Opportunities and Challenges for Neuromorphic Computing Algorithms and Applications

Catherine Schuman, Shruti R. Kulkarni, Maryam Parsa, J. Parker Mitchell,
Prasanna Date, and Bill Kay
Oak Ridge National Laboratory

August 1, 2022

Abstract

Neuromorphic computing technologies will be important for the future of computing, but much of the work in neuromorphic computing has focused on hardware development. Here, we review recent results in neuromorphic computing algorithms and applications. We highlight characteristics of neuromorphic computing technologies that make them attractive for the future of computing and we discuss opportunities for future development of algorithms and applications on these systems.

1 Promises of Neuromorphic Computing

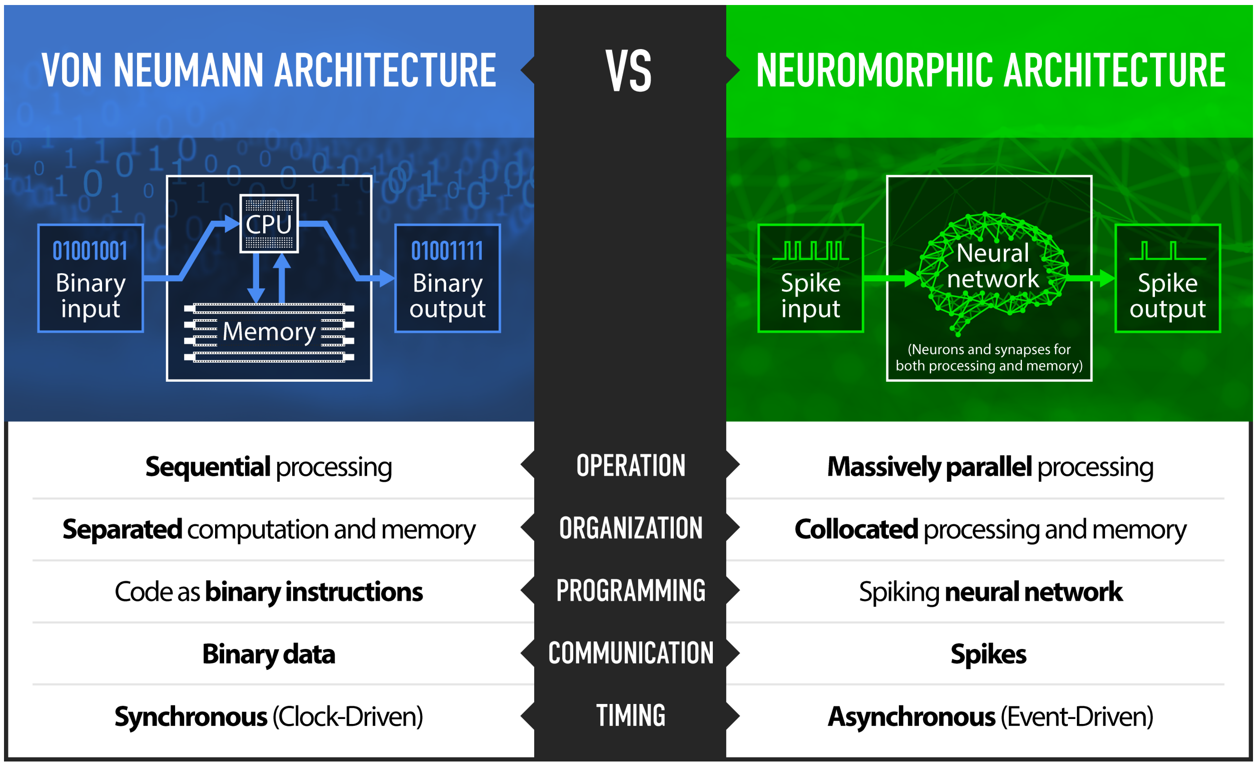
With the end of Moore's law approaching and Dennard scaling ending, the computing community is increasingly looking at new technologies to enable continued performance improvements. Neuromorphic computers are one such new computing technology. "Neuromorphic" was coined by Carver Mead in the late 1980's [63, 64], and at that time, primarily referred to mixed analog-digital implementations of brain-inspired computing. However, as the field has continued to evolve and with the advent of large-scale funding opportunities for brain-inspired computing systems such as the DARPA Synapse project and the European Union's Human Brain Project, the term neuromorphic has come to encompass a wider variety of hardware implementations.

We define neuromorphic computers as non-von Neumann computers whose structure and function are inspired by brains and that are composed of neurons and synapses. Von Neumann computers are composed of central processing units (CPUs) and memory units, where data and instructions are stored. In a neuromorphic computer, on the other hand, both processing and memory are governed by the neurons and the synapses. Rather than explicit instructions as in a von Neumann computer, programs in neuromorphic computers are defined by the structure of the neural network and its parameters. While Von Neumann computers encode information as numerical values represented by binary values, neuromorphic systems receive spikes as input, where the associated time at which they occur, their magnitude, and their shape can be used to encode numerical information. Binary values can be turned into spikes and vice versa, but the precise way to perform this conversion is still an area of study in neuromorphic computing [97].

Given the aforementioned contrasting characteristics between the two architectures, neuromorphic computers present some fundamental operational differences:

- Massively parallel operation: Neuromorphic computers are inherently massively parallel, where all of the neurons and synapses can potentially be operating simultaneously. However, the computations performed by neurons and synapses are relatively simple when compared with the parallelized von Neumann systems.

- Collocated processing and memory: There is no notion of a separation of processing and memory in neuromorphic hardware. Although neurons are sometimes thought of as "processing" units and



VON NEUMANN ARCHITECTURE	VS	NEUROMORPHIC ARCHITECTURE
Sequential processing	OPERATION	**Massively parallel** processing
Separated computation and memory	ORGANIZATION	**Collocated** processing and memory
Code as **binary instructions**	PROGRAMMING	Spiking **neural network**
Binary data	COMMUNICATION	**Spikes**
Synchronous (Clock-Driven)	TIMING	**Asynchronous** (Event-Driven)

Figure 1: Comparison of the von Neumann architecture with the neuromorphic architecture.

synapses are sometimes thought of as "memory," the neurons and synapses both perform processing and store values in many implementations. The collocation of processing and memory helps mitigate the von Neumann bottleneck i.e., the processor memory separation that causes a slowdown in the maximum throughput that can be achieved. Additionally, the data accesses from main memory consumes a significant amount of energy compared to the compute energy [110]) present in conventional computing systems.

- Inherent scalability: Neuromorphic systems are meant to be inherently scalable because adding additional neuromorphic chips is simply increasing the number of neurons and synapses that can be realized. It is possible to take multiple physical neuromorphic chips and treat them as a single large neuromorphic implementation to run larger and larger networks. This has been successfully accomplished across a variety of large-scale neuromorphic hardware systems, including SpiNNaker [62, 31] and Loihi [22].

- Event-driven computation: Neuromorphic systems leverage event-driven computation (i.e., computing only when data is available) and temporally sparse activity to allow for extremely efficient computation [70, 7]. Neurons and synapses only perform work when there are spikes to process, and typically, spikes are relatively sparse within the operation of the network.

- Stochasticity: Neuromorphic systems can include a notion of randomness, such as in the firing of neurons, to allow for noise and enable different algorithms.

The features of a neuromorphic computer are well noted in the literature and are given as motivations for implementing and using neuromorphic computers [98, 44, 109, 112, 23, 40]. One of the most attractive features of neuromorphic systems for computation is their extremely low power operation; they can often operate on orders of magnitude less power than traditional computing systems. This low power operation is due to their event driven nature and massively parallel nature, where typically only a small portion of the entire system is active at any given time and the rest is idle. Because of the increasing energy cost of computing, as well as applications in which there are energy constraints (such as edge computing applications), energy efficiency alone is a compelling reason to investigate the use of neuromorphic systems. Additionally, because they inherently implement neural network-style computation, neuromorphic systems are a natural platform for many of today's artificial intelligence and machine learning applications. There is also promise to leverage the inherent computational properties of neuromorphic computers to perform a wide variety of different types of computation [2].

Each of these features of neuromorphic computers are inspired by characteristics of the brain and have been prioritized in the implementation of neuromorphic computers in recent years. However, it is not clear whether they are the only aspects of biological brains that are important for performing computation. For example, though neurons and synapses have been chosen as the primary computational units of neuromorphic computers, there are a variety of other types of neural components that may be useful for computation, including glial cells [88, 41]. Moreover, neurons and synapses have been a convenient level of abstraction for neuromorphic systems, but whether they are the most appropriate level of abstraction is still an open question [89].

Unlike some of future computing technologies, many physical realizations of neuromorphic hardware are currently under development or are even available for use to the research community. Several large-scale neuromorphic systems have been developed with a variety of approaches and goals [36]. The European Union's Human Brain Project sponsored the development of SpiNNaker [31] and BrainScaleS [94] to enable neuroscience simulations at scale. An optimized digital neuromorphic processor called ODIN (Online-learning Digital spiking Neuromorphic) has also been proposed [35], allowing the use of slightly more complex neuron models. One of the neuromorphic platforms targeting more general computations for wider classes of applications is the Tianjic chip, a platform, that supports both neuromorphic spiking neural networks and the traditional artificial neural networks for different categories of problems [84]. Industry has also taken an interest to neuromorphic systems. Some examples include IBM's TrueNorth [65] and Intel's Loihi [22], and there are also a variety of academic efforts, including DYNAPs [69], Neurogrid [12], IFAT [112], and BrainScales2 [93]. The usefulness of neuromorphic hardware, such as BrainScales2, has been demonstrated in carrying out optimizations for learning to learn (i.e., where an optimization process is used to define how learning occurs) scenarios for spiking neural networks as it runs at a much accelerated timescales compared to biological timescales [15].

All the aforementioned large-scale neuromorphic systems are silicon-based and implemented using conventional CMOS (Complementary Metal Oxide Semiconductor) technology. However, there is a tremendous amount of research in the neuromorphic community on developing new types of materials for neuromorphic implementations, such as phase-change, ferroelectric, nonfilamentary, topological insulators, or channel-doped biomembranes [42, 75, 74]. One popular approach in the literature is using memristors as the fundamental device to have resistive memory to collocate processing and memory [45, 59], but other types of devices have also been used to implement neuromorphic systems, including optoelectronic devices [98]. Each device and material used to implement neuromorphic systems has unique operating characteristics, such as how fast they operate, their energy consumption, and the level of resemblance to biology. The diversity of devices and materials used to implement neuromorphic hardware today offers the opportunity to customize the properties required for a given application.

Most research in the field of neuromorphic computing today fall in the realm of the aforementioned hardware systems, devices, and materials. However, to most effectively use neuromorphic computers in the future, exploit all their unique computational characteristics, and help drive their hardware design, they must be connected to neuromorphic algorithms and applications. From this perspective, we provide an overview of the current state of the art in neuromorphic algorithms and applications and provide a forward-looking view on opportunities for the future for neuromorphic computing in computer science and computational science.

Additionally, it is worth noting that the term neuromorphic computing has been used for a wide array of different types of technologies. As noted previously, the original definition only encompassed mixed analog-digital implementations. In this work, we consider all types of hardware implementations (digital, mixed analog-digital, analog) as neuromorphic, but we restrict our attention to *spiking* neuromorphic systems, i.e., those that implement spiking neural networks.

2 Neuromorphic Algorithms and Applications

Programming a neuromorphic computer entails creating a spiking neural network (SNN) that can be deployed to that neuromorphic system (see Box 1). SNNs take an additional level of inspiration from biological neural systems in the way that they perform computation. In particular, SNNs include time in the way that they perform computation; neurons and synapses both include notions of time within most neuromorphic systems. For example, spiking neurons might leak charge over time based on a particular time constant, and neurons and/or synapses in SNNs might have an associated time delay.

Algorithms for neuromorphic implementations often entail how to define an SNN for a given application. There are a wide variety of algorithmic approaches for neuromorphic computing systems that fall into two broad categories: (1) algorithms for training or learning an SNN to be deployed to a neuromorphic system (Figure 2) and (2) non-machine learning algorithms in which SNNs are hand-constructed to solve a particular task. It is worth noting that here training and learning algorithms refer to the mechanism of optimizing the parameters of an SNN (typically the synaptic weights) for a particular problem.

This section provides an overview of some state-of-the-art approaches for neuromorphic computing algorithms, both on benchmarking datasets and demonstrating their potential applicability to real-world applications.

Spiking Neural Networks

Spiking neural networks (SNNs) are a particular type of artificial neural network in which the function of the neurons and the synapses in the network are more inspired by biology than other types of artificial neural networks, such as multi-layer perceptrons. The key difference between traditional artificial neural networks and SNNs is that SNNs take into account timing in their operation. Neuron models implemented in SNNs in the literature range from simpler integrate and fire models, in which charge is integrated over time until a threshold value is reached, to much more complex and biologically plausible models, such as the Hodgkin-Huxley neuron model, which approximates the functionality of specific aspects of biological neurons such as ion channels [98]. Both neurons and synapses in SNNs can include time components that affect their functionality.

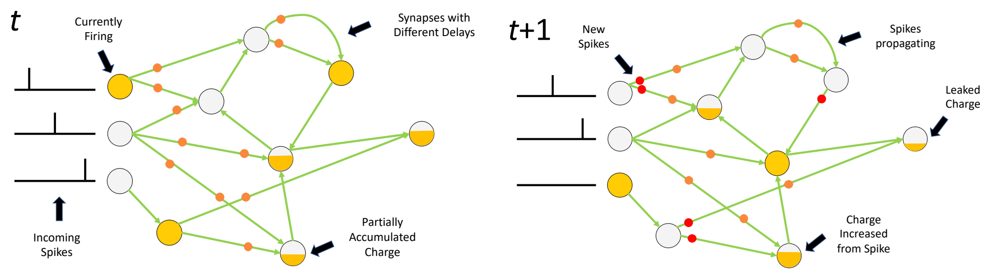
Neurons in spiking neural networks accumulate charge over time from either the environment (via input information to the network) or from internal communications (usually via spikes from other neurons in the network). Neurons have an associated threshold value, and when the charge value on that neuron reaches the threshold value, it fires, sending communications along all of its outgoing synapses. Neurons may also include a notion of "leak," where the accumulated charge that is not above the threshold dissipates as time passes. Additionally, neurons may have an associated axonal delay, in which outgoing information from the neuron is delayed before it affects its outgoing synapses. Synapses form the connections between neurons, and each synapse has a pre-synaptic neuron and a post-synaptic neuron. Synapses have an associated weight value, and that value may be positive (excitatory) or negative (inhibitory). Synapses may have an associated delay value such that communications from the pre-synaptic neuron are delayed in reaching the post-synaptic neuron. Synapses also commonly include a learning mechanism in which the weight value of the synapse changes over time based on activity in the network. Neuromorphic systems often realize a particular fabric of connectivity, but the synapses may be turned on and off to realize a network structure within that connectivity. Additionally, parameters of the neurons and synapses such as neuron thresholds, synaptic weights, axonal delays, and synaptic delays are often programmable within a neuromorphic architecture.

Unlike in traditional artificial neural networks where information is received at the input and then synchronously passed between layers in the network, in SNNs, even if input information is received at the same time and the SNN is organized in layers, because the delays on each synapse and neuron may be different, information is propagated asynchronously throughout the network, arriving at different times. Thus, this is beneficial while realizing SNNs on a neuromorphic hardware, which can be designed to operate in an event-driven or asynchronous manner that fits well with the temporal dynamics of spiking neurons and synapses. An example SNN and how it operates in the temporal domain is shown in the figure. In this example, synapses are shown with a time delay. Information is communicated by spikes passed throughout the network. In this example, the network's operation at time t (left) and time $t+1$ (right) is shown, to show how the network's state changes with time.

2.1 Machine Learning Algorithms

2.1.1 Spike-Based Quasi-Back-Propagation

Back-propagation and stochastic gradient descent have shown impressive performance in the field of deep learning. However, these approaches do not map directly to SNNs because spiking neurons do not have differentiable activation functions (i.e., many spiking neurons use a threshold function, which is not directly differentiable). Additionally, the temporal processing component of SNNs can add another difficulty in training and learning for these approaches. Algorithms that have been successful for deep learning applications



t
Currently Firing
Synapses with Different Delays
Incoming Spikes
Partially Accumulated Charge
t+1
New Spikes
Spikes propagating
Leaked Charge
Charge Increased from Spike

must be adapted to work with SNNs (top-left in Figure 2), and these adaptations can reduce the accuracy of the SNN compared with a similar artificial neural network [117, 53, 8, 9].

Some of the approaches that adapt deep learning-style training include using a surrogate gradient and having a smoothed activation function to compute the error gradients while performing weight adjustments in each of the successive layers [76, 119]. There have also been a few demonstrations on computing the spike error gradient [34, 57, 56] that have shown close to state-of-the-art classification performance on the Modified National Institute of Standards and Technology (MNIST) handwritten digits dataset. To make use of the inherent temporal dimension in SNNs, there have been efforts attempting to employ rules that have been used to train recurrent neural networks, albeit with several approximations. As surveyed by Zenke and Neftci [121], approaches such as backpropagation through time and real-time recurrent learning have been demonstrated on neuromorphic datasets, such as the Spiking Heidelberg Digits (SHD) and the Spiking Speech Command (SSC) dataset [20].

2.1.2 Mapping a Pre-Trained Deep Neural Network

Because deep neural networks (DNNs) have an established training mechanism, several efforts to deploy a neuromorphic solution for a problem begin by training a DNN and then performing a mapping process to convert it to an SNN for inference purposes (see top-right in Figure 2). Most of these approaches have yielded near state-of-the-art performance with potential for significant energy reduction in computation due to the use of only accumulate computations (AC) over multiply and accumulate computations (MAC) in DNNs on several commonly employed datasets, such as MNIST, Canadian Institute For Advanced Research (CIFAR)-10, and ImageNet [27, 39, 99, 101]. Most initial conversion techniques used weight normalization or activation normalization or employed average pooling instead of max pooling [27, 90, 99]. Other approaches involve training DNNs in a constrained manner so that the neuron's activation function iteratively starts resembling that of a spiking neuron [39, 101]. Stockl et al., have proposed a new mapping strategy where SNNs make use of Few Spikes neuron model (FS-neuron) that can represent complex activation functions temporally with at most two spikes [108]. They have shown close to deep neural network accuracies on benchmark image classification datasets with significantly fewer time-steps per inference compared to previously demonstrated conversion strategies. Several applications demonstrated on neuromorphic hardware have employed some of the aforementioned mapping techniques. Tasks such as keyword spotting, medical image analysis, and object detection have been demonstrated to run efficiently on existing platforms such as Intel's Loihi and IBM's TrueNorth [14, 32, 103].

It is worth noting that training a conventional DNN and then mapping to neuromorphic hardware, especially emerging hardware systems, can see a reduction in accuracy not only because of the change from DNNs to SNNs, but also because of the neuromorphic hardware itself. Often neuromorphic hardware systems that are implemented with emerging hardware devices such as a memristors will have reduced precision in the synaptic weight values they can realize, and they may also have cycle-to-cycle and device variation. When creating a mapping technique, it is important to take into account how these characteristics might influence the inference performance of a mapped network.

Algorithms that use deep learning-style training to train SNNs often do not leverage all the inherent computational capabilities of SNNs, and using those approaches limits the capabilities of SNNs to what

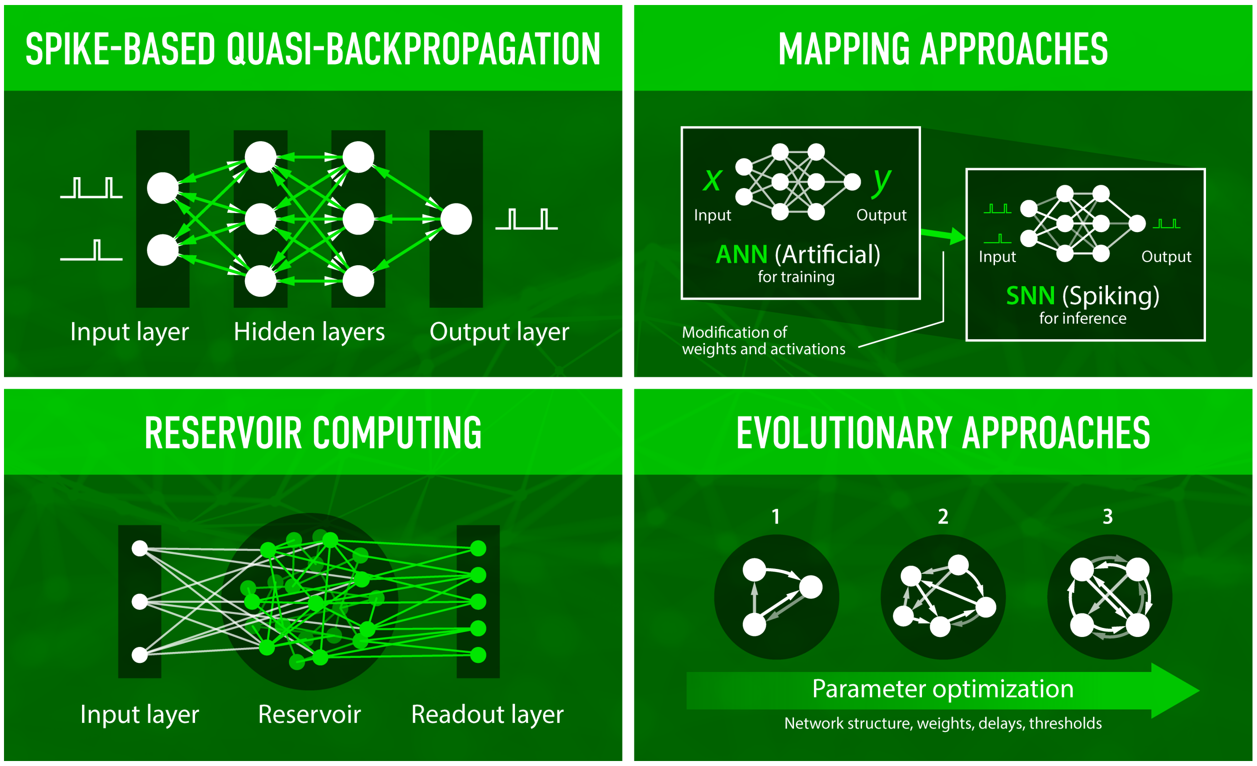


Figure 2: Common training approaches for SNNs. In the top-left panel, the structure of a network for a spike-based quasi-backpropagation is depicted. In this case, the training approach is performed directly on the SNN. The top-right panel shows the procedure for mapping approaches, where a traditional ANN is trained and then mapped into an SNN. The bottom-left panel shows the structure of a typical reservoir computing solution, including the input layer, the reservoir, and the readout layer. The bottom-right panel shows an evolutionary approach to evolving the structures and parameters of an SNN over time.

traditional artificial neural networks can already achieve. For example, most gradient descent style rules, including mapping approaches, do not focus on the temporal aspect of SNN computation.

2.1.3 Reservoir Computing

Another common algorithm used in SNNs is reservoir computing or liquid state machines (see bottom-left in Figure 2). In reservoir computing, a sparse recurrent SNN is defined to function as the "liquid" or reservoir. This liquid is typically randomly defined, but is required to have two properties: input separability, which requires that different inputs result in different outputs, and fading memory, which requires that signals do not continue to propagate infinitely through the reservoir and instead will eventually die out. In addition to the liquid itself, which is untrained, a reservoir computing approach also includes a readout mechanism, such as a linear regression, is trained to recognize the output of the reservoir. The key advantage of a reservoir computing is approach is that is does not require any training of the SNN component. Reservoir computing in SNNs uses the sparse and recurrent connections with synaptic delays in networks of spiking neurons to cast the input to a spatially and temporally higher dimensional space [111]. Several demonstrations of spike-based reservoir computing have shown their effectiveness at processing temporally varying signals [51, 29, 116]. Variants of this computing framework have ranged from simple reservoir networks for bio-signal processing and prosthetic control applications [51] to using hierarchical layers of liquid state machines—a type of reservoir network—interconnected with layers trained in supervised mode for video [106] and audio signal processing applications [116].

2.1.4 Evolutionary Approaches

Evolutionary approaches to training or designing SNNs (see bottom-right in Figure 2) have also been used [96, 92, 95]. In an evolutionary algorithm, a random collection of potential solutions is created to form an initial population. Each member of the population is evaluated and assigned a score, which are then used to perform selection (preferentially selecting better performing individuals) and reproduction (creating new individuals through recombination of old individuals and mutations) to produce a new population. In the context of SNNs for neuromorphic computing, evolutionary approaches can be used to determine parameters of the SNN, such as neuron thresholds or synaptic delays, or the structure of the network, such as the number of neurons and how they are connected to each other with synapses. These approaches are attractive because they do not require differentiability in the activation functions and do not rely on any particular network structure (i.e., feed-forward, recurrent). Additionally, they can be used to evolve the structure of the network and the parameters. However, their flexibility has a cost: evolutionary approaches can be slow to converge compared with other training approaches. Evolutionary approaches have been most successfully applied to control applications, such as video games [86] and autonomous robot navigation [92, 67].

2.1.5 Plasticity

Several neurobiological studies have reported the modulation of synaptic strength based on the activity of the connected neurons, which has been postulated as a learning mechanism for various tasks [13]. Spike timing-dependent plasticity (STDP), which operates on the underlying principle of adjusting the weights based on relative spike timings from pre- and post-synaptic neurons, is the most commonly implemented synaptic plasticity mechanism in neuromorphic literature [98]. Several different mathematical formulations of this rule have been demonstrated on the MNIST, CIFAR-10, and ImageNet datasets [102, 72, 55, 46, 11, 61]. Shrestha et al. presents a hardware-friendly modification of the exponential STDP rule, albeit the classification performance on MNIST was lower than the best results achieved so far with SNNs [102]. STDP-style rules have also been shown to approximate several machine learning approaches, such as clustering and Bayesian inference [73, 77]. STDP as a clustering mechanism has been demonstrated as a spike sorter in brain machine interface applications [73]. Combinations of spiking reservoirs and STDP have also been employed in an SNN approach called NeuCube [48]. NeuCube has been used to process electroencephalograms and functional magnetic resonance imaging signals in applications such as sleep state detection and prosthetic controllers [48, 17, 54].

A much broader class of SNNs for modeling dynamical systems are the recurrent networks with delays and synaptic plasticity. One such class of networks are the polychronization networks [43], which have been employed for different spatio-temporal classification tasks [114]. Alemi, et al. demonstrate a local learning rule with recurrent SNNs with fewer spikes to realize non-linear dynamical systems [4]. Such recurrent SNNs have shown greater classification ability with winner-take-all models [60, 79, 47]. To leverage the temporal dimension of SNN, some learning algorithms aim at generating single or multiple spikes at desired times, which have been applied in classification tasks [36, 16, 115, 58, 120]. Most of these algorithms also depend on the spike representation used to encode the input signals. There have been several approaches to encode signals in terms of spike rates, latency, and neuron population, etc. [97, 85].

2.2 Non-Machine Learning Algorithms

The typical use cases for neuromorphic computing have been mainly machine learning, but neuromorphic computers have also recently been considered for non-machine learning algorithms. One common class of algorithms that have been mapped onto neuromorphic implementations comes from graph theory [38] [19][49] [5]. A graph is a model that consists of vertices, or *nodes*, along with directed pairwise relationships between nodes called *edges*. The underlying architecture of a neuromorphic system is a directed graph, so when there is a graph of interest, it can be embedded directly into a neuromorphic architecture with suitable parameter settings, and the spike raster can reveal graph properties. For example, with the correct parameter sets, a given node can be spiked, and the time at which other nodes spike corresponds exactly with the length of the shortest path from the source node [3]. During the COVID-19 pandemic, neuromorphic computing was coupled with graph theory as a tool for analyzing the spread of disease [37].

Random walks have also been implemented within neuromorphic systems. In a random walk, a random node is selected as a starting point, and an agent moves along an edge departing from that node selected at random. The process is repeated for several steps, and the locations visited by the random agent can reveal an important characteristic related to the underlying network. Random walk analyses frequently involve performing many random walks and then aggregating the results for analysis. Although traditional hardware performs the parallel step well, the aggregation and analysis step requires high energy usage sequential operation and does not always benefit from parallel architectures, such as GPUs. Severa et al. [100] showed that in certain settings, random walks could be studied in low-energy neuromorphic settings and that the analysis can be done in an inherently parallel fashion. Random walks are central to many problems in computational physics. Smith, et.al. [105] used neuromorphic deployments of discrete time Markov chains to approximate solutions for both particle transport problems and heat flow on complex geometries with energy efficient time scalable approaches. Given that graphs are a special class of objects called relational structures, foundational work of Cook [18] on relational structures has proven to be compatible with neuromorphic hardware, finding application to learning in cortical networks [26] and unsupervised learning tasks [25].

Neuromorphic computing have also been used to find approximate solutions to NP-complete problems, which are some of the most difficult problems in computation. Neuromorphic systems can perform comparably in terms of time-to-solution and accuracy of the solution to their conventional counterparts, which use CPUs and GPUs to approximately solve NP-complete problems. The energy efficiency of the neuromorphic approaches makes them amenable to be used in edge computing applications. Thus, a considerable amount of energy can potentially be saved using a neuromorphic computer to address NP-complete problems without compromising on the time-to-solution and the accuracy of the solution. Several neuromorphic approaches for approximating NP-complete problems have been proposed in the literature. Alom et al. used the IBM TrueNorth Neurosynaptic system to approximately solve the quadratic unconstrained binary optimization (QUBO) problem, which is NP-hard [6].

Mniszewski converted the NP-complete graph partitioning problem to the QUBO problem and used the IBM TrueNorth system to solve it approximately [68]. In some cases, their solutions are more accurate than the solutions returned by the D-Wave quantum computer, which was designed to approximately solve the QUBO problem. Yakopcic et al. leveraged Intel Loihi to approximately solve the boolean satisfiablity

(SAT) problem [118]. They were able to obtain the solutions of many different SAT problems with varying sizes on the Loihi chip. Earlier work of Mostafa et al. developed neural network techniques for approximately solving many constraint satisfaction problems [71], later on dedicated neuromorphic hardware [70]. Fonseca and Furber develop a software framework for solving NP-complete constraint SAT problems on the SpiNNaker architecture [30]. Pecevski et al. [83] used neuromorphic hardware to perform inference and sampling on general graphical structures, such as Bayes' nets, which is NP complete for random variables with probabilities not bounded away from 0 [21].

3 Closing the Gap Between Promises and Reality

Though neuromorphic hardware is available in the research community and there have been a wide variety of algorithms proposed, the applications have been primarily targeted towards benchmark datasets and demonstrations. Neuromorphic systems are not currently being used in real-world applications, and there are still a wide variety of challenges that restrict or inhibit rapid growth in algorithmic and application development.

3.1 Limiting of Algorithmic Focus

There has yet to be a machine learning algorithm/application combination for which neuromorphic computing substantially outperforms deep learning approaches in terms of accuracy, though there have been compelling demonstrations in which neuromorphic solutions outperform other hardware implementations such as neural hardware and edge GPUs in terms of energy efficiency [14]. This has led to the argument that neuromorphic systems are primarily interesting because of their low power computing abilities. However, we believe that there is tremendous algorithmic opportunity for neuromorphic systems as well. There has been a focus on backpropagation-based training approaches because of their state-of-the-art performance in deep learning. By limiting focus to those algorithms, however, we may also be limiting ourselves to achieving results that are only comparable with (rather than surpassing) deep learning approaches. We believe that there is more opportunity to develop approaches that utilize the inherent features of spiking neuromorphic systems, such as evolutionary algorithms or neuroscience-inspired approaches. At the same time, though these approaches have also been iterated on for decades, they similarly have not achieved state-of-the-art results. Since these approaches use the native features of SNNs and thus require computing SNNs, iterating on and refining these algorithms is inherently bound by how efficiently SNNs can be computed. Neuromorphic computers have the opportunity to significantly speed up SNN evaluation and thus, provide the opportunity to accelerate development of SNN-based algorithms.

3.2 Limited Access to and Usability of Existing Hardware and Software

One key issue that inhibits algorithmic and application development for neuromorphic systems is the lack of readily accessible and usable software and hardware systems for the entire computational and computer science communities. Several neuromorphic systems are available; however, there are a limited number of these systems and they are typically only available via cloud access to the broader community. Several open-source neuromorphic simulators have support for different hardware back ends, such as multinode CPUs, GPUs, and emerging neuromorphic hardware (e.g., SpiNNaker [50]). Although simulators such as NEST [33], Brian [35], and Nengo [10] are available, none are universally used, and they are often built for a specific purpose. For example, NEST targets primarily computational neuroscience workloads, whereas Nengo implements computation as framed by the Neural Engineering Framework [107]. Because these software systems are developed for particular communities and use cases, their broader usability and accessibility are limited outside those communities. In the future, to enable broader usability, development of neuromorphic simulators, hardware, and software should take into account the more broad applicability of these systems. Many of these simulators also have limited performance when operating at scale [52]. With the current data explosion comes the need to process data quickly enough to keep up with data generation speeds, hence emphasising the need for highly performant and scalable neuromorphic simulators that can effectively leverage current high-performance computing systems to develop and evaluate neuromorphic workloads. The above mentioned limitations of simulators and also the large training times of current neuromorphic algorithms compared to non-spiking approaches have limited the usage of neuromorphic solutions to real-world applications, which actively needs to be addressed. Additionally, because the simulators are slow, it is very difficult to rapidly evaluate new algorithmic approaches, leading to slow algorithmic evolution. To enable more rapid advancement, the community needs performant hardware simulators or emulators that can be used when hardware is difficult or impossible to access.

3.3 Incorporation of Neuromorphic Computers into Broader Computing Environments

In addition to the lack of availability for neuromorphic hardware and performant simulators, many future use cases of neuromorphic computers are likely to be included as part of a broader heterogeneous computing environment rather than be operated in isolation. Because of performance constraints (e.g., energy usage or processing speed) in existing hardware, more exotic hardware systems, such as neuromorphic and quantum computers, will increasingly be included in the computing landscape to accelerate particular types of computation. Since neuromorphic computing is likely not the best accelerator for all possible computations but is likely to be the most performant on certain applications and certain metrics (i.e., energy efficiency for neural network computation and speed for neuroscience simulations), they will potentially be used alongside other accelerators or specialized hardware for other applications. For example, a neuromorphic system could be operating as a co-processor with a CPU with other nontraditional computing systems—such as reconfigurable, quantum, and approximate computing systems—also possibly operating as co-processors. Integrating these diverse systems into a single compute environment and developing programming models that enable the effective use of diverse heterogeneous systems is an ongoing challenge [113]. Additionally, neuromorphic systems are heavily reliant on host machines for defining the software structure that is deployed to the neuromorphic implementation and often for communication to and from the outside world (i.e., interfacing with sensors and actuators for real-world applications). This reliance can have a significant impact on the performance benefits of using a neuromorphic computer, to the point where factoring in communication and host machine costs eliminates the benefits of using a neuromorphic computer to implement an application [24]. A key challenge moving forward is how to minimize this reliance on traditional computers, as well as to optimize communication between them.

3.4 Lack of Benchmarks and Metrics

Another key challenge for neuromorphic algorithmic development is the lack of clearly established benchmarks, metrics, and challenge problems. Without common benchmarks and metrics, it is extremely difficult to evaluate which hardware system is most suitable for a given algorithm or application. Moreover, evaluating whether a new algorithm performs well can be extremely difficult without commonly defined metrics. Challenge problems, such as the ImageNet task for deep learning, drove significant advances in the field [66]. The field of neuromorphic computing does not have a well-defined task or set of tasks that the entire community is attempting to solve. Several groups have created datasets with event/spike based representation and having temporal dimension specifically for benchmarking neuromorphic training algorithms, such as the neuromorphic MNIST [78], DVS Gesture [7], and the Spiking Heidelberg audio datasets [20]. Though there are an increasing number of these neuromorphic-targeted datasets, these datasets have not yet been broadly adopted by the field at large as common benchmarks, limiting their utility at present. Datasets such as MNIST, CIFAR-10, and ImageNet dominate the benchmarks in neuromorphic, but these datasets do not require the native temporal processing capabilities present in neuromorphic systems, and as such, do not showcase the full capabilities of neuromorphic systems. Though the field needs benchmarks and challenge problems to target, it is also worth noting that creating a single challenge problem can also be dangerous because it may result in advances that target only that application, which can narrow the broader utility of the technology (an issue that affects the field of machine learning as a whole). Because of the wide variety of algorithms and applications of neuromorphic systems as detailed in the previous sections, we propose that instead of a single benchmark or challenge problem, there should instead be a suite of challenge problems, drawing from both machine learning and non-machine learning use cases.

3.5 Lack of Programming Abstractions

Finally, an additional challenge specific to the development of non-machine learning algorithms for neuromorphic deployment is the lack of programming abstractions for neuromorphic implementations. Currently, these approaches require that the programmer design the SNN for a particular task at the neuron and synapse level, defining all parameter values of those elements and how they are connected. Not only is this a fundamentally different way of thinking about how programming is performed but it is often also very time consuming and error prone to implement operations at this level. It is no coincidence that many of the non-machine learning algorithms for neuromorphic are centered on graph algorithms, because there is a very clear approach for mapping a graph into a network (i.e., nodes to neurons and edges to synapses). There have been attempts to describe programming abstractions at a higher level, such as the Neural Engineering Framework (NEF) [107] and Dynamic Neural Fields (DNF) [91]. However, these are often restricted to specific use cases and algorithms, such as biologically-plausible neural models in the case of NEF and modeling embodied cognition for DNF. We believe both NEF and DNF are important abstractions for neuromorphic, but we also believe that there is still a gap in defining abstractions for using neuromorphic computers more broadly. One approach that could be taken is defining subnetworks of spiking neurons and synapses to perform specific tasks that are familiar to programmers, such as binary operations, conditionals, and loops, in addition to those defined by NEF and DNF, as well as guidance for composing these subnetworks into larger networks capable of more complex tasks. There has been some initial work in this direction. In particular, Plank, et al. describe subnetworks that perform basic tasks such as AND, OR, and XOR using different spike encoding schemes [87]. However, there is still tremendous opportunity to influence how these subsystems should be defined and composed. It is clear that they can be used for more than just neural network computation; however, until clearer program abstractions are defined and/or the broader computing community becomes more familiar with the computational primitives of neuromorphic computing, non-machine learning neuromorphic algorithms will be slow to develop. It is also worth noting that although it is possible to implement a variety of different types of computations on neuromorphic computers, this does not mean that every problem *should* be mapped onto a neuromorphic computer because not every problem is likely to benefit from the computational characteristics of neuromorphic systems described in Section 1. It is better to think of neuromorphic computers as specialized processors than general purpose computer. However, we do want to emphasize with this work that the scope of specialized processors is not just neuroscience or machine learning algorithms, but a wide variety of other types of computation as well.

4 Outlook

Neuromorphic processors are energy efficient and adept at performing machine learning and some non-machine learning computations. They offer tremendous potential for computing beyond Moore's law. We envision at least three use cases for neuromorphic processors. First, because of their low power consumption, neuromorphic processors will be indispensable for edge computing applications, such as autonomous systems (e.g., vehicles, drones), robotics, remote sensing, wearable technology, and IoT. Second, neuromorphic computers are well poised to become the artificial intelligence accelerators and co-processors in personal computing devices, such as smart phones, laptops, and desktops. Accelerators and specialized architectures have already been widely adopted in mobile phones, and the need for extremely energy efficient operations to improve battery life in those systems as well as laptops continues to be an important factor. Neuromorphic systems can help realize those operations with potentially orders of magnitude less power than today's accelerators. Finally, because of their ability to perform certain non-machine learning computations, we envision that neuromorphic computers will be added on as co-processors in next-generation extremely heterogeneous high-performance computing systems. In this scenario, neuromorphic computers would be expected to enable spike-based simulations [37], run graph algorithms [49, 38], solve differential equations [104], and efficiently approximate NP-complete problems [68]. It is worth noting that the different use cases of neuromorphic computers, from edge devices to accelerators and co-processors are likely to look very different in their implementations. Neuromorphic systems deployed at the edge may be specialized to operate with one particular application and have a focus on, for example, extremely low power inference performance, whereas neuromorphic systems for broader types of computations in an HPC setting will have a focus on enabling reconfigurability and training acceleration. Though neuromorphic systems are not currently present in these use cases, we do expect that they will begin to emerge in these technologies in the future, likely first in the edge computing space as specialized processors, and later into future heterogeneous computers.

Although several large-scale neuromorphic hardware systems are already available to the research community, these systems are all being actively developed. Moreover, there is a wide variety of research efforts in developing new materials and devices to implement neuromorphic hardware. As such, there is an opportunity to engage in a software-hardware codesign process in the development of neuromorphic hardware [1]. Currently, most neuromorphic hardware design begins from the bottom of the compute stack (i.e., materials and devices) up to the algorithms and applications. That is, the hardware substrate is defined first, and the onus is then on the algorithms and applications developers to map them onto that particular hardware implementation. However, because the hardware is being actively developed and new materials and devices for neuromorphic computing are being actively investigated, there is tremendous opportunity to engage in codesign all across the compute stack, for example, so that the algorithms and applications can influence the underlying hardware design (Figure 3). There is an opportunity to tailor the underlying hardware implementation to suit a particular application's needs or constraints. This opens up new horizons to not only focus on digital computing, but also to rethink using analog, approximate, and mixed-signal computing [28], since biological neural computation itself is inherently analog and stochastic. Among several approaches proposed in the literature on software-hardware codesign, one is using Bayesian optimization and Neural Architecture Search approaches in which several stacks of computing that range from materials and devices to algorithm and applications are codesigned to optimize overall system performance [82, 80, 81]. For example, in a memristive crossbar-based accelerator, an automatic codesign optimization approach has the opportunity to define the number and sizes of crossbars to optimize the accuracy and energy efficiency of the design for different applications or datasets. In addition to the opportunity for whole-stack co-design driven by algorithms and applications noted here and by Aimone in [1], there is also the opportunity to allow for emerging materials and devices for neuromorphic computers to inspire our algorithmic approaches, for example, in the implementation of plasticity. Today, the process of implementing synaptic plasticity on devices begins with the inspiration of plasticity in biological brains, is implemented and demonstrated on emerging devices (top-down co-design), and then the specific plasticity algorithm is adapted to however plasticity functions on that device (bottom-up co-design). However, plasticity mechanisms in biological brain evolved to use biological materials and components. We believe there may be opportunities to look at the underlying physical behaviors of other devices and materials to inform new neuromorphic algorithms.

The potential of neuromorphic computers in the future of computing and computational science is only beginning to be understood, and there is tremendous opportunity to leverage the inherent computational characteristics of these systems for machine learning and certain non-machine learning computations as well. Using neuromorphic computers most effectively will require a paradigm shift in how researchers think about programming. We believe that there are opportunities to achieve unprecedented algorithmic performance in terms of speed and energy efficiency on many applications with neuromorphic computers. In particular, in addition to their clear benefits for neural network-style computation, we believe that two areas that have the opportunity to see tremendous benefits from neuromorphic computers are graph algorithms and optimization tasks. Both of these types of algorithms and applications have the opportunity to benefit from the massively parallel, event-driven, and/or stochastic operation of neuromorphic systems. With the confluence of many different types of algorithms and applications in neuromorphic, along with the active development of large-scale neuromorphic hardware and emerging devices and materials, now is the time for the greater computing community to begin considering neuromorphic computers a part of the greater computing landscape.

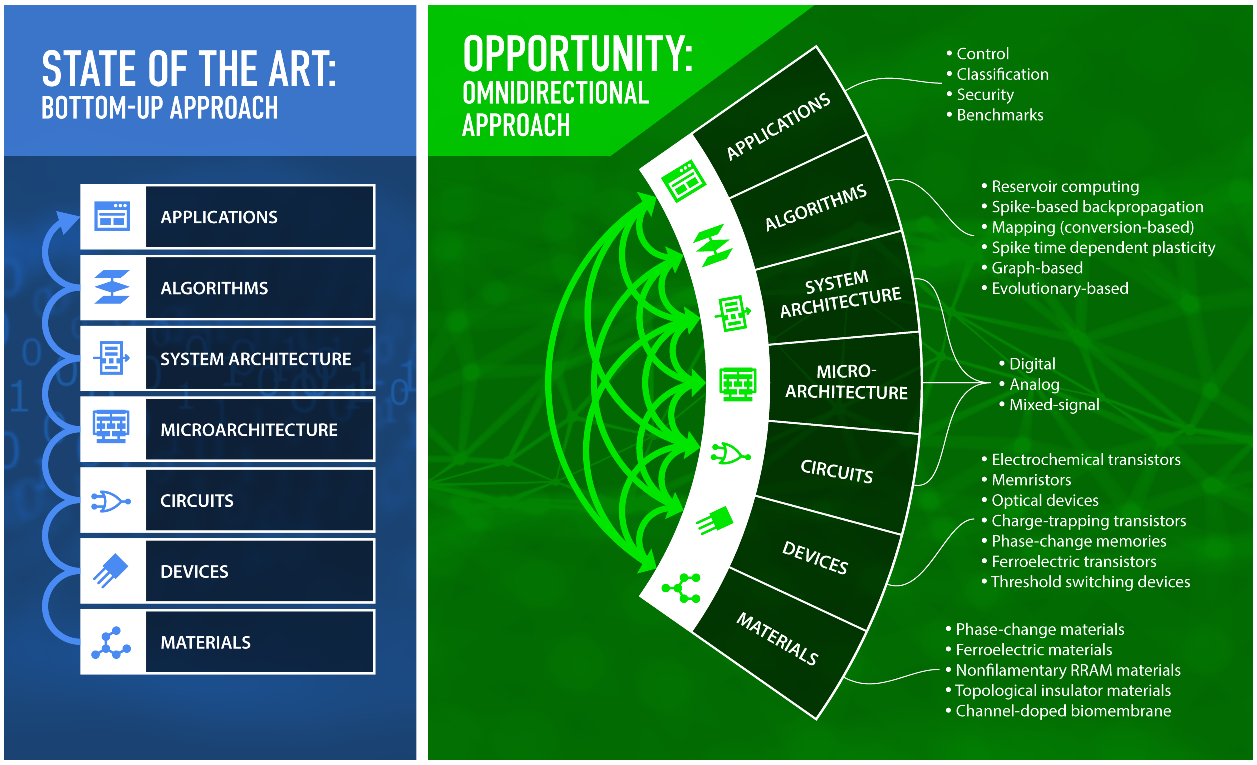


Figure 3: Opportunity for full compute stack co-design in neuromorphic systems. The current approach (shown on the left) is a bottom-up approach, where materials and devices are defined first, and those inform the architectures, algorithms, and applications sequentially. The opportunity for a future co-design approach (shown on the right) is for all aspects of the design stack to influence other components directly, e.g., for applications to directly influence the materials chosen or for the algorithms to directly influence the circuits used.

Acknowledgments

This material is based upon work supported by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under contract number DE-AC05-00OR22725.

We would like to thank Nathan Armistead for his aid in creating the graphics for this manuscript.

References

[1] James B Aimone. A roadmap for reaching the potential of brain-derived computing. *Advanced Intelligent Systems*, 3(1):2000191, 2021.

[2] James B Aimone, Kathleen E Hamilton, Susan Mniszewski, Leah Reeder, Catherine D Schuman, and William M Severa. Non-neural network applications for spiking neuromorphic hardware. In *Proceedings of the Third International Workshop on Post Moores Era Supercomputing*, pages 24–26, 2018.

[3] James B Aimone, Yang Ho, Ojas Parekh, Cynthia A Phillips, Ali Pinar, William Severa, and Yipu Wang. Provable neuromorphic advantages for computing shortest paths. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 497–499, 2020.

[4] Alireza Alemi, Christian Machens, Sophie Deneve, and Jean-Jacques Slotine. Learning nonlinear dynamics in efficient, balanced spiking networks using local plasticity rules. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[5] Abdullahi Ali and Johan Kwisthout. A spiking neural algorithm for the network flow problem. *arXiv preprint arXiv:1911.13097*, 2019.

[6] Md Zahangir Alom, Brian Van Essen, Adam T Moody, David Peter Widemann, and Tarek M Taha. Quadratic unconstrained binary optimization (qubo) on neuromorphic computing system. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3922–3929. IEEE, 2017.

[7] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7388–7397, 2017.

[8] Navin Anwani and Bipin Rajendran. Training multi-layer spiking neural networks using normad based spatio-temporal error backpropagation. *Neurocomputing*, 380:67–77, 2020.

[9] Alireza Bagheri, Osvaldo Simeone, and Bipin Rajendran. Training probabilistic spiking neural networks with first-to-spike decoding. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2986–2990. IEEE, 2018.

[10] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48, 2014.

[11] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):1–15, 2020.

[12] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[13] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 18(24):10464–10472, 1998.

[14] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pages 1–8, 2019.

[15] Thomas Bohnstingl, Franz Scherr, Christian Pehle, Karlheinz Meier, and Wolfgang Maass. Neuromorphic hardware learns to learn. *Frontiers in neuroscience*, 13:483, 2019.

[16] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.

[17] Sugam Budhraja, Basabdatta Sen Bhattacharya, Simon Durrant, Zohreh Doborjeh, Maryam Doborjeh, and Nikola Kasabov. Sleep stage classification using neucube on spinnaker: a preliminary study. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[18] Matthew Cook. *Networks of* relations*. California Institute of Technology, 2005.

[19] Kevin Corder, John V Monaco, and Manuel M Vindiola. Solving vertex cover via ising model on a neuromorphic processor. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[20] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[21] Paul Dagum and Michael Luby. An optimal approximation algorithm for bayesian inference. *Artificial Intelligence*, 93(1-2):1–27, 1997.

[22] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[23] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

[24] Alan Diamond, Thomas Nowotny, and Michael Schmuker. Comparing neuromorphic solutions in action: implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms. *Frontiers in neuroscience*, 9:491, 2016.

[25] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[26] Peter U Diehl and Matthew Cook. Learning and inferring relations in cortical networks. *arXiv preprint arXiv:1608.08267*, 2016.

[27] Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

[28] Rodney Douglas, Misha Mahowald, and Carver Mead. Neuromorphic analogue vlsi. *Annual review of neuroscience*, 18(1):255–281, 1995.

[29] Chao Du, Fuxi Cai, Mohammed A Zidan, Wen Ma, Seung Hwan Lee, and Wei D Lu. Reservoir computing using dynamic memristors for temporal information processing. *Nature communications*, 8(1):1–10, 2017.

[30] Gabriel A Fonseca Guerra and Steve B Furber. Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Frontiers in neuroscience*, 11:714, 2017.

[31] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[32] Neil Getty, Thomas Brettin, Dong Jin, Rick Stevens, and Fangfang Xia. Deep medical image analysis with representation learning and neuromorphic computing. *Interface Focus*, 11(1):20190122, 2021.

[33] Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[34] Julian Göltz, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Dominik Dold, Laura Kriener, Akos Ferenc Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, et al. Fast and deep neuromorphic learning with time-to-first-spike coding. *arXiv preprint arXiv:1912.11443*, 2019.

[35] Dan FM Goodman and Romain Brette. Brian: a simulator for spiking neural networks in python. *Frontiers in neuroinformatics*, 2:5, 2008.

[36] Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9(3):420–428, 2006.

[37] Kathleen Hamilton, Prasanna Date, Bill Kay, and Catherine Schuman D. Modeling epidemic spread with spike-based models. In *International Conference on Neuromorphic Systems 2020*, pages 1–5, 2020.

[38] Kathleen E Hamilton, Tiffany M Mintz, and Catherine D Schuman. Spike-based primitives for graph algorithms. *arXiv preprint arXiv:1903.10574*, 2019.

[39] Eric Hunsberger and Chris Eliasmith. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141*, 2016.

[40] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André Van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73, 2011.

[41] Yilda Irizarry-Valle and Alice Cline Parker. An astrocyte neuromorphic circuit that influences neuronal phase synchrony. *IEEE transactions on biomedical circuits and systems*, 9(2):175–187, 2015.

[42] Raisul Islam, Haitong Li, Pai-Yu Chen, Weier Wan, Hong-Yu Chen, Bin Gao, Huaqiang Wu, Shimeng Yu, Krishna Saraswat, and HS Philip Wong. Device and materials requirements for neuromorphic computing. *Journal of Physics D: Applied Physics*, 52(11):113001, 2019.

[43] Eugene M Izhikevich. Polychronization: computation with spikes. *Neural computation*, 18(2):245–282, 2006.

[44] Conrad D James, James B Aimone, Nadine E Miner, Craig M Vineyard, Fredrick H Rothganger, Kristofor D Carlson, Samuel A Mulder, Timothy J Draelos, Aleksandra Faust, Matthew J Marinella, et al. A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications. *Biologically Inspired Cognitive Architectures*, 19:49–64, 2017.

[45] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301, 2010.

[46] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020.

[47] David Kappel, Bernhard Nessler, and Wolfgang Maass. Stdp installs in winner-take-all circuits an online approximation to hidden markov model learning. *PLoS computational biology*, 10(3):e1003511, 2014.

[48] Nikola K Kasabov. Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, 52:62–76, 2014.

[49] Bill Kay, Prasanna Date, and Catherine Schuman. Neuromorphic graph algorithms: Extracting longest shortest paths and minimum spanning trees. In *Proceedings of the Neuro-inspired Computational Elements Workshop*, pages 1–6, 2020.

[50] James C Knight and Thomas Nowotny. Gpus outperform current hpc and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Frontiers in neuroscience*, 12:941, 2018.

[51] Dhireesha Kudithipudi, Qutaiba Saleh, Cory Merkel, James Thesing, and Bryant Wysocki. Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing. *Frontiers in neuroscience*, 9:502, 2016.

[52] Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, and Catherine D Schuman. Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing*, 2021.

[53] Shruti R Kulkarni and Bipin Rajendran. Spiking neural networks for handwritten digit recognition—supervised learning and network optimization. *Neural Networks*, 103:118–127, 2018.

[54] Kaushalya Kumarasinghe, Mahonri Owen, Denise Taylor, Nikola Kasabov, and Chi Kit. Faneurobot: A framework for robot and prosthetics control using the neucube spiking neural network architecture and finite automata theory. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4465–4472. IEEE, 2018.

[55] Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in neuroscience*, 12:435, 2018.

[56] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in neuroscience*, 14, 2020.

[57] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[58] Shenglan Li and Qiang Yu. New efficient multi-spike learning for fast processing and robust learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4650–4657, 2020.

[59] Yibo Li, Zhongrui Wang, Rivu Midya, Qiangfei Xia, and J Joshua Yang. Review of memristor devices in neuromorphic computing: materials sciences and device challenges. *Journal of Physics D: Applied Physics*, 51(50):503002, 2018.

[60] Wolfgang Maass. On the computational power of winner-take-all. *Neural computation*, 12(11):2519–2535, 2000.

[61] Erwann Martin, Maxence Ernoult, Jérémie Laydevant, Shuai Li, Damien Querlioz, Teodora Petrisor, and Julie Grollier. Eqspike: spike-driven equilibrium propagation for neuromorphic implementations. *Iscience*, 24(3):102222, 2021.

[62] Christian Mayr, Sebastian Hoeppner, and Steve Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning. *arXiv preprint arXiv:1911.02385*, 2019.

[63] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[64] Carver Mead. How we created neuromorphic engineering. *Nature Electronics*, 3(7):434–435, 2020.

[65] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[66] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, 2017.

[67] J Parker Mitchell, Grant Bruer, Mark E Dean, James S Plank, Garrett S Rose, and Catherine D Schuman. Neon: Neuromorphic control for autonomous robotic navigation. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, pages 136–142. IEEE, 2017.

[68] Susan M Mniszewski. Graph partitioning as quadratic unconstrained binary optimization (qubo) on spiking neuromorphic hardware. In *Proceedings of the International Conference on Neuromorphic Systems*, pages 1–5, 2019.

[69] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017.

[70] Hesham Mostafa, Lorenz K Müller, and Giacomo Indiveri. An event-based architecture for solving constraint satisfaction problems. *Nature communications*, 6(1):1–10, 2015.

[71] Hesham Mostafa, Lorenz K Müller, and Giacomo Indiveri. Rhythmic inhibition allows neural networks to search for maximally consistent states. *Neural computation*, 27(12):2510–2547, 2015.

[72] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. First-spike-based visual categorization using reward-modulated stdp. *IEEE transactions on neural networks and learning systems*, 29(12):6178–6190, 2018.

[73] Anand Kumar Mukhopadhyay, Atul Sharma, Indrajit Chakrabarti, Arindam Basu, and Mrigank Sharad. Power-efficient spike sorting scheme using analog spiking neural network classifier. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(2):1–29, 2021.

[74] Joseph S Najem, Graham J Taylor, Ryan J Weiss, Md Sakib Hasan, Garrett Rose, Catherine D Schuman, Alex Belianinov, C Patrick Collier, and Stephen A Sarles. Memristive ion channel-doped biomembranes as synaptic mimics. *ACS nano*, 12(5):4702–4711, 2018.

[75] SR Nandakumar, Shruti R Kulkarni, Anakha V Babu, and Bipin Rajendran. Building brain-inspired computing systems: Examining the role of nanoscale devices. *IEEE Nanotechnology Magazine*, 12(3):19–35, 2018.

[76] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[77] Bernhard Nessler, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Comput Biol*, 9(4):e1003037, 2013.

[78] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.

[79] Matthias Oster, Rodney Douglas, and Shih-Chii Liu. Computation with spikes in a winner-take-all network. *Neural computation*, 21(9):2437–2465, 2009.

[80] Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[81] Maryam Parsa, J Parker Mitchell, Catherine D Schuman, Robert M Patton, Thomas E Potok, and Kaushik Roy. Bayesian-based hyperparameter optimization for spiking neuromorphic systems. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4472–4478. IEEE, 2019.

[82] Maryam Parsa, John P Mitchell, Catherine D Schuman, Robert M Patton, Thomas E Potok, and Kaushik Roy. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in neuroscience*, 14:667, 2020.

[83] Dejan Pecevski, Lars Buesing, and Wolfgang Maass. Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons. *PLoS computational biology*, 7(12):e1002294, 2011.

[84] Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

[85] Balint Petro, Nikola Kasabov, and Rita M Kiss. Selection and optimization of temporal spike encoding methods for spiking neural networks. *IEEE transactions on neural networks and learning systems*, 31(2):358–370, 2019.

[86] J. S. Plank, C. Rizzo, K. Shahat, G. Bruer, T. Dixon, M. Goin, G. Zhao, J. Anantharaj, C. D. Schuman, M. E. Dean, G. S. Rose, N. C. Cady, and J. Van Nostrand. The TENNLab suite of LIDAR-based control applications for recurrent, spiking, neuromorphic systems. In *44th Annual GOMACTech Conference*, Albuquerque, March 2019.

[87] J. S. Plank, C. Zheng, C. D. Schumann, and C. Dean. Spiking neuromorphic networks for binary tasks. In *International Conference on Neuromorphic Computing Systems (ICONS)*, pages 1–8. ACM, 2021.

[88] Ioannis Polykretis, Guangzhi Tang, and Konstantinos P Michmizos. An astrocyte-modulated neuromorphic central pattern generator for hexapod robot locomotion on intel's loihi. In *International Conference on Neuromorphic Systems 2020*, pages 1–9, 2020.

[89] Thomas Potok, Catherine Schuman, Robert Patton, and Hai Li. Neuromorphic computing, architectures, models, and applications. *A Beyond-CMOS Approach to Future Computing, The Department of Energy (DOE) Office of Scientific and Technical Information (OSTI), Oak Ridge, TN*, 2016.

[90] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

[91] Yulia Sandamirskaya. Dynamic neural fields as a step toward cognitive neuromorphic architectures. *Frontiers in neuroscience*, 7:276, 2014.

[92] J David Schaffer. Evolving spiking neural networks for robot sensory-motor decision tasks of varying difficulty. In *Proceedings of the Neuro-inspired Computational Elements Workshop*, pages 1–7, 2020.

[93] Johannes Schemmel, Sebastian Billaudelle, Phillip Dauer, and Johannes Weis. Accelerated analog neuromorphic computing. *arXiv preprint arXiv:2003.11996*, 2020.

[94] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950. IEEE, 2010.

[95] Stefan Schliebs and Nikola Kasabov. Evolving spiking neural network—a survey. *Evolving Systems*, 4(2):87–98, 2013.

[96] Catherine D Schuman, J Parker Mitchell, Robert M Patton, Thomas E Potok, and James S Plank. Evolutionary optimization for neuromorphic systems. In *Proceedings of the Neuro-inspired Computational Elements Workshop*, pages 1–9, 2020.

[97] Catherine D Schuman, James S Plank, Grant Bruer, and Jeremy Anantharaj. Non-traditional input encoding schemes for spiking neuromorphic systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2019.

[98] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017.

[99] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

[100] William Severa, Rich Lehoucq, Ojas Parekh, and James B Aimone. Spiking neural algorithms for markov process random walk. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[101] William Severa, Craig M Vineyard, Ryan Dellana, Stephen J Verzi, and James B Aimone. Training deep neural networks for binary communication with the whetstone method. *Nature Machine Intelligence*, 1(2):86–94, 2019.

[102] Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 international joint conference on neural networks (IJCNN)*, pages 1999–2006. IEEE, 2017.

[103] Rohit Shukla, Mikko Lipasti, Brian Van Essen, Adam Moody, and Naoya Maruyama. Remodel: Rethinking deep cnn models to detect and count on a neurosynaptic system. *Frontiers in neuroscience*, 13:4, 2019.

[104] J Darby Smith, William Severa, Aaron J Hill, Leah Reeder, Brian Franke, Richard B Lehoucq, Ojas D Parekh, and James B Aimone. Solving a steady-state pde using spiking networks and neuromorphic hardware. In *International Conference on Neuromorphic Systems 2020*, pages 1–8, 2020.

[105] John Darby Smith, Aaron Jamison Hill, Leah Reeder, Brian C Franke, Richard B Lehoucq, Ojas D Parekh, William Mark Severa, and James Bradley Aimone. Neuromorphic scaling advantages for energy-efficient random walk computations. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2020.

[106] Nicholas Soures and Dhireesha Kudithipudi. Deep liquid state machines with neural plasticity for video activity recognition. *Frontiers in neuroscience*, 13:686, 2019.

[107] Terrence C Stewart. A technical overview of the neural engineering framework. *University of Waterloo*, 2012.

[108] Christoph Stöckl and Wolfgang Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 3(3):230–238, 2021.

[109] Dmitri Strukov, Giacomo Indiveri, Julie Grollier, and Stefano Fusi. Building brain-inspired computing. *Nature Communications*, (10):4838–2019, 2019.

[110] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang. Hardware for machine learning: Challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8. IEEE, 2017.

[111] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, 2019.

[112] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, 12:891, 2018.

[113] Jeffrey S Vetter, Ron Brightwell, Maya Gokhale, Pat McCormick, Rob Ross, John Shalf, Katie Antypas, David Donofrio, Travis Humble, Catherine Schuman, et al. Extreme heterogeneity 2018-productive computational science in the era of extreme heterogeneity: Report for doe ascr workshop on extreme heterogeneity. Technical report, USDOE Office of Science (SC), Washington, DC (United States), 2018.

[114] Felix Wang, William M Severa, and Fred Rothganger. Acquisition and representation of spatio-temporal signals in polychronizing spiking neural networks. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pages 1–5, 2019.

[115] Quan Wang, Constantin A Rothkopf, and Jochen Triesch. A model of human motor sequence learning explains facilitation and interference effects based on spike-timing dependent plasticity. *PLoS computational biology*, 13(8):e1005632, 2017.

[116] Parami Wijesinghe, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines. *Frontiers in neuroscience*, 13:504, 2019.

[117] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

[118] Chris Yakopcic, Nayim Rahman, Tanvir Atahary, Tarek M Taha, and Scott Douglass. Solving constraint satisfaction problems using the loihi spiking neuromorphic processor. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1079–1084. IEEE, 2020.

[119] Shihui Yin, Shreyas K Venkataramanaiah, Gregory K Chen, Ram Krishnamurthy, Yu Cao, Chaitali Chakrabarti, and Jae-sun Seo. Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations. In *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–5. IEEE, 2017.

[120] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

[121] Friedemann Zenke and Emre O Neftci. Brain-inspired learning on neuromorphic substrates. *Proceedings of the IEEE*, 2021.