Recent Progress in Real-Time Adaptable Digital Neuromorphic Hardware

Vladimir Kornijcuk and Doo Seok Jeong*

It has been three decades since neuromorphic engineering was first brought to public attention, which aimed to reverse-engineer the brain using analog, very large-scale, integrated circuits. Vigorous research in the past three decades has enriched neuromorphic systems for realizing this ambitious goal. Reverse engineering the brain essentially implies the inference and learning capabilities of a standalone neuromorphic system—particularly, the latter is referred to as embedded learning. The reconfigurability of a neuromorphic system is also pursued to make the system field-programmable. Bearing these desired attributes in mind, recent progress in digital neuromorphic hardware is overviewed, with an emphasis on real-time inference and adaptation. Real-time adaptation, that is, learning in realtime, highlights the feat of spiking neural networks with inherent rich dynamics, which allows the networks to learn from environments embodying an enormous amount of data. The realization of real-time adaptation imposes severe constraints on digital neuromorphic hardware design. Herein, the constraints and recent attempts to cope with the challenges arising from the constraints are addressed.

1. Introduction

Spiking neural networks (SNNs) are dynamic hypotheses that can be trained with data in both static and dynamic domains.1–3 The rich dynamics of SNNs arise from rich temporal kernels with which a time-varying input stimulus (for sensory neurons) and presynaptic neuronal responses (for postsynaptic neurons) are convolved. Often, multistage convolutions apply to neural response functions similar to that for a simple leaky integrate-and-fire (LIF) neuron model. The model first convolves a sequence of input spikes with a synaptic kernel to evaluate excitatory postsynaptic current (EPSC), which is subsequently convolved with a membrane potential kernel to evaluate the membrane potential. Such temporal kernels enrich the dynamics of SNN. Note that convolutional neuron networks (CNNs) convolve a static input with spatial, rather than temporal, kernels at an early stage of input processing. The convolutions with temporal kernels are easily implemented in spiking neuron models as shown in the spike response model.4 The timescale implemented using this dynamics of synaptic transmission reaches up to ≈100 ms with physiologically-plausible temporal kernels.5 The range of timescale considerably widens for the spiking dynamics of a group of spiking neurons with recurrent connections—up to a few tens of seconds beyond the range covered by dynamics of an individual synaptic transmission.4,6

Learning the sequence of data in a dynamic domain is deemed to leverage the rich dynamics of SNNs. Each component in the sequence is encoded as a decodable spatial and/or temporal pattern of a spike(s),4,7 and the sequence of components is encoded by the intervals between neighboring spike patterns. The interval may vary in a wide range of timescale given the wide range of time constant involved in spiking dynamics. A feasible algorithm for supervised sequence learning in stochastic recurrent SNNs was proposed by Brea, Senn, and Pfister,6 which is consistent with spike-based plasticity rules. This algorithm was successfully adopted to deterministic recurrent SNNs by Gardner and Grüning.7 Prior to these publications, a method to train deterministic recurrent SNNs was proposed, which is referred to as a remote supervised method8 that adjusts weights to produce a spike train identical to a target train by diminishing an error that does not include a derivative term.

Renowned deep neural networks (DNNs)9 are hypotheses that are also suitable for data in static and dynamic domains. Recurrent neural networks (RNNs) and their modifications are capable of learning sequences of inputs through recurrent connections10–14. Particularly, discrete-time RNNs15,16 can encode dynamic data with a fixed interval between neighboring time steps. Yet, the difference between the SNN and RNN lies in the sequence encoding method.

Neuromorphic engineering originally aimed at the hardware-implementation of SNNs using verylarge-scale integrated analog circuits and ultimately aimed to reverse engineer
Despite the vigorous research on neuromorphic hardware in the past three decades, the ultimate level has not been reached yet. The effort has, however, enriched implementation methods. To date, the available technologies embody conventional analog very-large scale integrated (VLSI) circuits, mixed analog/digital circuits, and fully digital circuits. Furthermore, nonvolatile memory-based approaches are emerging. That is, the research field of neuromorphic engineering has notably been expanding, and the consequent explosion of knowledge undoubtedly forms the solid basis for the achievement of the ambitious goal of neuromorphic engineering.

The modern goal of neuromorphic engineering is rather diverse but commonly application-oriented, for instance, acceleration of brain simulations using neuromorphic hardware and application of neuromorphic hardware to tasks that generally belong to the application domain of deep learning. BrainScaleS exemplifies the former goal in that wafer-scale large neuromorphic hardware significantly accelerates brain simulations even beyond real-time simulations. Additionally, a recent study on large-scale cortical circuit modeling using SpiNNaker identifies neuromorphic hardware as a powerful accelerator of brain simulations. The latter, however, views neuromorphic hardware as an energy- and time-efficient hardware platform for deep learning. This first needs to map deep learning onto deep SNNs by adopting deep learning techniques to SNNs—recent progress in this work is comprehensively reviewed in previous studies. Several prototypes of neuromorphic hardware, e.g., TrueNorth, SpiNNaker, DYNAPs, and Loihi, have shown their capability of object recognition. Other than these two main applications, the neural engineering framework offers a unique framework for the use of SNNs in particular function designs, which is successfully implemented in a recent neuromorphic hardware system. Yet, effort on unifying these seemingly different goals continues onward for the development of physiologically plausible sequence learning algorithms on the algorithm level. Developing a common platform across the goals fills the gap between the goals as noted for SpiNNaker. To either end, neuromorphic hardware that largely accelerates SNN simulations with excellent reconfigurability of network topology and learning algorithms is a common subject of research and development. Such hardware systems are the main concern of this article.

Among such diverse technologies, fully digital neuromorphic hardware recently attracts large attention, which may be triggered by the recent prototypes (TrueNorth and Loihi) of major chipmakers. In digital neuromorphic hardware, the key variables to neuronal response function, synaptic transmission, and learning are calculated in binary on discrete time steps. Note that the time steps are algorithmic steps, and thus, the algorithmic time is unnecessarily in sync with physical time. Yet, the synchronization with physical time is the key to real-time inference and adaptation of digital neuromorphic hardware. The main advantage of fully digital neuromorphic hardware over conventional analog hardware lies in its excellent scalability and reliability given the minimal effect of uncontrollable variables (e.g., size mismatch, inhomogeneous dopant profile, and so forth) on the hardware performance. Such uncontrollable variables considerably undermine the performance of analog circuits.

Regarding functionalities of neuromorphic hardware, two important keywords are inference and learning. An SNN produces output data in the output layer in response to input data with a given set of fixed synaptic weights, referred to as inference. Namely, inference is simultaneous neuronal encodings along parallel synaptic chains from the input to the output layer. Learning optimizes the set of synaptic weights given to the SNN according to the purpose. Much more data are associated with learning than inference, causing considerable difficulty in embedding learning algorithms in neuromorphic hardware. Notwithstanding the capacity of SNNs, it appears that the application hardly leverages its functional capabilities as evidenced by the current status of SNN applications to practical domains in comparison to deep learning. This may be due in part to difficulty in understanding SNNs and the consequent lack of application-specific learning algorithms and network architectures. The difficulty may arise from the use of time- and energy-inefficient test beds (general-purpose hardware) for SNN studies. Mapping SNNs to software inevitably causes a large gap between the runtime and simulated time. A workaround is the use of hardware dedicated to time- and energy-efficient realization of SNNs, which offers user-programmable network topology and learning algorithms. Digital neuromorphic hardware is the very hardware that considerably reduces the difficulty in SNN studies, and thus perhaps encourages the development of learning algorithms and SNN architectures that fully harness the potential capacity of SNNs.
In this regard, we address in this article recent developments of digital neuromorphic hardware as a user-programmable platform for SNN studies rather than particular application-specific hardware. To this end, this article overviews recent progress in digital neuromorphic hardware in conjunction with basics of its design. Particular emphasis will be placed on neuromorphic hardware design for real-time inference and adaptation. To this end, we begin with neuromorphic hardware with inference ability in Section 2. This section includes subsections dedicated to ideal neuromorphic hardware architecture compared with the von Neumann architecture (Section 2.1), introduction of neuromorphic cores to cope with the inefficiency of the ideal architecture (Section 2.2), and intra- and inter-core event-routing schemes, the key to real-time inference (Section 2.3). Section 3 is dedicated to real-time adaptable neuromorphic hardware to describe the additional constraints imposed on real-time inference. This section embodies desired features of embedded learning algorithms (Section 3.1), additional requirements for hardware (Section 3.2), and recent progress in eventrouting in an attempt to realize real-time adaptation (Section 3.3). Concluding remarks on challenges to boost the performance of neuromorphic hardware with regard to memory capacity are given in Section 4.

To begin with, we clarify the terms indicating neurons in different contexts (Figure 1). When invoking SNN topology, we used terms of “presynaptic neuron” and “postsynaptic neuron” by reference to unidirectionality in synaptic transmission. In Figure 1, Neuron 2 is a presynaptic neuron to Neuron 3 and a postsynaptic neuron to Neuron 1. Upon mapping the SNN to neuromorphic hardware, we switch the terms, presynaptic neuron and postsynaptic neuron, to “fan-in neuron” and “fan-out neuron”, respectively. Neuron 2 in Figure 1 is a fan-in and fan-out neuron to Neuron 3 and 1, respectively. Synapses are also referred to as “fan-in synapse” and “fan-out synapse” such that the synapses between a neuron and its fan-out neurons (fan-in neurons) are termed “fan-out synapses” (fan-in synapses). When addressing event delivery, “source neuron” and “destination neuron” indicate a neuron emitting a spike and a neuron receiving the spike, respectively. Note that any source neuron is a presynaptic and/or postsynaptic neuron, whereas all destination neurons are postsynaptic neurons in an event-delivery process.

2. Inference in Neuromorphic Hardware

2.1. Ideal Neuromorphic Architecture versus Von Neumann Architecture

A prevalent view on neuromorphic hardware is that it is of non-von Neumann architecture. To begin with, we briefly address von Neumann architecture underlying the modern computer. The architecture consists of separate devices—a central processing unit (CPU), memory, and input/output (I/O) devices. The memory commonly includes random access memory (RAM) and read-only memory (Figure 2a). The CPU is connected to the other devices through system buses (control bus, address bus, and data bus). 2^N-to-N encoded signals are placed on each bus so that the use of buses significantly reduces the number of lines for communication. Yet, a downside is that the bus limits the signal transfer rate to only one signal per clock cycle, causing the notorious von Neumann bottleneck. The control bus allows communication between the CPU and other devices through control signals. For instance, when reading data in the memory, the CPU sends a read signal through the control bus to the memory, which is decoded and subsequently triggers the read operation within the memory. A memory address which the CPU reads from or write to is sent to the memory through the address bus, preceding the read or write operation. Subsequently, the data to read or write are carried by the data bus. The feature of the von Neumann architecture lies in versatility in operation given the program instructions stored in the memory. The program instructions are programmable so that any user-programmed operations can be performed without changing any physical wiring.

In neuronal encoding, each neuron needs several variables and parameters that are exclusively dedicated to each neuron. The number of such variables and parameters differs for different neuron models implemented in the hardware. A point LIF neuron takes synaptic current and membrane potential as variables, and synaptic weight, current and potential decay constants, threshold for spiking, and refractory time as parameters. The larger the complexity in a neuron model implemented in the hardware, the richer is the dynamics exhibited by the model at the cost of more variables and parameters. For instance, multi-compartment neuron models incorporate separate dendrites whose potential values are separately evaluated alongside the somatic potential, so that the number of variables and parameters is proportional to the number of dendrites implemented.

A set of such variables and parameters is given to each neuron; they serve as local data to each neuron. To be specific, such data are local in network topology (topological local data) because no
other neurons need to address the data during simultaneous neuronal encoding. The corresponding (ideal) architecture of digital neuromorphic hardware is shown in Figure 2b—the memory for each processing unit (neuron) schematically contains its local data. Each processing unit in Figure 2b calculates the state variables of an implemented neuron model in binary, e.g., membrane potential and synaptic current for a point LIF neuron model. This architecture highlights the memory distributed over neurons in contrast to the von Neumann architecture in Figure 2a. Note that the memory is topologically, but unnecessarily spatially, local to the neurons—particularly, in digital neuromorphic hardware. In such hardware, topological local data can be placed in a single memory block that is arbitrarily accessed by neurons through parallel buses. The architecture of distributed processors and their local memories realize perfectly parallel computation that maximizes the number of synaptic operations per second (SynOps/s), an important measure of neuromorphic hardware performance. SynOps/s indicates the number of neuronal membrane potential updates triggered by presynaptic spikes per second. In this ideal architecture with hardwired neurons, this synaptic operation rate scales with the number of neurons given the perfect parallelism.

2.2. Introduction of Neuromorphic Cores to Neuromorphic Hardware Architecture

Albeit ideal regarding the parallelism, this architecture is hardly implementable due to the remarkable circuit overhead arising from the arithmetic logic circuit assigned to each neuron, which evaluates the neuronal variables on a given time step. Moreover, hardwiring the distributed neurons to each other is impractical for large-scale SNNs and significantly hinders the reconfigurability of neuromorphic hardware. A practical solution to the first issue is to group a number of neurons in a core as schematized in Figure 3. Each core is given an arithmetic logic circuit for neuronal variable update and binary comparator, which are shared by the neurons in the same core. The consequent reduction in the effective circuit area per neuron compared with the ideal (but impractical) architecture in Figure 2b is a significant advantage. Moreover, deploying cores enables efficient usage of memory such that the neurons in the same core share neuronal parameters in common. Such parameters in the LIF neuron model likely include current and potential decay constants, threshold for spiking, and refractory time.

Fan-in and fan-out neurons are occasionally placed in different cores, for instance, N1 and N2 in Figure 1 in two different cores. The synaptic weight can be placed in either the fan-out neuron’s core or the fan-in neuron’s one because duplicating the weight in both cores is a waste of memory. The former is preferred with regard to efficient neuronal encoding in that the fan-in synaptic weight (indispensable for neuronal encoding) can be retrieved within the same core as the fan-out neuron. Otherwise, the fan-in synaptic weight should be delivered from the fan-in neuron’s core to the fan-out neuron’s one over a data bus, giving rise to an increase in circuit overhead and power consumption.

Yet, this advantage of use of cores comes at the cost of a reduction in parallelism in that the shared arithmetic logic circuit can be accessed by only one neuron at a time. When an event from a particular fan-in neuron comes into a core, the destination (fan-out) neurons are searched in a parallel or serial manner, and their membrane potentials are serially updated (one at a time) because they use a common arithmetic logic circuit. That is,
the update uses time-division multiplexing. The time-division multiplexing also allows the neurons in the same core to share the common neuronal parameters in the memory, which are addressed by one neuron at a time. This serial update consumes time in proportion to the number of the fan-out neurons. This hardly holds for the ideal architecture in Figure 2b, where the membrane potentials of the fan-out neurons are simultaneously updated irrespective of the number of them. Recall that spike timings significantly matter in neuronal encoding, i.e., the SNN is prone to errors in spike and spike-arrival times. Thus, the delay in neuronal update—exceeding an allowable extent—due to the serial update process likely causes artifacts in neuronal encoding, which considerably undermines the reliability of inference. The allowable extent of delay is set aside until Section 2.3.2.

The delay in update is one of the major constraints in neuromorphic core designs. Therefore, the limit of number of fan-out neurons (equal to the number of fan-in synapses) in a core needs to be designed to reconcile with the downside of the reduction in parallelism.

Such neuromorphic core designs are instantiated by recent digital neuromorphic processors. Although there are different core designs for different processors, the core structure commonly includes an arithmetic logic circuit, memory, queue register, and router block. The arithmetic logic circuit updates neuronal variables on every timestep by retrieving the necessary data allocated in the memory block. The memory block stores the neuronal variables and parameters, synaptic weights, network topology (including fan-out neuronal addresses), axonal delays, and so forth. Given the time-multiplexed neuronal update, incoming events tagged with source or destination neurons’ indices need to be stored temporarily in a queue register. The events are sorted in order of event timing unless explicit axonal delays are set, and in order of axonal delay otherwise. The router block directs the events induced within a core to their destination neurons through local channels if they are in the same core as the source neurons. Otherwise, the events are sent out of the core with a tag of the destination core and axon indices or the source core and source neuron indices.

TrueNorth consists of 4,096 cores per chip—each with total 256 neurons, and each neuron is maximally allowed to take 256 fan-in connections (axons). All 256 neurons are available only if each of them is given a single fan-out connection because a single neuron with N fan-out connections in the SNN topology is achieved by grouping N neurons (each with a single fan-out) among the 256 neurons. In an extreme case, mapping a neuron with 256 fan-out connections onto the core takes the entire core as a single neuron. As such, all 256 neurons share a single arithmetic logic circuit. Notably, each of the 256 fan-in lines is connected to all 256 neurons via a 256 × 256 crossbar in which a switch (static random access memory [SRAM] bit) toggling between “0” and “1” (implying the existence and lack of a synapse, respectively) is placed at each cross-point. In this architecture, the entire fan-out neurons for each axon are addressed in parallel, which alleviates a workload of the communication channel between cores. Each core is given 256 × 410 bit memory (SRAM) of which 256 × 256 bits are dedicated to the crossbar. Thus, 154 bit memory is assigned to each neuron, storing neuronal variables and parameters. A queue register is referred to as a scheduler (16 × 256 SRAM) that contains binary flags for active axons over the total 256 axons in each row. Each row indicates the active axons on the same time step, and the scheduler is arranged in ascending order of spike-arrival timing. That is, axonal delay can be set up to 16 time steps, each of which corresponds to 1 ms.

Loihi consists of 128 cores; each embodies 1,024 spiking neuronal compartments. When a point neuron model is used, each of the 1,024 compartments serves as an independent neuron. Using a multi-compartment model with N compartments (including soma), 1,024/N independent neurons are available on a single core. Each core can accommodate up to 4,096 fan-ins (axons), each with multiple synapses with fan-out neurons. Multiple fan-outs (max. 4,096) from a core are supported by each core. Each core in Loihi also uses time-division multiplexing to update neuronal variables, leveraging the advantage of digital neuromorphic hardware. Different from the TrueNorth’s crossbar to direct an event to the corresponding fan-out neurons, a Loihi’s core uses a lookup table (LUT) of fan-out neuron and synapse indices—a compressed memory arrangement—in conjunction with a pointer array to accelerate searches for relevant synapses. A queue register for update schedules sorted in order of axonal delay is placed in the dendrite domain of the core. Total 2 Mb SRAM is available in each core. Half the memory is used to store the fan-in information for each synapse (64 bits), including the address of the corresponding destination neuron (10 bits if using a point neuron model), axonal delay (6 bits), synaptic weight (9 bits), synaptic tag (9 bits), etc. The memory is also allocated to the queue register and LUT of membrane potentials.

In Loihi, an event elicited from a neuron (source neuron) is sent out of the core with a tag of the destination core and axon indices. Each source neuron takes multiple axon indices as many as the number of its destination cores. Each destination core supports a maximum of 4,096 axon indices. The axon indices are predefined and stored in an LUT. Upon an event occurrence, the LUT is looked up to retrieve the source neuron’s axon indices that the event is labeled with. Once generated, every event (including local events whose destination core is the same as the source core) is delivered to the external router through the allocated slots. This differs from TrueNorth that is equipped with local channels dedicated to local events.

SpiNNaker is composed of multiple System-in-Package nodes, each incorporating a custom multiprocessor chip with 18 ARM968 cores and a standard 128 Mbyte synchronous dynamic random access memory (SDRAM) memory chip. The ARM cores in each node are commonly used to implement neuron cores. They can also be assigned custom roles, e.g., accelerating synaptic plasticity calculations, which can improve system performance at the cost of network size. Each core can be programmed to support different user-defined neuron and synapse models with various levels of complexity. The maximum number of elements a core can process in real time then differs for different models, because their complexity directly affects the number of processor instructions that should be sequentially performed upon each model-related calculation. Nevertheless, a typical figure used for estimating SpiNNaker network size is a few hundred point neurons (e.g., LIF neurons) with an average of 1,000 fan-in synapses per core. The node’s main SDRAM memory is shared between all cores and is used to store synapse information, such as destination neuron indices (topology LUT...
for each core), synaptic weights, axonal delays, and additional variables that are required for learning algorithms. It is noteworthy that the access to the shared memory is relatively slow, and thus, rapid timestamp-based synaptic variable modifications are not supported in SpiNNaker. Instead, synapse data are modified only upon input spikes or in long periodic iterations. Each SpiNNaker node is endowed with an additional router circuit that manages internal event traffic between the 18 ARM cores and external traffic to and from six neighboring nodes. In particular, the router houses 32 kb ternary content-addressable memory (CAM) and 24 kb RAM arrays that are used to map the events to their destination cores and/or output channels. Events that leave the node are further managed by the neighboring node routers, while the events that target internal cores are distributed and further routed by the topology LUTs stored in the SDRAM. The SpiNNaker architecture supports up to 65 000 nodes, potentially hosting over $10^6$ neuron cores in one system.

Korniucyk et al. have recently proposed a neuromorphic core design prototyped using a field programmable gate array (FPGA). The design highlights fast lookups of a topology LUT defining the topology of a given SNN to cope with the inevitable delay in neuronal variable update, arising from the shared pipelines in time-division multiplexing. The pointer LUT in the design indicates the locations of indices of local destination neurons in a main LUT of network topology. The pointer LUT significantly reduces the time-consumption on the retrieval of the destination neurons’ indices from the main topology LUT in comparison with entry-by-entry searches. A similar method applies to inverse lookups of synapse addresses upon fan-out neuronal spiking—the key to real-time weight adaptation. This will be elaborated in Section 3.3. As such, an arithmetic logic circuit is shared among all neurons in the core by using time-division multiplexing. The architecture consists of a neuron block, topology block, queue register, and synapse block. The neuron block is loaded with memory allocated to membrane potentials of all neurons and synaptic state variables (one variable for each neuron) for embedded learning, and the arithmetic logic circuit. The topology block hosts the topology LUT that occupies considerable space of the total memory. Communications between the neuron and topology blocks conforms to the handshake protocol underpinned by a queue register that temporarily stores events tagged with the source neuron addresses. Approximately half the total memory is allocated to synaptic weights as that for other core designs.

Akin to TrueNorth, this core design supports event routing over local channels when the source and destination cores are identical. Yet, a notable difference of this core design from TrueNorth as well as Loihi is that the events supposed to be delivered to different cores (referred to as global events) are sent out with a label of the source core’s and neuron’s indices in place of the destination cores’ and axons’ indices. Such global events are directed to their destination cores and their neurons by a global topology LUT that specifies the topology of the cores. Namely, the core in this core design sends out each event for lookups (outside the core) with the minimal label (i.e., source core’s and neuron’s indices), whereas the LUT for inter-core communication is embedded in the core for TrueNorth and Loihi. The LUT outside the core largely reduces memory usage in the core.

### 2.3. Event-Routing Schemes and Routing Delay

Digital event-routing schemes are commonplace in neuromorphic hardware irrespective of types of neuronal and synaptic circuits, which are based on address-event representation (AER). The AER passes an event from a source neuron to its destination neurons through a digital AER bus on the grounds of the binary indices of the source and destination neurons. The premise is access to arbitrary neurons in array with reference to their indices. The AER lays the foundation of reconfigurability of SNN topology as opposed to hardwired event routing that needs to rewire when the change in topology is required. The downside is event-routing congestion over the serial AER channel, which barely holds for the hardwired routing scheme capable of parallel event-deliveries over the parallel hardwired channels. An analogy with the AER is the system buses in von Neumann architecture. The buses allow the efficient communication between a CPU, working memory, and I/O, which endow the CPU with the full authority to access any arbitrary words—one word at a time—in the working memory through the address bus. Additionally, communication congestion over the serial communication channels is in common with the event-routing congestion in the AER scheme similar to that for the von Neumann bottleneck.

The aforementioned digital neuromorphic prototypes use AER channels within a core and/or between cores. For inference, event routing needs to figure out only the indices of destination neurons and synaptic weights to update neuronal variables consecutively along synaptic chains. We revisit the abovementioned prototypes with focus on the inter-core communication over shared pipelines in the following section.

#### 2.3.1. Event Routing in Digital Neuromorphic Hardware

As addressed in Section 2.2, an event elicited from a neuron (with a unique index) in a TrueNorth core is delivered to the destination cores through a chain of packet routers distributed as shown in Figure 4a. The cores are periodically distributed in two dimensions, each with a packet router. Note that local events (whose destination cores are identical to source cores) are delivered to their corresponding axons within the core through a local channel. TrueNorth supports event delivery across $256 \times 256$ cores over four chips, each with 64 $\times$ 64 cores. The global event is transferred to the core’s packet router with a tag of an axonal delay, axon index of the destination core, and destination core index. Figure 4a shows an event hopping through a router chain until it arrives at its destination core. Each packet router can cast one event at a time, i.e., bandwidth is strictly limited, so that traffic congestion likely comes into play, particularly, for simultaneous spiking at high activities. Furthermore, the serial event routing requires a queue register that temporarily stores the list of events subject to iterative delivery.

Loihi uses a similar event-routing method. Yet, unlike TrueNorth, four neighboring cores share a router that casts an event over 128 cores (Figure 4b), i.e., 32 routers per router network. Upon event generation, the event is delivered to its destination cores with a tag of the destination core and axon...
indices as explained in Section 2.2. Note that axonal delay is not included in the event data over the communication channel because the delay data are given to the destination core memory that will be looked up upon the arrival of the event. As such, the router network only supporting unicast delimits its bandwidth. To improve the bandwidth, Loihi uses two independent router networks that are alternatively used during iterative event delivery. All events generated from a core (irrespective of local and global events) are sent out of the core due to the lack of a local routing channel, imposing the entire routing tasks on the router networks.

The neuromorphic architecture by Kornijcuk et al. bases event routing on topology LUTs.\textsuperscript{[25,26]} The LUTs include a local LUT for local events within the same core and global LUT for global events across the cores. To this end, neurons in each core are classified as a local neuron and global neuron. A neuron whose entire destination neurons belong to the same core as the source neuron is referred to as a local neuron; the same logic applies to a global neuron. Provided that global neurons possibly form both local and global connections, each global neuron is given both local and global indices. The local LUT include a sub-LUT indicating global neurons. Upon event generation by a global neuron, the sub-LUT is searched for its neuron type (local or global), and the local topology LUT is simultaneously searched for its local destination neuron and synaptic weight indices to update their neuronal variables. The global event labeled with its global neuron index and the core index is sent to the global LUT that is subsequently searched for the destination cores and their destination neurons. Regarding the communication between the queue register and local topology block in a core, the communication between the core and global topology block conforms to the handshake protocol.

Hierarchical AER (HiAER)\textsuperscript{[45]} is an alternative spike routing architecture that achieves high level of interconnectivity using hierarchical router trees. On the ground level of the hierarchy, four neuron cores, each housing 16,000 neurons, are connected to a single digital router that uses two 2 Gb DDR3 dynamic random access memory (DRAM) chips for storing routing tables and synapse information (weight and axonal delay). Each router is composed of several queue registers, a module for implementing axonal delays, multiplexing circuitry, and a DDR3 controller. Upon receiving an event from either one of the four cores, the router checks the fan-out table entries that occupy a predefined DRAM address range assigned to the given neuron. Three possible event destinations can be distinguished: a neuron in the source core, a neuron in one of three neighboring cores, and a neuron in any other core in the system. In the first two cases, the local DRAM directly stores the postsynaptic neuron addresses and the related synaptic data; thus, the event is immediately routed to its destinations. In the last case, the postsynaptic neuron addresses are not explicitly stored in the local DRAM; instead, only the pointers to the routing table in the higher hierarchical level are available. The event is then passed one level “up” the hierarchy. The second hierarchical level is composed of four ground level nodes (four neuron cores and a router each) and uses the same router structure with two 2 Gb DDR3 DRAM chips. The routing table pointers passed from the ground level are used to access specific second level’s DRAM address ranges where the following routing instructions (up or down the hierarchy) alongside corresponding memory pointers are stored. The event is then accordingly duplicated and distributed to its target ground level nodes and/or passed to the higher hierarchical level. This routing architecture was used in a neuromorphic system containing $10^6$ neurons and $10^9$ synapses.\textsuperscript{[45]}

2.3.2. Criterion of Allowable Delay in Event Routing

Recall that the SNN is a rich dynamic hypothesis that takes spikes (events) as a trigger for dynamic neuronal encoding through synaptic chains as well as learning. Spike timing information
should therefore be intact to avoid unexpected results of inference and learning. Regarding the event routing over the AER channel, the corruption of spike timing information likely arises from event-routing congestion that causes undesirable delays in event routing. Conceding the inevitable delay due to the serial communication brings up the issue of the extent to which the delay is acceptable. This issue was addressed in a recent publication by Korniouch et al.\textsuperscript{[26]} First, the delay should be below 1 ms that is the approximate width of a physiological spike.\textsuperscript{[3]} Given that a spike is considered as a point event, its width (1 ms) is reasonably taken as a unit time step, i.e., the maximum temporal resolution. Thus, a delay in single-event routing within 1 ms unlikely causes artifacts in neuromorphic hardware operation. The second condition disallows event-routing delays exceeding the time interval between consecutive events—which is referred to as interspike interval (ISI). Otherwise, a new event occurring amid the previous event awaiting its turn to be routed is supposed to be discarded without a queue register in the router. With a queue register, the register will be loaded with source neuron indices to be routed one at a time, indicating event-routing congestion causing non-negligible delays.

Regarding the second condition, the delay in event routing can easily be evaluated if the digital routing scheme in use is understood. The difficulty, however, lies in estimating ISIs in a given SNN because it differs for different operation conditions given to the SNN, for instance, the activity of each neuron, irregularities in spiking, temporal correlation between spikes from different neurons, and so forth. Despite the complexity in ISI estimation, it is sensible to regard the events to follow Poisson processes unless spikes over the network are synchronous. This assumption leads to a simple relationship between a given activity and the corresponding ISI in that the average ISI equals the reciprocal of activity of a Poisson neuron.\textsuperscript{[1]} One may object to the assumption because Poisson neurons are indeed hypothetical—instead, the ISI distribution rather follows a gamma distribution function, mainly due to the presence of refractory periods.\textsuperscript{[3]} The event-routing schemes frequently make use of pipelines shared among many presynaptic neurons to a postsynaptic neuron(s). A group of spike sequences (each from a different presynaptic neuron) in the pipeline is therefore delivered to the destination neuron. A recent study has identified that a group of spike sequences—the ISIs of each sequence obey a gamma distribution function—equivalently behaves as Poisson spikes.\textsuperscript{[1]} This provides the grounds of theoretical estimation of ISIs.

Nevertheless, synchronous spiking over the SNN undermines the ISI estimation. The ISI on average over a spiking period hardly captures the spiking dynamics because of the strong temporal correlation between spikes from different neurons.

### 3. Neuromorphic Hardware with Embedded Learning

Inference-only neuromorphic hardware significantly accelerates inference tasks to the extent of real-time inference. The capability of real-time inference allows the neuromorphic hardware to receive input data from its natural environments and response to them. The neuromorphic prototypes introduced in Section 2 are certainly endowed with this capability. Yet, inference-only neuromorphic hardware rather passively interacts with the environments in that it can receive the data from the environments and infer, but cannot adapt itself to the data in real time. Training inference-only neuromorphic hardware thus requires a software replica of the SNN mapped onto the hardware. The software replica is trained with human-chosen datasets saved in memory devices. This off-chip learning method potentially causes significant issues. One of the most straightforward issues lies in difficulty in real-time simulation. The more complexity the SNN includes, the more likely the real-time simulation is infeasible such that the training runtime overwhelms the simulated time. This causes training to be time- and energy-inefficient. Another issue is the memory capacity for datasets. As such, neuromorphic hardware comes into its own when learning data in dynamic domains, e.g., videos, songs, etc. Certainly, data in dynamic domains are much larger than static-domain data; simple calculation of required memory of a video with a frame rate of 20 frames per second and each frame of 8-bit gray scale 100 × 100 pixels yields 1.6 Mb for a movie for a second. Large memory capacity is indispensable for training with a long video, for instance, training the SNN to evolve over a long-time period (days or months) in response to visual stimulus.

On-chip learning (referred to as embedded learning) is a proper solution to the abovementioned downside of off-chip learning. TrueNorth is inference-only hardware, whereas the prototypes introduced in Section 2.2 are equipped with embedded learning. To efficiently embed the capability to learn, additional indispensable attributes should be considered. They include learning algorithms with severe constraints, circuit- and energy-efficient learning engine (learner) design, and algorithm for fast searches for fan-in neurons and fan-in synapse weights. Note that only fast searches for fan-out neurons matter in inference as discussed earlier. Yet, embedded learning needs searches for both fan-out neurons (forward lookups) and fan-in neurons and synapse weights (inverse lookups) as elaborated in Section 3.2.

#### 3.1. Algorithms and Data for Learning

Data for SNNs are classified as local and global data. In turn, each of local and global data types is classified as topological (or spatial) and temporal data, providing a total of four distinguishable data types.\textsuperscript{[35]} The SNN needs only topological local data to infer inputs through consecutive neuronal encodings, in favor of ideally distributed processor and memory architecture. The data for inference are not only topologically local but also temporally local given that the consecutive neuronal encodings need mere current, rather than previous, topological local data. Needs for only temporal local data enable the minimal usage of memory for each core.

Learning, however, appears data hungry in need of other types of data alongside topological and temporal local data. For instance, celebrated backpropagation algorithms\textsuperscript{[36]} use an enormous amount of all four types of data to optimally modify the parameters in a DNN. Online learning allows a weight change to be determined for every input data with reference to the objective function. The objective function defines the distance between target and actual output vectors in the output.
activity space—such that the update moves the actual output vector closer to the target vector. The update evaluation is associated with a set of topological global data. Offline learning (also known as batch learning) uses the same type of topological global data as online learning plus temporal global data. The temporal global data indicate the weight changes from the previous data in the same batch, which are averaged to eventually update the weight at the end of the batch. The original backpropagation algorithm is hardly suitable for the SNN mainly due to the input-derivative of an activation function in backpropagating error. There have been vigorous attempts to modify the backpropagation algorithms to involve it in training the SNN.

The STDP rule offers a clue and its modifications still use global data as opposed to algorithm of locality (or local algorithm) that bases training on only local data.

In addition to derivative algorithms from deep learning, diverse local algorithms that originate from Hebb’s rule are available. Hebb’s rule explains the increase in synaptic weight (potentiation) for a synapse subject to high presynaptic and postsynaptic activities that are temporally correlated with each other. Training with Hebb’s rule needs activities of presynaptic and postsynaptic neurons for each synapse, and thus, the use of global topological data can be avoided. Yet, Hebb’s rule does not limit the growth of weight, causing training instability and lack of input selectivity. The modifications of Hebb’s rule mainly underpin these two issues by introducing the anti-Hebbian behavior, i.e., the decrease in synaptic weight (depression).

They include covariance rule, Bienenstock–Cooper–Munro (BCM) rule, Oja rule, and so forth. These local algorithms are functions of activities with firing rates equivalent to spike-firing probabilities when multiplied by time-bin size. That is, to update weights, the algorithm should be provided with presynaptic and postsynaptic spike-firing probabilities that are only evaluated from sufficient trials. Deploying a probability-evaluation circuit is hardly a circuit- and energy-efficient way to embedded learning. Spike-count rate can be a replacement for firing rate only if the spike-count rate outweighs the change rate of input to the neuron. Yet, the energy consumption by high spike-count rate and the consequent traffic congestion (as discussed in Section 2.3.2) are the cost.

Individual spikes, alternative to activities, are desirable trigger for weight update; algorithms to this end are referred to as an event-driven algorithm. Event-driven algorithms do not require statistical variables such as firing rate or spike-count rate, so that real-time ad hoc weight updates can be implemented. The spike timing-dependent plasticity (STDP) rule and its modifications exemplify event-driven algorithms. The STDP rule elucidates homosynaptic weight modification. That is, Hebbian (potentiation) behavior results from a postsynaptic event following a presynaptic event within a particular timing-window, whereas the opposite timing-order also within a timing-window results in anti-Hebbian behavior (depression). The implementation of a primitive STDP algorithm needs presynaptic and postsynaptic variables evolving with presynaptic and postsynaptic spikes that are convolved with pre and post trace temporal kernels, respectively. However, this primitive STDP algorithm is inconsistent with the abovementioned activity-based algorithms, particularly the BCM rule, where the algorithm is mapped onto an activity domain. Several event-driven algorithms that consistently explain physiological plasticity data in both spike-timing and activity domains have been proposed with a considerable degree of complexity (more variables). Note that the memory for state variables in an embedded learning algorithm scales with the number of state variables in a super-linear fashion, which is attributed to duplication of state variables over cores. This will be detailed in the following section.

The random backpropagation (RBP) algorithm offers a clue to neuromorphic hardware-suitable adaptation of the backpropagation algorithm. The RBP algorithm uses feedback channels separate from feedforward channels, which are given fixed weights through the entire training period. Instead, the weights of feedback channels are subject to update. A modification of the RBP algorithm to update weights by individual spikes—referred to as event-driven RBP (eRBP)—takes a step forward using energy- and data-efficient algorithm suitable for neuromorphic hardware.

3.2. Requirements for Real-Time Adaptable Neuromorphic Hardware

As such, the key to real-time inference in digital neuromorphic hardware is the delivery of events from source to destination neurons with acceptable delays. To this end, the forward lookup, i.e., search for the destination cores and neurons, should be sufficiently fast to avoid the delay. Real-time inference neuromorphic hardware is a subset of real-time adaptable neuromorphic hardware in that real-time adaptation should take into consideration several critical requirements alongside the fast forward lookup. Inference-only hardware merely delivers an event from a source to destinations by retrieving data of destination cores and neurons, axonal delay, and synaptic weight. However, in real-time adaptation, a subsequent modification of synaptic weight driven by the event should immediately follow the event delivery. A non-negligible delay in weight update likely undermines real-time adaptability. To this end, the event immediately wakes up the learner to evaluate a weight change using a state variable(s) and the current weight. Fortunately, implementing the weight modification upon a postsynaptic event minimally imposes an additional workload on the hardware given that addressing the corresponding fan-out synapses—one of the most time-consuming processes—is executed beforehand during the event delivery period.

A source neuron is a presynaptic and a postsynaptic neuron depending on the network topology. In inference, source neurons are merely viewed as presynaptic neurons because source neurons as postsynaptic neurons do not act on the presynaptic neurons given the unidirectional synaptic transmission (only from a presynaptic to a postsynaptic neuron). Postsynaptic events can, however, modify the weights of fan-in synapses as that for the STDP rule where presynaptic and postsynaptic events can respectively induce depression and potentiation. In this regard, the challenge lies in addressing the fan-in synapses and presynaptic state variables upon a postsynaptic event, which is unnecessary in inference. This inverse lookup is a time-consuming process, serving as a major obstacle to real-time adaptation. In common event-routing strategies, fast inverse lookups are
incompatible with fast forward lookups, and most commonly, the hardware is optimized for fast forward lookups that are more frequently applied to even learning than inverse lookups. Therefore, the key to real-time adaptable neuromorphic hardware is to accelerate the inverse lookup by keeping forward lookups fast.

3.3. Realization Learners in Neuromorphic Processor Cores

As discussed in Section 2.2, the introduction of neuromorphic cores to a neuromorphic processor reconciles parallel neural processing by individual cores with serial processing within each core. A learner consists of an arithmetic logic circuit and data buses to evaluate an event-driven change in weight according to a chosen learning algorithm. One learner per core suffices for the evaluation by using time-division multiplexing in that each event (incoming to and elicited from the core) shares the learner one at a time. Given that fan-in synapses and their fan-out neurons are present in the same core, the learner is placed in the same core as fan-in synapses.

Different learning algorithms need different data; algorithms of high complexity tend to require more data. For instance, the simple STDP rule takes presynaptic and postsynaptic state variables. Modified STDP rules, e.g., triplet-based STDP rule,[60] to enhance fidelity to biological observations take additional state variables. Learning algorithms with point neuron models commonly use state variables, each of which is given to a neuron. The single presynaptic state variable (of a presynaptic neuron) in the simple STDP rule is looked up when updating all fan-out synapses of the presynaptic neuron. The opposite holds true; the single postsynaptic state variable is looked up when updating all fan-in synapses of the postsynaptic neuron. As such, any neuron is a presynaptic and a postsynaptic neuron, and thus, each neuron is given its own presynaptic and postsynaptic variables. Theses state variables are desired to be placed in the same core as the host neuron for efficient variable update upon events.

The critical constraint is that such state variables need to be placed in the same core as the learner to avoid unexpected delays in learning. The constraint is satisfied when presynaptic and postsynaptic neurons are present in the same core. However, when they are placed apart, the learner (with postsynaptic neurons) should address the presynaptic state variable(s) in a different core. This inevitably causes delays in learning, additional energy consumption over inter-core communication channels, and complexity in communication channel design. A sensible workaround solution is to duplicate the presynaptic state variables in the cores whose learners need the variables. The learner can thus address the duplicated presynaptic state variables in the same core. Yet, such variable duplication imposes strict constraints on the complexity in learning algorithm with regard to the super-linearly incremental memory requirement with algorithm complexity and the limited memory capacity of each core.

Each of 128 cores in Loihi is with an independent programmable learner. The difficulty in inverse lookups holds for Loihi given that the core architecture is optimized for forward lookups (destination neurons and fan-out synapses of a source neuron). Albeit manageable to find fan-in synapses of a source neuron, it takes considerable time to finalize the updates of fan-in synaptic weights. Therefore, learning in Loihi is inevitably delayed.[24]

The neuromorphic processor architecture proposed by Kornijcuk et al. offers a feasible solution to the delay in inverse lookups.[25,26] The key to fast inverse lookup is the use of separate memories: a pointer memory (PTR_RAM), fan-out memory (FOUT_RAM), and fan-in memory (FIN_RAM) as schematized in Figure 5. Each entry of FOUT_RAM contains the postsynaptic neuron index of each synapse. Its length equals the number of synapses, and the entry indices correspond to synapse indices. FOUT_RAM is sorted according to presynaptic neuron index, so that the postsynaptic neuron indices of the same presynaptic neuron are adjacent in the memory. Given this memory sorting order, the postsynaptic neurons of an arbitrary presynaptic neuron can easily be addressed by pointing the start and end entries of FOUT_RAM in place of iterative memory searching. The start and end entry indices for each neuron are stored in PTR_RAM. PTR_RAM is sorted according to the neuron index. Note that each entry index of FOUT_RAM is a synapse index. Therefore, this scheme offers a fast means of forward lookups for the presynaptic neuron indices and synapse indices of an arbitrary neuron.

FIN_RAM stores the entire synapse indices, sorted according to postsynaptic neuron index. Thus, the synapse indices for the same postsynaptic neuron are adjacent in the memory. Likewise, the start and end entry indices for an arbitrary postsynaptic neuron are sufficient to find the fan-in synapse indices in FIN_RAM. The start and end entry indices are also stored in PTR_RAM. In this light, inverse lookups can be sufficiently fast to avoid delays in learning, realizing true real-time adaptation.

4. Outlook and Concluding Remarks

SNN is a dynamic hypothesis capable of adaptation with both static- and dynamic-domain data using inherent rich temporal and spatial kernels. The theoretical capacity and capability of SNN have barely been unveiled due in part to its innate complexity. SNN theories mostly originate from the brain—wetware closer to hardware than software. Mapping the wetware to software to simulate the SNN causes time- and energy-inefficiency as identified by the large difference between the runtime and simulated time. In contrast, mapping the wetware to hardware lays the time- and energy-efficient foundations of studies on SNN, which is the very neuromorphic hardware. The neuromorphic hardware addressed in this progress report certainly offers the solid foundations of fundamental as well as technical studies of SNN.

The digital neuromorphic hardware outlined in this report highlights reliable and time- and energy-efficient realization of large-scale SNN. To leverage digital neuromorphic hardware, the SNN needs to be scaled up to the level beyond the capacity of computer simulation. A daunting challenge to this end lies in the limited memory capacity in neuromorphic hardware. Indeed, the memory capacity limits the SNN size (the numbers of neurons and synapses), complexity in learning algorithm, learning speed, and so forth. A larger memory capacity allows (1) more neurons and synapses to be hosted in a core by endowing them with more memory space for membrane potentials and synaptic weights and by enlarging the topology LUT, (2) embedded learning to incorporate more state variables in a core to enhance
its learning ability, and (3) the delay in event delivery to be shortened by incorporating more LUTs as shown in Figure 5. In sum, the limited memory capacity is a notorious bottleneck for scaling up the SNN.

SRAM is commonplace in digital neuromorphic hardware. Low operation power, high operation speed, and high reliability are the attractive advantages of SRAM. Yet, its low memory density, due to the use of six transistors in a unit cell, is a disadvantage. DRAM is a conceivable replacement for SRAM in part in neuromorphic processors as that for SpiNNaker. A much higher memory density than SRAM is the attractive feat of DRAM; yet, the cost includes complexity in core design, a higher operation power, and needs for a secondary memory for DRAM’s hiccup periods.

The state-of-the-art commercial DRAM chips also offer high data transfer rates, which, in the best case scenarios, are even comparable to SRAM. For instance, the fourth generation double data rate (DDR4) SDRAM chip offers memory density up to 32 Gb with data transfer rates peaking above 1 GB s⁻¹. Such high data-transfer rates can be achieved when accessing large

Figure 5. a) Schematic of a pointer-based event-routing scheme. Forward and inverse lookup processes upon an event elicited from Neuron 2 in a toy network in (b) are shown. Reproduced with permission. Copyright 2019, Wiley-VCH.
structured data arrays, such as routing LUTs, where data are accessed with relatively long time intervals (e.g., upon each spike), and large amounts of data are required upon each access (e.g., all fan-out and fan-in neuron indices for a given neuron). In contrast, when rapid access to small data packets is required, the DDR4 SDRAM data transfer rate drops tremendously due to the long individual data access time. Such rapid memory access might be required, for instance, for reading/writing synaptic weights that may be scattered across the memory entries. In this case, the amount of data transferred during each access is small (one or several weights); however, the number of accesses is very large (number of synapses per neuron). In this scenario, DDR4 SDRAM is no longer a favorable solution. Fortunately, an attractive alternative is commercially available—reduced latency DRAM (RLDRAM). RLLDRAM uses a modified version of the internal DDR4 memory controller to improve random access speed. This improvement, however, comes at the cost of considerable reduction in memory density. For instance, the state-of-the-art third generation RLDRAM (RLDRAM3) chip only offers up to 1.1 Gb per chip in contrast to the 32 Gb per chip of the state-of-the-art third generation RLDRAM (RLDRAM3) chip.

As proposed by Kornijczuk et al., the use of a CAM as a topology LUT ideally copes with the delay in event delivery, allowing a larger SNN to be mapped to the hardware.[66] However, conventional SRAM-based CAM with a lower information density than SRAM critically undermines its status as a feasible replacement for SRAM. In this light, we need to pay attention to nonvolatile emerging memories of high information density and their application to CAM. Resistive random access memory (RRAM) is a frontrunner.[68] RRAM bases binary representation on the resistance state of a unit cell, i.e., high and low resistance states. Its high information density (one-transistor one-resistor unit cell), reliable read-out margin, and high scalability are major advantages. In light of the advantages, effort to build CAM using RRAM continues onward in various ways such as voltage read-out scheme[69–72] and current read-out scheme.[73] A renowned demerit of RRAM lies in its low endurance.[72] Yet, CAM as a LUT in neuromorphic hardware is hardly expected to be programmed and erased as frequently as working memory. Therefore, neuromorphic hardware appears to be one of the most suitable applications of RRAM-based CAM.

Acknowledgements
This work was supported by the research grant (NRF-2019R1C1C1009810) of the National Research Foundation of Korea.

Conflict of Interest
The authors declare no conflict of interest.

Keywords
digital neuromorphic hardware, embedded learning, real-time adaptation, spiking neural networks

Received: May 24, 2019
Revised: June 30, 2019
Published online: August 20, 2019

[1] D. S. Jeong, J. Appl. Phys. 2018, 124, 152002.
[2] W. Gerstner, W. M. Kistler, Spiking Neuron Models: Single Neurons, Populations, Plasticity, Cambridge University Press, New York 2002.
[3] P. Dayan, L. F. Abbott, Theoretical Neuroscience, The MIT Press, London 2001.
[4] E. M. Izhikevich, Neural Comput. 2006, 18, 245.
[5] B. Szatmáry, E. M. Izhikevich, PLOS Comput. Biol. 2010, 6, e1000879.
[6] J. Brea, W. Senn, J.-P. Pfister, J. Neurosci. 2013, 33, 9565.
[7] B. Gardner, A. Grüning, PLOS ONE 2016, 11, e0161335.
[8] F. Ponulak, A. Kasinski, Neural Comput. 2010, 22, 467.
[9] Y. LeCun, Y. Bengio, G. Hinton, Nature 2015, 521, 436.
[10] S. Hochreiter, J. Schmidhuber, Neural Comput. 1997, 9, 1735.
[11] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, in the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, 2014, p. 1724.
[12] K.-I. Funahashi, Y. Nakamura, Neural Networks 1993, 6, 801.
[13] B. A. Pearlmutter, IEEE Trans. Neural Netw. 1995, 6, 1212.
[14] A. C. Tsoi, A. Back, Neurocomputing 1997, 15, 183.
[15] R. J. Williams, D. Zipser, Neural Comput. 1989, 1, 270.
[16] D. R. Hush, B. G. Horne, IEEE Signal Process. Mag. 1993, 10, 8.
[17] C. Mead, Proc. IEEE 1990, 78, 1629.
[18] F. S. E. Chicca, C. Bartolozzi, G. Indiveri, Proc. IEEE 2014, 102, 1367.
[19] B. V. Benjamín, P. Gao, E. McQuinn, S. Choudary, M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, H. Schwenn, Y. Bengio, in the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, 2014, p. 1724.
[20] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumi, G. Indiveri, Front. Neurosci. 2015, 9, 141.
[21] N. Q. S. Moradi, F. Stefanini, G. Indiveri, IEEE Trans. Biomed. Circuits Syst. 2018, 12, 106.
[22] A. Necker, S. Fok, B. V. Benjamín, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Elaismith, R. Manohar, K. Boahen, Proc. IEEE 2019, 107, 144.
[23] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Io, S. K. Esser, R. Appuswamy, B. Tabia, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, D. S. Modha, Science 2014, 345, 668.
[24] M. Davies, N. Striniwasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathai, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, H. Wang, IEEE Micro 2018, 38, 82.
[25] V. Kornijczuk, D. S. Jeong, in 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, Piscataway, NJ 2018.
[26] V. Kornijczuk, J. Park, G. Kim, D. Kim, I. Kim, J. Kim, J. Y. Kwak, D. S. Jeong, Adv. Mater. Technol. 2019, 4, 1800345.
[27] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D.R. Lester, A. D. Brown, IEEE J. Solid-State Circuits 2013, 48, 1943.
[28] D. S. Jeong, I. Kim, M. Ziegler, G. Kohlstedt, RSC Adv. 2013, 3, 3169.
[29] S. Saighi, C. G. Mayr, T. Serrano-Gotarredona, H. Schmidt, G. Lecerf, J. Tomas, J. Grollier, S. Boyo, A. Vincent, D. Querlioz, S. La Barbera, F. Alibart, D. Vuillaume, O. Bichler, C. Gamrat, B. Linares-Barranco, Front. Neurosci. 2015, 9, 51.
[30] D. S. Jeong, C. S. Hwang, Adv. Mater. 2018, 30, 1704729.
[31] Z. Wang, S. Joshi, S. E. Savel’ev, H. Jiang, R. Miday, P. Lin, M. Hu, N. Ge, J. P. Strachan, Z. Li, Q. Wu, M. Barnell, G.-L. Li, H. L. Xin, R. S. Williams, Q. Xia, J. J. Yang, Nat. Mater. 2017, 16, 101.
[32] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, W. D. Lu, Nat. Nanotechnol. 2017, 12, 784.
[33] S. J. van Albada, A. G. Rowley, J. Senk, M. Hopkins, M. Schmidt, A. B. Stokes, D. R. Lester, M. Diesmann, S. B. Furber, Front. Neurosci. 2018, 12, 291.
[34] J. Schmehmel, D. Briiderle, A. Gribil, M. Hock, K. Meier, S. Millner, in Proc. IEEE Int. Symp. on Circuits and Systems, IEEE, Piscataway 2010, p.1947.
[35] E. O. Neftci, iScience 2018, 5, 52.
[36] S. B. Furber, F. Galluppi, S. Temple, L. A. Plana, Proc. IEEE 2014, 102, 652.
[37] M. Pfeiffer, T. Pfeil, Front. Neurosci. 2018, 12, 774.
[38] C. Eliasmith, C. H. Anderson, Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems, MIT Press, Cambridge, MA 2003.
[39] P. G. Drennan, C. C. McAndrew, IEEE J. Solid-State Circuits 2003, 38, 450.
[40] N. Tega, H. Miki, Z. Ren, C. P. D. Emic, Y. Zhu, D. J. Frank, J. Cai, M. A. Guillorn, D. Park, W. Haensch, K. Torii, in 2009 IEEE Int. Electron Devices Meeting (IEDM), IEEE, Piscataway, NJ 2009, p. 1.
[41] H. Miki, N. Tega, M. Yamaoka, D. J. Frank, A. Bansal, M. Kobayashi, K. Cheng, C. P. D. Emic, Z. Ren, S. Wu, J. Yau, Y. Zhu, M. A. Guillorn, D. Park, W. Haensch, E. Leobandung, K. Torii, in Proc. IEEE Int. Electron Devices Meeting, IEEE, Piscataway, NJ 2012, p. 19.1.1.
[42] F. Galluppi, X. Lagorce, E. Stromatias, M. Pfeiffer, L. A. Plana, S. B. Furber, R. B. Benosman, Front. Neurosci. 2015, 8, 429.
[43] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, D. Gillespie, IEEE Trans. Neural Netw. 1993, 4, 523.
[44] K. A. Boahen, IEEE Trans. Circuits Syst. II. Analog Digit. Signal Process. 2000, 47, 416.
[45] J. Park, T. Yu, S. Joshi, C. Maier, G. Cauwenberghs, IEEE Trans. Neural Netw. Learn. Syst. 2017, 28, 2408.
[46] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Nature 1986, 323, 533.
[47] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Norfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, D. S. Modha, Proc. Natl. Acad. Sci. U. S. A. 2016, 113, 11441.
[48] E. O. Neftci, C. Augustine, S. Paul, G. Detorakis, Front. Neurosci. 2017, 11, 324.
[49] H. Mostafa, IEEE Trans. Neural Netw. Learn. Syst. 2018, 29, 3227.
[50] D. O. Hebb, The Organization of Behavior, Wiley & Sons, New York 1949.
[51] E. Bienenstock, L. Cooper, P. Munro, J. Neurosci. 1982, 2, 32.
[52] K. D. Miller, D. J. C. MacKay, Neural Comput. 1994, 6, 100.
[53] T. J. Sejnowski, G. Tesauro, in Neural Models of Plasticity (Eds: J. H. Byrne, W. O. Berry), Academic Press, London 1989, p. 94.
[54] E. Oja, J. Math. Biol. 1982, 15, 267.
[55] L. N. Cooper, M. F. Bear, Nat. Rev. Neurosci. 2012, 13, 798.
[56] H. Markram, J. Lübke, M. Frotscher, B. Sakmann, Science 1997, 275, 213.
[57] G.-Q. Bi, M.-M. Poo, J. Neurosci. 1998, 18, 10464.
[58] S. Song, K. D. Miller, L. F. Abbott, Nat. Neurosci. 2000, 3, 919.
[59] R. C. Froemke, Y. Dan, Nature 2002, 416, 433.
[60] J.-P. Pfister, W. Gerstner, J. Neurosci. 2006, 26, 9673.
[61] E. M. Izhikevich, N. S. Desai, Neural Comput. 2003, 15, 1511.
[62] T. Toyoizumi, J.-P. Pfister, K. Aihara, W. Gerstner, Proc. Natl. Acad. Sci. U. S. A. 2005, 102, 5239.
[63] J. Gjorgjieva, C. Clopath, J. Audet, J.-P. Pfister, Proc. Natl. Acad. Sci. U. S. A. 2011, 108, 19383.
[64] C. Clopath, L. Büsing, E. Vasilaki, W. Gerstner, Nat. Neurosci. 2010, 13, 344.
[65] P. Baldi, P. Sadowski, Z. Lu, Artif. Intell. 2018, 260, 1.
[66] R. Waser, M. Aono, Nat. Mater. 2007, 6, 833.
[67] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, C. S. Hwang, Adv. Electron. Mater. 2016, 2, 1600090.
[68] D. S. Jeong, R. Thomas, R. S. Katyar, J. F. Scott, H. Kohlstedt, A. Petraru, C. S. Hwang, Rep. Prog. Phys. 2012, 75, 076502.
[69] K. Eshraghian, K. Cho, O. Kavehei, S. Kang, D. Abbott, S. S. Kang, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2011, 19, 1407.
[70] L. Zheng, S. Shin, S. S. Kang, in Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS), IEEE, Piscataway, NJ 2014, p. 2253.
[71] H. Li-Yue, C. Meng-Fan, C. Tien-Fu, S. Shin, S. S. Kang, in Proc. IEEE Symp. on VLSI Circuits Digest of Technical Papers, IEEE, Piscataway, NJ 2014, p. 1.
[72] F. Galluppi, X. Lagorce, E. Stromatias, M. Pfeiffer, L. A. Plana, S. B. Furber, R. B. Benosman, Front. Neurosci. 2015, 8, 429.
[73] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, D. Gillespie, IEEE Trans. Neural Netw. 1993, 4, 523.
[74] K. A. Boahen, IEEE Trans. Circuits Syst. II. Analog Digit. Signal Process. 2000, 47, 416.
[75] J. Park, T. Yu, S. Joshi, C. Maier, G. Cauwenberghs, IEEE Trans. Neural Netw. Learn. Syst. 2017, 28, 2408.
[76] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Nature 1986, 323, 533.
[77] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Norfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, D. S. Modha, Proc. Natl. Acad. Sci. U. S. A. 2016, 113, 11441.
[78] E. O. Neftci, C. Augustine, S. Paul, G. Detorakis, Front. Neurosci. 2017, 11, 324.
[79] H. Mostafa, IEEE Trans. Neural Netw. Learn. Syst. 2018, 29, 3227.