INVITED PAPER

The Next Generation of Deep Learning Hardware: Analog Computing

This paper explores the current state of neuromorphic deep learning architectures in silicon CMOS technology.

By Wilfried Haensch, *Fellow IEEE*, Tayfun Gokmen, and Ruchir Puri, *Fellow IEEE*

ABSTRACT | Initially developed for gaming and 3-D rendering, graphics processing units (GPUs) were recognized to be a good fit to accelerate deep learning training. Its simple mathematical structure can easily be parallelized and can therefore take advantage of GPUs in a natural way. Further progress in compute efficiency for deep learning training can be made by exploiting the more random and approximate nature of deep learning work flows. In the digital space that means to trade off numerical precision for accuracy at the benefit of compute efficiency. It also opens the possibility to revisit analog computing, which is intrinsically noisy, to execute the matrix operations for deep learning in constant time on arrays of nonvolatile memories. To take full advantage of this in-memory compute paradigm, current nonvolatile memory materials are of limited use. A detailed analysis and design guidelines how these materials need to be reengineered for optimal performance in the deep learning space shows a strong deviation from the materials used in memory applications.

KEYWORDS | Analog computing; deep learning; neural network; neuromorphic computing; nonvolatile memory; synapse

I. INTRODUCTION

Recent hardware developments for deep learning show a migration from a general-purpose design to more specialized hardware to improve compute efficiency, which can be measured in operations per second per watt (ops/s/W). The limited number of mathematical operations needed, and the reoccurring nature of these operations in the underlying algorithms, was first successfully exploited using graphics processing units (GPUs) for gaming and 3-D rendering [1]. GPUs allow a high degree of parallelism for such workloads and, therefore, significantly enhance the throughput compared to a conventional central processing unit (CPU). Since deep learning algorithms also use a limited amount of mathematical operations that are repeated they would also benefit from the parallel execution using a GPU [2]–[5]. To drive further improvement of compute efficiency, features of deep learning algorithms can be exploited that are unique to that space [6], [7]. For example, deep learning algorithms are resilient to noise and uncertainty and allow, in part, a tradeoff between algorithmic accuracy and numerical precision [8], [9]. This tradeoff is not present in the traditional application space for GPUs, and this key difference is driving a new generation of ASIC [10]–[12] chip designs for deep learning. This also opens the opportunity to revisit the use of analog computing. The analog approach to deep learning hardware we consider here is an extension of the in-memory compute [13] concept in which data movement is reduced by performing calculations directly in memory. Arrays of nonvolatile memory (NVM) can be used to execute matrix operations used in deep learning in constant time, rather than as a sequence of individual multiplication and summation operations. For instance, in an $n \times m$ array it is possible to execute $n \times m$ multiply and accumulate operations in parallel by exploiting Kirchhoff's law [14]. However, to effectively use analog computing based on NVM in deep learning applications, implementation details and material properties need to be aligned with the requirements of the algorithms.

Deep learning can be divided into two distinct operation modes: training and inference. The training phase is an

Manuscript received March 1, 2018; revised June 18, 2018; accepted September 1, 2018. Date of publication October 12, 2018; date of current version December 21, 2018. *(Corresponding author: Wilfried Haensch.)*

W. Haensch and **T. Gokmen** are with IBM Research, Yorktown Heights, NY 10598 USA (e-mail: whaensch@us.ibm.com).

R. Puri is with IBM Watson AI, Yorktown Heights, NY 10598 USA.

Digital Object Identifier 10.1109/JPROC.2018.2871057

optimization problem in a multidimensional parameter space to build a model that can be used to provide a wider generalization in the inference process. A model usually consists of a multilayer network with many free parameters (weights) whose values are set during the training process. The optimal form and structure of these networks is an intense area of current research in deep learning with neural networks. The optimal network structure depends on the task to be solved and on the computer hardware that is available. In the training phase, the backpropagation algorithm (BP) [15] is used to implement stochastic gradient descent (SGD) to solve the weight optimization problem. Backpropagation consists of three components: 1) the forward path—the presented data propagates forward through the network until the end where an error is computed; 2) the backward path—the error propagates backward through the network to compute the gradients in the parameters; and 3) the parameter (weight) update. During the training process, a large body of labeled data is repeatedly presented to the network to determine the best values for the parameters. Due to the volume of data that are processed repeatedly through the many layers of the neural net network, the training process tends to be time consuming and can take weeks for a realistic data set, which can require models with hundreds of millions of weight parameters. To find an optimal solution, fine tuning of the hyperparameters, e.g., parameters related to the structure of the network and the training algorithm, is also usually required. Optimizing the hyperparameters requires several complete learning cycles. Therefore, the development of customized hardware to reduce training time is desirable to speed up model development. For inference, the optimized (trained) network is only operated in the forward path mode, and the computational requirements can vary depending on the specific application. Inference in mobile applications will stress low power while in data center applications speed (latency) may be more important. Therefore, optimal solutions for training and inference can be quite different. From a software point of view, both training a model and executing it for inference do not depend on the underlying hardware. This can, however, lead to a suboptimal performance since hardware optimization for training and inference-only will tend to be different. As a practical matter, and with the advent of specialized hardware, it is advantageous to run training and inference on the same hardware platform for a seamless transition from training to inference. For example, if training and inference are performed on different hardware platforms, retraining or tuning of the model to accommodate the difference may be needed.

The remainder of this paper is organized as follows. In Section II, a short discussion of the basic types of networks is given, and in Section III, we briefly discuss the current state of various hardware implementations. This discussion is by no means complete but describes some of the tradeoffs that need to be addressed to optimize training performance. In Section IV, we give a detailed analysis of the design and material requirements for analog computing elements. Finally, in Section V, we briefly summarize some of the interesting recent material developments.

II. DEEP LEARNING NETWORKS

As data scientists seek to increase the accuracy and speed of deep learning, the complexity of network architectures has exploded [16], [17]. We can group the most commonly used deep learning neural networks (DNNs) into three principle classes: fully connected networks (FCNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). The latter two are designed to take advantage of spatial or temporal (or sequential) correlations in the data, for instance, in image processing, text, and speech processing. The building block of a multilayer FCN is a linkage that connects every element of an input layer with every element of an output layer. It can easily be represented as an $n \times m$ matrix with n input channels (rows) and m output channels (columns). The matrix elements w_{ij} determine the strength of the connection from input x_i $(i = 1, \ldots, n)$ to output y_j $(j = 1, \ldots, m)$. The multilayer network is then built up as a sequence of these connected layers, with the output of each layer serving as the input to the next. One complication is that a nonlinear activation function is applied to the output of each layer before it is passed to the next layer. Without this nonlinear element between the layers, the network would be equivalent to a simple two-layer linear regression model and the advantages of the BP algorithm to capture complex data structures would vanish. A CNN [2] consists of convolution and pooling layers. A convolution layer [Fig. 1(a)] consists of a set of filters or kernels of size $k \times k$ and d input channels. The input channels d can be considered as the decomposition of the input into appropriate components. For the first layer, that could be, for instance, the decomposition into red, green, and blue components for a color picture, and for successive layers, it is the number of independent kernels from the previous layer. The kernels are moved across the input data with a given stride s (the number of pixels moved) to scan the whole picture. There could be a number of kernels M in one convolution layer. The filters contain the weight elements w_{ij}^{l} $(i = 1, \ldots, k, j = 1, \ldots, k, l = 1, \ldots, d)$ and are moved across the entire range of the input data. The convolution process associated for one filter bank can be interpreted as a scan for a feature in the data. The convolution process is repeated for several distinct filter banks. Each of these filter banks will produce a feature map which is then reduced in dimensionality by the pooling process. One common approach is max pooling, in which a window is defined in the feature map and the maximum element in this window is retained. In the next convolution layer, these pooled feature maps will serve as the input data set and the process is repeated. Unlike the processing that occurs in FCNs, convolution and pooling cannot be represented as simple standard matrix multiplication. For example, since the same weights are

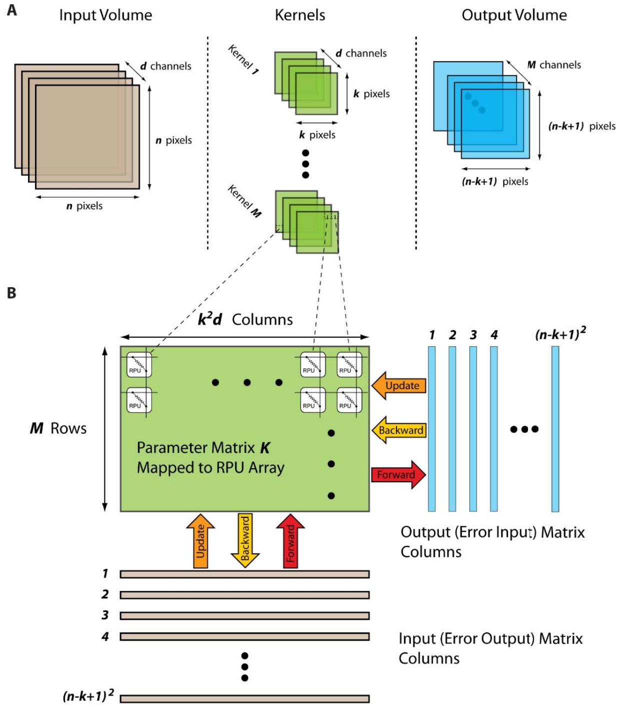


Fig. 1. ***(a) Convolution layer*** $n \times n$ ***input at*** d ***channels and*** M ***convolution kernels. (b) Mapping a convolution layer into standard matrix multiplication. The weight matrix has*** M ***rows, for*** M ***different kernels and each row contains the*** $k \times k$ ***filter pixels for*** d ***channels for each kernel. The input matrix maps the input data of dimension*** $n \times n$ ***into a matrix of dimension*** $[(n - k)/s \, \mathcal{C} \, 1]^2 \times k^2 d$.

used across the entire input for one filter, the weights are reused many times on the data. The input data stream into the convolution layer can, however, be transformed so that the convolution process can be cast into a standard matrix multiplication form, with the columns of the weight matrix containing the weight coefficients of the different filter banks [Fig. 1(b)]. The reuse of the weights is shifted now to a multiple use of data points in the input matrix. This transformation will have the effect that the simple data input vector of dimension n^2d is now replaced by an input matrix of dimension $((n - k)/s + 1)^2 \times k^2d$, with n^2d the number of pixels in the input data, feeding into the convolution matrix. The weight matrices are relatively small of dimension $M \times (k^2d + 1)$. For instance, for the first convolution layer for AlexNet, the weight matrix would have the dimension $96 \times [11 \times 11 \times 3 + 1] = 96 \times 364$ [the number of kernels × (kernel dimension squared × input channels + bias)] [5].

The most popular implementation of an RNN is the long-short-term-memory (LSTM) network [18]. The structure of this network enables the emphasis or deemphasis of portions of the data based on the data history, or its sequential (temporal) order, by feeding the output of the network back into itself. The complete input sequence is represented by an input vector that combines the training data with the feedback from the previous step. This vector is fed into a weight matrix that has components that either emphasize or deemphasize the processed data according to the data history (sequence). This combination of feedback and ingestion of new data can be repeated many times before the backward path and update process is executed. While the data handling is quite different in LSTM than in FCNs and CNNs, the weight matrix represents a fully connected network such as the FCN and is therefore also a computational bottleneck. However, there is some significant digital postprocessing of the matrix output required to accommodate the data history and intermediate results need to be stored in memory.

Matrix multiplications are at the core of the deep learning networks. For fully connected networks, data travel through the network in form of vector–matrix multiplications. For a better utilization of compute resources, several data points are usually batched together (mini-batch). To reduce the convolution process to a simple matrix multiplication requires the input to become a matrix, to begin with, and the convolution operation is a matrix–matrix multiplication. We will see later that this will have impact on the performance of an analog implementation.

III. DEEP LEARNING HARDWARE

The most popular acceleration scheme for deep learning training today involves the use of GPUs. Since GPUs are more efficient at handling matrix–matrix multiplication compared to vector–matrix multiplications, the input data are usually grouped into so-called mini-batches, combining *Mb* data input vectors into an input matrix that travels through the network together. The weight update is performed as an average across the mini-batch. In addition, the structure of the network is often designed to avoid costly memory access. These requirements drive, for example, smaller filter sizes and deeper networks. To further utilize parallel processing, n_g GPUs on a single node running on N nodes are used (Fig. 2) with a total number of $N_{\mathrm{GPU}} = n_g N$. The efficiency or speedup S of these distributed learning solutions depends strongly on balancing several system properties: GPU utilization, node and GPU I/O, and communication between the components as shown in

$$S = N_{\mathrm{GPU}} \frac{t_{io+\mathrm{GPU}}^{\mathrm{Mb}} + t_{\mathrm{GPU}}^{\mathrm{Mb}}}{t_{io+\mathrm{GPU}}^{\mathrm{Mb},n_g} + t_{\mathrm{GPU}}^{\mathrm{Mb}} + t_{\mathrm{comm}}^{N}}. \tag{1}$$

The quantity S ($S \leq N_{\mathrm{GPU}}$) is a measure how effective the distributed learning solution is. The optimal balance will depend on the exact details of the workload (network architecture and algorithm details) [19]–[21]. Since the mini-batch size is a critical parameter for the individual GPU utilization, distributed learning systems usually work with a very large mini-batch size that is then distributed across the GPUs for optimal results. Typically, mini-batch sizes for optimal use of a single GPU are in the range 32–512. For a system with 256 GPUs in parallel and

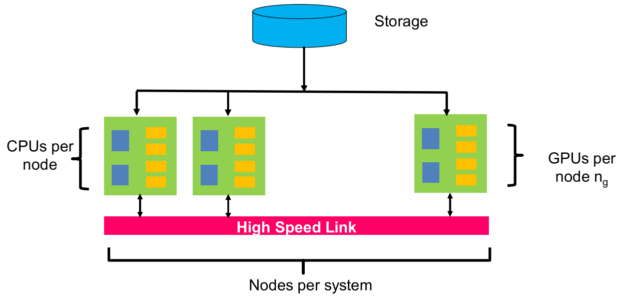


Fig. 2. ***Distributed learning system. A number of N nodes with n_g GPUs and a number of CPUs (here two as an example) are connected through network. Optimal operation requires balancing utilization of components and communication. CPUs serve as node control units and can also be used for additional computation.***

an individual GPU mini-batch *Mb* of, for example 128, that would result in a mini-batch size of 32 768 (for ImageNet ~5 GB). These data need to be read from the disk and distributed across the GPUs without creating a bottleneck. To find the right balance between compute and communication is a key challenge in hardware optimization [22]. Here $t_{io+\mathrm{GPU}}^{\mathrm{Mb}}$ and $t_{\mathrm{GPU}}^{\mathrm{Mb}}$ are the time to load the data for minibatch size *Mb* onto one GPU and the time to execute that minibatch, respectively. In the denominator $t_{io+\mathrm{GPU}}^{\mathrm{Mb},n_g}$ and t_{comm}^{N} are the time to load the data onto the node with n_g GPUs and the communication between the N nodes, respectively. A perfectly balanced system would hide the communication between nodes and would gate the data input/output (I/O) for a node by the GPU I/O. That would result in $S = N_{\mathrm{GPU}}$. Accelerating deep learning on a distributed system of GPUs is easily generalizable for any accelerator solution. The solution to the balancing problem will, however, depend on the accelerator architecture and the details of the network.

To accelerate deep learning with conventional digital hardware, the following strategies have been employed: 1) exploit the possibility of operating with reduced precision to increase compute efficiency; and 2) use data compression [23] to reduce the amount of data that is moved between components. Reduced precision provides a very effective mechanism to improve compute efficiency since it scales quadratically with the bit width [8], [7]. The use of reduced precision requires, however, a careful analysis of the complete algorithmic workflow to understand where, and to what extent, precision can be reduced without impacting classification accuracy [24]. The choice of fixed-point or floating-point arithmetic is another lever that can be exploited [8]. Again, it is important to understand the impact of these choices on the performance and versatility of a given network architecture. Since there is no fundamental theory of deep learning at present, most of these tradeoffs must be studied empirically, and a certain “safety margin” must be provided in order to apply these ideas to networks that have not been explicitly tested.

The appropriate metric for digital computation is the number of operations performed per second per unit power (ops/s/W), or per chip area (ops/s/mm^2). This measure is a reasonable metric if communication can be ignored, and if networks are confined to single chip. Unfortunately, this is typically not the case. However, if we consider the individual components of a larger system, these metrics are still a valid basis for comparison. Ultimately operation and system design will determine how much of this “raw performance” can be ultimately realized.

IV. ANALOG COMPUTE FOR DEEP LEARNING

We now turn to a more detailed discussion of the core component of a deep learning system. At the heart of the BP algorithm are three distinct operations: matrix multiplication, weight update, and the application of activation functions. For purely digital computation, these operations can be reduced to floating-point or fixed-point operations with an appropriate accuracy requirement. Alternatively, analog computing elements can be used to perform the matrix operations. Analog computations for matrix operation exploit the fact that a 2-D matrix can be mapped into a physical array (Fig. 3) with the same number of rows and columns as the abstract mathematical object. At the intersection of each row and column there will be an element with conductance G that represents the strength of connection between that row and column (i.e., the weight). If we now apply a voltage difference V across a given row and column, there will be a current flow j

$$j = GV. \tag{2}$$

We can easily generalize this concept for an $n \times m$ array (Fig. 3). At the n rows, we apply the components of a voltage vector v_i $(i = 1, \ldots, n)$ and collect the current at the n columns j_j $(j = 1, \ldots, m)$. Simple network analysis applying Ohm’s and Kirchhoff’s laws relates the current vectors to the voltage vectors

$$j_j = \sum\nolimits_{i=1}^{n} g_{j,i} V_i. \tag{3}$$

The above equation is exactly equivalent to the conventional result of a matrix multiplication if we identify the physical array with its connections g_{ij} with the abstract mathematical construct. The use of analog arrays allows us to replace two $n \times m$ floating-point operations ($n \times m$ multiplications and $n \times m$ additions) associated with vector–matrix multiplication by one single (parallel) operation. If we now further assume that the connection strengths g_{ij} $(i = 1, \ldots, n, j = 1, \ldots, m)$ can be simultaneously changed, the weight update operation can also be mapped into a single operation (in time). We would again replace two $n \times m$ floating-point operations by a single operation. The benefit is twofold: 1) we avoid moving the weight elements from memory to the chip for processing; and 2) we can replace two $n \times m$ floating-

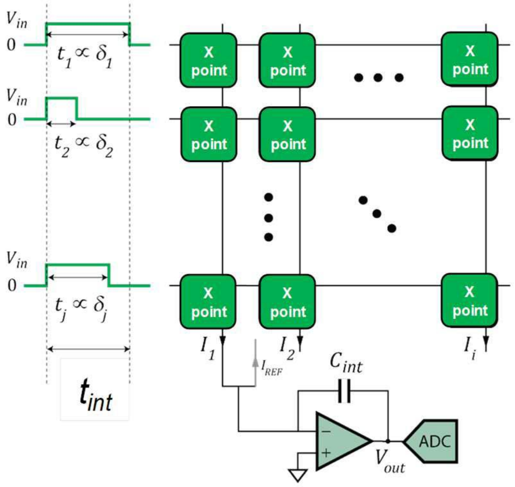


Fig. 3. ***Analog memory array—read operation. A time-encoded signal is applied to the rows and the current is integrated at each column. The ADC decodes the signal from analog into digital for further processing. If data flow between layers is time encoded, the ADC can be eliminated.***

point operations by one single operation. Both compute efficiency and communication are dramatically improved simultaneously [25]. For backpropagation, matrix multiplications is done with the transposed matrix, simply swapping the rows and columns, including the functionality of the peripheral circuits.

Of course, this begs the question: If it is that simple, why are we not doing it already? To understand the answer, we now will examine this analog process in more detail.

The use of arrays of conductive elements for matrix multiplication is not new; it was proposed many years ago [14]. With renewed interest in deep learning, it gained attention again as a possible solution to accelerate the required computations [26]–[28]. To maintain the benefits noted above, this would mean that the weight data are stored in a physical array, and that all operations are performed locally with the weights in place (i.e., not moved in and out of memory). The natural choice for such arrays come from memory technologies. We seek a memory solution that 1) can store and retain weights; 2) has a nondestructive readout mechanism; and 3) has the ability to read and write the entire memory array in one single operation. While 1) and 2) are conceivable, 3) is diametral to conventional memory operation which in its extreme implementations is optimized for random sequential access or at least will limited the accessible address space. This means we might be able to use conventional memory elements but we must create an array architecture that is different form the architecture of conventional memory. The basic array architecture that accomplishes vector–matrix multiplication as a single operation is a cross-point array with n rows and m columns, as shown in Fig. 3. A digital-to-analog (DAC) converts the components of a digital input vector of length n into a time or voltage encoded signal that is applied to the rows. The resulting column current is integrated at each column by charging a capacitor which feeds into an integrator/amplifier circuit that creates an output voltage V_{out} that is appropriate for further processing. The next step at the output would be to compute the activation function. This can be done either directly in the analog space, e.g., integrated into the amplifier function [29], or alternatively the output of the operational amplifier can be fed into an analog-to-digital converter (ADC) to calculate the activation digitally [we address the performance requirements of the input (DAC) and output (ADC) elements below]. The benefit of retaining a time-encoded signal at the output is the elimination of ADCs (power and real estate) at the cost, however, of the flexibility that a digital solution might offer in additional processing capabilities (choice of activation function, data renormalization, network configurability, etc.). A critical quantity in this scheme is the integration time t_{int} needed to accurately determine the integrated column charge. The integration time depends on the tolerated signal-to-noise ratio (SNR) and is influenced by the array size n, the dynamic range of the cross-point element $\beta = g_{\max}/g_{\min}$, the operating voltage V_{in}, and the cross-point device resistance $R_{\text{dev}} = 1/g_{\text{dev}}$. If we assume an SNR of 10% at the integrator/amplifier output, the design tradeoffs are captured by [30]

$$\frac{1}{10}\left(\frac{\beta-1}{\beta+1}\right) V_{in}\sqrt{\frac{t_{\text{int}}}{n R_{\text{dev}} k_B T}} > 1 \tag{4}$$

where k_B is the Boltzmann constant and T is the chip temperature. For the estimation of the feedback capacitance C_{int} (see also Fig 3), we have

$$C_{\text{int}} = 40\frac{V_{in}}{V_{\max}}\left(\frac{\beta-1}{\beta+1}\right)\frac{t_{\text{int}}}{R_{\text{dev}}}. \tag{5}$$

The last constraint comes from the fact that the voltage drop across the metal lines, i.e., within the rows and columns, should be no more than 10% of the voltage drop across the device itself

$$\frac{n^2 R_{\text{cell}}}{R_{\text{dev}}} < 0.1. \tag{6}$$

Here R_{cell} is the resistance of the metal line in a unit cell of the array. Equations (4)–(6) constrain the design of a suitable analog array for deep learning. They implicitly assume that no select device is needed. The introduction of a select device might be necessary, as discussed below, to compensate for nonideal behavior in the memory elements. The design constraints presented in (4)–(6) provide the densest array solution for optimal power and performance at the array level. An $n \times m$ array configuration can perform an equivalent of two $n \times m$ floating-point

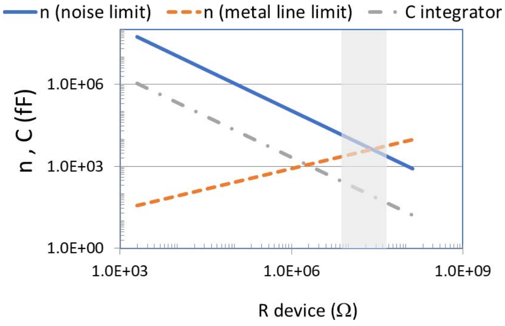


Fig. 4. ***Array sizing tradeoff at constant performance: resistive noise limit (blue solid line), metal line limit (brown dashed–dotted line), integrator capacitance limit (gray dashed–dashed line). Parameters used are:*** $t_{\text{int}} = 80$ ns, $R_{\text{cell}} = 144$ mΩ, V_{in} ***and*** $V_{\max} = 1$ V, $\beta = 5$***. Shaded area indicates optimal device operation range 5 to 25 M***Ω***.***

operations at constant time. With an integration time t_{int} this gives the equivalent performance of two $n \times m/t_{\text{int}}$ ops/s. For example, parallel processing of a square 1000×1000 array with $t_{\text{int}} = 1\mu s$ corresponds to 2 Tops/s. If the integration is reduced to 80 ns and the array size increased to 4000×4000, the throughput is equivalent to 400 Tops/s. Basic restrictions on noise, voltage drop in the metallic cross-bar lines, and integrator capacitance size will limit the practical array size. For example, the tradeoff between cross-point element resistance and array size is shown in Fig. 4, at constant performance, e.g., integration time. On the one hand, a larger device resistance enables a larger array at a given metallization technology because the voltage drop in the metal lines will remain relatively small compared to R_{dev}. On the other hand, a larger resistance increases the thermal noise. Array size and device resistance will be optimal at near the cross point of these two factors. For low resistance and array size tradeoff, the integrator capacitor will (at the constant performance scaling) be the limiting factor since the integrator circuit needs to fit into the array pitch which is difficult if the required capacitance is too large. Adjusting performance (smaller integration time for smaller arrays) will remedy this situation.

The vector–matrix multiply function is rather straightforward to implement. In the backpropagation algorithm, vector–matrix multiplication occurs in both the forward and backward paths. The third component of the BP algorithm is the weight update. During training, the weights (or conductances) are updated according to

$$w_{i,j} = w_{i,j} + \varepsilon x_i \delta_j \tag{7}$$

with x_i the forward path input vector into layer L, δ_j the backpropagated error vector coming from layer $L+1$ and ε the learning rate, which is a hyperparameter that is adjusted for optimal performance (accuracy and speed). In matrix multiplication, the conductances in the array are considered fixed, and are merely sampled by applying a voltage. The weight update process is considerably more complex. Here the weights must be changed in response to x_i and δ_j. The challenge is to execute the weight change locally—at each individual cross-point element—for all array elements at the same time. This requires a physical mechanism in which resistance of the cross-point material changes in response to a stimulus. One class of material that has this property is nonvolatile memory (NVM). Nonvolatility means that the conductance of the element, i.e., the weight value, persists over a considerable time. Typical NVM elements are used to store a small number of bits (e.g., one or two) in such a way that the stored information can be recovered individually for each element. For deep learning applications, many more states per device must be accessible to enable an incremental (or analog) weight change during training. We will see below that required properties of NVM elements for deep learning are qualitatively different from materials optimized for conventional memory applications. For example, the required changes in the conductance level are more gradual and that it is, in general, not necessary to recover the conductance of a single element.

There are many potential schemes for updating the conductances of the cross-point elements. For example, a selector device could be used so that each device in the array is independently updated in a serial fashion. Closed-loop iterative methods can be implemented to precisely adjust each conductance. However, these methods increase circuit complexity and are too slow for practical applications. Furthermore, as noted above, parallel weight update would result in significantly higher throughput. One scheme that enables open-loop, parallel weight update without a selector device is shown in Fig. 5. The approach exploits coincidence between voltage signals on both the rows and columns. A sequence of voltage pulses representing the input vector x is applied to the rows. Similarly, a sequence of voltage pulses representing the backpropagating error vector δ are applied to the columns. The components of these vectors are time-encoded representations [29], i.e., series of pulses of constant voltage and length, for both the input vector x and the error vector δ. The encoding scheme discretizes the analog input signal into a fixed bit stream of K bits. Each bit consists of a fixed-length voltage pulse with an amplitude of 0 or V, with the number of nonzero pulses for each input component k_i would be proportional to the input vector component x_i. The finite quantization will require a rounding to fit the grid. A stochastic rounding procedure turns out to be beneficial for the robustness of algorithm. An alternative approach is to encode row and column signal as stochastic bit stream of length K [30]. The number of nonzero pulses k_i in that bit stream would then, in average, be proportional to the input signal x_i. In both cases, K could be used as an adjustable parameter that will modulate the learning rate. Encoding the signals into bit streams with constant voltage pulse sequences will simplify the peripheral

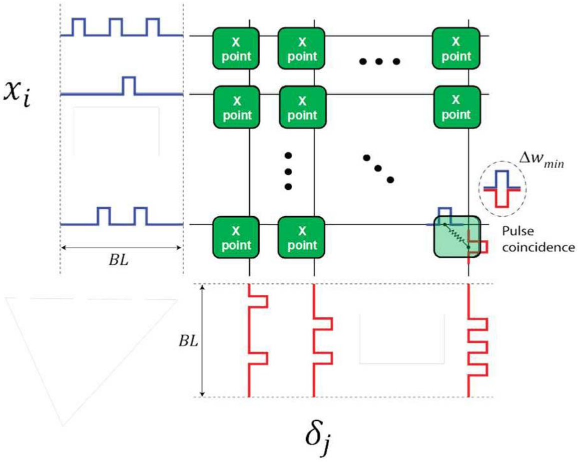


Fig. 5. ***Analog array—weight update. Rows (x) and columns (δ) receive a bit stream BL of K constant voltage pulses. Weight update occurs where row and column pulses coincide.***

circuits. At the core of this scheme is the response of the material to coinciding pulses at the cross points of the rows and columns. Ideally, the conductance at the cross point will change by a maximum amount Δg_{ij} when there is a coincidence between pulses on the row and column and the voltage across the device is 2 V. "Half-select" condition, e.g., a pulse on the row but not the column, can also cause a response in the material. For materials with a switching threshold, the signal voltage V can be chosen to be lower than the threshold to avoid the half-select problem. This is not always possible and the "half-select" problem can degrade performance. However, with stochastic bit stream encoding, the half-select problem can be mitigated due to random averaging of the half-select signals. Coincidence events permit a certain tolerance which would result in a fluctuation of the learning rate. The goal is to compute all weight updates in parallel. Since both the row and column inputs could carry a sign, the array update will require four cycles (++, -+, +-, - -) with the correct pulse polarity. The time that is needed for that will be determined by the pulse length t_{pulse} and the length K_{BL} of the stimulus bit stream

$$t_{\text{update}} = 4K_{BL}t_{\text{pulse}}. \tag{8}$$

Both are hyperparameters that need to be adjusted for optimal performance. Their values will depend on the materials used for the cross point and on the network architecture [31].

To estimate the cycle time per layer of the network we add the times for the forward and backward passes to the weight update time

$$t_{\text{cycle}} = 2t_{\text{int}} + t_{\text{update}}. \tag{9}$$

To this we must add the overhead that comes from the DAC, ADC, other digital computations, and the communication between digital and analog components. With a proper choice of architecture some of these can be hidden through proper pipelining and others represent a genuine constraint. For instance, when the ADCs are sampling the output voltage at the opamp, the array can already process the vector–matrix multiplication and data could be encoded at the DAC. The number of analog arrays, or tiles, that can be operated in parallel will depend on the available on-chip data rate and the on-chip memory. From a network-on-chip (NoC) prospective, an optimal solution requires balance between the optimal number of analog tiles, digital backbone, on-chip memory, and on-chip communication bandwidth. The tradeoffs are like those discussed above [see (1)], however without the weight movement which reduces the data amount significantly. In addition, these must be balanced with off-chip I/O. We will not discuss the system design aspects further, but will focus on the operation of a single analog tile.

There are two key considerations that need to be addressed to understand performance quantitatively: 1) the digital/analog interface requirements; and 2) the material properties of the cross-point elements. To study the impact of the A/D interface and the impact of cross-point material properties, we constructed a simulation tool [30] that captures the impact of the peripheral circuits, e.g., DAC and ADC accuracy, SNR sensitivity of the integrator/amplifier, as well as the switching behavior of the cross-point element, including device-to-device variation, stochasticity, and cycle-to-cycle variations in a single device. We do not consider a detailed circuit model for the periphery [32] but maintain an abstraction level that captures the interaction of the A/D and D/A conversion with the properties of the analog elements on the algorithmic performance for easy scalability to larger more complex networks. The typical switching behavior of NVM materials depends very strongly on the switching mechanism. The weight elements can be positive and negative and will, during training, move up and down, often changing sign. Since physical conductivities are always positive, signed weights can only be realized using a differential signal. The differential signal can be obtained either by associating two conductivities G^+ and G^- to one cross point

$$G = G^+ - G^- \tag{10}$$

or comparing a local cross-point conductance G_{local} to a global reference G_{global} for all elements

$$G = G_{\text{local}} - G_{\text{global}}. \tag{11}$$

Changes in the conductivity need to be small to avoid convergence problems in the algorithm. In deep learning software solutions, the change in weight is controlled by the learning rate ε [see (7)]. Weight changes that

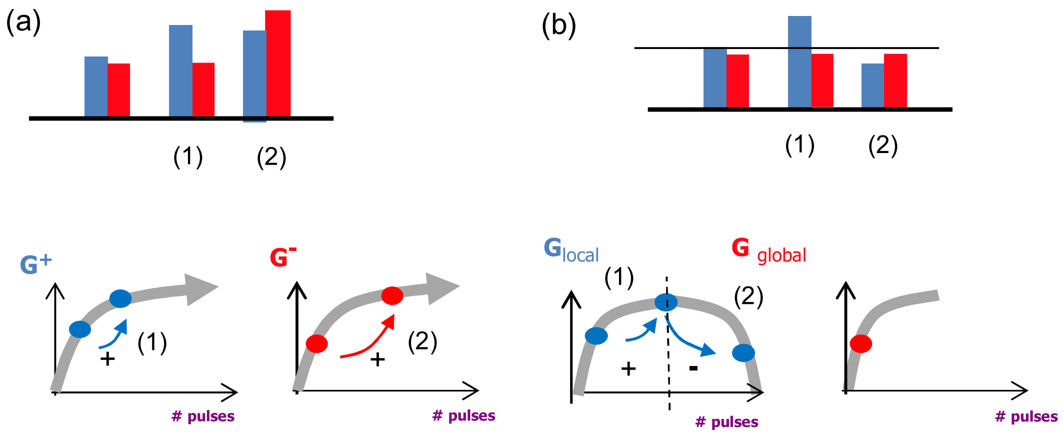


Fig. 6. *(a) One-sided switching: Conductivity can only change gradually in one direction. Conductance level will eventually saturate and the differential signal will not change anymore. (b) Two-sided switching: Conductivity can gradually switch up and down. Differential signal is measured against a fixed reference (global or local).*

increase the weight value are called potentiation, and weight changes that decrease the weight value are called depression. Material properties for NVM do not map in a natural way into the requirements for the weight movements, mainly because the response to a given stimulus depends both on the value of the weight and the sign of the stimulus [27]. Ideally, the response would be linear (independent of the weight value) and symmetric (independent of the sign of the stimulus). For many NVM materials, conductance increase is associated with the set operation and conductive decrease with the reset operation, e.g., phase change memory (PCM) and resistive random-access memory (RRAM). In general, for existing NVM materials, these two processes are very different due to the underlying physical mechanisms. NVM materials can be distinguished by the detailed switching properties of these two branches. There are several materials available that fall under the following categories (Fig. 6).

1) One-sided or unipolar switching. These are materials that show a gradual change of conductivity in one branch, usually the set branch, and are abrupt in the other [Fig. 6(a)]. In practice, the conductivity can only be gradually changed in one direction.
2) Two-sided or bipolar switching. These materials show gradual change in both set and reset branch [Fig. 6(b)]. Gradual conductivity changes can be made in either direction.

The switching behavior can be further classified as either linear or nonlinear depending on the number of stimulus pulses and the conductance state. As indicated in Fig. 6(a), the nonlinearity can lead to a saturation since conductance change is only in one direction. For a two-sided switching device, symmetric switching means that under the same number of potentiation and successive depression pulses the device returns to its original conductance level. This last requirement is more general than linear switching since symmetric switching does not require linearity. In operation, one-sided switching devices must be paired, with one element carrying the potentiation and the other the depression signal as shown in Fig. 6(a). The net conductance is the difference between the conductances of the individual elements. To achieve symmetric weight update, the pair needs to match in their linearity. For the two-sided device, it is possible to use a global reference (a column for forward or row for backward propagation) since the conductivity in the cross point can move locally up and down [Fig. 6(b)]. The reference element however must be the same for all rows and columns because the weights need to be the same for forward and backward propagation. A local fixed reference would eliminate this restriction at the cost of array size or process complexity (stacked arrays). The requirements for the cross-point materials for use in deep learning training can be explored using a fully analog deep learning model. The subtleties in the material properties are incorporated into our device switching model. It incorporates spatial variations (device-to-device) and temporal stochastic behavior (coincidence-to-coincidence) shown in Fig. 7.

To understand the interdependence of the digital interface, material properties, and the BP algorithm, we implemented a three-layer FCN (784, 256, 128, 10) for the handwritten character data set MNIST and investigated the training performance, sensitivity to device parameters, and the A/D interface specifications [30]. We find that the performance of the FCN is robust against stochastic behavior for certain properties and less tolerant to others, as summarized in Fig. 7. Regarding the material properties, the most important attributes are: 1) weight movement for potentiation and depression must be symmetric within 2%; and 2) the granularity of the weight update requires 1000 steps (10 b), on average, between minimum and maximum conductance values in the case that all variations are considered simultaneously. Relaxed requirements are observed for individual components which is a nonphysical situation. A 5-b DAC and a 9-b ADC are

Parameter	Variation	Individual	Combined
Δw_{min}	Global	**0.01**	**0.001**
$\lvert w_{ij}\rvert$	Global	**0.3**	**0.6**
Δw_{min}	Cycle	**150%**	**30%**
Δw_{min}	Device	**110%**	**30%**
$\lvert w_{ij}\rvert$	Device	**80%**	**30%**
Δw_{min}^{-}	Global	**5%**	**small**
Δw_{min}^{+}	Global	**5%**	**small**
$\frac{\Delta w_{min}^{-}}{\Delta w_{min}^{+}}$	Device	**6%**	**2%**
$Noise$	Cycle	**10%**	**6%**

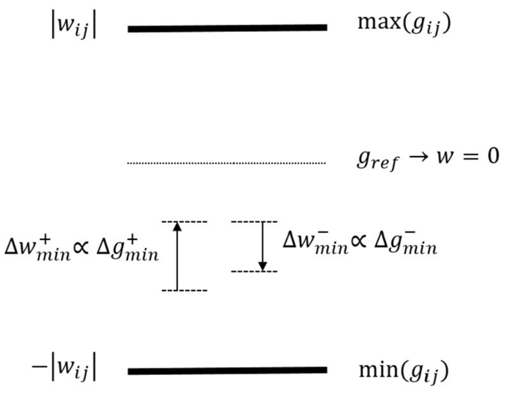

$|w_{ij}|$
$\max(g_{ij})$
$g_{ref} \rightarrow w = 0$
$\Delta w_{min}^{+} \propto \Delta g_{min}^{+}$
$\Delta w_{min}^{-} \propto \Delta g_{min}^{-}$
$-|w_{ij}|$
$\min(g_{ij})$

Fig. 7. ***Important device variability components.*** $\Delta \mathrm{w}_{\min}$ ***minimum step from device to device (spatial) and from coincidence to coincidence (temporal). Device to device weight range*** w***. Overall up (+) and down (-) change*** $\Delta \mathrm{w}_{\min}^{-/\mathcal{C}}$ ***and up (+) and down (-) ratio for a single device*** $\Delta \mathrm{w}_{\min}^{\mathcal{C}}/\Delta \mathrm{w}_{\min}^{-}$***. The table shows individual sensitivity and combined sensitivity at 0.3% penalty from floating-point results.***

required, and a noise level of 6% can be tolerated at the integrator/amplifier output. The MNIST data set is very small, with only about 60 000 images, and the FCN that is used for training has 235 000 weight parameters. For comparison, the recent networks [6] used for training on the ImageNet data set, with 1.2 million pictures, have tens of millions of parameters and many layers [Alexnet 61 million weights, five convolutions, and three fully connected layers; ResNet 50 25.5 million weights, 53 convolutions, and one fully connected layer(s)]. It is not clear, at present, if the sensitivity analysis presented here holds true when the networks are massively scaled. In a more recent analysis, we have shown that convolution layers can be implemented with the same device parameters if noise and bound management is introduced at the digital peripheries [31] and that stochastic rounding is beneficial for larger LSTM networks [33]. The sensitivity analysis provides a set of target material properties for useful cross-point elements to implement the BP algorithm for deep learning networks. Given the material parameters, we can estimate the performance of a given tile. In Fig. 8, we show an example for the performance as a function of array size for one layer in a fully connected network (FCN, LSTM). For a convolutional layer, power and performance will degrade significantly due to the mapping discussed above (Fig. 1). In contrast to the trend in GPU-based convolutional networks, analog arrays would favor larger kernels. The operation conditions for the example are: integration time 80 ns, sharing 16 rows (columns) per ADC [34] with a sampling rate of 200 M samples/s at an energy of 23 pJ per sample and R_{dev}24 MΩ. An on-chip data rate of 90 GB/s is required for the largest tile. In modern NoC, TB/s of on-chip band width is available which gives enough headroom for the communication between tiles. Latency due to the additional digital operations (activation, data renormalization, etc.) can be mostly hidden by pipelined design. Building a NoC with multiple analog tiles as a primary building block can therefore provide a performance potential in the thousands of Tops/W/s on chip.

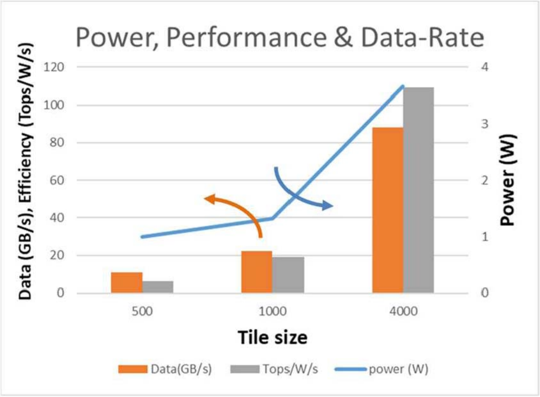


Fig. 8. ***Performance, power, and data rate for a single tile with an input vector of size 500, 1000, and 4000.***

V. MATERIALS

To capitalize on this performance potential, further innovation is required. Either more-ideal material systems must be identified or circuit solutions must be developed that can accommodate imperfect materials. The latter will, however, come at a cost of performance, power, and area since it will need an overhead that can be avoided if suitable materials are identified. A third solution are innovations or modifications of existing learning algorithms that can function with imperfect devices and take advantage of the array architecture. Ultimately, the classification accuracy of analog-based solutions needs to be on-par with that of digital solutions, however with the benefit of better power and performance. Conventional NVM materials are optimized for memory applications that require a large SNR ratio [Fig. 9(a)] to securely recover the

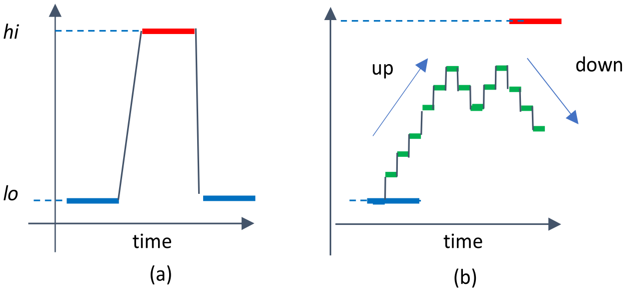


Fig. 9. ***(a) Memory elements have well separated resistive states that allow individually read out. (b) Deep leaning requires gradual symmetric changes. Individual states will not be read out except in case that an occasional reset is required (one-sided switch). (a) Binary change. (b) Incremental change.***

stored information. Therefore, only a few, or even just two, conductance levels are supported in conventional NVM. Since the cross-point elements in a deep learning array are never accessed in a sequential fashion the individual state is never captured. What matters is the accumulated column (row) signal (Fig. 3), which is an averaged quantity. Our sensitivity analysis shows that, unlike a conventional memory, a certain degree of variation is tolerable in a single element. The deep learning training process is self-correcting with self-consistent weight updates. However, small and symmetric weight changes, as shown in Fig. 9(b), are strict requirements. For one-sided switching elements, the requirement for symmetric switching will require locally matched linear switching behavior for G^+ and G^- to realize accurate differential operation. Nonlinearities will impact the network performance significantly [35]. For the two-sided switching devices, symmetric switching in both the set and reset branch is required while linearity is less important. There are several NVM materials that have been explored for deep learning. No winner has emerged as a competitive solution for deep learning training. For inference-only the material constraints are relaxed: no symmetric switching is required, and the granularity (number of states) can be significantly reduced. The weight transfer from a trained floating-point model to an analog array can be done sequentially (node by node, row by row, or column by column) with closed-loop feedback to guarantee accuracy. We find that a replication of the floating-point weights within a 5% error is sufficient to replicate the classification error of the original model. However, inference-only will stress cross-point array yield since the training process implicitly assumes all cells are functional.

Popular materials that are investigated for use in analog arrays for deep learning networks are as follows.

A. Phase Change Memory (PCM)

In PCM, different conductance levels are created by changing the morphology of chalcogenide layers from amorphous to crystalline [36]. This transition is thermally activated and therefore requires a heater. This heating element can be the PCM material itself (i.e., Joule heating) if fed with a critical current density. Typically, the memory element (chalcogenide layers) is in series with a low resistive contact material of small diameter to supply the high current density that provides the energy for the phase transition. The amorphous material gradually crystalizes in a moving front, lowering the resistance. This mechanism eventually saturates, and no further conductance increase is possible. To return the element into the high resistive state a high and fast current pulse will melt the material and return it to the amorphous (high resistance) state. The conductance change in this process is very abrupt. PCM is therefore a 1-D switch with a gradual set (amorphous –> crystalline) and an abrupt reset (crystalline –> amorphous). Since the switching process is related to a change of crystal structure, PCM switching shows stochasticity and relaxation phenomena during the set process which can influence the training process due to unacceptable (and unintended) weight changes between updates [37], [38]. Despite this, PCM materials have been successfully used for deep learning [29], [39]. The work around for the saturation of the conductivity is to periodically reset both the G^+ and G^- conductances while maintaining the difference which is proportional to the weight [35]. This operation requires a select device because the reset operation needs to be executed on a small number of devices and not on the array. Recently significant progress has been reported by combining the PCM element with a capacitor and separating leading and trailing digits of the weight [40]. Frequent weight updates during training are performed by modulating the charge on a capacitor and periodically the weight information is transferred to the PCM element when a critical charge state is reached. This separation ensures a symmetric update for frequently changing weight increments and provides nonvolatility for the most significant digits of the weight, thus, circumventing the limitations of the unipolar switching PCM material.

B. Resistive Random-Access Memory (RRAM)

The basic architecture of an RRAM device is a metal–oxide film sandwiched between appropriate metal contacts [41], [42]. The top contact controls the infusion of oxygen vacancies to form a conductive filament that consists of substoichiometric oxide. The conductivity of the device is determined by the proximity of the filament to the bottom contact. The conductivity is controlled by growing (set) or shrinking (reset) the size of the filament. The filament formation (set) and dissolution (reset) are reversible which enables conduction changes in both directions. Although both branches can show a gradual change in conductivity the observed changes are not symmetric. Most RRAM devices require a formation process that establishes the conducting filament. This formation process will determine the base resistance of the RRAM device and it is known that a proper current control during the formation process can serve to modulate the operating resistance range of the device. As with PCM, the change of the conductance depends on structural changes at the atomic level, and therefore is intrinsically stochastic. Controlling the filament formation and dissolution, as well as engineering symmetric set and reset behavior, are the key challenges for RRAM. Convincing hardware demonstration for deep learning training with RRAM is still lacking. However, simulations that incorporate RRAM devices with improved device characteristic [43] are encouraging. The use of a select device for RRAM can be avoided if a material combination is found that has a controlled filament formation that does not require the active current limiting needed for current devices.

C. Conductive Bridge Random Access Memory (CBRAM)

In contrast to RRAM, in a CBRAM, a conductive path is formed by mobile metal ions that move through an electrolyte or dielectric [44], [45]. The typical stack of an CBRAM consists of an inert electrode, a solid electrolyte or dielectric, and an electrochemically active electrode. The motion of the metal ions is controlled by applying a voltage to the stack and it is reversible. There are a variety of material combinations discussed in the literature for this type of device, mostly addressing their use in memory applications. More recently they have also been considered as cross-point elements for analog arrays for deep learning. A simple stack of $Cu/SiO_2/W$ showed interesting gradual switching properties for set and reset branches [46], however with the caveat that variable voltage pulses were used, which is impractical for an array implementation. That work suggests that a two-layer diffusive model can explain the linear switching of this stack and would possibly reduce the stochasticity of the switching process.

D. Ferroelectric Devices

A ferroelectric device is a stack of a thin dielectric, ferroelectric material (FE) located between a suitable metal electrode and a substrate. While initial material stacks were based on ferroelectric perovskites, HfO_2-based stacks are easier to integrate with conventional CMOS [47]. Ferroelectric materials will respond to an external field by changing their electric polarization, either by moving domain wall boundaries or by directly flipping the polarization of a small crystalline domain. In FE devices, the polarization modulates the interface barriers in the stack (Fig. 10) that can either be used to tune a threshold voltage of a field-effect transistor (FET) or to modulate a current through a tunable tunnel junction. Both applications have been proposed for a synaptic device. While the FET solution is a three-terminal device, the tunnel junction is a two-terminal device that will provide higher density for a cross-point array due to its smaller size. Proposals to use the adjustable channel conductance of ferroelectric (FE) FETs as a synaptic weight date back to the early 1990s. Significant progress has been made using perovskite ferroelectrics such as $Pb(Zr,Ti)O_3$. Yet, implementation on a conventional silicon CMOS platform remains challenging, due to incompatibilities with CMOS processing. The recent discovery of a previously unknown FE phase of HfO_2 (FE–HfO_2) [48], [49], has the potential to remove the integration challenges of the traditional perovskite-based FE materials. Ferroelectric two-terminal (capacitor or resistor) and three-terminal (transistor) devices can thus be built from conventional high-k/metal gate materials [50], [51] used in commercial CMOS logic FETs, albeit with different processing and doping to achieve ferroelectric behavior. This opens the possibility of implementing a variety of tunable solutions on a CMOS platform, e.g., FeFETs, FE capacitors controlling conventional FET gates, or metal–FE–metal (MFM) ferroelectric tunnel junctions (FTJs) that might be useful as synaptic device. The outstanding issues are the demonstration of gradual symmetric switching under constant voltage pulse stimulation, switching distributions that meet the requirements outlined above, operating conditions that allow energy-efficient operation, and dimensional scalability.

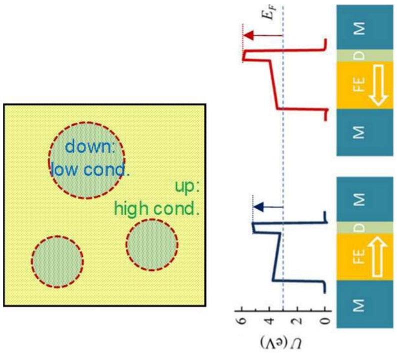


Fig. 10. ***Metal/ferroelectric/dielectric/metal tunnel junction. Polarization domains are modulated by an applied voltage and changing the interface barriers with respect to the Fermi energy E_F to modulate the tunnel current [51].***

E. Electrochemical Device

Electrochemical devices are a newcomer in the field of contenders for an analog array element for deep learning. The device idea, however, has been around for a long time and is related to the basic principle of a battery [52]. Compared to the previously discussed switches, which only required two terminals, this switch required three terminals. The device structure, shown in Fig. 11, is a stack of an insulator that forms the channel between two contacts source and drain, an electrolyte, and a top electrode (reference electrode). Proper bias between the reference electrode and the channel contacts will drive a chemical reaction at the host/electrolyte interface in which positive ions in the electrolyte react with the host, effectively doping the host material. Charge neutrality requires the free carriers to enter the channel through the

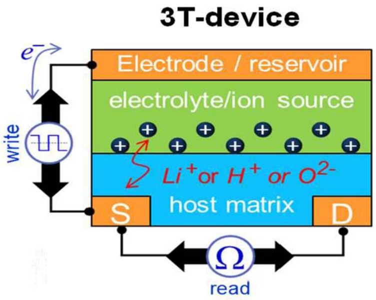


Fig. 11. *Three-terminal electrochemical device. Applying bias between the reference (top) and S/D electrodes moves ions in and out of the channel (host matrix). To maintain charge state of the channel reference electrode must be disconnected after write.*

channel contacts. If the connection between the reference electrode and the channel contacts is terminated after the write step the channel will maintain its state of increased conductivity. The read process is simply the current flow between the two channel contacts, source and drain, with the reference electrode floating. It has been shown [53] that almost symmetric switching can be achieved if the reference electrode is controlled with a current source. Regarding the switching requirements, the group obtained similar criteria [54] that are shown above. Voltage control of the reference electrode leads to strongly asymmetric behavior due to the buildup of an open circuit voltage (VCO) that can depend on the charge state of the host. If the reference electrode voltage compensates for VCO, almost symmetric switching can be achieved as well. For a voltage controlled analog array for deep learning, this device is not suited since every cell would require an individual compensation depending on its conductivity. Possible solutions are low VCO material stacks.

With the tunable resistive elements at different states of maturity, the question is as follows: Can we implement analog arrays for deep learning with existing CMOS technology options? Charge is the natural agent in the CMOS world to represent the weights. Charges can be stored either on a floating gate of an EPROM device or in a capacitor. Both possibilities are explored and for both the main switching properties we discussed above hold: symmetricity in potentiation and depression and sufficient granularity in the change.

F. Floating Gate Devices

Floating gate devices for use in analog arrays for DNN were proposed in the early 1990s [55], [56] coinciding with the emerging Flash memory technology. The weights are represented by charges stored in the floating gate of cross-point cell device. The analysis of the required switching properties matches our results with respect to symmetry and weight granularity. To meet these requirements, a modified cell design was proposed, albeit with a very large cell size. There are two additional concerns for using floating gate devices for deep learning analog arrays: write speed and durability. During the write process, the charge injection into the floating gate is either accommodated by hot electron effects or by tunneling. Both processes are relatively slow and require high voltages. They also tend to damage the gate dielectric and lead to a limit on durability of about $10^5 - 10^6$ write cycles. The number of weight updates for training on the 1.2 million ImageNet samples with a minibatch size of 256 at 50 epochs is $2.3{*}10^5$. Recently, proposals have emerged to take advantage of the 3-D stacked architecture for NAND flash or solid-state drive (SSD) configurations [57] for deep learning applications. Due to the limitations on endurability and high voltage operation, it is questionable if floating gate devices are competitive for deep learning training. They might, however, be useful for inference as only the read operation is required.

G. Analog CMOS

In a DRAM, stored charge is used to represent a single bit. Conceptually, the amount of stored charge could be used to represent an analog weight. However, the translation from DRAM to an analog array for deep learning is unfortunately not as straightforward as one might imagine [58]. In DRAM, the charge state of the capacitor is destroyed during the read operation. However, the read mechanism remembers the state and writes it right back. Unfortunately, the writeback operation can only restore the signal to the rails: high or low, which is sufficient for the DRAM but not for a deep learning array where the weights are an arbitrary level between the high and low state. An addition concern is that charge leaks out of the capacitor, and to compensate for this, the entire DRAM array is periodically refreshed. The typical period for the refresh operation is in the order 32–64 ms, while the characteristic leakage time, the retention time τ_{ret}, in which 50% of the cells fail to give the correct signal, is in the several-second regime. For deep learning, we need to avoid destructive reads and compensate for charge leakage. If we assume a time Δ between weight updates of the order of 200 ns, and a retention time in the order of seconds, the updated weight will decay according to

$$w \leftarrow w\left(1-\frac{\Delta}{\tau_{\text{ret}}}\right). \tag{12}$$

This has the same form as the weight decay produced by L_2 regularization [59] which is a method to avoid overfitting. Typical values for the decay in the software world are about 10^{-5}–10^{-6}. By comparison we find the required retention of the order of several seconds. A word of caution is in order since we assumed a cycle time of 200 ns between updates. For convolutional networks as discussed above the input is a matrix with $[(n-k)/s+1]^2$ columns. The time between updates will then be approximately $[(n-k)/s+1]^2 \times 200$ ns, which would increase the required retention time τ_{ret} by $[(n-k)/s+1]^2$, which is difficult to achieve in hardware. The input matrix size, for instance, for the first convolution layer for the small CNN LeNET (MNIST) is 576 and for AlexNET (ImageNET) is 3025 which would require retention times τ_{ret} in access of several minutes or more which are unrealistic to achieve. A controlled symmetrical change of charge requires the use of current sources: one for injecting charges, the other for extracting charges. These are usually done with simple FETs that operate in saturation. In the dormant state, they are turned off and leak. This leakage will discharge(charge) the capacitor which is needed to hold the charges to a certain level. With the definition of the retention time, we find

$$C \approx \frac{I_{\text{lkg}}\tau_{\text{ret}}}{V_{\text{cap}}}. \tag{13}$$

With the retention time in 1-s range and V_{cap} in the range of 1 V, a very low leakage CMOS technology is required. With an area capacitance of 470 $\text{fF}/\mu\text{m}^2$ that can be achieved for an eDRAM technology [60] real estate for the capacitor would scale as $2(I_{\text{lkg}}/\text{pA}) \times (\tau_{\text{ret}}/\text{s})\mu\text{m}^2$. Therefore, an ultralow leakage CMOS technology is required for a competitive arrays size. In addition to these simple scaling considerations, the effect of the device variations on the scaling behavior of the cell needs to be explored in more details for the implementation of a robust learning algorithm.

VI. CONCLUSION

Now artificial intelligence (AI) is synonymous with deep learning. The desire to apply deep learning to all facets of life is reminiscent of the pervasive use of microelectronics enabled by traditional scaling. We are far away from a similar scaling law for deep learning; in fact we do not even have a fundamental theory that can guide us. Progress is being made by brute force: we develop more complex neural networks with tens of millions of parameters, collect and curate huge labeled data sets, and find the hardware to run the algorithms. For a pervasive use of deep learning, cost is a major issue. Cost means the time to build models and the computational resources that are needed to train and execute them. The realization that GPUs are a good fit for these tasks was a critical enabling step. However, it is now clear that specialized hardware that is customized for deep learning can do better than conventional GPUs. We are already seeing the emergence of a new generation of deep learning accelerator hardware: trading general use for compute efficiency, which ultimately means cost. Unfortunately, the shear complexity of building and training models forces us to look at the solution at the system level, where several deep learning accelerators work together to solve the problem. We have only briefly touched on the system aspects for deep learning machines, but these issues will ultimately determine the viability of new AI hardware accelerators. Our discussion focused on the basic design and material properties issues that need to be addressed for analog accelerators. Only if we can show, in a convincing manner, that these are solvable do questions concerning system-level integration become relevant. We do not expect that analog computing for deep leaning will drive a fundamentally new ecosystem but rather will augment the existing, digital one. We will see a continued push to improve neural networks and to push digital hardware solutions to the limit of what is possible. The analog solutions, if successful, should be ready to fit seamlessly into this evolution. ■

REFERENCES

[1] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar./Apr. 2010.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer (2017). "Efficient processing of deep neural networks: A tutorial and survey." [Online]. Available: https://arxiv.org/abs/1703.09039

[7] A. Agrawal *et al.*, "Approximate computing: Challenges and opportunities," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, San Diego, CA, USA, 2016, pp. 1–8.

[8] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, Lille, France, 2015, pp. 1–10.

[9] 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." [Online]. Available: https://arxiv.org/abs/1602.02830

[10] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[11] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Toronto, ON, Canada, Jun. 2017, pp. 1–12.

[12] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[13] M. Di Ventra and F. L. Traversa, "Memcomputing: Leveraging memory and physics to compute efficiently," *J. Appl. Phys.*, vol. 123, no. 18, p. 180901, 2018.

[14] K. Steinbuch, "Die lernmatrix," *Kybernetik*, vol. 1, no. 1, pp. 36–45, 1961.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[16] A. Canziani, E. Culurciello, and A. Paszke (2016). "An analysis of deep neural network models for practical applications." [Online]. Available: https://arxiv.org/abs/1605.07678

[17] L. Lazebnik. (2017). *Convolutional Neural Network Architectures: From LeNet to ResNet*. [Online]. Available: http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1223–1231.

[20] P. Goyal *et al.* (2017). "Accurate, large minibatch SGD: Training ImageNet in 1 hour." [Online]. Available: https://arxiv.org/abs/1706.02677

[21] M. Cho, U. Finkler, S. Kumar, D. Kung, V. Saxena, and D. Sreedhar (2017). "PowerAI DDL." [Online]. Available: https://arxiv.org/abs/1708.02188

[22] S. Shi, Q. Wang, and X. Chu (2017). "Performance modeling and evaluation of distributed deep learning frameworks on GPUs." [Online]. Available: https://arxiv.org/abs/1711.05979

[23] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan (2017). "AdaComp: Adaptive residual gradient compression for data-parallel distributed training." [Online]. Available: https://arxiv.org/abs/1712.02679

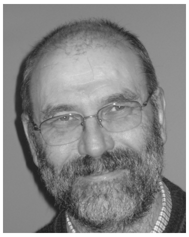
[24] B. Fleischer *et al.*, "A scalable multi-TeraOPS deep learning processor core for AI training and inference," in *Proc. Symp. VLSI Circuits*, Honolulu, HI, USA, 2018.

[25] C. Lehmann, M. Viredaz, and F. A. Blayo, "A generic systolic array building block for neural networks with on-chip learning," *IEEE Trans. Neural Netw.*, vol. 4, no. 3, pp. 400–407, May 1993.

[26] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature*

Nanotechnol., vol. 8, pp. 13–24, Dec. 2013.
[27] G. W. Burr *et al.*, "Neuromorphic computing using non-volatile memory," *Adv. Phys. X*, vol. 2, no. 1, pp. 89–124, 2017.
[28] S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," *Proc. IEEE*, vol. 106, no. 2, pp. 260–285, Feb. 2018.
[29] G. W. Burr *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element," in *IEDM Tech. Dig.*, San Francisco, CA, USA, Dec. 2014, pp. 29.5.1–29.5.4.
[30] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," *Frontiers Neurosci.*, vol. 10, p. 333, Jul. 2016.
[31] T. Gokmen, M. Onen, and W. Haensch, "Training deep convolutional neural networks with resistive cross-point devices," *Frontiers Neurosci.*, vol. 11, p. 538, Oct. 2017.
[32] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *IEDM Tech. Dig.*, Dec. 2017, pp. 6.1.1–6.1.4.
[33] T. Gokmen, M. Rasch, and W. Haensch (2018). "Training LSTM networks with resistive cross-point devices." [Online]. Available: https://arxiv.org/abs/1806.00166
[34] L. Kull *et al.*, "A 10 b 1.5 GS/s pipelined-SAR ADC with background second-stage common-mode regulation and offset calibration in 14 nm CMOS FinFET," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2017, pp. 474–475.
[35] G. W. Burr *et al.*, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *IEDM Tech. Dig.*, Washington, DC, USA, Dec. 2015, pp. 4.4.1–4.4.4.
[36] G. W. Burr *et al.*, "Recent progress in phase-change memory technology," *IEEE J. Emerg. Sel. Topics Power Electron.*, vol. 6, no. 2, pp. 146–162, Jun. 2016.
[37] S. Kim *et al.*, "A phase change memory cell with metallic surfactant layer as a resistance drift stabilizer," in *IEDM Tech. Dig.*, Washington, DC, USA, Dec. 2013, pp. 30.7.1–30.7.4.
[38] W. W. Koelmans, A. Sebastian, V. P. Jonnalagadda, D. Krebs, L. Dellmann, and E. Eleftheriou, "Projected phase-change memory devices," *Nature Commun.*, vol. 6, Sep. 2015, Art. no. 8181.
[39] S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian, and E. Eleftheriou (2017). "Mixed-precision training of deep neural networks using computational memory." [Online]. Available: https://arxiv.org/abs/1712.01192
[40] S. Ambrogio *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, pp. 60–67, Jun. 2018.
[41] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories—Nanoionic mechanisms, prospects, and challenges," *Adv. Mater.*, vol. 21, nos. 25–26, pp. 2632–2663, 2009.
[42] G. Bersuker *et al.*, "Toward reliable RRAM performance: Macro- and micro-analysis of operation processes," *J. Comput. Electron.*, vol. 16, no. 4, pp. 1085–1094, 2017.
[43] J. Woo *et al.*, "Improved synaptic behavior under identical pulses using AlOx/HfO2 bilayer RRAM array for neuromorphic systems," *IEEE Electron Device Lett.*, vol. 37, no. 8, pp. 994–997, Aug. 2016.
[44] D. Jana *et al.*, "Conductive-bridging random access memory: Challenges and opportunity for 3D architecture," *Nanoscale Res. Lett.*, vol. 10, Dec. 2015, Art. no. 188.
[45] J. R. Jameson, P. Blanchard, J. Dinh, and E. al, "Conductive bridging RAM (CBRAM): Then, now, and tomorrow," *ECS Trans.*, vol. 75, no. 5, pp. 41–54, 2016.
[46] W. Chen *et al.*, "A CMOS-compatible electronic synapse device based on Cu/SiO2/W programmable metallization cells," *Nanotechnology*, vol. 27, no. 25, pp. 255202-1–2552029, 2016.
[47] Z. Fan, J. Chen, and J. Wang, "Ferroelectric HfO2-based materials for next-generation ferroelectric memories," *J. Adv. Dielectrics*, vol. 6, no. 2, p. 1630003, 2016.
[48] T. S. Böscke, J. Müller, D. Bräuhaus, U. Schröder, and U. Böttger, "Ferroelectricity in hafnium oxide thin films," *Appl. Phys. Lett.*, vol. 99, no. 10, p. 102903, 2011.
[49] M. H. Park *et al.*, "Ferroelectricity and antiferroelectricity of doped thin HfO2-based films," *Adv. Mater.*, vol. 27, no. 11, pp. 1811–1831, 2015.
[50] M. Jerry *et al.*, "Ferroelectric FET analog synapse for acceleration of deep neural network training," in *IEDM Tech. Dig.*, San Francisco, CA, USA, Dec. 2017, pp. 6.2.1–6.2.4.
[51] M. M. Frank *et al.*, "Analog resistance tuning in TiN/HfO$_2$/TiN ferrolelectric tunnel junctions," in *Proc. IEEE SISC*, San Diego, CA, USA, Dec. 2018
[52] J. Janek and W. G. Zeier, "A solid future for battery development," *Nature Energy*, vol. 1, Sep. 2016, Art. no. 16141.
[53] E. J. Fuller *et al.*, "Li-ion synaptic transistor for low power analog computing," *Adv. Mater.*, vol. 29, no. 4, p. 1604310, 2017.
[54] S. Agarwal *et al.*, "Resistive memory device requirements for a neural algorithm accelerator," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 929–938.
[55] O. Fujita and Y. Amemiya, "A floating-gate analog memory device for neural networks," *IEEE Trans. Electron Devices*, vol. 40, no. 11, pp. 2029–2035, Nov. 1993.
[56] T. Morie and Y. Amemiya, "An all-analog expandable neural network LSI with on-chip backpropagation learning," *IEEE Trans. Electron Devices*, vol. 29, no. 9, pp. 1086–1093, Sep. 1994.
[57] H. Choe, S. Lee, H. Nam, S. Park, S. Kim, and E. Y. Chung, "Near-data processing for machine learning," in *Proc. ICLR*, 2016, pp. 1–12.
[58] S. Kim, T. Gokmen, H.-M. Lee, and W. Haensch (2017). "Analog CMOS-based resistive processing unit for deep neural network training." [Online]. Available: https://arxiv.org/abs/1706.06620
[59] S. Raschka, *Python Machine Learning*. Birmingham, U.K.: Packt Publishing, 2015.
[60] G. Freeman *et al.*, "Performance-optimized gate-first 22-nm SOI technology with embedded DRAM," *IBM J. Res. Develop.*, vol. 59, no. 1, pp. 5:1–5:14, Jan./Feb. 2015.

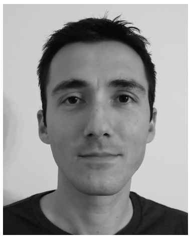
ABOUT THE AUTHORS



Wilfried Haensch (Fellow, IEEE) received the Ph.D. degree in the field of theoretical solid state physics from the Technical University of Berlin, Berlin, Germany, in 1981.

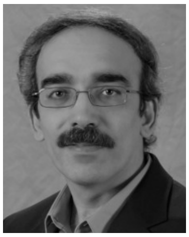
He started his career in Si technology in 1984 at SIEMENS Corporate research, Munich, Germany. There he worked on high field transport in MOSFETs and later in DRAM development and manufacturing. In 2001, he joined the IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, to lead a group for novel devices and applications. He was responsible for the exploration of device concepts for future technology nodes and new concepts for memory and logic circuits, including 3-D integration, early FinFET work, and the exploration of carbon nanotubes for VLSI circuits. He also was active in CMOS integrated silicon photonics to provide high-bandwidth low-cost links for future compute systems. He is currently responsible for novel technologies for neuromorphic computation with emphasis on exploring memristive elements (such as PCM, RRAM, FeRAM, etc.) in neural network arrays. He is the author of a text book on transport physics and author/coauthor of more than 150 publications.

Dr. Haensch was awarded the Otto Hahn Medal for Outstanding Research in 1983.



Tayfun Gokmen received B.S. degrees (double major) from the Department of Physics and the Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey, in 2004. As a Francis Robbins Upton Fellow, he received the Ph.D. degree from the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, in 2010, studying 2-D electron systems in the quantum Hall regime.

After working as a Software Developer at Bloomberg L.P. for a year, in 2011, he joined IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, as a Postdoctoral Researcher in the photovoltaics program. IBM group demonstrated the world's first copper, zinc, tin, sulfur, and selenium (CZTSSe)-based solar cell that can perform above 10% efficiency in 2011. In 2013, after being appointed as a Research Staff Member at IBM and then IBM Research AI, he began working on projects focused on exploring new hardware solutions for machine learning and artificial intelligence. He proposed the concept of resistive processing unit (RPU) devices that can accelerate machine learning, specifically deep neural network training algorithms, by many orders of magnitude compared to conventional digital approaches. He has 20 pending/issued patents, coauthored a book chapter, and published over 50 papers in various fields ranging from condensed matter physics, solar cells, to machine learning.



Ruchir Puri (Fellow, IEEE) is CTO and Chief Architect, IBM Watson AI, Yorktown Heights, NY, USA and an IBM Fellow, who has held various technical, research, and engineering leadership roles across IBM's AI and Research businesses. He has been an Adjunct Professor at Columbia University, New York, NY, USA and a visiting scientist at Stanford University, Stanford, CA, USA. He was honored with John Von-Neumann Chair at the Institute of Discrete Mathematics, Bonn University, Bonn, Germany. He is an inventor of over 50 U.S. patents and has authored over 100 scientific publications on software, hardware automation methods, and optimization algorithms.

Dr. Puri has been an ACM Distinguished Speaker, an IEEE Distinguished Lecturer, and was awarded 2014 Asian American Engineer of the Year.