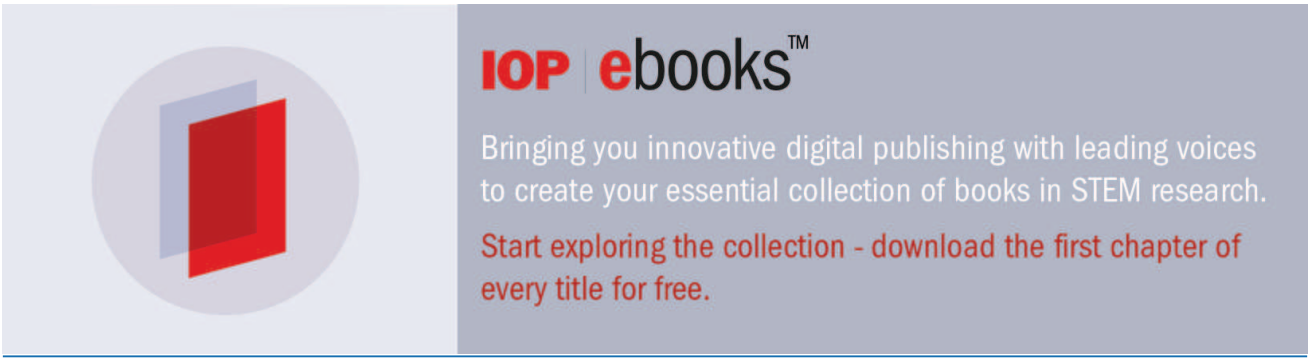
Recent progress in analog memory-based accelerators for deep learning



OPEN ACCESS
IOP Publishing
Journal of Physics D: Applied Physics
J. Phys. D: Appl. Phys. **51** (2018) 283001 (27pp)
https://doi.org/10.1088/1361-6463/aac8a5

Topical Review

Recent progress in analog memory-based accelerators for deep learning

Hsinyu Tsai, Stefano Ambrogio, Pritish Narayanan, Robert M Shelby and Geoffrey W Burr

IBM Research–Almaden, 650 Harry Road, San Jose, CA 95120, United States of America

E-mail: gwburr@us.ibm.com

Received 8 February 2018, revised 25 April 2018
Accepted for publication 30 May 2018
Published 21 June 2018

Abstract

We survey recent progress in the use of analog memory devices to build neuromorphic hardware accelerators for deep learning applications. After an overview of deep learning and the application opportunities for deep neural network (DNN) hardware accelerators, we briefly discuss the research area of customized digital accelerators for deep learning. We discuss how the strengths and weaknesses of analog memory-based accelerators match well to the weaknesses and strengths of digital accelerators, and attempt to identify where the future hardware opportunities might be found. We survey the extensive but rapidly developing literature on what would be needed from an analog memory device to enable such a DNN accelerator, and summarize progress with various analog memory candidates including non-volatile memory such as resistive RAM, phase change memory, Li-ion-based devices, capacitor-based and other CMOS devices, as well as photonics-based devices and systems. After surveying how recent circuits and systems work, we conclude with a description of the next research steps that will be needed in order to move closer to the commercialization of viable analog-memory-based DNN hardware accelerators.

Keywords: analog memory, non-volatile memory, hardware accelerators, deep learning

(Some figures may appear in colour only in the online journal)

1. Introduction

Over the past five decades, information technology has been transformed by the virtuous combination of three intersecting trends: Moore's law, Dennard scaling, and the von Neumann (VN) architecture. Moore's law described exponential reductions in the cost per transistor [1] that then drove similarly exponential increases in the number of transistors per wafer, making multi-billion-transistor microprocessors (CPUs) and graphical processing units (GPUs) not just possible but profitable. Dennard scaling supplied a set of 'scaling laws' [2] that allowed those smaller transistors to be, fortuitously, both faster **and** lower in power. Also, the flexibility of the 'stored program' VN architecture allowed programmers to build a wide diversity of complex computational systems by leveraging these CPUs and GPUs as modularized 'building blocks'.

Over the past few years, this intersection of virtuous trends has begun to dissipate. Device scaling has slowed due to power- and voltage-considerations. It has become difficult (and thus extremely costly) to guarantee perfect functionality across billions of devices. Finally, the time and energy spent transporting data between memory and processor (across the so-called 'von-Neumann bottleneck') has become problematic, especially for data-centric applications such as real-time image recognition and natural language processing [3].

One avenue for continuing to evolve the capabilities of future computing systems is to take inspiration from the human brain. Characterized by its massively parallel

architecture connecting myriad low-power computing elements (neurons) and adaptive memory elements (synapses), the brain can readily outperform modern VN processors on many tasks involving unstructured data classification and pattern recognition. By taking design cues from the human brain, neuromorphic hardware systems could potentially offer strong potential as an intriguing non-VN computing paradigm supporting fault-tolerant, massively parallel, and energy-efficient computation [4].

However, the number of different projects and proposals—many of them completely distinct from each other—that are now described as 'neuromorphic' computing has grown very large. (Recent surveys of neuromorphic hardware end up listing hundreds or even thousands of different citations [5, 6].) Many of these efforts involve circuits, sometimes including novel devices, that attempt to carefully mimic something that we can currently observe with moderate accuracy within the brain, usually at the scale of a few neurons. This could be the exact neural/synaptic response, precise local connection patterns, or local learning rules such as spike-timing-dependent-plasticity [7–14]. Other efforts involve new software algorithms partly or completely inspired by the architecture of the brain [15, 16]. The motivations for such neuromorphic hardware research can range from improving our fairly limited understanding of exactly how our brains function, to the hope of engineering computers that could potentially operate at ultra-low power through sparse utilization (in both time and space) of computational resources that are tremendously large, in both sheer number and interconnectivity.

We recently surveyed the potential role that novel analog memory devices could play in these areas [17] (also see earlier assessments by other authors [18–20]). However, in this paper, we refine our scope much more tightly, focusing on the use of analog memory devices to build neuromorphic hardware accelerators for deep learning [21].

Artificial neural networks (ANNs), first conceived in the mid-1940s to mimic what was then known about neural systems [22, 23], perform computations in a naturally parallel fashion. Modern graphical processing units (GPUs) have greatly increased both the size of the networks and the datasets that can be trained in reasonable time, giving rise to deep learning, or deep neural networks (DNNs) [24–26]—essentially, ANNs with many layers of neurons. Over the past few years, DNN performance has improved—on tasks such as classifying images [27], understanding speech [28], playing video games [29] and complex board games [30], and translating between languages [31–33]—to near-human (or sometimes even better than human) capabilities. More importantly, these developments have allowed DNN systems to become commercially pervasive—influencing social media sites, shopping and recommender systems, automated call centers, banking and finance, numerous cloud-based computing applications, and even our mobile phones and living rooms.

While some researchers occasionally attempt to connect DNNs back to biology [34], most deep learning practitioners do not concern themselves too much with neuromorphism. They are primarily focused on maximizing performance while finessing the limitations of commercially available VN hardware, which up until recently has meant hardware that was originally designed for something other than deep learning [35]. However, the intense interest in deep learning has led to research on [36, 37] and the introduction of [38] custom-ASIC (application specific integrated circuit) chips for deep learning. While some of these chips also double as neuromorphic hardware [39], these efforts primarily focus on re-imagining the GPU as if it had been expressly designed for deep learning. Leveraging conventional digital circuit design techniques, numerous design teams are seeking to deliver hardware acceleration for high energy-efficiency, high throughput (Tbit/second), and low latency DNN computation without sacrificing neural network accuracy [37, 40]. Thus, it is critical for researchers working on analog-memory-based hardware accelerators to both understand and take into account the advances that can be expected to arrive soon with such digital accelerators.

In the first section of this paper, we briefly overview deep learning and the major application opportunities for DNN hardware accelerators. Then we briefly discuss the research area of customized digital accelerators [36, 37, 40]. We discuss how the strengths and weaknesses of analog memory-based accelerators match up to digital accelerators, and attempt to identify where the most promising future hardware opportunities might be found. We survey the extensive but rapidly developing literature on what would be needed from an analog memory device to enable such a DNN accelerator, and then summarize recent progress using various analog memory candidates. These include NVMs such as resistive RAM (RRAM) and memristors, phase change memory (PCM) devices, Li-ion-based devices, capacitor-based and other CMOS devices. In addition to these efforts focused on integrating analog resistive-type electronic memories onto CMOS wafers, we also survey recent work on photonics-based devices and systems and discuss their potential impact on deep learning accelerators. After surveying recent work in advancing and demonstrating the circuits and systems behind analog-memory-based accelerators, we conclude with some thoughts on the next research steps that will be needed to move closer to commercializing viable analog-memory-based DNN hardware accelerators.

2. Overview of deep learning

In this section, we briefly discuss the basic computational needs of deep learning, including both forward inference and training with the backpropagation algorithm. Readers interested in a more complete overview should consult other recent tutorials in the hardware accelerator space [21, 37, 40]; those interested in truly learning the field should consult one of the many excellent online resources [41–44].

In general, the topology of a deep neural network is fixed by a designer before any training occurs. The size of the first layer—e.g. the number of neurons in the input layer—is typically chosen to match the size of the incoming data of interest: the pixels of a standard-size image, reduced audio content from digitized speech, encoded letters or words from

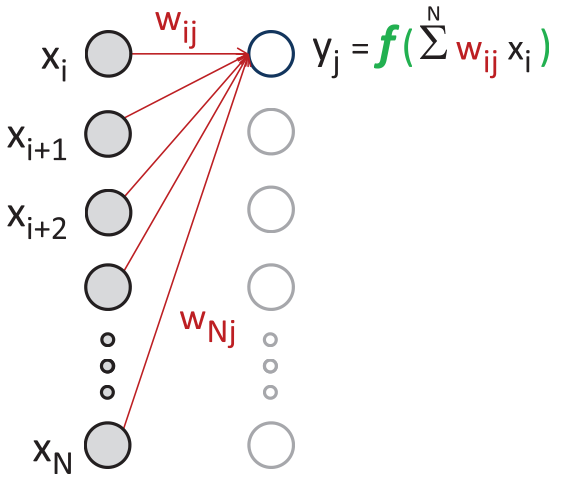


Figure 1. An important computational task for DNN forward inference is a vector-matrix-multiply (VMM), in which a vector of neuron excitations, x_i, is multiplied by a matrix of weights, w_{ij}, via a multiply-accumulate (MAC) ($\Sigma w_{ij}x_i$), and then put through a nonlinear squashing function ($f()$), to generate the new vector of neuron excitations for the next layer, y_j.

a written document, etc. The size of the output layer is chosen based on the task the DNN should accomplish, such as classifying an image into one of a number of pre-defined classes ('this is a handwritten seven', 'this is a border collie'.) or an output vector of interest. Examples of the latter include Mel-frequency cepstral coefficients (MFCCs) for speech recognition [28] or a vector of predicted probabilities of the next letter or word in a sentence.

Forward inference is the evaluation of what an already-trained neural network 'thinks' of one or more new data-examples, using weights that have already been optimized by training. It turns out that, despite the apparent complexity of deep learning [44], the set of computational tasks involved in this computation is not very large. Better yet, most of the computational effort is spent implementing only a small subset of those tasks. Figure 1 shows one of the most important of these computational tasks: the vector-matrix-multiply, or VMM. (Here we note that while there are many different DNN architectures, in general most differ only in how many 'balls' and 'sticks' (neurons and weights) are involved and how they are organized. In contrast, the actual operations performed *with* those resources are reasonably consistent across different DNN models.) Many DNNs contain some variant of a VMM, in which a vector of neuron excitations, x_i, must be multiplied by a matrix of weights, w_{ij}, generating a new vector of neuron excitations for the next layer, y_j. This breaks down into a series of multiply-accumulate (MAC) operations ($\Sigma w_{ij}x_i$), followed by a nonlinear squashing function, $f()$.

From a neural network perspective, f is very important. Without this nonlinearity, forward-evaluate of a multi-layer neural network of any number of layers would simply collapse into a single linear equation. From a computational perspective, evaluating the squashing function takes much less effort than the preceding MAC. The newest DNNs tend to use many 'rectified linear units', or ReLU functions. ReLU is a simple piece-wise linear function with only two segments: one along the x-axis, outputting zero for any input sum that is negative; and a second segment along the diagonal $f(x) = x$ directly passing any positive sum as the output. The ReLU helps avoid problems stemming from saturating excitations, and also helps keep gradients from vanishing for deep networks. However, recurrent networks such as long short term memory (LSTM) [45] and gated recurrent units (GRUs) [32]—which tend to suffer from gradients that explode instead of vanish—still tend to use saturating nonlinearities such as the logistic or hyperbolic tangent functions.

The neuron excitations at the input layer come directly from the data-example being evaluated, including any pre-processing so that it looks 'just like' the data-examples that were used for training. The network is evaluated serially from input to output. (Pipelining can be introduced, so that layer #1 can already start working on data-example #2 while layer #2 is still working on the excitations just passed to it from layer #1.) At the output layer, a softmax operation is frequently performed. Here, each raw excitation y_j is put through an expanding nonlinearity (such as an exponential), and the intermediate result, $q_j = \exp(y_j)$, is then normalized by the sum of all such intermediate results across the entire output layer. This produces outputs that are guaranteed both to fall between zero and one, and to sum up to one as well. The softmax operation produces a vector of probabilities, representing the predictions (or the guess/vote) of the pre-trained DNN, given that particular data-example.

Training is the process of tuning the weight matrices to a set of values that can provide good performance (e.g. accurate classifications, predictions, translations, high game scores, etc). This begins with the same forward inference step described above. At first, however, since the weights are randomly chosen, the output result is likely to be nowhere near the desired target. The backpropagation algorithm [46] is a supervised training algorithm that attempts to tune the weights by altering between forward inference of training data for which the desired output vector (or label) is known, reverse propagation of errors based on the difference between the current guess of the DNN and the correct label or other 'ground truth', and then weight update of each weight based on the excitation it saw during forward inference and the error it induced using that excitation, as computed by reverse propagation.

The additional computation associated with training is also dominated by a MAC or VMM, this time proceeding from right to left. Figure 2 shows the reverse propagate step, as a vector of errors (δ_j) is multiplied by the transpose of the original weights w_{ij}. Instead of putting this sum through a nonlinear squashing function, however, the sum is multiplied by the derivative of the squashing function as evaluated at the original excitation, x_i. This formula arises from the use of the chain-rule to compute the derivative of an 'energy function', E, for the overall DNN as a function of each individual weight, w_{ij} [44]. If forward inference produces a guess, y, which should have been g, we can choose an energy function that is minimized only when $y = g$. Backpropagation then allows the DNN to compute the derivative with respect to each

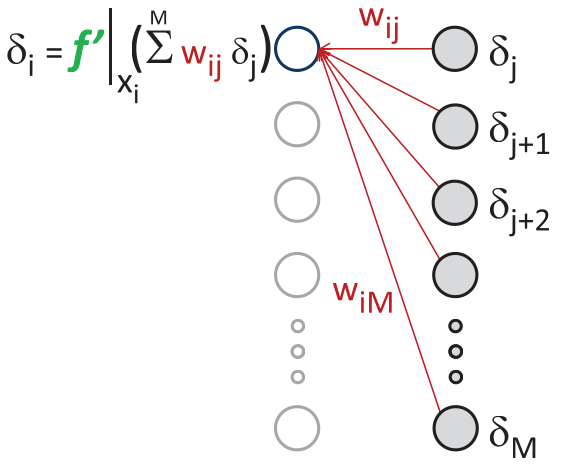


Figure 2. The reverse propagate step in training involves a second MAC, between a vector of errors (δ_j) and (the transpose of) the same weight matrix (w_{ij}).

weight, or how that weight needs to change in order to allow the DNN to do a better job the next time it sees that same training example.

As we proceed from the output of the network back towards the input, the very first δ vector typically comes from the raw difference between the network output y and the label vector, g. Interestingly, it turns out that multiplying this difference by the derivative of the squashing function used at the output neuron can cause problems. One way to deal with this is to train against 'cross-entropy loss', so that the underlying energy function applies a logarithm to the softmax outputs. The advantage of this is that the derivative at this output neuron cancels out, and thus training does not get stuck even if the initial choice of weights produces excitations at the output layer that are not just wrong, but large in magnitude. Since such large output excitations start out already in the regime where the derivative of f gets very small, multiplying by this derivative would strongly suppress the very corrections that the network will be applying to fix this. As a result, training with this final-layer derivative can be very slow to get started.

Another method is to skip the softmax, the logarithm, and the derivative at the output layer. While this second approach technically does not implement the exact chain-rule, it seems to work fine for simple networks. While this can simplify an implementation, one would need to confirm that this second approach would still work on the very deep networks that have become popular in the past few years.

Reverse propagate takes place throughout the network from output towards input, and can terminate once the accumulated error values (δ_j) have been delivered to the first hidden layer. Weight update for each weight is then the product of the original upstream neuron excitation, x_i, and the downstream neuron's error, δ_j. Typically this is scaled by a fairly small number, η, called the learning rate. Note that, during training, each neuron needs to hold onto its original excitation, x_i, until it is used for weight update. In contrast, if we are only performing forward inference, that same excitation can be discarded as soon as it has contributed into all the associated MAC operations. As a result, introducing pipelining is somewhat more complicated for a full training implementation than for a forward inference-only implementation.

There are many subtleties to deep learning beyond the above discussion. The learning rate (and other 'hyperparameters') must be chosen carefully, not too small so that learning is too slow, but not so large that the changes induced by each training example inhibit convergence. Often learning rate is modified during training. The set of training data must be chosen extremely carefully in order to represent the intended test data, and then divided into a subset of verification data in order to evaluate the performance of the network while it is being trained. The training data should preferably be supplied to the network in a random order (this is the 'stochastic' in 'stochastic gradient descent').

The initial distribution of random weights needs to be chosen carefully, so that the accumulated sums ($\Sigma w_{ij} x_i$) land in a useful region of the squashing function. Then, as the weights get trained, these 'internal covariate' distributions will shift [47], which can end up moving the sums out of that regime and thus requiring corrective steps such as 'weight normalization' [48] or 'batch normalization' [47].

Probably the most important aspect of deep learning to convey is the realization that the mathematics of the training algorithm is only guaranteed to compute the weight adjustments that will move the DNN towards better performance on exactly those example(s) that were just examined with forward inference. There is no guarantee that adjusting the weights for the next set of training examples (often referred to as a 'mini-batch') will not completely ruin the improvements for the first set. Fortunately, if the learning rate is small and the training examples are repeatedly supplied to the network, stochastic gradient descent tends to get better on the entire training set.

At that point, however, the algorithm is going to attempt to perfect this, driving the energy-function to zero, at which point the network has literally memorized the training dataset. In contrast, the commercial interest in DNNs stems from good generalization performance: how well can a trained DNN handle handwritten digits, pictures of dogs, spoken sentences, written sentences, etc, that it has *never* seen before. A deep learning practitioner spends considerable effort finessing the subtle but important distinction between the mathematical goal of the DNN (memorizing the training set) and the actual desired engineering goal (good generalization performance on a much larger and effectively unknowable 'test' set). Tricks such as dropout [49] and early stopping [28] are some of the many approaches used to maximize the generalization performance of DNNs. Much more information can be obtained from books [44], online resources [41–43] and conferences such as ICML and NIPS.

Table 1 lists the two main opportunities for hardware accelerators—those designed for just forward inference of pre-trained DNNs, and those designed to accelerate DNN training. For forward inference, there is a set of hardware opportunities in highly-power-constrained environments such as internet-of-things (IoT) devices, edge-of-network devices and sensors, mobile phones, and autonomous vehicles. There are also numerous forward inference opportunities in the cloud or server room, as described quite well in [38].

Table 1. Hardware accelerator opportunities break into two major application areas: hardware for the evaluation of pre-trained DNNs (forward inference), either in extreme-power-constrained environments (IoT, edge-of-network, autonomous vehicles, etc) or in the server room [38]; and hardware for DNN training, typically performed in a distributed manner in server room harnessing many compute nodes working in either a data– or model–parallel fashion [44].

Application	Critically requires…	Might be OK with only …
Forward inference	• High energy efficiency (TOP/s/W) • High areal efficiency (TOP/s/mm^2) • Low latency	• Modest throughput (TOP s^{-1})
Training	• High throughput • Efficient use of network bandwidth between chips	• Modest latency

Despite these distinct opportunities, the performance aspects that are likely to be more or less important for forward inference are relatively similar. While throughput in terms of tera-operations per second (TOP s^{-1}) or equivalently, in data-examples-per-second, is always important, a forward inference application is quite likely to value low latency over throughput. This is likely to be as true in edge-computing (an IoT sensor reading just changed, an autonomous vehicle must respond to its sensors, etc) as in the cloud (a customer is waiting for this particular search/recommendation/translation/recognition result). While both scenarios always profit from lower power, obviously edge systems will require extremely power-efficient computation.

For the near future, it appears as if training can be expected to take place mostly in the cloud. In the future, there might certainly be opportunities for training in the field—but this would be much easier if the problems of 'catastrophic forgetting' during DNN training [50] can be solved. This would allow an edge-based training chip to update a network on new data-examples without sacrificing the performance on training examples that are no longer readily available. Typically, training is now performed in a distributed manner using many parallel workers, either working on the same model with different data (*data parallelism*) or on multiple instances (*model parallelism*) that can improve performance by averaging the different model outputs [44]. For data parallelism, it is important that the necessary communication between the workers over an interconnecting network does not itself become the bottleneck that determines the total time needed for training [51]. This then favors approaches that can harness the improved network performance (e.g. generalization accuracy) offered by having multiple workers while using the interconnect between the workers wisely [52]. The overall goal of a hardware accelerator is to complete training in a shorter total time. Thus, in contrast to forward inference, latency on any one training example is not as critical as raw throughput. Power and area-efficiency are important simply as a means to packing as much compute as possible into each card-slot of a given standardized volume and power envelope (e.g. 75 W or 300 W).

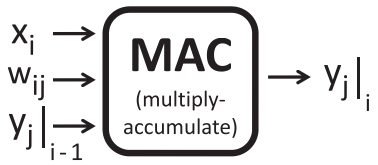


Figure 3. In a digital accelerator, MAC operations are implemented by processing elements that work with three pieces of data (x_i, w_{ij}, and the partial sum so far $y_j|_{i-1}$) in order to produce the new partial sum, $y_j|_i$.

3. Digital accelerators for deep learning

The recent history and the apparent emergence of deep learning owes much to graphical processing units (GPUs). Deep learning can be considered as the fortuitous convergence of a scalable learning algorithm (e.g. one that drives better performance as the models and the training data get larger), the easy availability of vast amounts of training data via the Internet, and the raw computation needed to train and implement very large networks. The first two components have been available for 30 and 20 years, respectively. The final ingredient was the fast, parallel computation provided by GPUs [26].

In a GPU implementation, the VMM operations described in figures 1 and 2 are turned into matrix-matrix, or even into tensor-tensor operations [44]. This allows mini-batches of examples to be computed at the same time, with the MACs for each layer taking place in parallel on the many SIMD (single-instruction multiple-data) processors within a modern GPU. GPUs are particularly efficient when multiplying large matrices of roughly unity aspect ratio, and thus the size of the mini-batch is chosen in order to fully utilize either the compute or memory resources of the GPU. (Note, however, the inherent tension between the large mini-batch sizes that optimize computation, and the small mini-batch sizes that would help keep latency low.) The advent of sophisticated layers of middleware and hardware drivers such as CuDNN have allowed deep learning practitioners to focus solely on high-level scripting languages such as TensorFlow and Caffe, yet still harness the full computational capabilities of GPUs. As we mentioned earlier, the fact that only a fairly small set of fundamental operations are involved has helped greatly.

Research in custom digital accelerators primarily focuses on re-designing a GPU-like processor, but as if it had been designed explicitly for deep learning. This can either be done with full ASIC designs [38, 53, 54] or with more flexible field-programmable-gate-arrays (FPGAs) [55]. The fundamental building block for the critical MAC operation looks something like figure 3: a processing element that receives three pieces of data (x_i, w_{ij}, and the partial sum so far $y_j|_{i-1}$) and outputs the new partial sum, $y_j|_i$. While this seems rather simple, there is a strong incentive to carefully organize the complex 'systolic' data-flow into and among these processing elements [37, 40].

The overarching concern driving all deep learning accelerators is the enormous cost of moving large amounts of data over any long distance. For example, bringing data onto a processor chip from off-chip memory is much more expensive than retrieving it from a local register. One way to reduce the volume of incoming data is to reduce the precision (number of bits)

with which the data is encoded. This can be done with fixed-point arithmetic (integers with a scale divider to help tune the dynamic range where it is needed) or fewer bits in the mantissa and/or exponent in a floating-point number. Precision in forward inference implementations has been aggressively tuned all the way down to 1 or 2 bits, using binary $(0, 1)$ or trinary $(-1, 0, 1)$ encoding [56]. Much more typical are weights encoded using 8 bits. One of the advantages here is that the encoding can be introduced during the training process and its impact both measured and minimized during training. A similar approach is to 'prune' the network, eliminating neurons during the final stages of training that can be identified as unimportant [57]. Individual weights that are unimportant can even be removed, if the matrix can be stored and delivered to the accelerator efficiently using sparse matrix techniques. Alternatively, unimportant weights that cannot be removed can be set to zero and the circuitry simply instructed to skip over such weights, eliminating unnecessary computations. Compression techniques can reduce the on-chip bandwidth, at some increase in computation associated with decompression [57]. All these approaches can help reduce the amount of data that must be brought on-chip in order to feed the MAC units shown in figure 3.

While forward inference appears to work for many DNNs even at low precision, DNN training appears to call for higher precision in order to avoid sacrificing significant accuracy. One issue with DNN training is the large contrast between the absolute magnitude of the weights and the magnitude of the tiny weight changes requested by a large mini-batch. As training proceeds, weight updates naturally get smaller, both because learning rate is typically reduced during training, but inherently as well, because the errors are getting smaller as the network does a better job on each example. At any given precision, there will be a requested weight update that is effectively smaller than the least significant bit (LSB). Various tricks such as stochastic rounding can help reduce the precision beyond this limit while still achieving good training accuracy [58]. Other tricks are being developed to help reduce the amount of data conveyed between the various chips ('workers') participating in distributed training.

An important part of optimizing the data-flow into and among these processing elements is designing the hardware to match the inherent re-use of data within the algorithm. A family of DNN networks offering many opportunities for such data re-use are convolutional neural networks (CONV-net) [24]. As discussed earlier, the main difference between various DNNs is how the 'balls' and 'sticks' are organized. Figure 4 shows two important types of layers within DNNs: the fully-connected (FC) layer (at left, part (a)), in which every pair of neurons across the two neighboring neuron layers shares a unique weight, and a CONV-layer (figure 4(b)). (Note that neither of these configurations have any connections *within* layers.)

A CONV-layer contains many neurons, often organized into planes. For instance, the input color images to a CONV-net trained on ImageNet contains three planes (red, green, blue). In most CONV-nets, the number of planes increases rapidly as one moves away from the input layer. Instead of a unique weight between all possible upstream and downstream neurons, there are small weight kernels (frequently a 3×3 array

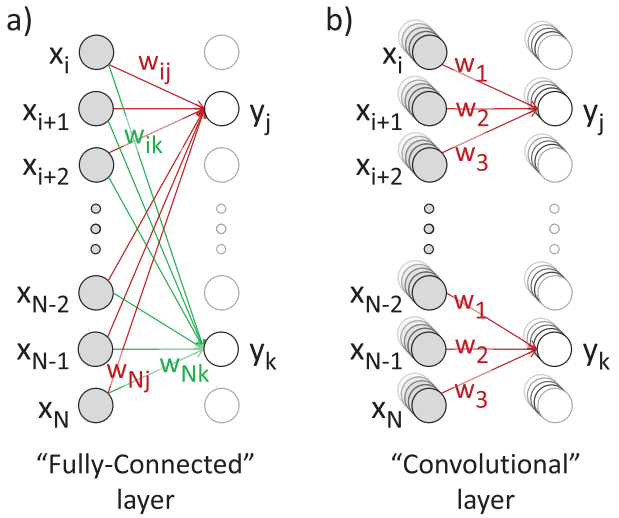


Figure 4. (a) In a fully-connected (FC) DNN layer, every pair of neurons across the two neighboring neuron layers shares a unique weight (but no connections within layers). (b) In contrast, a convolutional (CONV) layer contains many neurons, often organized into planes.

for each input plane) which are convolved across the input planes to produce the output planes. Since the same kernel is needed in order to produce y_j from x_i, x_{i+1}, x_{i+2} and y_k from x_{N-2}, x_{N-1}, x_N, there is significant data reuse. As the number of planes increases from CONV-layer to CONV-layer, pooling layers and larger strides (e.g. stepping the convolutional kernel in jumps of 2 pixels at a time rather than just 1) help quickly reduce the lateral dimensions. Convolution makes enormous sense for image processing, inherently allowing a system to learn and apply specific kernels to recognize features within images independent of the specific location within the image. Much of the success of deep learning has come from the rapid progress of CONV-nets on very impressive image processing tasks [27, 42].

A few years ago, CONV-nets such as AlexNet [27] included multiple FC-layers near the output layer. The number of unique weights in a CONV-layer is quite low, sometimes $1000\times$ smaller than the number of neurons. Given that memory and memory bandwidth are the first things one runs out of in a GPU- or digital-accelerator implementation, the trend by DNN practitioners has been to increase the number of CONV-layers and decrease the number of FC-layers to the bare minimum [59].

4. Analog-based accelerators

As we noted in the previous section, the most important priority in designing any DNN accelerator is minimizing both the amount of data that needs to be moved, and the distance that it needs to be moved. As a result of this realization, a large fraction of the activity in digital accelerators has focused primarily on optimizing the computations behind memory-light DNN models such as CONV-nets [24]. In fact, recent reviews of digital accelerators have focused solely on forward inference-only accelerators for CONV-nets [37, 40].

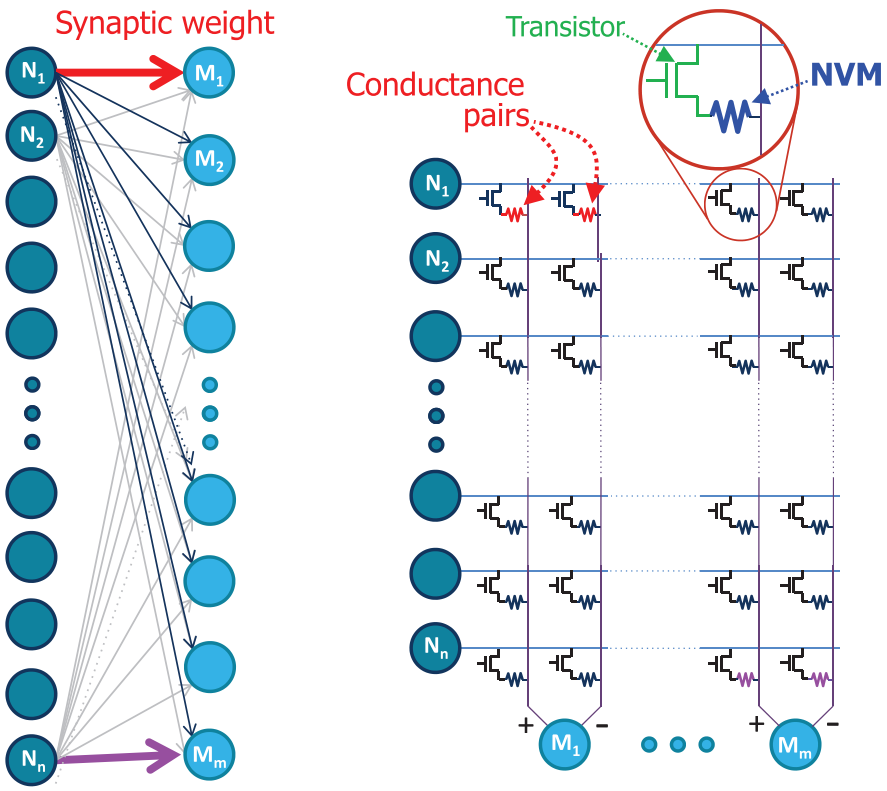


Figure 5. An analog-based deep learning accelerator uses a crossbar memory array to store the values of the weight matrix in an analog fashion, so that the VMM and MAC operations inherent to DNN computation can be performed in a highly parallel and energy-efficient manner.

This is great for applications such as image processing, but less ideal for other applications that depend on FC-layers—including families of recurrent neural networks mentioned earlier such as LSTMs [45] and GRUs [32], which have fueled recent advances in machine translation, captioning, and other natural language processing. Fortunately, in the same way that digital accelerators seem uniquely well-suited for CONV-layers, analog-memory-based accelerators seem to be uniquely well-suited for FC-layers.

The heart of any analog-based accelerator is a memory array that can store the values of the weight matrix in an analog fashion (figure 5). Weights are encoded into device conductances (inverse of resistance), typically (but not always) using NVM devices. In analog-based accelerators, the MAC operations within each VMM are performed in parallel at the location of the data, using the physics of Ohm's law and Kirchhoff's current law. This can completely eliminate the need to move weight data at all.

Conventionally, NVM devices are used as digital memory devices. A high conductance or SET state might represent a digital '1' and a low conductance or RESET state might represent a '0.' In a crossbar array of such memory cells (figure 5), access devices allow addressing of a single memory cell by appropriate activation of word- and bit-lines, for reading device conductance to retrieve stored data and for programming device conductance to update the stored digital data values.

Such an NVM array can readily be used as an accelerator for deep neural networks. As shown in figure 5, each FC neural network layer—connecting N neurons to M neurons—maps reasonably well to a crossbar array of $N \times M$ weights. (As we will describe below, typically we use multiple conductances per weight.) For forward inference, signals are applied to the horizontal row-lines of the array-core, and a small trickle read current is generated in each device along the row, just as they were in a memory application.

However, unlike the memory application, we do not activate just one row at a time, and uniquely sense each small trickle currents at the ends of each column-line to retrieve digital data. Instead, we will activate all the rows simultaneously, and allow these trickle currents to aggregate along the entire column-line. If we are careful to encode each upstream neuron activation into the voltage that is applied to 'its' row, then Ohm's law at each stored conductance implements the multiplication between neuron excitation x and weight w (figure 6). Once Ohm's law has performed the multiply operation, then the summation along the column-lines via Kirchhoff's current law implements the accumulate operation.

In order to be able to encode signed weights w using positive-only conductances G, we typically take the difference between a pair of conductances, so that $w = G^+ - G^-$. In some cases, we can use a 'shared' column of devices, or even a dedicated reference current instead of G^-. However, this requires that each device can be tuned both up and down in a

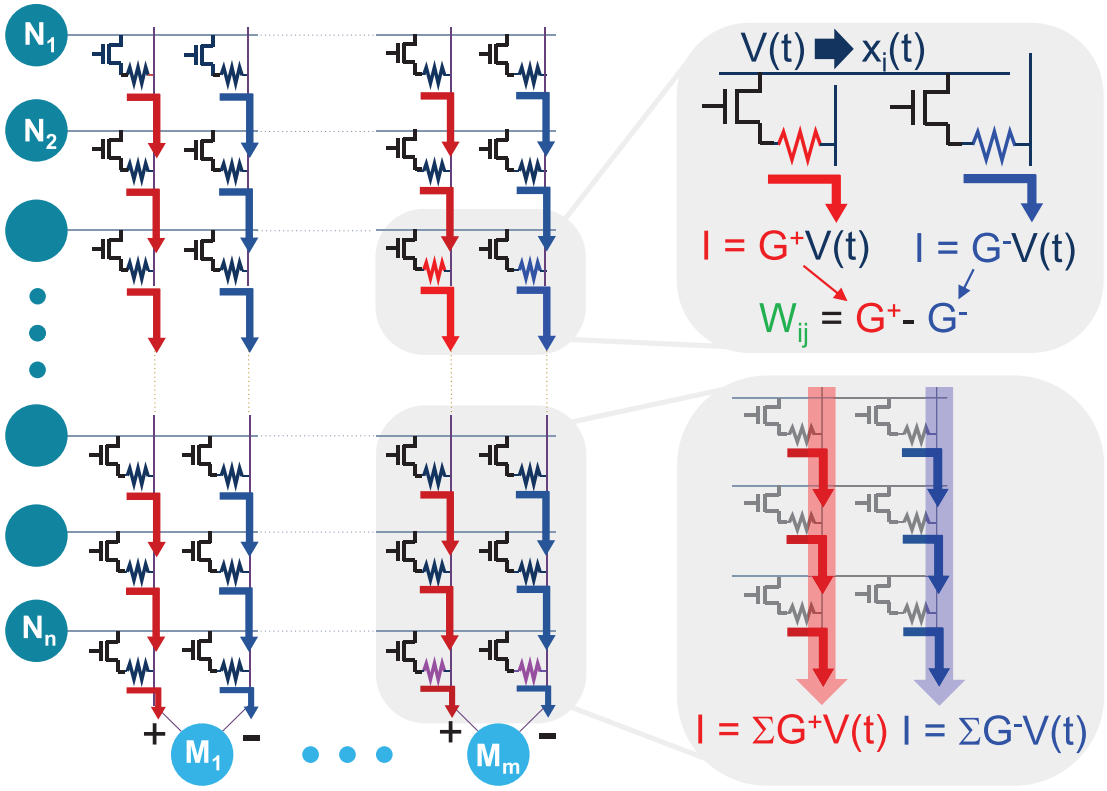


Figure 6. The physics of Ohm's law and Kirchhoff's current law allow the implementation of analog MAC operations, in parallel, at the location of the weight data.

gradual manner, which is not available for some well-known NVM devices such as PCM and filamentary RRAM.

Note that the neuron excitation can be encoded onto the voltages in one of two ways. If the x value is mapped to a unique voltage, then the instantaneous aggregated current along the column-line encodes the MAC result. While this can be measured as soon as the current stabilizes, there are a few drawbacks. First, a dedicated D/A converter is required at every row to supply the voltages, meaning that the target resolution must be specified at fabrication. Second, since the NVM devices could be read anywhere within a range of different read voltages, their I-V characteristics must now be highly linear. Finally, we have no remedy if the instantaneous power involved with activating all the row-lines simultaneously turns out to be excessive.

In contrast, by encoding the neuron excitation x into the duration for which a constant read voltage is supplied, many of these drawbacks are removed. We do not need any D/A converters, and the NVM device could be significantly non-ohmic because we are going to use only one read voltage. The signal pulse conveying the analog data within its duration can be manipulated across the chip using digital circuits, right up until the voltage conversion at the edge of the array to the desired read voltage. Since the data is no longer in the raw current, we do need to integrate the aggregated current (say, onto a capacitor) for some length of time. This also means that if instantaneous power were an issue, we could distribute the application of these pulses as needed within a slightly longer integration window. Additionally, resolution could be dynamically adjusted as needed by adjusting the maximum duration allowed. Note, however, that this does create an undesirable tradeoff between effective resolution with which excitations can be encoded, and the speed and latency of the VMM operations.

By turning an analog-memory-read operation into an in-memory-compute operation, we perform an entire VMM without any motion of weight data, and entirely in parallel. This is the most attractive feature of this analog-memory-based approach: this could potentially be both quite fast *and* quite energy-efficient. However, while most of the computation in deep learning are VMMs, there are more steps that are needed to turn this simple VMM operation into a viable DNN accelerator.

During forward inference each such NVM array performs all the MAC operations constituting the VMM for one FC-layer of a deep network. The outputs of the array must then be processed by neuron circuitry that applies a nonlinear squashing function (the $f()$ function from figure 1). We have now computed the excitation of this downstream neuron that is needed for the next layer of the neural network. However, the weights of this next layer are encoded within another crossbar array, sitting elsewhere on the chip. Thus, each hidden neuron within the network is implemented by circuitry sitting at the periphery of two different array-blocks. A first circuit collects column output and implements the nonlinear squashing function (such as a logistic function or its piecewise-linear (PWL)

approximation); and a second circuit then introduces this neuron activation into the corresponding row of the second array-block. The former represents the 'output' to a neuron from an upstream layer; the latter, that same neuron's 'input' to the downstream layer. A routing network must then be able to connect all columns of a first array-block to the corresponding rows of a second array-block, connecting the two halves of each hidden neuron with each other, preferably in a flexible and reconfigurable manner [21].

Alternatively, the column-portions of each neuron circuit can include an analog-to-digital converter (ADC) to convert aggregate excitations to digital representations, which can then be bussed to digital logic for processing steps such as the nonlinear squashing function [60]. The resulting excitations would then be bussed to the row-neuron circuits and converted from digital representations back to appropriate excitation pulses. While this approach offers the flexibility and familiarity of a digital bus, the need for high parallelization in processing each neuron layer mandates ADCs that are very fast, leading to significant power dissipation and silicon real-estate.

If the application is the implementation of a forward-inference accelerator, then once the routing network is able to pass data from one crossbar array to another efficiently, one need only apply the softmax operation at the output neurons (if desired) to compute the network output.

For training, things get more complicated. First of all, if we intend to perform training of any sort, the neuron excitations need to be stored temporarily—preferably within the upstream neuron circuitry to minimize energy spent transporting and storing this data. The training label must be made available and the raw δ corrections computed by subtraction at the output layer. Then for the reverse propagation of errors, we perform a very similar operation to forward inference, except that δ corrections from downstream neurons are applied to the 'south' side of the array, and the errors for the upstream layer are accumulated on the 'west' side. (This is effectively a VMM using the transpose of the original weight matrix.)

For stochastic gradient descent, each weight receives an update for each training example proportional to the backpropagated error for the downstream neuron and the activation of the upstream neuron during forward propagation. This is why these forward activations had to be stored, to have them available to combine with the backpropagated error in order to perform the weight update. For weights represented by pairs of NVM conductances, weight updates are typically performed by firing programming pulses at the NVM elements to increase or decrease their conductances. It is essential that this should be as fully parallel as possible, as the time required to individually program all conductances for each example would result in unacceptably long training times. Parallel weight updates are facilitated by schemes in which downstream and upstream neuron circuits independently fire programming pulses according to their knowledge of backpropagated error and downstream activation, respectively, resulting in the correct conductance programming when these pulses overlap in time [61, 62].

It is during this weight-update step that the imperfections of real NVM devices can cause serious problems. As the neural network examines each example from the training dataset, the backpropagation algorithm computes the weight changes needed to improve classification performance on that example, implementing gradient descent along the objective function designed to force the network outputs to match the target labels. For any particular weight which gets increased during this step, the network is quite likely to request, during training of some later example, a counteracting decrease. Many thousands of increases may be requested, and over some period of time, nearly but not quite the same number of decreases.

In an ideal world, these increase and decrease requests would exactly cancel. When they do not cancel, serious problems can arise. It turns out that neural networks have a surprising degree of tolerance for stochastic variability. If the cancellation of increases and decreases were to be random from synaptic weight to synaptic weight, or better yet, random over time, accuracy could still be reasonably high. Unfortunately, nonlinearity in the conductance response of real NVM devices means that at a given conductance, each conductance-increase pulse might *consistently* be more effective than the conductance-decrease starting from that same absolute conductance (or vice-versa). Since this is systematic across every single device in all the crossbar arrays, this means that the weight updates that are supposed to cancel do not. Worse yet, since the cancellation error has the same general trend on all weights at all times (typically towards weights of smaller absolute magnitude), touching a weight at all means that it invariably shifts in that same direction. And since the network is firing many hundreds if not thousands of update requests yet expecting that most of them will cancel, these weights are touched all the time. As a result of all this, neural network accuracy of NVM-based systems can markedly fail to match what would be expected of a GPU- or CPU-based system of the same network size. Our IBM colleague Tayfun Gokmen has shown that an asymmetry between the size of conductance increases and decreases as small as 5% can have a marked effect on accuracy [60].

The above discussion accentuated the efficiency of computing an entire VMM for each FC-layer in one time step. This situation, where each weight is used exactly once, turns out to exactly match the strengths of an analog-based accelerator. In contrast, an FC-layer is problematic for a digital-based accelerator, because the number of weights brought onto the chip is enormous yet there is minimal opportunity to be clever with data re-use. For a CONV-layer, the situation is exactly reversed. Since many excitations need to be multiplied by the same weights, an analog-based accelerator will either spend time to implement this (we apply each set of excitations one by one to the crossbar-array encoding the one copy of the weights), or area (we maintain multiple copies of the weights on different crossbar arrays, and route the excitations to the various copies). Either choice will inherently depress computational efficiency in units of TOP/s/mm^2. For training of a CONV-layer, since the weight updates for each weight are actually the sum of the $x\delta$ products across all the copies,

Table 2. As with the hardware accelerator opportunities, the requirements for analog-memory devices for deep learning accelerators break into the two same major application areas: devices for storing the weights of pre-trained DNNs (forward inference); and devices for performing *in situ* DNN training.

	Critically requires…	Might be OK with only…
Forward inference	• Long-term retention • Excellent conductance stability	• Modest endurance • Modest programming speed & energy
Both	• LRS $> 100\ \mathrm{k}\Omega$	• Inefficient programming between full LRS and full HRS
Training	• Multiple analog states • Highly linear & gradual conductance update • High endurance • Fast programming speed	• Modest retention

the complexity—orchestrating the all-reduce of the various contributions to each weight update, and the broadcast of the accumulated weight-update back out to the various copies—of implementing all this efficiently is extremely daunting, to say the least.

So digital accelerators are naturally good for layers with a lot of neurons per weight (like CONV-layers). Also, analog accelerators—if the effective precision is suitable and the data routing does not sacrifice the inherent efficiency of the crossbar-based VMM—will be naturally good for layers with a lot of weights per neuron (like FC-layers). As a result, one can expect that a hybrid analog/digital accelerator would be an ideal blend of these complementary characteristics, leading to the best of both worlds for DNNs that can benefit from a mix of various types of layers. An example would be CONV-nets in which the first layers are CONV-layers naturally suited to applications such as image processing, implemented on digital cores, which then feed highly-efficient analog cores implementing FC-layers for the final layers of the DNN. In the near term, the yet-to-be-answered research questions that must be addressed for analog-based DNN acceleration are effectively identical, whether the final goal is an all-analog accelerator or a hybrid analog-digital accelerator.

In the next section, we discuss the specific requirements of analog memory devices for the application of deep learning accelerators.

5. Requirements of analog memory devices

Analog-based accelerators promise significant improvement in speed and power. However, such improvements are useful only if the performance in terms of accuracy is reasonable. Ideally, training or inference results with analog MAC operations should produce comparable accuracy to a full software implementation with high precision weights stored as digital bits. A common method for studying how analog memory devices affect deep learning accuracy is to substitute ideal weights with values predicted from a single or ensemble analog device model. Such a model can include a wide range of non-ideal properties. For example, conductance change per programming pulse can be a nonlinear function of the current conductance state of the analog memory device, with a conductance that typically saturates at some maximum value. The response to input pulses can be very asymmetric depending on whether conductance is increasing or decreasing. There are also variations from device to device, and from one programming event to another for each device. Some devices may be defective, resulting in no response and either 'stuck on' or 'open' conductance values.

In this section, we review how the specifications of analog memory devices affect accelerator performance. We survey various proposals on how to mitigate such device limitations through altered algorithm or more sophisticated circuit designs. Table 2 summarizes the underlying device specifications that can be expected to be more (or less) important when seeking an ideal analog memory device for deep learning accelerators for both forward inference and training.

In order to benchmark accuracy for analog-based accelerators, a deep learning dataset that can be solved reasonably well with FC networks, such as MNIST, is commonly used. The MNIST hand written digit recognition dataset consists of 60 000 training examples and 10 000 test examples. The deep learning network chosen for benchmarking varies in terms of number of layers and neurons per layer, usually to accommodate the size of available device hardware. As a result, the target accuracy can differ from around 90–99%, depending on which network, how many layers and how many neurons per layer. It should be noted that MNIST is a much easier network to train than cutting-edge DNNs. Thus, success at training or inferencing MNIST must be considered as absolutely *necessary*, but in no way *sufficient* to predict success as a generic DNN accelerator.

5.1. System-level simulations

Gokmen *et al* [60] introduced the concept of a resistive processing unit (RPU) and identified several RPU device and system specifications, including minimum/maximum conductances, number of conductance steps, device non-linearity, weight update asymmetry, device-to-device variation, and noise. The specifications differ significantly from parameters typical for NVM technologies as the algorithm can tolerate up to 150% of noise in the weight updates and up to 10% reading noise. Even with the intrinsically high variability of states in RRAM due to the physical movement of ions, which limits its use as conventional memory, the intrinsic variability does not impose a major problem in this application because

the algorithm adapts its weights to accommodate device variations [63]. Impact on accuracy from time-dependent variation (TDV) in RRAM is more severe for high resistance synapses, and during backpropagation, because of the narrow distribution of resistances in a trained network, accuracy can be affected by TDV [64]. Endurance requirements are also relaxed as RPU devices also only need high endurance to small incremental conductance changes, rather than the large conductance changes needed for digital memory applications. On the contrary, a large number of conductance steps are required and weight update asymmetry (between conductance increase and decrease) becomes the most demanding specification, which is quite unlike any of the restrictions typically imposed upon conventional memory devices [60].

Chen *et al* [65] also looked into impact of device non-idealities with device models of $Pr_{0.7}Ca_{0.3}MnO_3$ (PCMO), conductive-bridging RAM (CBRAM) (Ag:a-Si), and TaO_x/TiO_2 RRAM. A sparse-coding feature extraction network was used as the benchmarking problem. The authors considered properties needed for array access/selection device and looked at the effects of device nonlinearity, variation, stochasticity, and limited dynamic range. Multiple analog memory devices (up to nine in one example) were used as one weight element to average out variability.

NeuroSym+ [66] provides a framework for modeling NVM-based networks, including similar device properties, and aims at evaluating system-level performance. The simulator yields circuit area, leakage power, latency, and energy consumption during training. A comparison was conducted among SRAM-based synapse, 'analog' NVM synapses, and 'digital' NVM synapses, where weights are stored as digital bits in NVM devices. SRAM showed advantages for online learning, analog NVM was found suitable for offline classification, and digital NVM was judged to be better for low standby power design.

Finally, Gokmen *et al* [67] discussed implementation of CONV-nets with RPU arrays. Device variability, noise, optimal array size for best weight re-use, and power consumption were analyzed.

5.2. Device asymmetry

In real hardware demonstrations, device asymmetry is difficult to avoid. PCM and RRAM are the leading choices for implementing analog-based accelerators, but both exhibit asymmetric response between SET (increasing conductance) and RESET (decreasing conductance) operations. When PCM is programmed with SET pulses, it is possible to increase the conductance of the device in small enough increments to make weight updates reasonably effective in training networks. However, incremental RESET of PCM devices is difficult to achieve, as a pulse that produces any RESET response typically fully resets the device to the high resistance state. Filamentary RRAM has the opposite behavior, in that these devices can be incrementally RESET, but SET is abrupt to the low resistance state. As a result, it is common to use a pair of analog memory devices to represent one weight, not only to represent both positive and negative weights, but also to mitigate weight update asymmetry by choosing to program one or the other devices in the same SET/RESET direction when applying positive/negative weight updates [61, 62]. Efforts to improve device characteristics by engineering the device physics will be discussed in the next section.

5.3. Device dynamic range and weights of varying significance

An interesting approach to extend the device conductance range is the periodic carry method proposed by Agarwal *et al* [68]. This introduces a method for encoding a wider dynamic range for weights, as compared to the size of the smallest possible weight change. This helps increase the number of effective conductance steps, thus training to higher DNN accuracies. Four devices with varying significance per weight were used. Weight updates were performed only on the least significant device, while weights were always read from all four devices combined. When the updated device saturates, either at its minimum or maximum conductance values, the second least significant device is updated to take into account the information from the least significant device. Training then continues on the least significant device after it is initiated to an intermediate conductance value well away from saturation.

Similarly in [69], multiple RRAM cells along one vertical pillar electrode together define one weight value. Each layer in the 3D vertical RRAM crosspoint array represents a weight contribution of varying significance, allowing higher resolution and effective dynamic range. RRAM weights were only ternary, i.e. −1, 0, or 1. Parallel read is implemented for forward inference, but weight update is read-before-write, one row at a time.

In Ambrogio *et al* [70], our research group at IBM Almaden proposed a new weight structure exploiting the multiple conductances of varying significance using a combination of different analog memory devices to both extend available conductance range and improve weight-update linearity. A pair of PCM devices are used to represent the more significant contribution to its weight, while a pair of transistors with gates connected to a capacitor are used for the less significant part of the weight. Training is performed by adding and subtracting charge from the capacitor, thus avoiding PCM device endurance and non-linearity issues. After training with a certain number of examples, the entire weight from the transistor pair is transferred to the pair of PCM devices with a scaling factor, thereby extending the weight dynamic range beyond the limits of a single pair of PCM conductances. This is similar to the approach of Agarwal *et al* except that no additional ADCs are required as additional conductances are added.

Finally, a third option is to implement multiple conductances of equal significance [71]. Here, a single weight is computed from the sum of many PCM devices, typically 7. Since the weight update is performed by programming only one of the PCM devices at a time, more conductance steps can be achieved. An arbitration clock ensures that all PCM devices receive a similar number of programming requests, to avoid early saturation or endurance failure of any single

PCM device. This method also improves linearity in weight update and allows a more gradual RESET transition. Weight update asymmetry can also be mitigated by controlling the relative update rate between positive and negative updates, at the expense of missing some update events. This architecture also reduces device degradation due to limited endurance since each device is only programmed once per seven updates. The downside of this technique is the increase in array size and power consumption.

Note that while the last two methods were demonstrated with PCM devices as the analog memory, the same concepts could readily be applied to many other types of NVM devices, with fairly minor modifications.

5.4. Non-linearity

Most analog memory devices exhibit some level of non-linearity, either between measured conductance and device voltage, or between the amount of weight update and current conductance value.

This first type of non-linearity is particularly important when neuron excitations are encoded into analog read voltages [72]. Effects of non-linearity were shown to be more severe for deeper networks with many synapse layers. This effect can be mitigated by applying a nonlinear transformation of upstream activations before multiplying by weights, effectively linearizing the combined activation-device response or by using pulse duration rather than amplitude to represent analog input to synapses. As mentioned earlier, the advantage of encoding analog signals as pulse duration comes at a cost of increased computation time, which could reduce performance on accelerator speed.

The second type of non-linearity, which is the non-linearity in conductance update, has been identified by multiple researchers as the most restrictive requirement for analog memory devices [60, 62, 66]. This is because during training, each weight element sees numerous update pulses in both increasing and decreasing directions, yet it is critical that a positive update and a negative update with the same magnitude can cancel each other. When implementing synapse weights using one single analog memory device, this cancellation relies on the symmetry between positive and negative conductance updates. When implementing weights using a pair of memory devices, as most hardware implementations do, both positive and negative weight updates become conductance updates in the same direction, just on different devices. Therefore, the update symmetry requirement becomes a linearity requirement, i.e. the amount of conductance update should be independent of the particular conductance value. Studies using modeling with experimentally measured 'jump tables'—tabulating the induced conductance change as a function of the starting conductance—from a variety of devices, including TaO_x/TiO_2-based RRAM, $AlOx/HfO_2$-based RRAM, PCMO, and Cu/Ag-based CBRAM [65, 66, 73, 74], show the effect of non-linearity on training accuracy.

5.5. Weight mapping for inference only

When DNN weights are pre-trained offline in software and then loaded into the analog memory array for forward inference, inaccuracies in setting weight values lead to poor performance. The device requirements in this case are slightly more relaxed compared to the case where training takes place directly in memory, because there is no need to implement backpropagation and many fewer weight tuning steps are required. As a result, one can afford to be quite careful when tuning the resistances of individual weights, and one can apply more complicated mapping schemes.

By having a sparse collection of weights represented by the NVM conductance plus a value stored in digital memory, the in-memory values can be trained further to improve performance. Because only a small fraction (5%) of the weights are in the sparse collection, some portion of the inherent advantages of NVM array can be retained [75]. Yan *et al* [76] contrasts two weight mapping schemes for weight quantization: evenly-spaced levels in resistance are compared to equal conductance difference between levels. The authors also investigated resistance shift due to read disturb and proposed alternating read polarity to minimize this effect. Wang *et al* [77] considered the limitation of device dynamic range, i.e. how many distinguishable weight values are needed for accuracy. They considered networks with binary weights and proposed to assign different analog values to the binary weights in different layers of the network, according to the distributions the weights would have if continuous-valued.

6. Analog memory device candidates

Established memories range from high density, slow and low cost NAND to low-density, fast and expensive DRAM and SRAM [78]. In recent years, the semiconductor industry has shown growing interests in the development of novel memories to replace or enhance functionalities of existing CMOS memory. Various candidates show multilevel programmability by applying electrical pulses, including RRAM, PCM, magnetic RAM (MRAM) and ferroelectric RAM [79–82]. This capability fits well with the basic needs of an analog-memory-based deep learning accelerator. Progress on other device options, including emerging battery-like devices, capacitor-based devices, photonics, and more exotic devices is also covered in this section.

6.1. Resistive RAM (RRAM)

RRAM is one of the more mature novel NVM device candidates, with commercially available memory arrays fabricated with CMOS technology (albeit in small size arrays, or at low density using older technology nodes). Filamentary RRAM offers promising properties such as very low programming energy, fast switching on the nanosecond timescale and relatively high endurance [83]. On the contrary, the

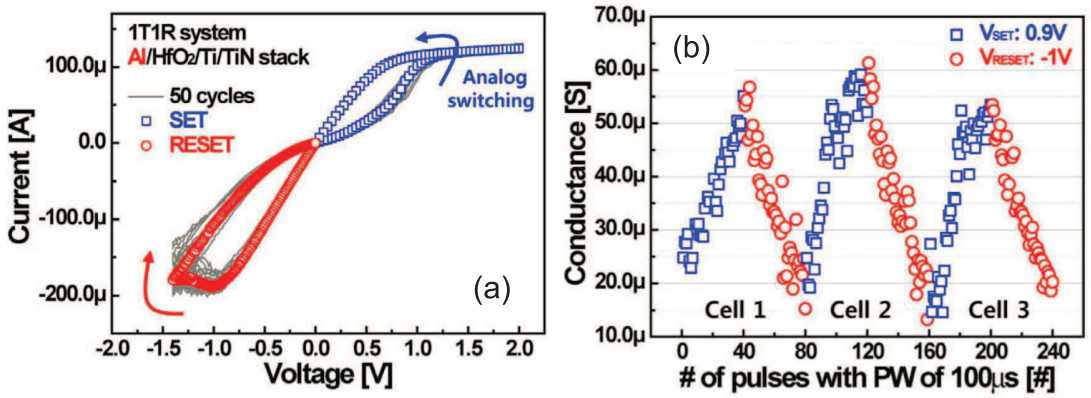


Figure 7. (a) *IV* curves for Al/HfO_2/Ti/TiN RRAM for 50 cycles. Analog switching appears at both SET and RESET transitions. (b) Subsequent conductance steps obtained with 100 μs pulses for both conductance increases and decreases. © 2016 IEEE. Reprinted, with permission, from [89].

resistance window of RRAM is generally not larger than a factor of 50×, which, together with an inherent intrinsic variability, poses limitations towards implementation of a large number of intermediate levels at low programming currents [84]. Other types of filamentary RRAM include unipolar RAM, where transitions are thermally driven, but reliability and endurance are relatively poor [85]. Conductive-bridge RAM (CBRAM) usually shows a resistance window larger than a factor of 100× [86, 87]. In a crossbar array, devices are typically located at the intersections between wordlines (WL) and bitlines (BL). When the memory device is in series with a select device, such as a diode, a selector or a transistor, the crossbar is *active*; otherwise the crossbar is *passive*. In the last few years, hardware demonstrations implementing FC networks have been limited by the number of available devices in a single crossbar array and by device variability.

6.1.1. Device optimization. RRAM comprises a family of devices which can be divided in two categories: filamentary switching devices and uniform (non-filamentary) switching devices [83].

Filamentary-RRAM typically consists of a metal-insulator-metal structure, where the formation of a conductive filament (CF) through the insulator (mostly metal oxide layers based on Hf, Ti, Si, Ta, but also chalcogenides) provides a high conductance state [83]. In many cases, the filament is composed of oxide defects; in some cases, however, the filament is composed of metal atoms, usually originally coming from one of the two metal electrodes. The CF formation (SET transition) and dissolution (RESET transition) are reversible and can be induced by electrical pulses, providing switching capability between high (SET state) and low (RESET state) conductance states. If CF formation and dissolution take place under the same voltage polarity, the device is defined as unipolar. If, instead, SET and RESET require different voltage polarity, then the device is bipolar.

Bipolar devices have shown superior performance in terms of endurance, variability and reliability. In bipolar RRAM, SET is temperature-accelerated and driven by the electric field [88], and the transition is typically abrupt, although non-abrupt transitions can be obtained with careful engineering of the oxide interface [89]. On the other hand, RESET transition is usually gradual due to the gradual dissolution of the conductive filament. This latter transition is of interest in deep learning applications, since it enables analog tuning of device conductances. Another option to gradually change the device conductance is to vary the CF diameter by changing the maximum allowed (or 'compliance') current that can flow into the device during SET transition. This leads to different SET conductance states with a higher degree of controllability, while RESET states typically show stronger non-linear dependence on applied voltage. This is caused by an exponential relationship between the conductance and the gap length during the RESET transition. In contrast, the dependence during SET state is linear in the area of the CF cross-section [90]. This asymmetry between conductance update during SET and RESET transitions is highly detrimental to deep learning accuracy, as we discussed in the previous section.

Several works have been published concerning improvements in RRAM device switching properties. Woo *et al* [89] developed a device stack based on Al/HfO_2/Ti/TiN in order to symmetrize RRAM switching by slowing down the SET transition. Figure 7(a) shows the obtained IV curves with gradual SET and RESET transitions, while figure 7(b) shows the conductance evolution as a function of identical pulses of 100 μs width. The simulation of this device as inserted into a three layer FC network showed an accuracy around 90% on the MNIST dataset [24]. Other approaches involve a careful and more elaborate sequence of programming pulses [91], with gradual SET states obtained through the application of consecutive SET and RESET pulses. Wu *et al* [92] used a thermally resistive top layer to smooth out the temperature distribution during programming, allowing multiple filaments and smoother bidirectional SET-RESET response. A small network for face recognition and a one-hidden layer perceptron for MNIST with binarized weights in the hidden layer were demonstrated.

Uniform-switching, or non-filamentary RRAM that could reach acceptable linearity and number of states, has also been developed. The non-localized switching strongly reduces

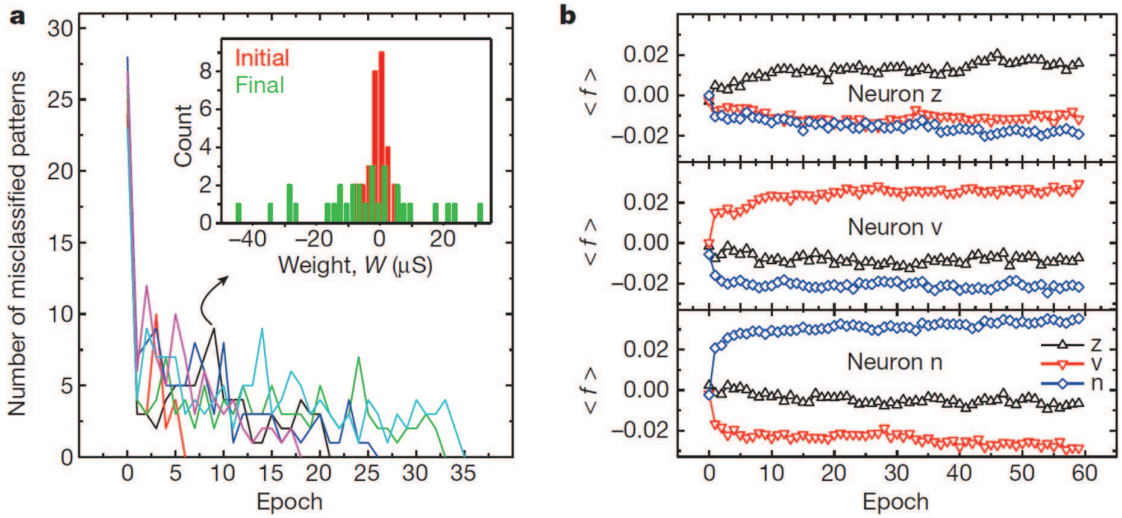


Figure 8. (a) Number of recognition errors as a function of training epochs for different experimental runs. The inset shows the weight distribution before and after training. (b) Average traces of neuron outputs during one training run, for input letters equal to z (top) v (center) and n (bottom). [101] 2015 Copyright © 2015, Springer Nature. With permission of Springer.

variability and enables gradual tuning of the conductance through electrical pulses [93]. Among them, $Pr_{0.7}Ca_{0.3}MnO_3$ (PCMO) devices and vacancy-modulated conductive oxide (VMCO) RAM [94] are most promising that have been used for neural network simulations [93, 95, 96]. PCMO devices show a conductance change due to migration of oxygen ions at the interface between electrode and PCMO layer [97]. In these devices, the adoption of molybdenum electrodes has been demonstrated to increase data retention [93], which is one of the important factors enabling multilevel programming in deep learning networks, together with other aspects such as low read noise, negligible conductance drift and resilience to device instabilities [98]. In addition, other architectures have been employed, such as 1T2R (one transistor-two resistors) weights, where one of the two resistors is the PCMO device. Here, a two-resistor voltage divider controls the transistor gate voltage. The application of pulses on this divider changes the resistance of the PCMO and this reflects in a modified gate voltage. By reading the current from the transistor from the transistor source, the number of conductance levels and linearity strongly increases [96].

Another non-filamentary device is based on a TiO_x oxide layer [99]. Here, limitations arise due to asymmetry between SET and RESET currents. To overcome this issue, Park *et al* [99] suggest the adoption of a Mo/TiO_xTiN stack. Since workfunctions for molybdenum and TiN are equal, the device shows enhanced SET/RESET symmetry.

In addition to engineering the physical switching mechanism, the application of dedicated voltage or current [99] pulse shapes can also relax the constraints on device characteristics. However, the benefits come at the cost of peripheral circuit overhead, energy dissipation and, in cases involving a large number of full RESETs, larger device endurance degradation [100]. Furthermore, improved device linearity was demonstrated with relatively long pulse-widths, around hundreds of μs or even ms, which are not practical for hardware accelerators. Thus, exploration of the proposed techniques with shorter pulse widths and large number of cycles will be important to prove feasibility of such methods.

6.1.2. Fully connected RRAM network demonstrations. Hardware demonstrations fall into two major categories: those where the weights encoded into RRAM device conductances are trained *in situ*, directly within the crossbar array; and those where weights are trained in software (*ex situ*) and then programmed into the crossbar.

A first hardware implementation by Alibart *et al* [63] reports a 9×1 neuron one-layer classifier implemented in a passive crossbar, which was able to classify 3×3 images of letters X and T. Here, the output neuron was providing $+1$ for X and -1 for T. Weights were encoded in the conductance difference of a pair of devices, thus providing both positive and negative weights. The crossbar size was 10×2, implemented with Pt/TiO_{2-x}/Pt devices. Training was performed both offline in software and directly in the crossbar memory array.

A later work from Prezioso *et al* [101] demonstrated a more advanced implementation with a 12×12 Al_2O_3/TiO_{2-x} passive crossbar. The network was directly trained in the crossbar array with 3×3 input images, taken from three classes representing the letters z, v and n and their noisy versions. Training was performed with the Manhattan update rule, which is a simplified version of the usual Delta rule. The Manhattan rule takes into account the sign of the sum of all δ values obtained after the forward propagation of all the training images. Therefore, weight update is performed only once per training epoch. Figure 8(a) shows the experimental accuracy error during different training runs. The inset shows the distribution of weights before and after training, while figure 8(b) shows the average output neurons signals for inputs corresponding to z, v and n letters.

Recently, Bayat *et al* [102] reported a bilayer network with one crossbar array divided in two portions of 17×20 and 11×8 Pt/Al_2O_3/TiO_{2-x}/Pt devices. The network is able to classify 4×4 images representing letters A, B, C and D. Training was performed both in software and in the crossbar

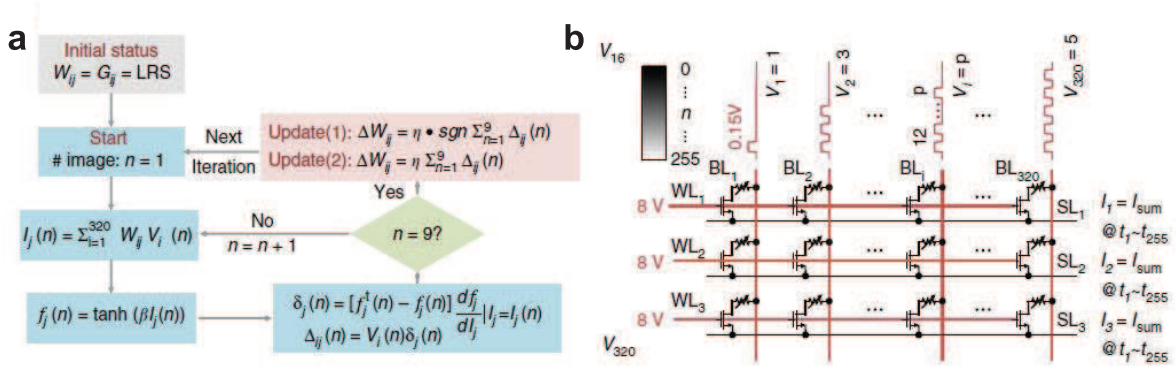


Figure 9. (a) Adopted flow chart of the training process in [104]. (b) hardware implementation with one device per weight, and corresponding encoding of gray-scale image (face images from [105], not shown here). Reproduced from [104]. CC BY 4.0.

array, with software (*ex situ*) training yielding higher recognition accuracy.

Demonstrations of relatively large networks still do not employ crossbar arrays due to reliability issues. Yu *et al* [103] use a 16 Mb TaO_x/HfO_2 RRAM macro where they implement a $400 \times 200 \times 10$ neuron FC network for MNIST handwritten digit recognition. To overcome the variability issues which arise in the multilevel programming operation, weights are trained in software and then programmed into the crossbar array with 1-bit precision, thus providing a large error margin to device conductance. To perform training, weights are encoded in 6-bits and programmed into six different devices. Simulations show less than 1% discrepancy from the software case. The drawback of this implementation is that it requires six times more memory, which also leads to higher power consumption.

All previous cases implemented weights as the difference of a couple of device conductances. Instead, Yao *et al* [104] reported pattern classification with a 128×8 active crossbar where weights are encoded in a single, bidirectional device stacked as $TiN/TaO_x/HfAlO_x/TiN$. Figure 9(a) shows the adopted algorithm, which is the backpropagation algorithm with the Delta rule for write-and-verify tuning, or the Manhattan rule for write-only tuning. The crossbar implementation is shown in figure 9(b). The network was able to recognize faces extracted from the Yale face database (face images not shown here, [105]). In this demonstration, higher accuracy was obtained by using a write-and-verify procedure, which enables more precise conductance tuning, and therefore more accurate weights.

6.1.3. Dot-product accelerator. Crossbar memory arrays can also be used to compute dot-product operations, $\mathbf{x} \cdot \mathbf{y} = \Sigma xy$, in one clock step. Given vectors x and y of size n, the computational complexity of a dot product operation goes from $O(n^2)$ in digital hardware to $O(1)$ in crossbars [106]. For this reason, researchers at HP Labs extensively studied how to develop an accelerator to efficiently map and calculate dot products in crossbar arrays. Due to device non-idealities, voltage drops on the wires, and circuit non-linearity, mapping of software weights into crossbar memory using a trivial linear conversion would degrade computational accuracy. Non-linear mapping techniques are developed to program crossbar weights in such a way that the final dot-product result in hardware matches the expected software value, as shown in figure 10. In [106], a first technique to reduce the voltage drop on the word lines consists of biasing a word line from both array edges (figure 10(b)). This leads to the highest error in central columns, then corrected with a static signal restoration that amplifies the read current from central columns (figure 10(c)). A major drawback is that this approach is data-dependent since the read current depends on the input signal. The image in figure 10(a) shows an input example with a Gaussian noise distribution.

A second approach is to use a non-linear conversion algorithm for weight programming. The ideal linear crossbar behavior is calculated in software. Then, by using a careful resistive device model of $Pt/TaO_x/Ta$, the actual response is obtained. Finally, the devices are fine-tuned in order to close the gap between ideal and actual crossbar simulation results. This technique is still dependent on input data, but results are very accurate, reaching 99% accuracy on the MNIST dataset in simulations, with no degradation from full software implementation [107].

6.1.4. Convolutional RRAM network demonstrations. Most hardware neural network implementations focus on FC networks because of the large number of weights employed in these networks. Here, the parallelism deriving from the large crossbars strongly accelerates speed for both training and forward inference. Instead, CONV-nets implement relatively fewer devices, which are organized in kernels and used several times during convolutions. Garbin *et al* demonstrated for the first time the impact of RRAM devices in CONV-nets by means of device modelling, figure 11(a), characterization and simulation [108]. The network was trained on the MNIST dataset where pixel intensity is encoded with a train of spikes whose frequency is proportional to pixel brightness, figure 11(b).

Accuracy was close to the software equivalent (98.3% against 98.5%) under strong programming conditions (thus maximizing the device resistance window around a factor $100\times$, but greatly reducing the device endurance), and weights were encoded with 20 binary RRAM in parallel. Accuracy is decreased to 94% when considering weak programming

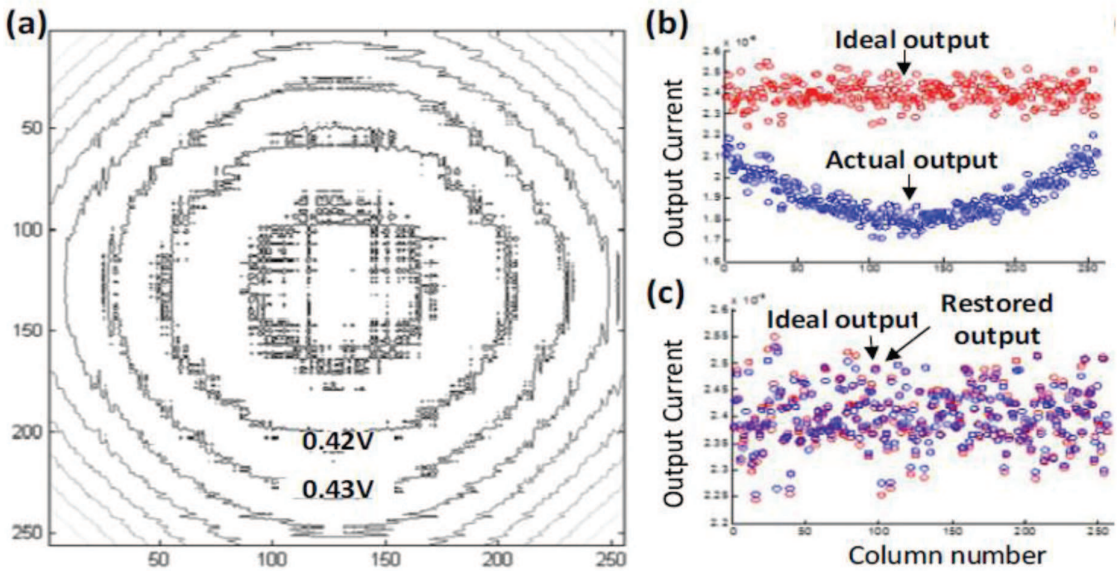


Figure 10. (a) Input pattern used as an example for linear mapping with double biasing and static signal restoration. (b) The ideal output obtained with a perfect crossbar, while the actual output shows a minimum due to voltage drop in correspondence of the center of the 256 × 256 crossbar. The application of signal restoration provides nearly ideal results. © 2016 IEEE. Reprinted, with permission, from [106].

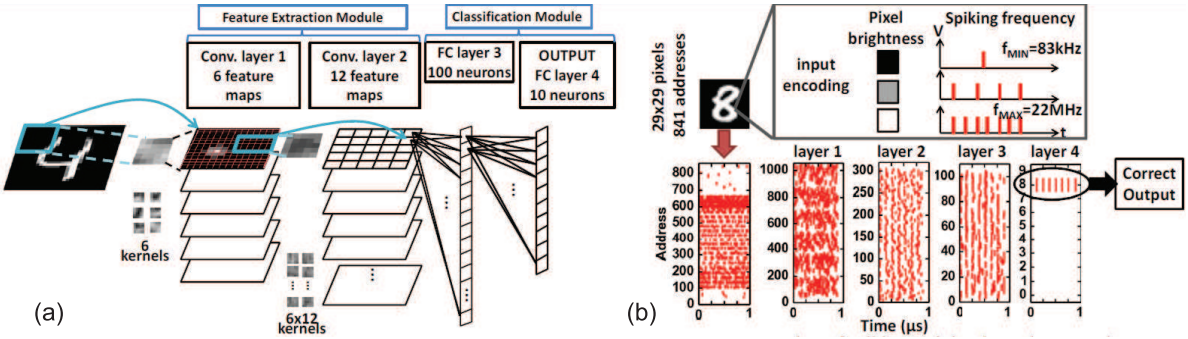


Figure 11. (a) CONV-net used in [108]. Simulated RRAM devices encoded the weights in the kernels. (b) Input image conversion with a train of spikes with frequency proportional to pixel brightness. Only neuron corresponding to '8' fires at the last layer. © 2014 IEEE. Reprinted, with permission, from [108].

conditions, with a resistance window around 10× and a higher dependence on device variability [109] These conditions, however, preserve the device from early failure [110]. Unlike FC networks, CONV-nets use fewer weights, thus reducing the impact of using many devices in parallel. On the other hand, the number of SET/RESET cycles on each device is three orders of magnitude larger, thus degrading the device faster [110]. In general, crossbar implementation of CONV-nets shows difficulty in achieving speed improvement over existing GPUs during training because weights need to be convolved with the entire input image, thus breaking the parallelism that exists in FC networks.

6.2. Phase-change memory (PCM)

PCM relies on the creation of different conductance levels by switching the material properties of chalcogenide layers, such as $Ge_2Sb_2Te_5$, from amorphous, at low conductance, to crystalline, at high conductance. Different architectures exist, but they all rely on controlled heating of a chalcogenide material. The SET transition is gradual, since crystallization implies a local reordering of the atomic lattice, while RESET transition is abrupt, as the entire region needs to be melted and then quenched into the amorphous state [111]. Both SET and RESET processes can be driven by electrical pulses, enabling the implementation of analog acceleration for neural network training.

6.2.1. Fully connected PCM network demonstrations. In recent years, our research group at IBM Almaden has developed an approach to accelerate on-chip training of FC networks with weights weights encoded as the difference in the conductances of a pair of PCM devices in an active crossbar array [61, 62]. The backpropagation algorithm is implemented in three steps. Training images are forward propagated through a 528 × 250 × 125 × 10 neurons network, results are compared against the correct labels, errors are backpropagated from the last to the first layer, and then the weights

are updated following a crossbar-compatible procedure that enables parallel update of all the weights in the crossbar array [61, 62]. Results performed on real PCM devices (but with the neuron circuitry simulated, not integrated with the NVM devices) reported a 82.2% test accuracy on MNIST dataset. As mentioned before, later developments with the same PCM in a larger and more complex unit-cell have recently achieved software-equivalent training accuracies [70].

6.2.2. PCM for memory computing. In addition to fully implementing neural networks on crossbars, hybrid solutions have also been adopted. Le Gallo *et al* [112] propose a general method for solving systems of linear equations in the form $Ax = b$ where the solution comes from two interacting modules, namely a high-precision processing unit and a low-precision computational memory unit, i.e. the PCM crossbar. The main idea is to split the calculation of the solution into two parts: a low-precision z solution from $Az = r$, followed by high-precision calculation of the solution update $x = x + z$ and error $r = b - Ax$. After that, successive iterations refine the solution to the desired degree of tolerance. Interestingly, this method speeds up the overall computation time since the calculation of the inexact z, which involves the calculation of many multiplications and sums, represents the most computationally expensive operation in digital computation, thus fully exploiting the capabilities of analog-based acceleration.

This concept has also been applied in simulations of FC neural network for MNIST digit recognition by Nandakumar *et al* [113] (see also [114]). Here, the high-precision unit calculates forward, back-propagation and weight update. Therefore, the network is implemented in CMOS with the PCM array used to perform the compute-intensive multiply-accumulate operation, creating a hybrid architecture which accelerates training on the MNIST dataset. Weight updates are summed into a high precision variable χ. Since update on PCMs shows a certain granularity ϵ, meaning that it is not possible to program conductance changes smaller than ϵ, weight update is only performed when $\chi > \epsilon$. After the effective weight update, χ is updated to $\chi = \chi_{previous} - n\epsilon$, where n represents the number of steps the network asked to program into the crossbar. Simulations show test accuracy within 1% from full software implementation.

6.3. Battery-like devices

While well-known NVMs such as PCM and RRAM dominate the landscape of emerging technologies for deep learning, there have been attempts at exploring other devices with better linearity, symmetry, scalability, and higher dynamic range. Two recent papers [115, 116] report novel devices exploiting electrochemical reactions derived from working principles of batteries [117]. Fuller *et al* [115] describe a Li-ion synaptic transistor (LISTA) based on the intercalation of Li-ion dopants into a channel of $Li_{1-x}CoO_2$. A negative gate voltage V_G recalls Li-ions from the channel region to the gate, providing additional electronic carriers and thus increasing the source-drain conductance. Similarly, a positive V_G pushes ions into the channel region, decreasing the source-drain conductance and enabling a six-orders of magnitude dynamic range in conductance. Figure 12(a) shows the electrical characterization of this device, with application of many pulses. The corresponding jump-tables [61, 62] for current-controlled ((b) and (c)) or voltage-controlled ((d) and (e)) positive ((b) and (d)) or negative ((c) and (e)) weight update reveal a highly linear device. This improvement on device characteristics translates into MNIST performance on accuracy, achieving less than 1% accuracy degradation from full software implementation [115].

Similar results are obtained by van de Burgt *et al* [116] demonstrating an organic neuromorphic device with a similar experimental behavior (with H^+ as the mobile ion) and MNIST accuracy only 1% below its software baseline [115]. These novel devices show promising results for neural networks, but research on these devices is still at its early stage. Programming times are on the order of milliseconds, since shorter pulses induce only a short term conductance change [116]. Scalability of such devices, and operation within an array will also need to be explored.

6.4. Capacitor-based CMOS devices

Given the inherent non-linearity and asymmetry in existing NVMs that make on-chip training challenging, Kim *et al* [118] proposed an analog synapse based on capacitance (figure 13). The weight of the synapse is proportional to the capacitor voltage, and is sensed through a read transistor. The authors proposed using a logic block in every unit cell to make a local determination on whether an up or down pulse needs to be fired during weight update. While the proposed guideline of 1000 states per unit cell implies that the capacitor dominates unit-cell area in initial designs, this assumption could well change with either multiple conductances of varying significance and/or adopting other capacitance manufacturing processes such as deep trenches, stacks or metal insulator metal capacitors. In this case, the many transistors in the design would make achieving area-efficient unit-cells (and consequently large numbers of synapses per die) a challenge. Furthermore, even with elimination of some of the logic devices, managing the random variation-induced asymmetry between the pull-up and pull-down FETs (P3 and N3 in figure 13) would still require very large devices and/or other circuitry techniques. Although the synaptic state is decaying continuously, it can be shown that at high learning rates, the network can accommodate this so long as the ratio between the RC time constant (governing the charge decay) and the time-per-trained-data-example is extremely large ($\gg 10^4$) [118, 119].

6.5. Ferroelectric devices

Ferroelectric materials have also been studied for analog memory devices with hafnium–zirconium–oxygen (HZO) stoichiometries being a popular choice [120, 121]. Applying short pulses can cause polarization domains to flip, changing

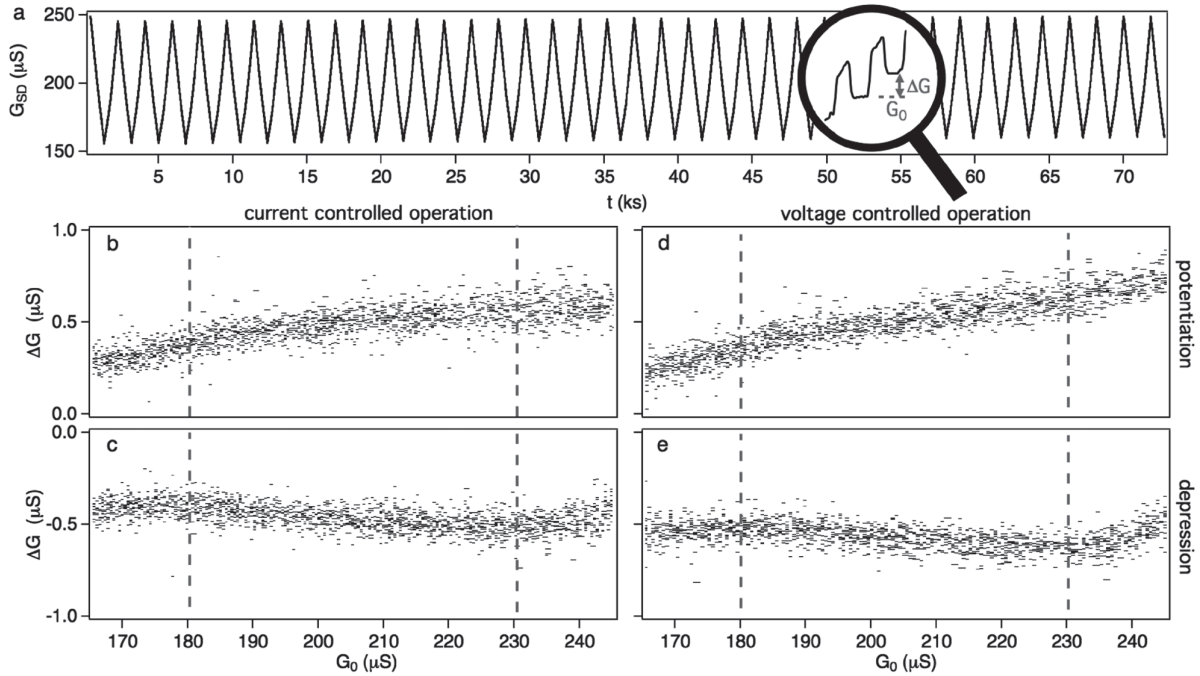


Figure 12. (a) Conductance evolution for several cycles. The device can be voltage or current controlled, leading to similar behavior. (b), (c) The respective jump-table for conductance increase or decrease as a function of initial conductance for current controlled operation, while (d), (e) show the corresponding results for voltage controlled operation. [115] John Wiley & Sons. © 2016 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim.

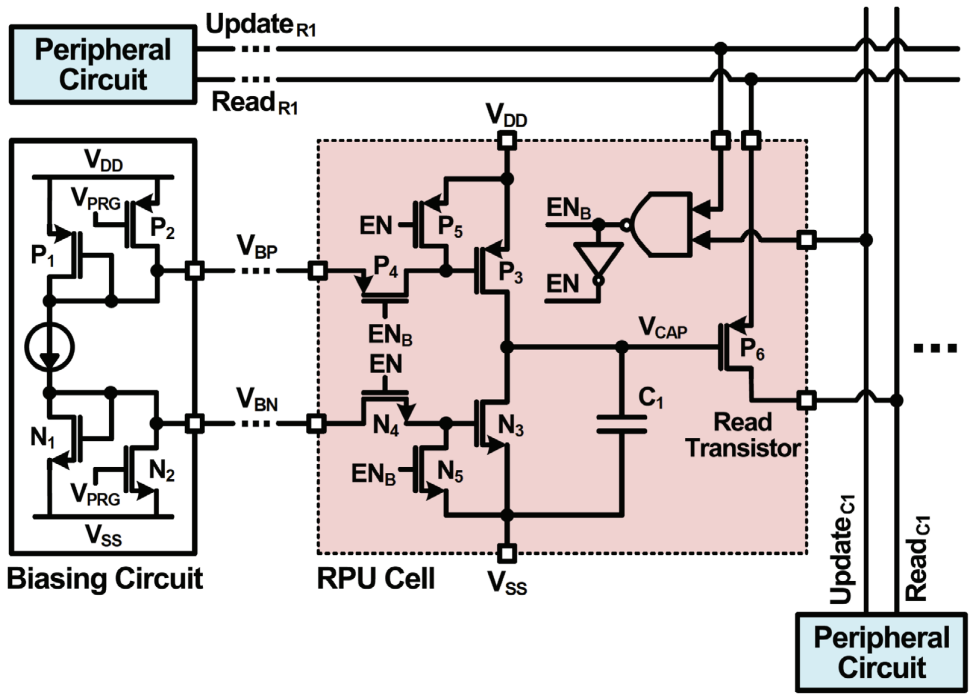


Figure 13. The CMOS RPU from [118]. © 2017 IEEE. Reprinted, with permission, from [118].

the threshold voltage of a FeFET device. However, for gradual changes in conductance over a wide range, first domains with smaller coercive voltages and then domains with larger coercive voltage would need to be flipped. This implies that programming of weights would require a read-before-write scheme to choose the right programming pulse amplitude. This would severely hamper speed for on-chip training, but may still be applicable for inference, where weights are only programmed once.

In [122], the authors proposed using ferroelectric capacitors not as a continuously tunable analog device, but as strongly-ON or strongly-OFF switch devices allowing current

to flow through resistive elements of varying significance. This reduces the requirements on ferroelectric devices, and the authors showed through simulation that they can achieve well-separated weights. Also, using a hardware-aware regularization approach during training led to good accuracy during inference. This is an insight that may be valuable for other inference researchers as well. Nevertheless, the area cost of building unit cells with multiple resistive elements (the suggested implementation is as distinct FETs), and power/performance benefits were not quantified.

6.6. Photonics

The drive to reduce power consumption and increase throughput in the execution of deep neural networks has spurred novel approaches, including the emerging field of photonic networks. Photonic implementations promise high speed due to the high communication bandwidth of optics and low power consumption due to the low dissipation associated with the transmission of light in waveguides. Early efforts in this area included optical implementation of the Hopfield network and proposals for holographic MAC operations [123–126]. More recently, silicon nanophotonics is becoming a mature technology for producing versatile photonic integrated circuits. Although photonic devices are larger than CMOS logic and NVM memory devices, techniques such as wavelength division multiplexing allow large numbers of signals to be simultaneously transmitted through the same physical waveguides and devices.

Although the field is still emerging, several building blocks relevant for neuromorphic computing have been shown. These include optical versions of neurons with leaky-integrate-and-fire response [127, 128], MAC operation using wavelength division multiplexing and optical filters [129], and adaption of the intrinsic nonlinear dynamics of optical feedback networks for application to reservoir computing [130–133]. Here, we give some highlights of this work. For recent reviews focused on this area, see [134–136].

Optical gain media used in laser oscillators and amplifiers are intrinsically nonlinear, and this nonlinearity has been exploited to implement functions needed for neuromorphic computing. Using a semiconductor optical amplifier as an integrator and a nonlinear fiber loop mirror as thresholder, an optical leaky-integrate-and-fire neuron was demonstrated [127, 128, 137]. A similar approach was used to demonstrate a simple neuromorphic processor [138]. Non-linear microring resonators [139, 140] could serve a similar role.

Wavelength division multiplexing (WDM) has played a key role in optical communications, allowing a single physical waveguide to carry many signals simultaneously. If the activation of a neuron is represented by the optical intensity of one of these wavelength channels, with each neuron assigned a different wavelength channel, WDM provides a means of transmitting a multiplicity of signals from one network layer to the next. A series of optical filters, implemented, for example, with silicon photonic microring resonators [141], can transmit individually chosen fractions of each wavelength, producing upon photodetection a weighted sum of the outputs of the upstream neurons [129, 142]. In this scheme, the microring resonators are the optical synapses, with synaptic weights programmed by the detuning of the resonators.

In addition to ring resonators, several alternatives for realizing optical synapses are being explored using photonic technologies. Silicon photonics resonators have been fabricated on a ferroelectric barium titanate film [143, 144]. The transmission of the resonator at a particular wavelength could be incrementally tuned by changing the domain configuration of the ferroelectric layer with in-plane electric field pulses. By integrating phase-change materials onto an integrated photonics chip, the analog multiplication of an incoming optical signal by a synaptic weight encoded in the state of the phase-change material was achieved [145]. In this device, the weight could be adjusted with optical write pulses carried by the same waveguide. This is one example of an optical synaptic element that can potentially have its weight tuned *in situ* for online learning. This scheme of embedding a phase change element as an optically programmable attenuator has also been used for another example of optical 'in-memory' computing, an 'optical abacus' that can perform numerical operations with optical pulses as inputs [146].

One relatively advanced photonic ANN implementation uses coherent optical nanophotonic circuits [147]. Processing is done by arrays of Mach–Zehnder interferometers and phase shifters to realize matrix multiplication of arbitrary real-valued matrices. In this case, the matrix of weights that represents the synaptic connections between neuron layers is factored via singular value decomposition into the product of two unitary (i.e. lossless) matrices that are implemented using Mach–Zehnder interferometers and phase shifters, and a diagonal matrix whose elements are represented by optical transmissions. Effectively, the diagonal matrix encodes the synaptic weights, represented as optical transmission, and the unitary matrices the connectivity. A simple four-layer network is shown that recognizes vowel sounds with 76.7% accuracy, compared to 91.7% for an ideal network, limited by the precision for controlling optical phase and photodetection noise. For this application, training is done offline and the network programmed with the resulting weight matrices.

The devices discussed above were used for forward inference only, the synaptic weights for a given application having been pre-computed offline. Given that forward propagation through an optical network is cheap, researchers have proposed computation of the gradient for each weight directly, one weight at a time, which would bypass the need to implement backpropagation [147]. Another approach is to use a neuromorphic computation model that requires relatively few tunable weights. Reservoir computing [136, 148, 149] is one such paradigm that uses a recurrent neural network with fixed weights, exhibiting nonlinear dynamics with a sufficiently rich state-space to effectively represent a large variety of inputs. This recurrent network is the reservoir. Typically, a small number of the reservoir neurons are coupled to output neurons to serve as a classifier, and only these output weights are adjusted during the learning phase. Optical systems with

feedback are one possible implementation of this type of recurrent network and have been shown using semiconductor optical amplifiers [130, 131, 150], nonlinear electro-optic oscillators using delayed feedback [151–153]. These have been applied to simple tasks such as spoken digit recognition [132, 150, 151], or time series prediction [132, 133, 150].

To date, many basic neural network operations have been demonstrated using photonic devices [134–136], but the numbers of neurons and synaptic elements are far from the scale of, for example, deep CONV-nets that embody today's state of the art. Implementing a network for forward inference is conceptually straightforward, and a significant amount of work has been done to understand the impact of issues like weight resolution, variability and noise on the expected performance. Online learning has not yet been addressed in a satisfactory way, nor has the widely used backpropagation algorithm. Reservoir computing is an area to which photonic networks seem to adapt well, and this network model may be useful in applications where recurrent networks could be important such as classifying sequences. The low power dissipation and high processing speed that photonics brings to ANNs will be attractive only if photonic implementations succeed at solving problems of strong interest to computer scientists and AI practitioners.

6.7. Other devices

In this section, we summarize other recent research on new device exploration, including but not limited to other CMOS devices, flash and organic devices.

Bae *et al* [154] propose using Schottky diodes whose work-function can be modified by charge-trapping using a back gate. The material stack proposed uses Si–SiO_2–Si_3N_4, which involves well-established CMOS unit processes and can fit in a $6F^2$ unit cell area, comparable to most DRAMs or 1T1R designs. However, the authors' proposal for dealing with non-linearity uses a read-before-write scheme which would be better suited to inference-only schemes as opposed to high performance training.

In [155], the authors use a charge-trapping $HfSiO_x$ layer as part of the gate dielectric stack to induce a threshold voltage shift on 28nm CMOS planar SOI devices. This can modulate the current flow through the device, enabling analog synaptic behavior. The authors propose using this device as part of a forward inference engine, and include full mixed-signal circuit and architecture design to build a test prototype. However, at the time of writing, experimental results from the prototype are not available. Simulation results suggest 8-bit charge-trap-transistor (CTT) weight resolution may be needed for software-equivalent accuracies on MNIST, but could benefit from recent work on other inference engines demonstrating aggressive quantization of weights [156]. Also, it is not clear if the simulations capture threshold voltage increases due to non-zero source-to-body voltage, which is strongly dependent on the current being integrated in the array.

In [157], a single-crystalline SiGe layer epitaxially grown on Si was used as an analog memory device, called epiRAM. Conductance tuning is achieved through modulating confinement of conductive Ag filaments into dislocations in the SiGe layer. A defect-selective etch step was required before cation injection to widen dislocation pipes to enhanced ion transport in the confined paths and therefore increase on/off ratio of the device. With the one-dimensional confinement for filament formation, the epiRAM devices showed improved set voltage variation both spatially and temporally. A 3 layer MNIST FC network simulation with experimentally measured device characteristics showed online learning accuracy of 95.1% (97% in software.)

As opposed to building new devices (albeit with existing unit processes) to exploit charge-trapping, Guo *et al* [158] used modified NOR flash memory arrays for inference, as shown in figure 14. They implemented a $784 \times 64 \times 10$ neural network on a test site, and demonstrate $<1\ \mu$s inference latency, $\sim$20 nJ average energy consumption on MNIST and discuss prospects for further improving these numbers. They also demonstrate resilience to drift (in measured NN performance), over a timescale of 7 months, and temperature invariance. The reduced classification accuracy ($\sim$94% in hardware versus 97.7% in software), may be attributed either to the weight tuning itself (only 30% of the weights were tuned to within 5% error), or to device and circuit variations, although it is unclear what the relative contributions were.

Lin *et al* [159] used organic memristive synapses based on Iron (tris-bipyridine) redox complexes. While the devices show gradual conductance change with both SET and RESET pulses, these devices still need considerable improvement and a compelling use case. The pulse width of 100 μs, as well as the need for increasing voltage amplitudes makes high-performance training difficult. The authors discuss a complete test setup, including an FPGA interface and different programming modes to tune the conductances using a Delta learning rule. Experimental demonstrations include successful learning of a three-input boolean function, along with simulations of other functions and MNIST under different assumptions of variability.

7. Computing-in-memory architectures

In addition to materials, devices and process integration efforts on building ideal analog memory devices for deep learning, an important research direction is the realization of larger-scale systems that can translate the raw benefits of analog computation to tangible improvements at the application level.

This includes several design challenges, e.g. area and power-efficiency in peripheral circuitry that handles communication and computations outside of the analog MAC operations, IO interfaces, resource allocation for maximizing throughput-per-unit area, control schemes, etc. It also requires circuit and/or architectural simulation frameworks to demonstrate speedup or power/energy benefits over competing CMOS CPUs, GPUs or ASICs designs on various benchmarks. Finally, an often overlooked yet equally important research challenge is achieving equivalent accuracies on these benchmark tasks in the presence of imperfect devices, circuit

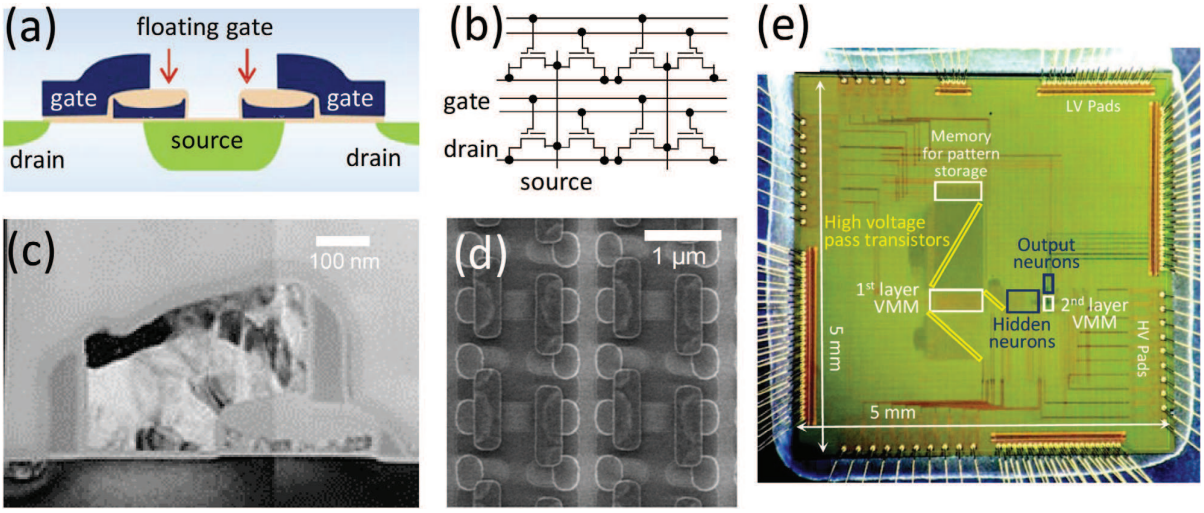


Figure 14. 180 nm ESF1 NOR Flash used in [158]. (a), (c) Schematic and TEM of the cross-section, (b) is array organization, (d) is top-view SEM, (e) shows micrograph of chip with MLP regions implementing vector matrix multiply. © 2017 IEEE. Reprinted, with permission, from [158].

variability and analog noise. There is little point in being faster or more area-efficient if the hardware accelerator does not 'do the same job' as software.

This section presents an overview of several computing-in-memory architectures that address one or more of these aspects. We begin with approaches for forward inference on CONV-nets and multi-layer perceptrons (MLPs), and then discuss architectures for training.

7.1. Architectures for inference

The ISAAC accelerator [160] is positioned as a processing-in-memory architecture for forward inference of CONV-nets. MAC operations occur on 128 × 128 memristor arrays, with 2-bit memristors and eight memristors-per-synapse (16-bit weight). ADCs are used at the periphery of the arrays, with one ADC shared among 128 columns, and achieving a sampling rate of 1.28 *Gigasamples* s^{-1} to meet a target read time of 100ns. Embedded DRAM (eDRAM) is used for buffering intermediate terms and digitized array outputs that are yet to be used in the next layer. For CONV-net forward inference, the authors propose a pipelining scheme that allows convolution operations on the next layer as soon as a sufficient number of pixels in that layer have been generated. They also observe that ADCs consume the most power in the design (58%), and present a weight flipping scheme that allows reduction in the ADC resolution. While the peak throughput-per-unit area of 479 GigaOps/s/mm^2 exceeds modern GPUs, it is somewhat unclear if one can achieve 100% utilization of the memristive arrays on more modern CONV-nets such as VGG [161], especially in the first few layers where the number of inputs is far larger than the number of weights. The impact of memristor imperfections on classification accuracies is also not discussed.

The PRIME architecture [162] is a similar RRAM-based inference engine with some important distinctions. Firstly, device assumptions were more aggressive—including 4-bit (16 state) RRAMs, two of which are combined for an 8-bit weight, multiple analog voltage levels for read (which assumes I–V linearity over the entire span of read voltages), and eschewing all eDRAMs (which places a high demand on RRAM device endurance). Secondly, at the circuit-level, the authors proposed to repurpose the sense amplifiers as ADCs, and the write drivers as DACs (figure 15) in order to save area and power. They also provided a means for interfacing their architecture to a software stack, allowing mapping of several different NN topologies including MLPs and variants of VGG. Benchmarking showed potential for 3 orders of magnitude speedup on VGG over a 4-core CPU, but did not compare to GPU architectures. Classification accuracy and memristor imperfections/variability were not discussed.

In contrast to the above approaches that assume digital communication of signals between arrays, the RENO approach [163] presents a reconfigurable interconnect scheme (figure 16) that can be repurposed to transmit either analog or digital signals. ADCs or other digitizing schemes are not required except at the I/O interfaces. This approach still requires multiple analog read voltages and I–V linearity for the memristor device. The authors considered several small MLPs for benchmarks such as MNIST. However, classification accuracies are somewhat below their software counterpart. Furthermore, speed and power numbers are shown in comparison only to an Intel Atom CPU.

A paper by Fan *et al* [164] targeted low-power inference, as opposed to the other approaches where high performance is the point of emphasis. The use of STT-MRAM allows for several orders-of-magnitude higher endurance than either PCM or RRAM, which is necessary for the in-memory logic scheme that the authors proposed. To overcome the issue of low resistance contrast, the authors proposed using a binarized CONV-net, which has been shown to achieve comparable accuracies on benchmarks such as AlexNet. While the authors showed nearly two orders-of-magnitude less energy compared to GPUs, their reconfigurable computation scheme involves

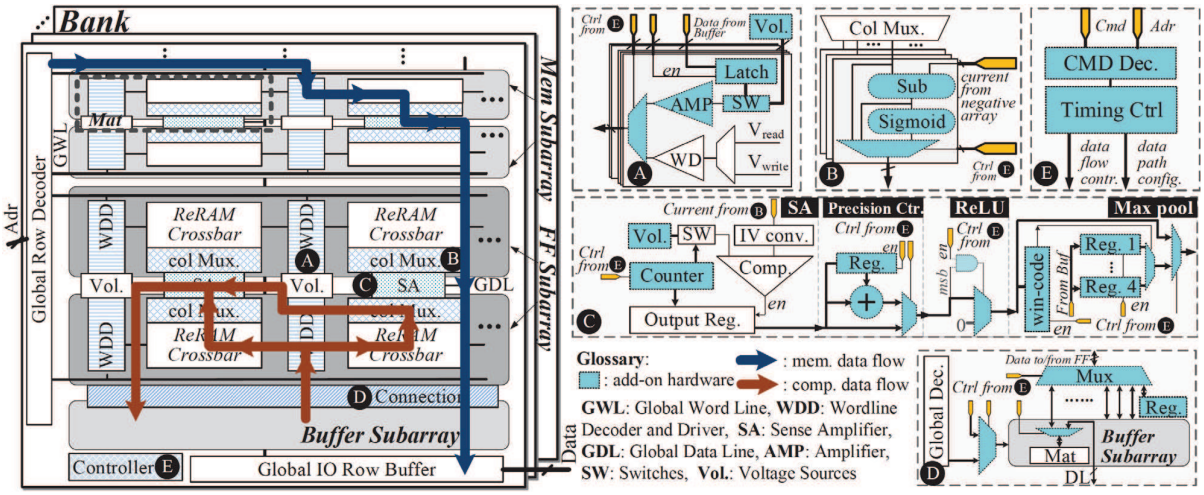


Figure 15. The PRIME memory bank from [162] showing a typical memory read write path (left—blue) and a typical compute path (left—brown). Schematics (A)–(E) on the right show repurposing of standard memory circuitry with additional components for implementing compute-in-memory. © 2016 IEEE. Reprinted, with permission, from [162].

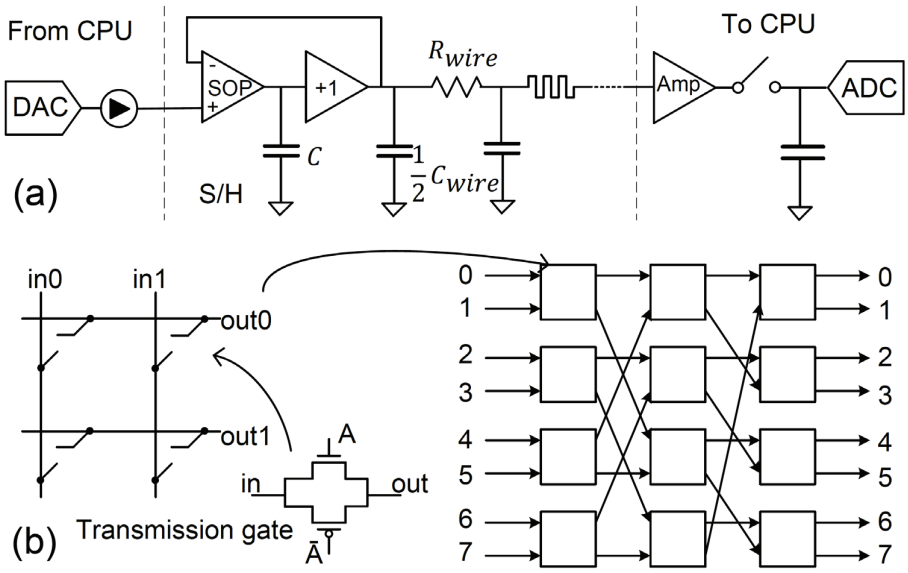


Figure 16. Reconfigurable routing channel used in RENO ([163]) that can transmit both analog and digital signals. © 2015 IEEE. Reprinted, with permission, from [163].

setting different reference voltages on the column sense-amps. This will likely be extremely challenging for analog-memory-based accelerators, given the aforementioned low resistance contrast and associated variability, and only gets exacerbated as higher fan-in functions are considered.

Finally, DRISA [165] is a CMOS-based approach that seeks to use 3T1C and 1T1C DRAM arrays for in-memory compute. The technological challenge here is integrating logic and DRAM, as opposed to using other NVMs. While this may seem more achievable, the upside for such a technology is low. The paper demonstrated one order-of-magnitude speedup and energy efficiency over GPUs at software-equivalent classification accuracies. However, the caveat is that this was evaluated at a mini-batch size of 1, which is inherently inefficient for GPUs. Increasing the mini-batch size to 64, which is standard for GPUs, nearly eliminated the benefits. Forward inference use cases where input data is infrequent (implying that it may not be trivial to fill up a mini-batch) yet latency and power consumption are critical, may benefit from the DRISA approach.

7.2. Architectures for training

In addition to forward inference, architectures for training also need to include mechanisms for backpropagation and open-loop weight update. This is especially challenging to implement for convolution layers, where typically multiple copies of the weights are needed for efficient forward inference, yet the same gradient needs to be applied to all copies of the weights. This requirement to consolidate weight updates from x and δ values that arrive at different points of the crossbar and the fact that many convolution layers may not be

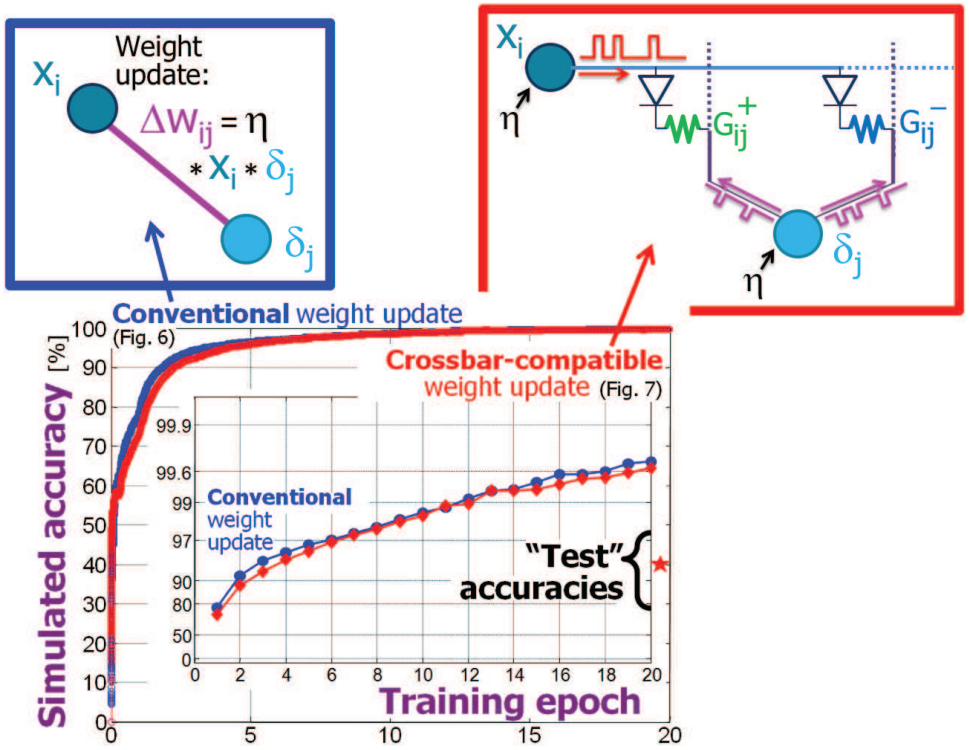


Figure 17. Hardware-friendly massively parallel crossbar compatible weight update from [61]. © 2014 IEEE. Reprinted, with permission, from [61].

memory-bound in the first place makes the prospects for hardware acceleration unclear. To our knowledge, no one has yet to propose an end-to-end training architecture for convolution layers. The papers below discuss variants of FC-networks, including MLPs and Boltzmann machines.

In [166, 167], the authors discuss an early architecture for training with backpropagation. The authors proposed using a separate training unit that needs to generate the weight updates required for all the pulses and transmit it back to the arrays. However, the challenge with implementing training separately is that the latency and temporary storage requirements for any intermediate terms needs to be carefully considered. The authors also did not assume any access devices to cut off sneak path leakage, which will likely be a problem for weight update operations.

In [21], our research group at IBM Almaden described a generic architecture for training using backpropagation on NVM arrays, using approaches for circuit implementation of forward, reverse and weight update with input x, δ, and update signals all in the analog domain. We described several trade-offs for peripheral circuitry, including several approximations to reduce area overhead and minimize time multiplexing of neuron circuits while supporting standard forward, reverse and weight update operations. In this approach, weight update is implemented directly on the crossbar array with upstream x and downstream δ firing a finite number of pulses based on their values and associated learning rate(s). This 'crossbar-compatible' and highly parallel weight update step (figure 17) was shown to achieve the same accuracy as the full weight update for the MNIST dataset [62]. In addition, we discussed the issue of 'network freeze-out' wherein NVMs whose conductance changes are gradual only in one direction (such as PCM or RRAM) eventually saturate to zero net weight. We described an occasional RESET procedure (occasional SET for RRAM) that would be needed in addition to the three NN modes, to allow training to continue.

In [168], the authors proposed a memristive Boltzmann machine, that uses RRAM crossbars to accelerate both the well-known restricted Boltzmann machines (RBMs) used in deep learning, and more general Boltzmann machines that are often applied to various optimization problems. Computation involves a three way partial product between the downstream neuron (implemented as time gating on bit lines) an upstream neuron (implemented as time gating on word lines) and a crosspoint weight (implemented as RRAM conductances). A sense-amp-as-ADC approach similar to [162] was used. 57× speedup was shown for some deep belief network configurations compared to a 4-core CPU (no comparison against GPUs.) The use of this architecture to train using contrastive divergence is likely limited by the need for a separate controller to compute weight updates from the obtained energy, and to perform the write operations in the array. However, this is likely not a problem for some of the other Boltzmann machine problems, where energies are recalculated in every iteration yet the weights of the network do not change.

8. Conclusion

Innovations at the device level targeting improved conductance symmetry and linearity are still essential for hardware

acceleration of training. At the same time, it is important to not view device development in isolation, according to some fixed set of criteria and specifications. Instead, device development must proceed in the context of integrated systems wherein different algorithms, architectures and circuit schemes can place different requirements on the underlying devices. For instance, techniques such as multiple conductances of varying significance [68, 70], local gains [169] or other approaches can potentially change device requirements, making them both less challenging but also subtly retargeting these requirements.

In this system-centric view, the applicability of the device will be quantified based on whether it can achieve competitive machine learning accuracy not just on MNIST but on much larger datasets while accounting for the full extent of variability. While an eventual large-scale hardware demo may seem appealing, early efforts to demonstrate equivalent accuracy on commercially interesting scales would likely involve either mixed hardware-software approaches or simulations that can reasonably accurately capture real device behavior. Such experiments would greatly benefit from hardware-aware simulation frameworks that could allow mapping of networks from deep learning platforms such as TensorFlow and Caffe to real systems. Small-scale hardware demonstrations investigating the core critical modules or tasks that will eventually be an integral part of a large-scale system (for example, implementing parallel vector-vector multiplies needed in LSTMs, or block-wise reconfigurable routing) will be an extremely important stepping stone.

Identifying the right networks and tasks suitable for hardware acceleration is critical. Recurrent nets based on LSTM/GRU, RBMs and MLPs are all promising candidates, and further advancements in approximate computing for DNNs (e.g. reduced precision [58]) are a good fit with custom-hardware approaches. While convolution layers have widespread use in image classification systems, the relatively small number of weights with respect to neurons, especially in earlier layers make efficient implementation challenging. While it may be possible to use multiple copies of weights and/or pipelining schemes, this limits the effective throughput per unit area while also setting up additional difficulties for high-speed training. In that sense, it may be beneficial to use approaches such as transfer learning [170] that utilize pre-trained weights for convolution. It must also be noted that the evolution of deep learning has been closely tied to the evolution of the existing hardware paradigm, which is better suited to handle convolution. The emergence of reliable high-performance Non-VN architectures could, in a similar fashion, fuel further innovations in the algorithms.

At the circuit and micro-architecture levels, there are several open avenues of research. For instance, analog-to-digital converters at the edge of each array might provide the maximum flexibility for implementing arbitrary neuron functions in the digital domain. However, the tradeoffs in power and performance need to be carefully quantified. There may still be good use cases for hybrid analog-digital chips, for e.g. in the area of memcomputing [112]. Memcomputing applications that could use the exact same hybrid chips designed for forward-inference of deep learning acceleration would be particularly attractive.

Similarly, encoding DNN neuron activations in voltage levels as opposed to time durations at fixed voltage may seem promising, however device non-linearity with respect to changing read voltages needs to be carefully considered. Furthermore, routing a large number of analog voltage levels between arrays would need specialized operational amplifier circuitry that would be both area- and power-inefficient, as opposed to simple buffering. Analog signal noise sources could negate improvements in device characteristics, especially in large arrays. Reconfigurable routing for mapping arbitrary NN topologies onto the same piece of hardware is necessary for reuse, and must potentially tie in with higher level software frameworks. This is closely tied to finding mechanisms for fast export and import of weight information from/to the chip, allowing accelerated distributed deep learning, which is absolutely a must for competing against GPUs for training applications.

The requirements for forward inference are in general less stringent, although there are one or two unique challenges. Firstly, linear and symmetric conductance response is not needed as closed loop weight tuning can be used. Secondly, inference could possibly work even with a limited dynamic range of weights, benefitting from recent work on weight quantization [156], as well as hardware-aware training regularization approaches such as the one described in [122]. However, as discussed in table 2, devices would need to demonstrate excellent long-term resilience to drift in conductance, even at elevated temperatures. Furthermore, defect rates should (eventually) be low enough that many thousands or even millions of chips can be programmed with the exact same set of pre-trained weights, with minimal provisioning for spare rows/columns. As with training, a key challenge will be demonstrating larger hardware demos that can show equivalent accuracy to software. However, achieving significantly lower throughput per unit area than GPUs may still be okay, if these chips can deliver on ultra-low power inference that could make them ubiquitous in the mobile/embedded/consumer space.

ORCIDs

Geoffrey W Burr https://orcid.org/0000-0001-5717-2549

References

[1] Moore G E 1965 *Electronics* **38** 114–7
[2] Dennard R H, Gaensslen F H, Rideout V L, Bassous E and LeBlanc A R 1974 *IEEE J. Solid-State Circuits* **9** 256–68
[3] Conte T M, Track E and DeBenedictis E 2015 *Computer* **48** 10–3
[4] Mead C 1990 *Proc. IEEE* **78** 1629–36
[5] Nawrocki R A, Voyles R M and Shaheen S E 2016 *IEEE Trans. Electron Devices* **63** 3819–29
[6] Schuman C D, Potok T E, Patton R M, Birdwell J D, Dean M E, Rose G S and Plank J S 2017 arXiv:1705.06963
[7] Indiveri G *et al* 2011 *Frontiers Neurosci.* **5** 1–23

[8] Merolla P A *et al* 2014 *Science* **345** 668–73
[9] Benjamin B V, Gao P, McQuinn E, Choudhary S, Chandrasekaran A R, Bussat J M, Alvarez-Icaza R, Arthur J V, Merolla P A and Boahen K 2014 *Proc. IEEE* **102** 699–716
[10] Orchard G, Lagorce X, Posch C, Furber S B, Benosman R and Galluppi F 2015 Real-time event-driven spiking neural network object recognition on the spiNNaker platform *2015 IEEE Int. Symp. on Circuits and Systems* (IEEE) pp 2413–6
[11] Schmitt S *et al* 2017 Neuromorphic hardware in the loop: training a deep spiking network on the BrainScaleS wafer-scale system *2017 Int. Joint Conf. on Neural Networks* pp 2227–34
[12] Markram H, Gerstner W and Sjöström P J 2011 *Frontiers Synaptic Neurosci.* **3** 00004
[13] Jackson B L *et al* 2013 *ACM J. Emerg. Technol. Comput. Syst.* **9** 12
[14] Kim S *et al* 2015 NVM neuromorphic core with 64 k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous *in situ* learning *IEDM Technical Digest* (IEEE) p 17.1
[15] Lukoševičius M and Jaeger H 2009 *Comput. Sci. Rev.* **3** 127–49
[16] Hawkins J, Ahmad S, Purdy S and Lavin A 2016 Biological and machine intelligence (BAMI) initial online release 0.4 https://numenta.com/biological-and-machine-intelligence/
[17] Burr G W *et al* 2017 *Adv. Phys.* X **2** 89–124
[18] Yang J J, Strukov D B and Stewart D R 2013 *Nat. Nanotechnol.* **8** 13–24
[19] Chang T, Yang Y and Lu W 2013 *IEEE Circuits Syst. Mag.* **13** 56–73
[20] Li H H, Chen Y, Liu C, Strachan J P and Davila N 2017 *IEEE Consum. Electron. Mag.* **6** 94–103
[21] Narayanan P, Fumarola A, Sanches L, Lewis S, Hosokawa K, Shelby R M and Burr G W 2017 *IBM J. Res. Dev.* **61** 1–11
[22] McCulloch W and Pitts W 1943 *Bull. Math. Biophys.* **7** 115–33
[23] Rosenblatt F 1957 The perceptron—a perceiving, recognizing automaton *Technical Report* 85-460-1 and Cornell Aeronautical Laboratory
[24] LeCun Y, Bottou L, Bengio Y and Haffner P 1998 *Proc. IEEE* **86** 2278–324
[25] Schmidhuber J 2015 *Neural Netw.* **61** 85–117
[26] LeCun Y, Bengio Y and Hinton G 2015 *Nature* **521** 436–44
[27] Krizhevsky A, Sutskever I and Hinton G E 2012 Imagenet classification with deep convolutional neural networks *Advances in Neural Information Processing Systems* pp 1097–105
[28] Hinton G *et al* 2012 *IEEE Signal Process. Mag.* **29** 82–97
[29] Mnih V *et al* 2015 *Nature* **518** 529–33
[30] Silver D *et al* 2016 *Nature* **529** 484–9
[31] Sutskever I, Vinyals O and Le Q V 2014 Sequence to sequence learning with neural networks *Advances in Neural Information Processing Systems* pp 3104–12
[32] Cho K, Van Merriënboer B, Bahdanau D and Bengio Y 2014 arXiv:1409.1259
[33] Bahdanau D, Cho K and Bengio Y 2014 arXiv:1409.0473
[34] Bengio Y, Lee D, Bornschein J and Lin Z 2015 Towards biologically plausible deep learning arXiv:1502.04156
[35] Owens J D, Houston M, Luebke D, Green S, Stone J E and Phillips J C 2008 *Proc. IEEE* **96** 879–99
[36] Wang C, Lou W, Gong L, Jin L, Tan L, Hu Y, Li X and Zhou X 2017 arXiv:1712.04771
[37] Sze V, Chen Y H, Yang T J and Emer J S 2017 *Proc. IEEE* **105** 2295–329
[38] Jouppi N P *et al* 2017 In-datacenter performance analysis of a tensor processing unit *Proc. 44th Annual Int. Symp. on Computer Architecture* (ACM) pp 1–12
[39] Esser S K *et al* 2016 *Proc. Natl Acad. Sci.* **113** 11441–6
[40] Sze V 2017 *IEEE Solid-State Circuits Mag.* **9** 46–54
[41] Ng A Machine learning cs229.stanford.edu
[42] Li F F and Karpathy A Convolutional neural networks for visual recognition cs231n.stanford.edu
[43] Socher R Deep learning for natural language processing cs224d.stanford.edu
[44] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (Cambridge, MA: MIT Press)
[45] Hochreiter S and Schmidhuber J 1997 *Neural Comput.* **9** 1735–80
[46] Rumelhart D E, Hinton G E and Williams R J 1986 *Nature* **323** 533–6
[47] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift *Int. Conf. on Machine Learning* pp 448–56
[48] Salimans T and Kingma D P 2016 Weight normalization: a simple reparameterization to accelerate training of deep neural networks *Advances in Neural Information Processing Systems* pp 901–9
[49] Srivastava N, Hinton G, Krizhevsky A, Sutskever I and Salakhutdinov R 2014 *J. Mach. Learn. Res.* **15** 1929–58
[50] French R M 1999 *Trends Cogn. Sci.* **3** 128–35
[51] Hunter H 2018 IBM research achieves record deep learning performance with new software technology www.ibm.com/blogs/research/2017/08/distributed-deep-learning
[52] Chen C Y, Choi J, Brand D, Agrawal A, Zhang W and Gopalakrishnan K 2018 arXiv:1712.02679
[53] Chen T, Du Z, Sun N, Wang J, Wu C, Chen Y and Temam O 2014 *ACM SIGPLAN Not.* **49** 269–84
[54] Chen Y H, Krishna T, Emer J S and Sze V 2017 *IEEE J. Solid-State Circuits* **52** 127–38
[55] Zhang C, Li P, Sun G, Guan Y, Xiao B and Cong J 2015 Optimizing fpga-based accelerator design for deep convolutional neural networks *Proc. 2015 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays* (ACM) pp 161–70
[56] Courbariaux M, Bengio Y and David J P 2015 Binaryconnect: training deep neural networks with binary weights during propagations *Advances in Neural Information Processing Systems* pp 3123–31
[57] Han S, Mao H and Dally W J 2015 arXiv:1510.00149
[58] Gupta S, Agrawal A, Gopalakrishnan K and Narayanan P 2015 Deep learning with limited numerical precision *Proc. 32nd Int. Conf. on Machine Learning* pp 1737–46
[59] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* pp 770–8
[60] Gokmen T and Vlasov Y 2016 *Frontiers Neurosci.* **10** 333
[61] Burr G, Shelby R M, di Nolfo C, Jang J, Shenoy R, Narayanan P, Virwani K, Giacometti E U, Kurdi B and Hwang H 2014 *IEDM Technical Digest* T29.5
[62] Burr G W *et al* 2015 *IEEE Trans. Electron Devices* **62** 3498–507
[63] Alibart F, Zamanidoost E and Strukov D B 2013 *Nat. Commun.* **4** ncomms3072
[64] Kang J *et al* 2017 Time-dependent variability in RRAM-based analog neuromorphic system for pattern recognition *IEEE Int. Electron Devices Meeting* pp 6.4.1–4
[65] Chen P Y, Gao L and Yu S 2016 *IEEE Trans. Multi-Scale Comput. Syst.* **2** 257–64
[66] Chen P Y, Peng X and Yu S 2017 NeuroSim+: an integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures *IEEE Int. Electron Devices Meeting* pp 6.1.1–4
[67] Gokmen T, Onen M and Haensch W 2017 *Frontiers Neurosci.* **11** 538
[68] Agarwal S, Gedrim R B J, Hsia A H, Hughart D R, Fuller E J, Talin A A, James C D, Plimpton S J and Marinella M J 2017 Achieving ideal accuracies in analog neuromorphic

computing using periodic carry *2017 Symp. on VLSI Technology* (IEEE) pp T174–5
[69] Li Z, Chen P Y, Xu H and Yu S 2017 *IEEE Trans. Electron Devices* **64** 2721–7
[70] Ambrogio S *et al* 2018 *Nature* **558** 60–7
[71] Boybat I, Gallo M L, Nandakumar S R, Moraitis T, Parnell T, Tuma T, Rajendran B, Leblebici Y, Sebastian A and Eleftheriou E 2017 arXiv:1711.06507
[72] Kim H, Kim T, Kim J and Kim J J 2017 arXiv:1703.10642
[73] Jacobs-Gedrim R B, Agarwal S, Knisely K E, Stevens J E, van Heukelom M S, Hughart D R, Niroula J, James C D and Marinella M J 2017 Impact of linearity and write noise of analog resistive memory devices in a neural algorithm accelerator *IEEE Int. Conf. on Rebooting Computing* (IEEE) pp 1–10
[74] Agarwal S, Plimpton S J, Hughart D R, Hsia A H, Richter I, Cox J A, James C D and Marinella M J 2016 Resistive memory device requirements for a neural algorithm accelerator *Int. Joint Conf. on Neural Networks* (IEEE) pp 929–38
[75] Mohanty A, Du X, Chen P Y, Yu S and Cao Y 2017 Random sparse adaptation for accurate inference with inaccurate multi-level RRAM arrays *IEEE Int. Electron Devices Meeting* pp 6.3.1–4
[76] Yan B, Liu C, Liu X, Chen Y and Li H 2017 Understanding the trade-offs of device, circuit and application in ReRAM-based neuromorphic computing systems *IEEE Int. Electron Devices Meeting* pp 11.4.1–4
[77] Wang Y, Wen W, Song L and Li H H 2017 Classification accuracy improvement for neuromorphic computing systems with one-level precision synapses *22nd Asia and South Pacific Design Automation Conf.* (IEEE) pp 776–81
[78] Burr G W, Kurdi B N, Scott J C, Lam C H, Gopalakrishnan K and Shenoy R S 2008 *IBM J. Res. Dev.* **52** 449–64
[79] Russo U, Kamalanathan D, Ielmini D, Lacaita A L and Kozicki M N 2009 *IEEE Trans. Electron Devices* **56** 1040–7
[80] Nirschl T *et al* 2007 Write strategies for 2 and 4-bit multilevel phase-change memory *IEEE Int. Electron Devices Meeting* pp 461–4
[81] Vincent A F, Larroque J, Locatelli N, Romdhane N B, Bichler O, Gamrat C, Zhao W S, Klein J O, Galdin-Retailleau S and Querlioz D 2015 *IEEE Trans. Biomed. Circuits Syst.* **9** 166–74
[82] Mulaosmanovic H, Ocker J, Muller S, Noack M, Muller J, Polakowski P, Mikolajick T and Slesazeck S 2017 Novel ferroelectric FET based synapse for neuromorphic systems *2017 Symp. on VLSI Technology (IEEE)* pp T176–7
[83] Wong H S P, Lee H Y, Yu S, Chen Y S, Wu Y, Chen P S, Lee B, Chen F T and Tsai M J 2012 *Proc. IEEE* **100** 1951–70
[84] Ambrogio S, Balatti S, Cubeta A, Calderoni A, Ramaswamy N and Ielmini D 2014 *IEEE Trans. Electron Devices* **61** 2912–9
[85] Baek I *et al* 2004 *IEDM Technical Digest* pp 587–90
[86] Jameson J *et al* 2013 *IEDM Technical Digest* pp 30.1.1–4
[87] Suri M, Querlioz D, Bichler O, Palma G, Vianello E, Vuillaume D, Gamrat C and DeSalvo B 2013 *IEEE Trans. Electron Devices* **60** 2402–9
[88] Larentis S, Nardi F, Balatti S, Gilmer D C and Ielmini D 2012 *IEEE Trans. Electron Devices* **59** 2468–75
[89] Woo J, Moon K, Song J, Lee S, Kwak M, Park J and Hwang H 2016 *IEEE Electron Device Lett.* **37** 994–7
[90] Woo J, Padovani A, Moon K, Kwak M, Larcher L and Hwang H 2017 *IEEE Electron Device Lett.* **38** 1220–3
[91] Woo J, Moon K, Song J, Kwak M, Park J and Hwang H 2016 *IEEE Trans. Electron Devices* **63** 5064–7
[92] Wu H *et al* 2017 Device and circuit optimization of RRAM for neuromorphic computing *IEEE Int. Electron Devices Meeting* pp 11.5.1–4
[93] Moon K, Cha E, Park J, Gi S, Chu M, Baek K, Lee B, Oh S H and Hwang H 2016 *IEEE Electron Device Lett.* **37** 1067–70
[94] Govoreanu B, Redolfi A and Zhang L 2013 Vacancy-modulated conductive oxide resistive ram (VCMO-RRAM): an area-scalable switching current, self-compliant, highly nonlinear and wide on/off-window resistive switching cell *Proc. 2013 IEEE Int. Electron Devices Meeting* (IEEE) pp 256–9
[95] Moon K, Fumarola A, Sidler S, Jang J, Narayanan P, Shelby R M, Burr G W and Hwang H 2018 *IEEE J. Electron Devices Soc.* **6** 146–55
[96] Moon K, Kwak M, Park J, Lee D and Hwang H 2017 *IEEE Electron Device Lett.* **38** 1023–6
[97] Seong D J, Hassan M, Choi H, Lee J, Yoon J, Park J B, Lee W, Oh M S and Hwang H 2009 *IEEE Electron Device Lett.* **30** 919–21
[98] Ambrogio S, Balatti S, McCaffrey V, Wang D and Ielmini D 2015 *IEEE Trans. Electron Devices* **62** 3812–9
[99] Park J, Kwak M, Moon K, Woo J, Lee D and Hwang H 2016 *IEEE Electron Device Lett.* **37** 1559–62
[100] Wang Z, Ambrogio S, Balatti S, Sills S, Calderoni A, Ramaswamy N and Ielmini D 2016 *IEEE Trans. Electron Devices* **63** 4279–87
[101] Prezioso M, Merrikh-Bayat F, Hoskins B, Adam G, Likharev K K and Strukov D B 2015 *Nature* **521** 61–4
[102] Bayat F M, Prezioso M, Chakrabarti B, Kataeva I and Strukov D 2017 arXiv:1712.01253
[103] Yu S, Li Z, Chen P Y, Wu H, Gao B, Wang D, Wu W and Qian H 2016 Binary neural network with 16 Mb RRAM macro chip for classification and online training *IEEE Int. Electron Devices Meeting* (IEEE) p 16
[104] Yao P *et al* 2017 *Nat. Commun.* **8** 15199
[105] Belhumeur P N, Hespanha J P and Kriegman D J 1997 *IEEE Trans. Pattern Anal. Mach. Intell.* **19** 711–20
[106] Hu M *et al* 2016 Dot-product engine as computing memory to accelerate machine learning algorithms *17th Int. Symp. on Quality Electronic Design* (IEEE) pp 374–9
[107] Hu M, Strachan J P, Li Z, Grafals E M, Davila N, Graves C, Lam S, Ge N, Yang J J and Williams R S 2016 Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication *53nd ACM/EDAC/IEEE Design Automation Conf.* (IEEE) pp 1–6
[108] Garbin D, Bichler O, Vianello E, Rafhay Q, Gamrat C, Perniola L, Ghibaudo G and DeSalvo B 2014 Variability-tolerant convolutional neural network for pattern recognition applications based on oxram synapses *IEEE Int. Electron Devices Meeting* pp 28.4.1–4
[109] Garbin D, Vianello E, Bichler O, Azzaz M, Rafhay Q, Candelier P, Gamrat C, Ghibaudo G, DeSalvo B and Perniola L 2015 On the impact of oxram-based synapses variability on convolutional neural networks performance *Proc. 2015 IEEE/ACM Int. Symp. on Nanoscale Architectures* pp 193–8
[110] Garbin D, Vianello E, Bichler O, Rafhay Q, Gamrat C, Ghibaudo G, DeSalvo B and Perniola L 2015 *IEEE Trans. Electron Devices* **62** 2494–501
[111] Wouters D, Waser R and Wuttig M 2015 *Proc. IEEE* **103** 1274–88
[112] Gallo M L, Sebastian A, Mathis R, Manica M, Giefers H, Tuma T, Bekas C, Curioni A and Eleftheriou E 2018 *Nat. Electron.* **1** 246
[113] Nandakumar S R, Gallo M L, Boybat I, Rajendran B, Sebastian A and Eleftheriou E 2017 arXiv:1712.01192v1
[114] Le Gallo M, Sebastian A, Mathis R, Manica M, Giefers H, Tuma T, Bekas C, Curioni A and Eleftheriou E 2018 *Nat. Commun.* **1** 246–53
[115] Fuller E J, Gabaly F E, Léonard F, Agarwal S, Plimpton S J, Jacobs-Gedrim R B, James C D, Marinella M J and Talin A A 2017 *Adv. Mater.* **29** 1604310
[116] van de Burgt Y, Lubberman E, Fuller E J, Keene S T, Faria G C, Agarwal S, Marinella M J, Talin A A and Salleo A 2017 *Nat. Mater.* **16** 414–8

[117] Yang J J and Xia Q 2017 *Nat. Mater.* **16** 101–8
[118] Kim S, Gokmen T, Lee H M and Haensch W E 2017 Analog CMOS-based resistive processing unit for deep neural network training *IEEE 60th Int. Midwest Symp. on Circuits and Systems* pp 422–5
[119] Sanches L L *et al* 2017 Multilayer perceptron algorithm: Impact of nonideal conductance and area-efficient peripheral circuits *Neuro-Inspired Computing Using Resistive Synaptic Devices* (Berlin: Springer) pp 209–31
[120] Jerry M, Chen P Y, Zhang J, Sharma P, Ni K, Yu S and Datta S 2017 Ferroelectric fet analog synapse for acceleration of deep neural network training *IEEE Int. Electron Devices Meeting* pp 6.2.1–4
[121] Oh S, Kim T, Kwak M, Song J, Woo J, Jeon S, Yoo I K and Hwang H 2017 *IEEE Electron Device Lett.* **38** 732–5
[122] Obradovic B, Rakshit T, Hatcher R, Kittl J, Sengupta R, Hong J G and Rodder M S 2017 *IEEE J. Electr. Dev. Soc.* **6** 438–48
[123] Farhat N H, Psaltis D, Prata A and Paek E 1985 *Appl. Opt.* **24** 1469–75
[124] Macukow B and Arsenault H H 1987 *Appl. Opt.* **26** 924–8
[125] Jang J S, Jung S W, Lee S Y and Shin S Y 1988 *Opt. Lett.* **13** 248–50
[126] White H J and Wright W A 1988 *Appl. Opt.* **27** 331–8
[127] Kravtsov K, Fok M P, Rosenbluth D and Prucnal P R 2011 *Opt. Express* **19** 2133–47
[128] Nahmias M A, Shastri B J, Tait A N and Prucnal P R 2013 *IEEE J. Sel. Top. Quantum Electron.* **19** 1–12
[129] Tait A N, Nahmias M A, Shastri B J and Prucnal P R 2014 *J. Lightwave Technol.* **32** 4029–41
[130] Vandoorne K, Dambre J, Verstraeten D, Schrauwen B and Bienstman P 2011 *IEEE Trans. Neural Netw.* **22** 1469–81
[131] Duport F, Schneider B, Smerieri A, Haelterman M and Massar S 2012 *Opt. Express* **20** 22783–95
[132] Vandoorne K, Mechet P, Vaerenbergh T V, Fiers M, Morthier G, Verstraeten D, Schrauwen B, Dambre J and Bienstman P 2014 *Nat. Commun.* **5** 3541
[133] Duport F, Smerieri A, Akrout A, Haelterman M and Massar S 2016 *Sci. Rep.* **6** 22381
[134] Ferreira de Lima T, Shastri B J, Tait A N, Nahmias M A and Prucnal P R 2017 *Nanophotonics* **6** 8029–44
[135] Prucnal P R, Shastri B J, Lima T F D, Nahmias M A and Tait A N 2016 *Adv. Opt. Photon.* **8** 228–99
[136] Van der Sande G, Brunner D and Soriano M C 2017 *Nanophotonics* **6** 561–76
[137] Rosenbluth D, Kravtsov K, Fok M P and Prucnal P R 2009 *Opt. Express* **17** 22767–72
[138] Fok M P, Deming H, Nahmias M, Rafidi N, Rosenbluth D, Tait A, Tian Y and Prucnal P R 2011 *Opt. Lett.* **36** 19–21
[139] Vaerenbergh T V, Fiers M, Mechet P, Spuesens T, Kumar R, Morthier G, Schrauwen B, Dambre J and Bienstman P 2012 *Opt. Express* **20** 20292–308
[140] Mesaritakis C, Papataxiarhis V and Syvridis D 2013 *JOSA* B **30** 3048–55
[141] Tait A N, Wu A X, Lima T F d, Zhou E, Shastri B J, Nahmias M A and Prucnal P R 2016 *IEEE J. Sel. Top. Quantum Electron.* **22** 312–25
[142] Tait A N, Lima T F, Zhou E, Wu A X, Nahmias M A, Shastri B J and Prucnal P R 2017 *Sci. Rep.* **7** 7430
[143] Abel S, Stferle T, Marchiori C, Caimi D, Czornomaz L, Stuckelberger M, Sousa M, Offrein B J and Fompeyrine J 2016 *J. Lightwave Technol.* **34** 1688–93
[144] Abel S, Stark D J, Eltes F, Caimi D, Frompeyrine J and Ortmann J E 2017 *Proc., ICRC 2017*
[145] Rios C, Youngblood N, Cheng Z, Gallo M L, Pernice W H, Wright C D, Sebastian A and Bhaskaran H 2018 arXiv:1801.06228
[146] Feldmann J, Stegmaier M, Gruhler N, Rios C, Bhaskaran H, Wright C D and Pernice W H P 2017 *Nat. Commun.* **8** 1256
[147] Shen Y *et al* 2017 *Nat. Photon.* **11** 441
[148] Maass W, Natschlaeger T and Markram H 2002 *Neural Comput.* **14** 2531–60
[149] Appeltant L, Soriano M C, Sande G V D, Danckaert J, Massar S, Dambre J, Schrauwen B, Mirasso C R and Fischer I 2011 *Nat. Commun.* **2** 468
[150] Brunner D, Soriano M C, Mirasso C R and Fischer I 2013 *Nat. Commun.* **4** 1364
[151] Larger L, Soriano M C, Brunner D, Appeltant L, Gutierrez J M, Pesquera L, Mirasso C R and Fischer I 2012 *Opt. Express* **20** 3241–9
[152] Martinenghi R, Rybalko S, Jacquot M, Chembo Y K and Larger L 2012 *Phys. Rev. Lett.* **108** 244101
[153] Paquot Y, Duport F, Smerieri A, Dambre J, Schrauwen B, Haelterman M and Massar S 2012 *Sci. Rep.* **2** 287
[154] Bae J H, Lim S, Park B G and Lee J H 2017 *IEEE Electron Device Lett.* **38** 1153–6
[155] Du Y, Du L, Gu X, Wang X and Chang M C F 2017 arXiv:1709.06614
[156] Tang W, Hua G and Wang L 2017 How to train a compact binary neural network with high accuracy? *31st AAAI Conf. on Artificial Intelligence* pp 2625–31
[157] Choi S, Tan S H, Li Z, Kim Y, Choi C, Chen P Y, Yeon H, Yu S and Kim J 2018 *Nat. Mater.* **17** 335
[158] Guo X, Bayat F M, Bavandpour M, Klachko M, Mahmoodi M R, Prezioso M, Likharev K K and Strukov D B 2017 Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded nor ash memory technology *IEEE Int. Electron Devices Meeting* pp 6.5.1–4
[159] Lin Y P, Bennett C H, Cabaret T, Vodenicarevic D, Chabi D, Querlioz D, Jousselme B, Derycke V and Klein J O 2016 *Sci. Rep.* **6** 31932
[160] Shafiee A, Nag A, Muralimanohar N, Balasubramonian R, Strachan J P, Hu M, Williams R S and Srikumar V 2016 Isaac: a convolutional neural network accelerator with *in situ* analog arithmetic in crossbars *Proc. 43rd Int. Symp. on Computer Architecture* (IEEE) pp 14–26
[161] Simonyan K and Zisserman A 2014 arXiv:1409.1556
[162] Chi P, Li S, Xu C, Zhang T, Zhao J, Liu Y, Wang Y and Xie Y 2016 Prime: a novel processing-inmemory architecture for neural network computation in reram-based main memory *Proc. 43rd Int. Symp. on Computer Architecture* (IEEE) pp 27–39
[163] Liu X *et al* 2015 Reno: A high-efficient reconfigurable neuromorphic computing accelerator design *52nd ACM/EDAC/IEEE Design Automation Conf.* (IEEE) pp 1–6
[164] Fan D and Angizi S 2017 Energy efficient in-memory binary deep neural network accelerator with dual-mode sot-mram *IEEE Int. Conf. on Computer Design* pp 609–12
[165] Li S, Niu D, Malladi K T, Zheng H, Brennan B and Xie Y 2017 Drisa: a dram-based reconfigurable *in situ* accelerator *Proc. 50th Annual IEEE/ACM Int. Symp. on Microarchitecture* (New York: ACM) pp 288–301
[166] Hasan R and Taha T M 2014 Enabling back propagation training of memristor crossbar neuromorphic processors *Int. Joint Conf. on Neural Networks* (IEEE) pp 21–8
[167] Taha T M, Hasan R and Yakopcic C 2014 Memristor crossbar based multicore neuromorphic processors *27th IEEE Int. System-on-Chip Conf.* (IEEE) pp 383–9
[168] Bojnordi M N and Ipek E 2016 Memristive boltzmann machine: a hardware accelerator for combinatorial optimization and deep learning *IEEE Int. Symp. on High Performance Computer Architecture* pp 1–13
[169] Boybat I *et al* 2017 Improved deep neural network hardware-accelerators based on non-volatile-memory: the local gains technique *IEEE Int. Conf. on Rebooting Computing* pp 1–8
[170] Donahue J *et al* 2014 Decaf: a deep convolutional activation feature for generic visual recognition *Int. Conf. Machine Learning* pp 647–55