based deep neural networks," in 54th ACM/EDAC/IEEE Design Automation Conference (DAC) (2017), pp. 1–6.
[157]X. Sun, S. Yin, X. Peng, R. Liu, J. Seo, and S. Yu, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in Design, Automation Test in Europe Conference Exhibition (DATE) (2018), pp. 1423–1428.
[158]L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, "Switched by input: Power efficient structure for RRAM-based convolutional neural network," in 53nd ACM/EDAC/IEEE Design Automation Conference (DAC) (2016), pp. 1–6.
[159]M. Santos, N. Horta, and J. Guilherme, "A survey on nonlinear analog-to-digital converters," Integr. VLSI J. **47**, 12–22 (2014).
[160]E. Rosenthal, S. Greshnikov, D. Soudry, and S. Kvatinsky, "A fully analog memristor-based neural network with online gradient training," in IEEE International Symposium on Circuits and Systems (ISCAS) (2016), pp. 1394–1397.
[161]G. Khodabandehloo, M. Mirhassani, and M. Ahmadi, "Analog implementation of a novel resistive-type sigmoidal neuron," IEEE Trans. Very Large Scale Integration (VLSI) Syst. **20**, 750–754 (2012).
[162]F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," Nanotechnology **23**, 075201 (2012).
[163]B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA) (2018), pp. 367–382.
[164]B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," in IEEE International Symposium on High Performance Computer Architecture (HPCA) (2018), pp. 52–65.
[165]Y. Kim, H. Kim, D. Ahn, and J.-J. Kim, "Input-splitting of large neural networks for power-efficient accelerator with resistive crossbar memory array," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '18)* (Association for Computing Machinery, New York, 2018).
[166]L. Ni, H. Huang, Z. Liu, R. V. Joshi, and H. Yu, "Distributed in-memory computing on binary RRAM crossbar," J. Emerging Technol. Comput. Syst. **13**, 1–36:18 (2017).
[167]S. Yin, Y. Kim, X. Han, H. Barnaby, S. Yu, Y. Luo, W. He, X. Sun, J. Kim, and J. Seo, "Monolithically Integrated RRAM- and CMOS-based in-memory computing optimizations for efficient deep learning," EEE Micro **39**, 54–63 (2019).
[168]T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on RRAM," in 22nd Asia and South Pacific Design Automation Conference (ASP-DAC) (2017), pp. 782–787.
[169]L. Ni, Z. Liu, H. Yu, and R. V. Joshi, "An energy-efficient digital ReRAM-crossbar-based CNN with bitwise parallelism," IEEE J. Explor. Solid-State Comput. Devices Circuits **3**, 37–46 (2017).
[170]E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in International Conference on Field-Programmable Technology (FPT) (2016), pp. 77–84.
[171]F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," Nat. Commun. **4**, 2072 (2013).

[172]S. N. Truong and K.-S. Min, "New memristor-based crossbar array architecture with 50-% area reduction and 48-% power saving for matrix-vector multiplication of analog neuromorphic computing," J. Semicond. Technol. Sci. **14**, 356–363 (2014).
[173]Y. Zhang, X. Wang, and E. G. Friedman, "Memristor-based circuit design for multilayer neural networks," IEEE Trans. Circuits Syst. I **65**, 677–686 (2018).
[174]L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined reram-based accelerator for deep learning," in IEEE International Symposium on High Performance Computer Architecture (HPCA) (2017), pp. 541–552.
[175]See https://developer.nvidia.com/deep-learning-performance-training-inference for "NVIDIA Data Center Deep Learning Product Performance;" accessed 13 May 2020.
[176]See https://habana.ai/wp-content/uploads/2019/06/Goya-Datasheet-HL-10x.pdf for "Habana Labs Goya HL-1000–Inference card;" accessed 13 May 2020.
[177]K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016), pp. 770–778.
[178]Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," arXiv:1609.08144 (2016).
[179]V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "MLPerf inference benchmark," arXiv:1911.02549 (2019).
[180]P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," Nature **577**, 641–646 (2020).
[181]M. Collins, J. Hasler, and S. George, "An open-source tool set enabling analog-digital-software co-design," J. Low Power Electron. Appl. **6**, 3 (2016).
[182]M. A. Zidan, Y. Jeong, J. H. Shin, C. Du, Z. Zhang, and W. D. Lu, "Field-programmable crossbar array (FPCA) for reconfigurable computing," arXiv:1612.02913 (2016).
[183]I. Kataeva, F. Merrikh-Bayat, E. Zamanidoost, and D. Strukov, "Efficient training algorithms for neural networks based on memristive crossbar circuits," in International Joint Conference on Neural Networks (IJCNN) (2015), pp. 1–8.
[184]S. Choi, J. H. Shin, J. Lee, P. Sheridan, and W. D. Lu, "Experimental demonstration of feature extraction and dimensionality reduction using memristor networks," Nano Lett. **17**, 3113–3118 (2017).
[185]T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," Neural Networks **2**, 459–473 (1989).
[186]P. Narayanan, A. Fumarola, L. L. Sanches, K. Hosokawa, S. C. Lewis, R. M. Shelby, and G. W. Burr, "Toward on-chip acceleration of the backpropagation algorithm using nonvolatile memory," IBM J. Res. Develop. **61**, 11:1–11:11 (2017).
[187]L. Gao, I.-T. Wang, P.-Y. Chen, S. Vrudhula, J. sun Seo, Y. Cao, T.-H. Hou, and S. Yu, "Fully parallel write/read in resistive synaptic array for accelerating on-chip learning," Nanotechnology **26**, 455204 (2015).
[188]D. Kadetotad, Z. Xu, A. Mohanty, P. Chen, B. Lin, J. Ye, S. Vrudhula, S. Yu, Y. Cao, and J. Seo, "Parallel architecture with resistive crosspoint array for dictionary learning acceleration," IEEE J. Emerging Sel. Top. Circuits Syst. **5**, 194–204 (2015).
[189]B. D. Hoskins, M. W. Daniels, S. Huang, A. Madhavan, G. C. Adam, N. Zhitenev, J. J. McClelland, and M. D. Stiles, "Streaming batch eigenupdates for hardware neural networks," Front. Neurosci. **13**, 793 (2019).
[190]S. Agarwal, R. B. J. Gedrim, A. H. Hsia, D. R. Hughart, E. J. Fuller, A. A. Talin, C. D. James, S. J. Plimpton, and M. J. Marinella, "Achieving ideal accuracies in analog neuromorphic computing using periodic carry," in *Symposium on VLSI Technology* (IEEE, 2017), pp. T174–T175.
[191]S. R. Nandakumar, M. L. Gallo, C. Piveteau, V. Joshi, G. Mariani, I. Boybat, G. Karunaratne, R. Khaddam-Aljameh, U. Egger, A. Petropoulos, T. Antonakopoulos, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Mixed-precision deep learning based on computational memory," Front. Neurosci. **14**, 406 (2020).
[192]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 IEEE International Electron Devices Meeting (IEDM) (2015), pp. 4.4.1–4.4.4.
[193]M. Suri, O. Bichler, D. Querlioz, O. Cueto, L. Perniola, V. Sousa, D. Vuillaume, C. Gamrat, and B. DeSalvo, "Phase change memory as synapse for ultra-dense neuromorphic systems: Application to complex visual pattern extraction," in International Electron Devices Meeting (2011), pp. 4.4.1–4.4.4.
[194]O. Bichler, M. Suri, D. Querlioz, D. Vuillaume, B. DeSalvo, and C. Gamrat, "Visual pattern extraction using energy-efficient "2-PCM synapse" neuromorphic architecture," IEEE Trans. Electron Devices **59**, 2206–2214 (2012).
[195]Y.-P. Lin, C. H. Bennett, T. Cabaret, D. Vodenicarevic, D. Chabi, D. Querlioz, B. Jousselme, V. Derycke, and J.-O. Klein, "Physical realization of a supervised learning system built with organic memristive synapses," Sci. Rep. **6**, 31932 (2016).
[196]A. Fumarola, P. Narayanan, L. L. Sanches, S. Sidler, J. Jang, K. Moon, R. M. Shelby, H. Hwang, and G. W. Burr, "Accelerating machine learning with non-volatile memory: Exploring device and circuit tradeoffs," in IEEE International Conference on Rebooting Computing (ICRC) (2016), pp. 1–8.
[197]I. Boybat, C. di Nolfo, S. Ambrogio, M. Bodini, N. C. P. Farinha, R. M. Shelby, P. Narayanan, S. Sidler, H. Tsai, Y. Leblebici, and G. W. Burr, "Improved deep neural network hardware-accelerators based on non-volatile-memory: The local gains technique," in IEEE International Conference on Rebooting Computing (ICRC) (2017), pp. 1–8.
[198]T. Gokmen and W. Haensch, "Algorithm for training neural networks on resistive device arrays," Front. Neurosci. **14**, 103 (2020).
[199]R. B. Jacobs-Gedrim, S. Agarwal, R. S. Goeke, C. Smith, P. S. Finnegan, J. Niroula, D. R. Hughart, P. G. Kotula, C. D. James, and M. J. Marinella, "Analog high resistance bilayer RRAM device for hardware acceleration of neuromorphic computation," J. Appl. Phys. **124**, 202101 (2018).
[200]Y. Jeong, M. A. Zidan, and W. D. Lu, "Parasitic effect analysis in memristor-array-based neuromorphic systems," IEEE Trans. Nanotechnol. **17**, 184–193 (2018).
[201]S. Agarwal, R. L. Schiek, and M. J. Marinella, "Compensating for parasitic voltage drops in resistive memory arrays," in IEEE International Memory Workshop (IMW) (2017), pp. 1–4.
[202]Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Proceedings of the 56th Annual Design Automation Conference (DAC '19)* (ACM, New York, 2019), pp. 57:1–57:6.
[203]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 (ICCAD '17)* (IEEE Press, Piscataway, NJ, 2017), pp. 533–540.
[204]J. Lin, Z. Zhu, Y. Wang, and Y. Xie, "Learning the sparsity for ReRAM: Mapping and pruning sparse neural network for ReRAM based accelerator," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC '19)* (Association for Computing Machinery, New York, 2019), pp. 639–644.
[205]H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," arXiv:1608.08710 (2016).
[206]W. Huangfu, L. Xia, M. Cheng, X. Yin, T. Tang, B. Li, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Computation-oriented fault-tolerance schemes for RRAM computing systems," in 22nd Asia and South Pacific Design Automation Conference (ASP-DAC) (2017), pp. 794–799.
[207]I. Boybat, M. L. Gallo, S. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, and E. Eleftheriou, "Neuromorphic computing with multi-memristive synapses," Nat. Commun. **9**, 2514 (2018).
[208]S. Ambrogio, M. Gallot, K. Spoon, H. Tsai, C. Mackin, M. Wesson, S. Kariyappa, P. Narayanan, C. Liu, A. Kumar, A. Chen, and G. W. Burr, "Reducing the impact of phase-change memory conductance drift on the inference of large-scale hardware neural networks," in IEEE International Electron Devices Meeting (IEDM) (2019), pp. 6.1.1–6.1.4.
[209]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), pp. 19–24.
[210]L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in 54th ACM/EDAC/IEEE Design Automation Conference (DAC) (2017), pp. 1–6.
[211]V. Joshi, M. L. Gallo, I. Boybat, S. Haefeli, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," arXiv:1906.03138 (2019).

[212]C. H. Bennett, T. P. Xiao, R. Dellana, V. Agrawal, B. Feinberg, V. Prabhakar, K. Ramkumar, L. Hinh, S. Saha, V. Raghavan *et al.*, "Device-aware inference operations in SONOS nonvolatile memory arrays," arXiv:2004.00802 (2020).
[213]M. Klachko, M. R. Mahmoodi, and D. B. Strukov, "Improving noise tolerance of mixed-signal neural networks," arXiv:1904.01705 (2019).
[214]C. H. Bennett, V. Parmar, L. E. Calvet, J. Klein, M. Suri, M. J. Marinella, and D. Querlioz, "Contrasting advantages of learning with random weights and backpropagation in non-volatile memory neural networks," IEEE Access **7**, 73938–73953 (2019).
[215]T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," Nat. Commun. **7**, 13276 (2016).
[216]D. Negrov, I. Karandashev, V. Shakirov, Y. Matveyev, W. Dunin-Barkowski, and A. Zenkevich, "An approximate backpropagation learning rule for memristor based neural networks using synaptic plasticity," Neurocomputing **237**, 193–199 (2017).
[217]C. H. Bennett, D. Garland, R. B. Jacobs-Gedrim, S. Agarwal, and M. J. Marinella, "Wafer-scale TaO_x device variability and implications for neuromorphic computing applications," in IEEE International Reliability Physics Symposium (IRPS) (2019), pp. 1–4.
[218]S. Agarwal, http://cross-sim.sandia.gov for "CrossSim;" accessed 7 December 2019.
[219]P. Chen, X. Peng, and S. Yu, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **37**, 3067–3080 (2018).