MEMRISTIVE ARRAYS ARE A NATURAL fit to implement spiking neural network (SNN) acceleration. Representing information as digital spiking events can improve noise margins and tolerance to device variability compared to analog bitline current summation approaches to multiply–accumulate (MAC) operations. Restricting neuron activations to single-bit spikes also alleviates the significant analog-to-digital converter (ADC) overhead that mixed-signal approaches have struggled to overcome. Binarized, and more generally, limited-precision, NNs are considered to trade off computational overhead with model accuracy, but unlike conventional deep learning models, SNNs do not encode information in the precision-constrained amplitude of the spike. Rather, information may be encoded in the spike time as a temporal code, in the spike frequency as a rate code, and in any number of standalone and combined codes. Even if activations and weights are bounded in precision, time can be thought of as continuous and provides an alternative dimension to encode information in. This article explores the challenges that face the memristor-based acceleration of NNs and how binarized SNNs (BSNNs) may offer a good fit for these emerging hardware systems.

PERSPECTIVE

Nature has engineered the most efficient computational processor, and yet the blueprint of the brain remains a mystery. As deep learning models scale to unsustainably large sizes, the question begs to be asked: How does the brain achieve within 20 W what it takes data centers hundreds of thousands of watts to process [1]–[3]? With NN training runtimes and data generation experiencing exponential rates of growth during the past decade [4], [5], there is a justified concern that advances in hardware will struggle to keep up, and the benefits derived from modern deep learning may soon saturate.

Optimizations need to be made across the full stack of computing. High-precision weights are unlikely to be the most efficient data type to encode information in NNs [6].

Memristor-Based Binarized Spiking Neural Networks

Challenges and applications.

JASON K. ESHRAGHIAN, XINXIN WANG, AND WEI D. LU

Digital Object Identifier 10.1109/MNANO.2022.3141443
Date of current version: 26 January 2022

1932-4510/22©2022IEEE

The isolation of memory and processing compounds data transmission costs [7], [8]. Synchronous neuron activations demand regular (and often redundant) memory accesses [9]. Emerging memory technologies, such as memristors/resistive random-access memory (RAM), are reducing the gap between the physical and algorithmic layers of computing from a bottom-up approach, while SNNs draw inspiration from the brain's spike-based computational paradigm, providing top-down integration opportunities [10]. Combining these approaches can overcome several challenges that face the development of memristive accelerators while reducing the adverse impact of limited-precision computation. A high-level overview of how conventional von Neumann processing isolates the various layers, and how in-memory computing aims to converge these, is depicted in Figure 1.

This article provides our perspective on the challenges that face NN acceleration using memristive hardware and how BSNNs are positioned to overcome many of them. We present early empirical evidence of how spiking neurons can be parametrized to facilitate learning convergence in BSNNs and how this reduces the burden of mixed-signal processing in memristive accelerators.

> Representing information as digital spiking events can improve noise margins and tolerance to device variability.

OPTIMIZING ACROSS THE STACK

Processing a NN, spiking or otherwise, relies heavily on frequent data movement between the processor and memory, and emerging memory technologies that can be directly integrated with advanced CMOS processes offer a promising way to reduce the cost of regular memory access [11], [12]. Most neuromorphic designs and NN accelerators address this by distributing memory arrays across processing units, which represents a form of near-memory processing [13]–[15]. Similarly, in-memory processing physically unites memory and computation within the same substrate [8], [16]–[19] and is thought to be analogous to how the brain can both store and operate on information within synapses (Figure 1).

A common way to implement MAC operations using memristors is to synchronously draw current through an array of memristors, where the magnitude of current through each memristor is the product of the input voltage amplitude with the programmable conductance of the memristor. Charge is accumulated along the bitline such that the resultant

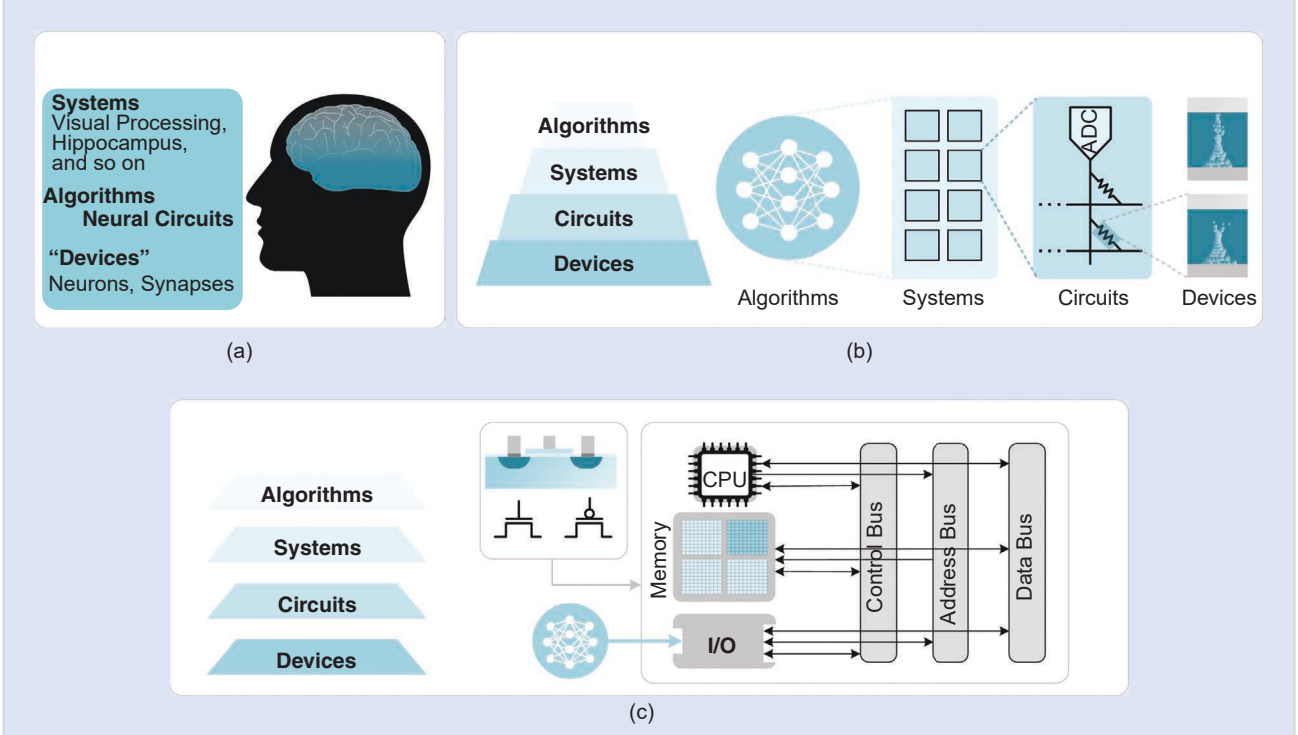

FIGURE 1 (a) Biological processing colocates the neural algorithm with the physical substrate it is processed on. Modern computing, on the other hand, optimizes for generality such that algorithms are treated somewhat independently of the hardware they are processed on. Each instruction is independently executed with repeated calls to and from memory, which means algorithms that are suited for parallelization are processed in a slow and sequential manner. (b) In-memory computing paradigms aim to reduce the gap between various abstractions by using devices to directly implement a small set of algorithms in a parallel and efficient approach. (c) The von Neumann architecture. I/O: input–output.

current represents an analog sum of products [20], [21]. Given that the dominant operation of NNs concerns matrix–vector multiplications, this approach provides a highly parallelized implementation of MACs and theoretically offers among the highest energy efficiency within constrained resources [Figure 2(a)] [22], [23]. In practice, such an implementation faces several challenges to widespread adoption and commercialization, but a shift to spike-based computing can potentially overcome these issues, which are summarized in Table 1.

PERFORMANCE-LEVEL CHALLENGES: ENERGY CONSUMPTION AND HEAT DISSIPATION

Conventional artificial NNs, such as deep learning models, are, by default, synchronous algorithms. Every time an input is passed to a neuron, an activation is guaranteed to be returned in lockstep. The highly parallelized processing in memristive arrays from Figure 2(a) comes at the cost of highly parallel resistive heat dissipation. Heat dissipation is already a bottleneck in high-performance, single-layer CPUs [24], and 3D integration scales power density with increasing layers. This problem is exacerbated by pushing additional layers farther from bottom-mounted heatsinks, which drives junction temperatures higher [25]. Continued transistor scaling and 3D integration increase on-chip power density, which leads to higher temperatures and has harmful effects on reliability and performance. Many advances in deep learning through the past decade have depended on architecture expansion and model scaling [1], [26], and, unfortunately, heat dissipation may fundamentally impose an upper bound on the number of operations and memory accesses that can take place.

The distributed representation of data through time as spikes enables an asynchronous mode of processing, as illustrated in Figure 2(b). Even when an input excitation is applied to a neuron, there is no guarantee that it will trigger an activation, which eliminates the need for downstream processing of the effect a spike has on subsequent layers, unlike conventional deep learning models. Current accumulation along a bitline may be traded for either charge-based integration (to calculate the membrane potential of an integrating neuron model; see the "Spiking Neuron Model" section) or low-precision voltage conversion. When spikes are sparse, only a small proportion of the input spikes are applied to a single array, and the total bitline current can be reduced by the same factor as the number of rows in the array, which is often in excess of two orders of magnitude [12], [27]. Such an extreme reduction of bitline current significantly lessens resistive dissipation in large-scale memristor arrays. SNN activation sparsity enables reduced processing and fewer memory read/writes and offers an algorithmic solution to lessening the heat dissipation burden on vertically integrated memristive arrays.

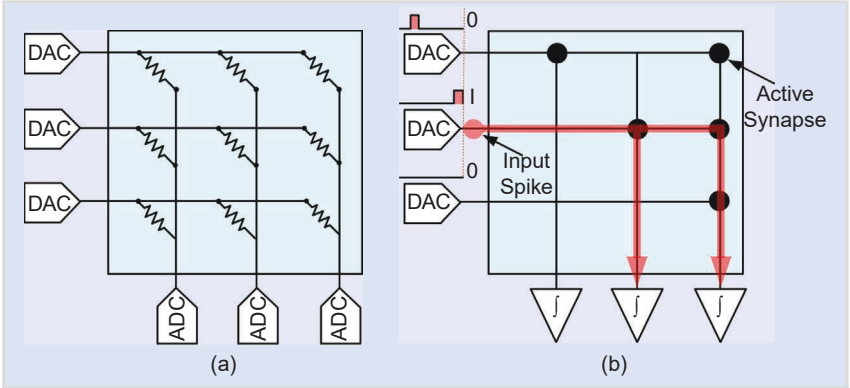

FIGURE 2 NN acceleration using memristive in-memory computing. (a) Memristors store weights as programmable conductances. MAC operations are performed on the basis of Ohm's law and Kirchhoff's current law by accumulating charge along the bitline. An analog current is converted into a digital voltage by an ADC for further processing and communication in the digital domain. (b) SNN acceleration using memristive hardware. The same operating principle is taken as in the preceding, where the input is constrained to sparsely triggered spikes. Eliminating the need for bitline charge accumulation enables the use of low-precision ADCs for subsequent integration or simply a sense amplifier to trigger a spike if the bitline capacitance is used to integrate charge. A typical array may have between 64 and 256 bitlines. DAC: digital-to-analog converter.

DEVICE-LEVEL CHALLENGES: VARIATIONS AND CONDUCTANCE DRIFT

The analog characteristics of memristors are susceptible to device variations and

TABLE 1 The challenges and solutions of memristor-based NN acceleration.

ABSTRACTION	MEMRISTIVE ARRAY CHALLENGES	SNN-BASED SOLUTIONS
Device		
Variance	D2D and C2C variation	Binarized weights and activations
conductance drift	Destructive readout	Sparsity and activity regularization
Circuit		
ADC overhead	Peripheral circuitry dominates on-chip power/area	Temporal data encoding
Sneak path currents	Scaling, IR drop, charge leakage	Large R_{on}/R_{off} ratio, lateral inhibition
System		
Data path stalling	Shared weights/limited resources increase latency	Sparse interlayer communication
Collisions	Synchronous data processing	
Algorithm		
Parameter quantization	Compressed models are less tolerant to quantization noise	High-precision time steps

D2D: device-to-device; C2C: cycle-to-cycle.

perturbations. For example, electrochemical metallization and filamentary valence change mechanism cells rely on ion migration to modulate resistance, and the fundamental stochastic filament formation and dissolution processes can result in large variations of device conductance [28]–[33]. When using single-bit activations that are emitted only once a neuron threshold is met, any incident noise is absorbed into the subthreshold dynamics of the neuron and thus largely eliminated. The variance of the spike amplitude no longer has a significant effect, as all necessary information is retained in the timing of the spike. The tradeoff of noise absorption is that relevant signals may also be eliminated. When errors are introduced into the weight update process, it has been shown that learning convergence can take a larger number of training epochs [34], [35].

Repetitive application of a driving voltage may lead to destructive readout and unintended programming in synchronous systems. At nanoscale dimensions, even small voltages can cause a large buildup of electric fields in any given cell, thus providing sufficient energy to delocalize ions, which ultimately manifests as conductance drift [36]–[38]. Drift is tolerable to some extent for the same reasons that device variations are acceptable for threshold-gated activations. But SNNs can be treated as asynchronous, where a stored weight need only be accessed if a spike arrives at the corresponding presynaptic terminal. The frequency of spikes can also be decreased by training a network to promote sparsity, for example, by using techniques such as L1 regularization, which corresponds to lower memory access frequency and a reduced probability of read perturbations [39].

Device variance and conductance drift are highly undesirable where precise weight values are required [40]. The stringent noise margins for incremental analog programming have made it extremely challenging for large-scale implementations. In fact, all commercial memristive processes that are presently available conservatively opt for digital switching instead [41], [42]. To implement anything other than binarized weights will require additional cells and therefore increased area and power consumption. While binarized weights can adversely impact metrics such as accuracy in deep learning models, reliance on the continuous temporal dimension in SNNs may amortize this cost.

> Restricting neuron activations to single-bit spikes also alleviates the significant analog-to-digital converter overhead that mixed-signal approaches have struggled to overcome.

CIRCUIT-LEVEL CHALLENGES: ADC OVERHEAD AND SNEAK PATHS

The cost of data converters that interface analog bitline currents to downstream digital processing as well as path currents are two of the biggest challenges in deploying large-scale memristive accelerators. Nanoscale memristors offer extremely high density and vertical integration, and utilizing them as synaptic weights in bitline current summation is an incredibly efficient way to parallelize MAC operations in the analog domain, but conversion into a digital voltage offsets a considerable portion of this benefit. As major functional blocks of any mixed-signal memristor-based accelerator, ADCs have been shown to account for a significant portion of the area usage and power consumption [12], [27], [43], [44]. An estimate of the total area and power usage of the various blocks of an integrated memristor-CMOS chip [45] performing feedforward processing is provided in Figure 3. In this instance, ADCs occupy approximately 2.5× more area and consume 18× more power than the analog memristor modules.

The results of the MAC operations are often postprocessed through activation units and sent for further computation in subsequent layers, which makes the conversion process unavoidable; communicating digital signals is far more robust than for analog. Spike-based activation preserves the digitization process by generating a binarized

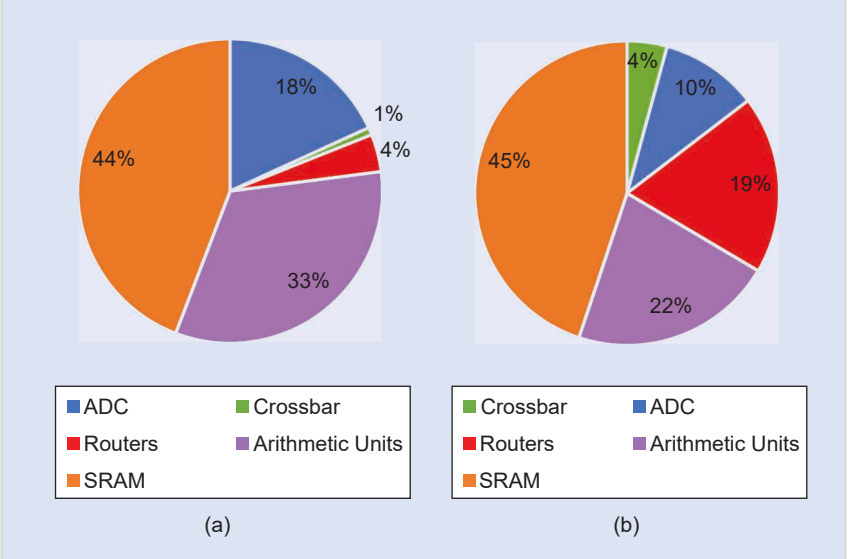

FIGURE 3 The (a) power and (b) area of every in-memory computing module proposed in [45]. ADCs consume 18% of the power and occupy 10% of the area in each module. SRAM: static RAM.

> Even if activations and weights are bounded in precision, time can be thought of as continuous and provides an alternative dimension to encode information in.

output (spike or not) and can potentially eliminate the need for ADCs, replacing them with current sense amplifiers and comparators to determine if the bitline current (or charge) exceeds the neuron's firing threshold.

Sneak path currents occur where charge leaks through low-resistance pathways across an array [46], [47]. In the absence of selectors and high-resistance devices, this can have a detrimental effect on analog computation, and spike-based computation addresses this issue in several ways. First, as with device variation, the noise tolerance of thresholded activations is much greater than that for nonspiking NNs. Second, using high temporal precision for each time step makes it increasingly unlikely for two spikes to occur in exact unison, such that only the largest bitline current needs to be transmitted to subsequent layers. That is to say, any sneak path currents may be neglected. Finally, SNNs may integrate lateral inhibition mechanisms that suppress firing from all neurons (other than the most excitable one) in a given layer [48]. In doing so, it can be guaranteed that only one neuron per layer is firing at any time, providing an alternative, biologically plausible mechanism to simply ignore all sneak path currents in this "winner-take-all" mechanism, while also sparsifying network activity [49], [50].

SYSTEM-LEVEL CHALLENGES: DATA PATH STALLING AND COLLISIONS

Where the number of data paths exceeds the number of parameters, the parameters are likely reused with each forward pass. This is especially the case for convolutions, where parameters/weights are shared among nodes. As an example, when processing a 224×224 image with the Visual Geometry Group (VGG) 16 architecture, 99% of the total operations are convolutions, and yet 99% of the memory is allocated to parameters in dense layers [51]. Unless weights are duplicated in memory, this often means the data path must be stalled while each input is sequentially applied to the weight, thus increasing latency [52]. This challenge is not limited to memristive hardware accelerators [53], [54]. Data path stalling may become a necessity where the bandwidth of peripheral circuitry is insufficient to cope with reading out the data from every bitline in a memristive array in parallel, which is especially the case for high-speed ADCs, which are commonly shared across multiple bitlines as they exceed the minimum pitch. Sharing is also commonly used to reduce the total ADC power and area, at a cost of increased data path latency. This is another instance where activation sparsity is highly beneficial, as there are fewer data in the signal path that may be stalled [55].

In terms of the interlayer communication of data, routing spikes between layers is far less expensive than directing multibit activations. Sparsity again plays a critical role, as the infrequent occurrence of spikes means communication channels are often free to route spikes to their designated target, which reduces the chance of collisions and thus having to stall the signal path for channels to be cleared. This benefit extends to all NN accelerators beyond memristor-based processing [56].

ALGORITHM-LEVEL CHALLENGES: UNDERPARAMETERIZED ARCHITECTURES

Processing NN algorithms on embedded and resource-constrained hardware typically requires using limited-precision techniques to reduce the amount of memory and resources required. This might appear to manifest as a device-level issue, where improving the analog programming reliability of memristors could theoretically solve the problem. However, there is significant interplay between weight/activation precision and model-specific accuracy. In general, quantization and binarization necessarily incur accuracy degradation as a result of the computational approximations that are made [57], [58]. While techniques such as quantization-aware training [59]–[61], asymmetrical quantization [62], [63], and knowledge distillation [64], [65] have been devised to reduce the impact of quantization noise, different types of models still show varying degrees of tolerance to such approximations.

In general, large-scale networks appear to be far more tolerant of quantization errors than constrained models are. For example, the work in [27] demonstrates how an overparameterized model, such as VGG16, with 138 million parameters, is far more robust to quantization noise than compressed model families, such as MobileNets, with 13 million parameters. When processed on tiled memristive arrays, VGG16 showed an accuracy degradation of only 2.41%, whereas MobileNet dropped far further, by 5.9%. The underlying issue here is that quantization and model compression techniques have been designed with the same goal of resource-constrained processing, but somewhat ironically, reducing the precision of compressed models is far more detrimental to accuracy. While various methods, such as those used in Tiny Machine Learning, are continuously being developed to help overcome these challenges to various degrees [66], SNNs may once again be considered a case that is less sensitive to the precision loss problem. By encoding data in the temporal domain, this partially sidesteps the challenges associated with limited-precision activations since even if weights were to be binarized, the temporal precision of a simulation may be enhanced to offset this. To summarize, SNNs have the potential to address several major challenges of memristive NN acceleration across the stack. The benefits of temporal data encoding,

single-bit interlayer communication via spikes, and sparse activations trickle down to extract more value out of devices, circuits, and architectures.

> There is a justified concern that advances in hardware will struggle to keep up, and the benefits derived from modern deep learning may soon saturate.

BSNNs

To ease hardware implementation and algorithm development, we consider SNNs that are topologically identical to conventional NNs, where the artificial neuron model is swapped for a spiking neuron model. This means the same types of layers can be represented, including dense, convolutional, residual, and recurrent layers. Since SNNs are also treated as acyclic graphs, the backpropagation-through-time (BPTT) algorithm can be used to train them in a manner almost identical to how it would be performed with a recurrent NN [67].

SPIKING NEURON MODEL

The leaky integrate-and-fire (LIF) neuron model is commonly used in conjunction with large-scale network models [68]. Although it is a very coarse simplification of biological neurons, the relatively simple dynamics make it computationally inexpensive and easily trainable with the BPTT algorithm. The LIF neuron is based on a passive membrane model consisting of a resistor and capacitor connected in parallel, where a spike is emitted from the neuron only when the membrane potential exceeds a firing threshold. This spike is typically encoded by the value "1," which is emitted to downstream neurons, and the membrane potential of the activated neuron is reset. Otherwise, no activation is emitted from the neuron (i.e., "0").

The first-order linear differential equation representing the *RC* circuit can be solved using the forward Euler method, where input current is substituted for a weighted input, $I_{\text{in}}[t] = WX[t]$, and several simplifications are made to reduce the number of tunable hyperparameters. A complete derivation is provided in [69], where the resulting time-discretized neuron model is represented by

$$U[t+1] = \underbrace{\beta U[t]}_{\text{decay}} + \underbrace{WX[t+1]}_{\text{input}} - \underbrace{S_{\text{out}}[t]\theta}_{\text{reset}}, \tag{1}$$

where $U[t]$ is the membrane potential of the neuron, or equivalently, the hidden state; $\beta \in [0,1]$ is the decay rate of the membrane potential; and $S_{\text{out}}[t] \in \{0,1\}$ is the output spike generated by the neuron. When the neuron is at rest, $S_{\text{out}} = 0$, and if the membrane potential exceeds the threshold θ, then $S_{\text{out}} = 1$, which activates the reset term, thus driving the membrane potential back down. The following equation formalizes the spike generation condition:

$$S_{\text{out}}[t] = \begin{cases} 1, & \text{if } U[t] \geq \theta \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The qualitative behavior of (1) is as follows. When an excitatory (or inhibitory input) is applied by $WX[t+1]$, the membrane potential experiences a sudden jump (or decrease), followed by an exponential relaxation back to $U \to 0$. A spike is emitted to subsequent layers if a positive jump causes $U[t]$ to reach the threshold θ. A full network implementation vectorizes U, X, and S_{out}. Here, W is a weight matrix, and in our experiments, β is treated as a global constant, though it can also be set to a learnable parameter [70], [71].

The result of (2) is already binarized, whereas several additional constraints must be imposed on (1) for binarized memristive processing. Specifically, $W \in \{-1,+1\}$, such that if a neuron is at rest, $U[t] = 0$, and an incoming spike $X[t+1] = 1$ is weighted by an excitatory synapse, $W = +1$, the membrane potential at the next time step will be $U[t+1] = 1$. If the threshold is normalized to $\theta = 1$, it guarantees that a spike is triggered, $S_{\text{out}} = 1$, and no long-term temporal leakage dynamics are experienced by the neuron as a result of the reset mechanism from (1) being activated. Therefore, W, X, and S_{out} are all binarized quantities, and U is also binarized in the absence of inhibitory inputs.

When a neuron is inhibited, which necessarily arises due to a negative weight (recalling that a spiking input X is restricted to either "1" or "0"), the membrane potential becomes unbounded. We apply membrane potential clipping by capping $U \in [-1,+1]$. This condition affects only neurons that are subject to an aggregate of multiple excitatory (or inhibitory) inputs. While it is not a necessary constraint, in practice, this limitation can be implemented by current limiting the bitline. It also eliminates the impact of any spike collisions that would otherwise cause the magnitude of the current to fall out of bounds.

TRAINING BSNNs

Two approximations are made to overcome the nondifferentiable operators that are applied to weights and hidden states during binarization and thresholded spiking. During the forward pass, (2) is applied to the membrane potential and acts as a threshold-shifted Heaviside operator. As the derivative of this function evaluates to "0" across almost the entire domain, the first approximation is to use a proxy for the gradient term during the backward pass, commonly referred to as a *surrogate gradient* [72]–[74]. In our experiments, we use the derivative of a threshold-shifted fast sigmoid function:

$$\tilde{S} = \frac{(U-\theta)}{(1+k|U-\theta|)}, \tag{3}$$

$$\frac{\partial \tilde{S}}{\partial U} = \frac{1}{(1+k|\theta-U|)^2}, \tag{4}$$

where the tilde above $\tilde{S}$ indicates that (3) is an approximation of (2); k is a hyperparameter that varies the steepness of the squashing function.

The second approximation accounts for the binarization of weights, where the

SNNs draw inspiration from the brain's spike-based computational paradigm, providing top-down integration opportunities.

SNN is trained using low-precision inference coupled with high-precision weight updates [74]. A floating-point version of the weights is stored, and a quantized version of the weights is used during the forward pass to calculate the loss at the output. During error backpropagation, the gradient is computed with respect to each layer's activations and binary weights. The gradients are used to update the high-precision weights during the update step. This process is repeated for all time steps.

Formally, consider W_r to be a full-precision proxy for the binarized weight W_b. During the forward pass,

$$W_b = \text{sign}(W_r). \tag{5}$$

Because the sign function is nondifferentiable, a straight-through estimator (STE) is used to backpropagate an estimation of the gradient to the full-precision proxy instead [74]. The STE treats the gradient process as though the binarization step had not occurred and takes the following form:

$$\frac{\partial W_b}{\partial W_r} = 1. \tag{6}$$

If gradient clipping is applied, (6) would otherwise evaluate to zero. To complete the picture, the backpropagation step approximates the gradient of the loss with respect to the binarized weight as follows, where it is assumed that the gradient with respect to each spike $\partial \mathcal{L}/\partial \tilde{S}$ is available:

$$\frac{\partial \mathcal{L}}{\partial W_r} = \frac{\partial \mathcal{L}}{\partial S}\frac{\partial \tilde{S}}{\partial U}\frac{\partial U}{\partial W_b}. \tag{7}$$

Each term in (7) can be analytically solved and is then used to calculate the magnitude and direction of the weight update. The form of (7) may be generalized using the chain rule to handle weight updates in deeper layers.

SPIKE-BASED OBJECTIVES

A variety of loss functions can formulate the supervised learning task as either a classification problem or classification disguised as a multivariate regression problem. In the former case, the neuron representing the correct class can be promoted to fire the most frequently in a rate code or fire first in a temporal code. The latter case may require correct and incorrect classes to have prespecified targets in a continuous range, such as a spike count, spike time, or target membrane potential [69].

Our experiments apply the cross-entropy loss function to the total spike count of the output layer. The aim is to maximize the spike count of the correct class $\vec{c} \in \mathbb{R}^{N_C}$, where N_C is the number of classes; c_i is the ith element of $\vec{c}$ and used as the logits in the softmax function:

$$p_i = \frac{e^{c_i}}{\sum_{i=1}^{N_C} e^{c_i}}. \tag{8}$$

TABLE 2 The network structure.

LAYER	INPUT CHANNEL	KERNEL SIZE	OUTPUT SIZE
Convolutional 1	1	5	16
Average pooling 1	16	2	16
Convolutional 2	16	5	64
Average pooling 2	64	2	64
Fully connected	$64 \times 7 \times 7$	2	10

The cross entropy between p_i and the one-hot encoded target vector $y_i \in \{0, 1\}^{N_C}$ is obtained using

$$\mathcal{L} = \sum_{i=0}^{N_C} y_i \log(p_i). \tag{9}$$

The sparsity of the spikes in BSNNs can be enhanced through the following approaches. A fast decay rate for the neuron model can prevent the neuron from firing spikes frequently, which may be suitable for static tasks that do not have memory dependence. However, this does not exploit the temporal dynamics of SNNs. Second, lateral inhibition between neurons in the same layer can be leveraged to prevent all other neurons from firing after the earliest spike occurs. Third, the postsynaptic current can be bounded within a fixed range, which can suppress excessive firing rates and also referred to as *activation clipping*.

EXPERIMENTAL RESULTS

On the basis of the spiking neuron model and training strategies discussed in the preceding, we designed several BSNNs to assess the use of biased gradient estimators that can be used to leverage memristive accelerators. The architecture of the proposed convolutional SNN is detailed in Table 2, where average pooling is applied to the weighted inputs rather than the output spikes. If applied to output spikes, the sparse nature of activations would cause each spike to be averaged with zeros and reduce each spike's influence on downstream layers. A similar issue occurs with maximum pooling, where better results can be obtained by downsampling the membrane potential rather than tiebreaking zeros and ones.

We performed a simulation sweep for training the proposed BSNN across the Modified National Institute of Standards and Technology (MNIST) and Fashion MNIST (FMNIST) data sets by using a variety of hyperparameters, where each simulation was conducted for 100 steps of duration trained using the Adam optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) [76]. The batch size of all experiments was fixed to 128, and batch normalization is applied to each convolution layer.

Full precision and BSNNs are used, with varying gradient, weight, and activation clipping conditions; surrogate gradient slope k; the initial learning rate; and dropout at the final dense layer. One hundred separate trials are conducted for each experiment of five epochs, each using a tree-structured Parzen estimator algorithm to randomly sample from the wide hyperparameter space [77]. The parameter sweep is depicted in a contour plot of the test set accuracy for the MNIST in Figure 4. The best hyperparameters from the sweep are then used for a longer simulation run using early stopping and a cosine annealing learning rate scheduler [78]. Final simulations were conducted in *snnTorch* [69], with the results on the test set listed in Table 3.

While accuracy degradation is expected for the BSNN compared with the full-precision models, inspection of the results indicates that BSNNs with larger thresholds and smaller surrogate gradient slopes [k from (4)] minimize the accuracy loss. Both observations are quite intuitive. First, larger thresholds are preferred, as this provides the spiking neuron with a wider dynamic range. While a threshold of "1" would immediately elicit an output spike in response to an input spike (assuming an initial condition of $U \geq 0$), a higher threshold affords the neuron with a larger range of permissible states before spiking may take place. However, the firing threshold cannot be arbitrarily increased since it will lead to reduced firing in deeper layers, which may cause problems with training convergence, due to vanishing gradients.

Second, the smaller gradient slope in BSNNs reduces the bias in the gradient estimator. As $k \rightarrow 0$, the gradient becomes more precise, at the expense of inducing dead neurons. When weights are binarized, error is introduced during the quantization-aware training step. To compensate for this, our results indicate that the slope must be decreased relative to the high-precision case to balance the amount of error injection during training. Since BSNNs are already injecting bias by using STEs at the binarization step and surrogate gradients at the thresholded firing step, we expect that the tendency to smaller surrogate gradient slopes is the network's way of reducing bias. Analogous to the threshold, k cannot be indefinitely decreased; otherwise, it will drive gradients to zero, i.e., the dead neuron problem. Although vanishing gradients and dead neurons are often described interchangeably, this result highlights a nuanced distinction between the two. Vanishing gradients occur as a result of the activation variance falling below "1," here as a result of reduced spiking. Dead neurons occur as a result of a nondifferentiable gradient.

Considering a typical memristor device that draws 3 μA of current in the low-resistance state, with a 0.3-V read voltage, an on/off ratio of 10, a 256 × 64 crossbar array size, and a 40-μW, 40-MHz, 8-bit successive-approximation

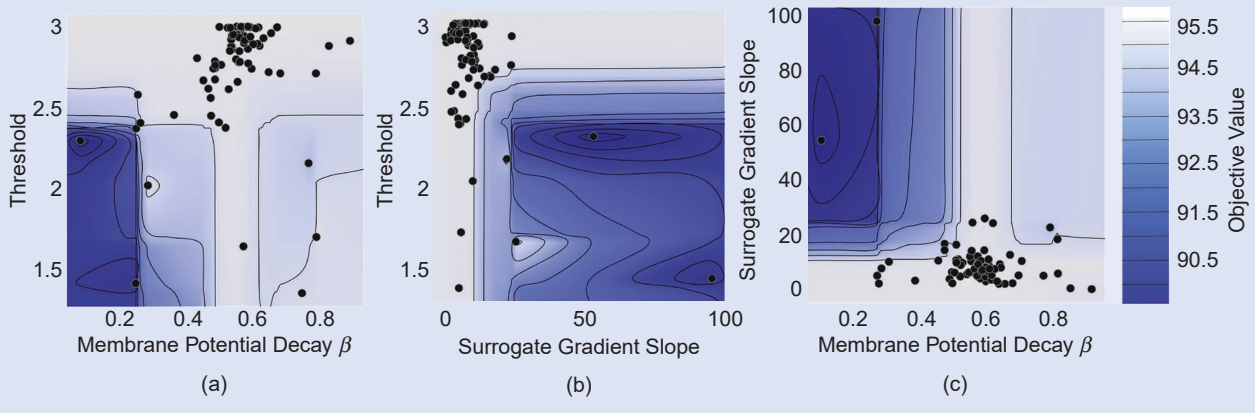

FIGURE 4 The test set accuracy after an MNIST parameter sweep across 100 training trials of five epochs each. Trials that do not appear promising on the basis of optimizing the test set accuracy are pruned using the asynchronous successive halving algorithm [75]. This explains the significant clustering of experiments, while other regions seem to be unexplored. (a) Membrane potential decay rate versus threshold. (b) Surrogate gradient slope versus threshold. (c) Membrane potential decay rate versus surrogate gradient slope.

TABLE 3 The experimental results.

DATA SET	NETWORK	BETA	LEARNING RATE	THRESHOLD	SLOPE	ACTIVATION CLIP	DROPOUT	RESULTS
MNIST	flt32 SNN	0.92	$1.9e-3$	2	6.03	True	0.1	99.29%
MNIST	BSNN	0.83	$2.7e-3$	6.4	2.8	True	0.1	98.9%
FMNIST	flt32 SNN	0.39	$2e-3$	1.5	7.66	True	0.1	91.13%
FMNIST	BSNN	0.87	$1.6e-2$	6.7	3.9	False	0	87.91%

The benefits of temporal data encoding, single-bit interlayer communication via spikes, and sparse activations trickle down to extract more value out of devices, circuits, and architectures.

(SAR) ADC, the throughput of the full-precision artificial NN proposed here is ~74 tera operations per second (TOPS)/W, where the ADC consumes 4.77% of the total power. For the BSNN with activation clipping, only 1-bit ADCs are required for the readout, and they can be implemented using charge-based comparators. The SAR ADC power consumption scales exponentially with the resolution, and the throughput of the three-layer network can be improved to ~153 TOPS/W, where the data conversion step consumes only 0.03% of the total power.

CONCLUSION

The challenges of mixed-signal NN memristive acceleration can be partially offset using SNNs, and we have provided a perspective on how characteristics of BSNNs can overcome issues that exist across the various abstraction layers. We conducted a simulation analysis to intuit the impacts of network parameters on accuracy and explored training strategies that may be used to improve performance. We showed that the preferred hyperparameters of BSNNs shift toward those that reduce the bias in gradient estimators while aiming to increase the available range in the subthreshold region of membrane potential. Our findings give an indication of how one might improve performance on temporally dynamic data sets, such as those procured from dynamic vision sensors, on resource-constrained hardware. While accuracy degradation may be intolerable on more challenging data sets, the power savings and circuit-level simplifications are a sound justification for further exploring how BSNNs can be pushed toward achieving similar performance on high-precision networks.

ACKNOWLEDGMENTS

This work was supported, in part, by Semiconductor Research Corporation and DARPA, through the Applications Driving Architectures Research Center, and the National Science Foundation, through awards ECCS-1915550 and CCF-1900675.

ABOUT THE AUTHORS

Jason K. Eshraghian (jasonesh@umich.edu) is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, 48109, USA.

Xinxin Wang (xinxinw@umich.edu) is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, 48109, USA.

Wei D. Lu (wluee@umich.edu) is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, 48109, USA.

REFERENCES

[1] T. B. Brown *et al.*, "Language models are few-shot learners," 2020, arXiv:2005.14165.

[2] P. Dhar, "The carbon impact of artificial intelligence," *Nature Mach. Intell.*, vol. 2, no. 8, pp. 423–425, 2020, doi: 10.1038/s42256-020-0219-9.

[3] L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," 2020, arXiv:2007.03051.

[4] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," 2020, arXiv:2007.05558.

[5] D. Amodei and D. Hernandez, "AI and compute," OpenAI, 2019. https://openai.com/blog/ai-and-compute/

[6] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn., PMLR*, 2015, pp. 1737–1746.

[7] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electron.*, vol. 1, no. 6, pp. 333–343, 2018, doi: 10.1038/s41928-018-0092-2.

[8] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electron.*, vol. 1, no. 1, pp. 22–29, 2018, doi: 10.1038/s41928-017-0006-8.

[9] F. Naveros, N. R. Luque, J. A. Garrido, R. R. Carrillo, M. Anguita, and E. Ros, "A spiking neural simulator integrating event-driven and time-driven computation schemes using parallel CPU-GPU co-processing: A case study," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 7, pp. 1567–1574, 2014, doi: 10.1109/TNNLS.2014.2345844.

[10] C. Frenkel, D. Bol, and G. Indiveri, "Bottom-up and top-down neural processing systems design: Neuromorphic intelligence as the convergence of natural and artificial intelligence," 2021, arXiv:2106.01288.

[11] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnol.*, vol. 15, no. 7, pp. 529–544, 2020, doi: 10.1038/s41565-020-0655-z.

[12] F. Cai *et al.*, "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," *Nature Electron.*, vol. 2, no. 7, pp. 290–299, 2019, doi: 10.1038/s41928-019-0270-x.

[13] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Architecture*, 2017, pp. 1–12, doi: 10.1145/3079856.3080246.

[14] M. Laskin *et al.*, "Parallel training of deep networks with local updates," 2020, arXiv:2012.03837.

[15] J. Zhu, J. Jiang, X. Chen, and C.-Y. Tsui, "SparseNN: An energy-efficient neural network accelerator exploiting input and output sparsity," in *Proc. 2018 Design, Autom. Test Europe Conf. Exhib. (DATE)*, pp. 241–244, doi: 10.23919/DATE.2018.8342010.

[16] J. K. Eshraghian, S.-M. Kang, S. Baek, G. Orchard, H. H.-C. Iu, and W. Lei, "Analog weights in ReRAM DNN accelerators," in *Proc. 2019 IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, pp. 267–271, doi: 10.1109/AICAS.2019.8771550.

[17] B. Chen, F. Cai, J. Zhou, W. Ma, P. Sheridan, and W. D. Lu, "Efficient in-memory computing architecture based on crossbar arrays," in *Proc. 2015 IEEE Int. Electron Devices Meeting (IEDM)*, pp. 17.5.1–17.5.4, doi: 10.1109/IEDM.2015.7409720.

[18] M. Rahimi Azghadi *et al.*, "Complementary metal-oxide semiconductor and memristive hardware for neuromorphic computing," *Adv. Intell. Syst.*, vol. 2, no. 5, p. 1,900,189, 2020, doi: 10.1002/aisy.201900189.

[19] S. M. Kang *et al.*, "How to build a memristive integrate-and-fire model for spiking neuronal signal generation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 12, pp. 4837–4850, 2021, doi: 10.1109/TCSI.2021.3126555.

[20] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, p. 19, doi: 10.1145/2897937.2898010.

[21] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Architecture News*, vol. 44, no. 3, pp. 14–26, Oct. 2016, doi: 10.1145/3007787.3001139.

[22] Z. Li *et al.*, "RRAM-DNN: An RRAM and model-compression empowered all-weights-on-chip DNN accelerator," *IEEE J. Solid-State Circuits*, vol. 56, no. 4, pp. 1105–1115, 2020, doi: 10.1109/JSSC.2020.3045369.

[23] W. Wan *et al.*, "A 74 TMCAS/W CMOSRRAM neurosynaptic core with dynamically reconfigurable dataflow and in-situ transposable weights for probabilistic graphical models," in *Proc. 2020 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pp. 498–500.

[24] M. S. Bakir *et al.*, "3D heterogeneous integrated systems: Liquid cooling, power delivery, and implementation," in *Proc. 2008 IEEE Custom*

Integrated Circuits Conf., pp. 663–670, doi: 10.1109/CICC.2008.4672173.
[25] D. Rich *et al.*, "Heterogeneous 3D nano-systems: The N3XT approach?" in *NANO-CHIPS 2030*, B. Murmann and B. Hoefflinger, Eds. Cham: Springer-Verlag, 2020, pp. 127–151.
[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inform. Process. Syst.*, 2012, vol. 25, pp. 1097–1105.
[27] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A deep neural network accelerator based on tiled RRAM architecture," in *Proc. 2019 IEEE Int. Electron Devices Meeting (IEDM)*, pp. 14.4.1–14.4.4, doi: 10.1109/IEDM19573.2019.8993641.
[28] I. Valov, R. Waser, J. R. Jameson, and M. N. Kozicki, "Electrochemical metallization memories—fundamentals, applications, prospects," *Nanotechnology*, vol. 22, no. 25, p. 254,003, 2011, doi: 10.1088/0957-4484/22/25/254003.
[29] S. Menzel, U. Bottger, and R. Waser, "Simulation of multilevel switching¨ in electrochemical metallization memory cells," *J. Appl. Phys.*, vol. 111, no. 1, p. 014501, 2012, doi: 10.1063/1.3673239.
[30] B. Choi *et al.*, "Resistive switching mechanism of TiO_2 thin films grown by atomic-layer deposition," *J. Appl. Phys.*, vol. 98, no. 3, p. 033715, 2005, doi: 10.1063/1.2001146.
[31] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008, doi: 10.1038/nature06932.
[32] N. Raghavan *et al.*, "Stochastic variability of vacancy filament configuration in ultra-thin dielectric RRAM and its impact on OFF-state reliability," in *Proc. 2013 IEEE Int. Electron Devices Meeting*, pp. 21.1.1–21.1.4, doi: 10.1109/IEDM.2013.6724674.
[33] Y. Li *et al.*, "Filament-free bulk resistive memory enables deterministic analogue switching," *Adv. Mater.*, vol. 32, no. 45, p. 2,003,984, 2020, doi: 10.1002/adma.202003984.
[34] B. A. Richards *et al.*, "A deep learning framework for neuroscience," *Nature Neurosci.*, vol. 22, no. 11, pp. 1761–1770, 2019, doi: 10.1038/s41593-019-0520-2.
[35] Y.-C. Chen, J. K. Eshraghian, I. Shipley, and M. Weiss, "Analog synaptic behaviors in carbon-based self-selective RRAM for in-memory supervised learning," in *Proc. 2021 IEEE 71st Electron. Components Technol. Conf. (ECTC)*, pp. 1645–1651, doi: 10.1109/ECTC32696.2021.00261.
[36] A. Chanthbouala *et al.* "A ferroelectric memristor," *Nature Mater.*, vol. 11, no. 10, pp. 860–864, 2012, doi: 10.1038/nmat3415.
[37] W. Shim, Y. Luo, J.-s. Seo, and S. Yu, "Impact of read disturb on multilevel RRAM based inference engine: Experiments and model prediction," in *Proc. 2020 IEEE Int. Reliab. Phys. Symp. (IRPS)*, pp. 1–5, doi: 10.1109/IRPS45951.2020.9129252.
[38] J. K. Eshraghian, K. Cho, and S. M. Kang, "A 3-D reconfigurable RRAM crossbar inference engine," in *Proc. 2021 IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1–5, doi: 10.1109/ISCAS51556.2021.9401672.
[39] P. T. P. Tang, T.-H. Lin, and M. Davies, "Sparse coding by spiking neural networks: Convergence theory and computational results," 2017, arXiv:1705.05475.
[40] C.-Y. Lin *et al.*, "Adaptive synaptic memory via lithium ion modulation in RRAM devices," *Small*, vol. 16, no. 42, p. 2,003,964, 2020, doi: 10.1002/smll.202003964.
[41] "eflash," TSM Corporation, Reston, VA, USA, 2020. [Online]. Available: https://www.tsmc.com/english/dedicatedFoundry/technology/eflash.htm
[42] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3D XPoint technology," *Proc. IEEE*, vol. 105, no. 9, pp. 1822–1833, 2017, doi: 10.1109/JPROC.2017.2731776.
[43] J. M. Correll *et al.*, "A fully integrated reprogrammable CMOS-RRAM compute-in-memory coprocessor for neuromorphic applications," *IEEE J. Explor. Solid-State Computat.*, vol. 6, no. 1, pp. 36–44, 2020, doi: 10.1109/JXCDC.2020.2992228.
[44] P. Yao *et al.*, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020, doi: 10.1038/s41586-020-1942-4.
[45] X. Wang *et al.*, "TAICHI: A tiled architecture for in-memory computing and heterogeneous integration," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, early access, 2021, doi: 10.1109/TCSII.2021.3097035.
[46] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectron. J.*, vol. 44, no. 2, pp. 176–183, 2013, doi: 10.1016/j.mejo.2012.10.001.
[47] C.-Y. Lin *et al.*, "A highspeed MIM resistive memory cell with an inherent vanadium selector," *Appl. Mater. Today*, vol. 21, p. 100,848, Dec. 2020, doi: 10.1016/j.apmt.2020.100848.
[48] C. Blakemore, R. H. Carpenter, and M. A. Georgeson, "Lateral inhibition between orientation detectors in the human visual system," *Nature*, vol. 228, no. 5266, pp. 37–39, 1970, doi: 10.1038/228037a0.
[49] I. Ebong and P. Mazumder, "Memristor based STDP learning network for position detection," in *Proc. 2010 Int. Conf. Microelectron.*, 2010, pp. 292–295, doi: 10.1109/ICM.2010.5696142.
[50] J. K. Eshraghian *et al.*, "Neuromorphic vision hybrid RRAM-CMOS architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, pp. 2816–2829, 2018, doi: 10.1109/TVLSI.2018.2829918.
[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, arXiv:1409.1556.
[52] X. Wang, M. A. Zidan, and W. D. Lu, "A crossbar-based in-memory computing architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 12, pp. 4224–4232, 2020, doi: 10.1109/TCSI.2020.3000468.
[53] S. Dong, X. Gong, Y. Sun, T. Baruah, and D. Kaeli, "Characterizing the microarchitectural implications of a convolutional neural network (CNN) execution on GPUs," in *Proc. 2018 ACM/SPEC Int. Conf. Performance Eng.*, pp. 96–106, doi: 10.1145/3184407.3184423.
[54] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proc. 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pp. 25–34, doi: 10.1145/3020078.3021698.
[55] P. Knag, C. Liu, and Z. Zhang, "A 1.40 mm^2 141mW 898GOPS sparse neuromorphic processor in 40nm CMOS," in *Proc. 2016 IEEE Symp. VLSI Circuits (VLSI-Circuits)*, pp. 1–2, doi: 10.1109/VLSIC.2016.7573526.
[56] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, 2015, doi: 10.1109/JSSC.2014.2386892.
[57] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inform. Process. Syst.*, 2016, vol. 29.
[58] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pp. 65–74, doi: 10.1145/3020078.3021744.
[59] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, arXiv:1510.00149.
[60] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," 2019, arXiv:1902.08153.
[61] S. Uhlich *et al.*, "Mixed precision DNNs: All you need is a good parametrization," 2019, arXiv:1905.11452.
[62] T. Zhang, Z. Lin, G. Yang, and C. D. Sa, "Qpytorch: A low-precision arithmetic simulation framework," 2019, arXiv:1910.04540.
[63] J. Huang, J. Reed, D. Khudia, and J. Park, "FBGEMM." 2021, GitHub. https://github.com/dskhudia/FBGEMM (Accessed: 2021).
[64] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, 2021, doi: 10.1007/s11263-021-01453-z.
[65] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," 2018, arXiv:1802.05668.
[66] C. R. Banbury *et al.*, "Benchmarking TinyML systems: Challenges and direction," 2020, arXiv:2003.04821.
[67] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990, doi: 10.1109/5.58337.
[68] L. Lapique, "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization," *J. Physiol. Pathol.*, vol. 9, pp. 620–635, 1907.
[69] J. K. Eshraghian *et al.*, "Training spiking neural networks using lessons from deep learning," 2021, arXiv:2109.12894.
[70] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 2661–2671.
[71] N. Perez-Nieves, V. C. Leung, P. L. Dragotti, and D. F. Goodman, "Neural heterogeneity promotes robust learning," *Nature Commun.*, vol. 12, no. 1, p. 5791, 2021, doi: 10.1038/s41467-021-26022-3.
[72] F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks," *Neural Comput.*, vol. 33, no. 4, pp. 899–925, 2021, doi: 10.1162/neco_a_01367.
[73] E. Hunsberger and C. Eliasmith, "Spiking deep networks with LIF neurons," 2015, arXiv:1510.08829.
[74] Y. Bengio, N. Leonard, and A. Courville, "Estimating or propagating´ gradients through stochastic neurons for conditional computation," 2013, arXiv:1308.3432.
[75] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proc. 19th Int. Conf. Artif. Intell. Statist.*, 2016, pp. 240–248.
[76] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, arXiv:1412.6980.
[77] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, vol. 24.
[78] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," 2016, arXiv:1608.03983.