PERSPECTIVE
https://doi.org/10.1038/s43588-021-00184-y

nature computational science

Opportunities for neuromorphic computing algorithms and applications

Catherine D. Schuman[1,2]✉, Shruti R. Kulkarni[1], Maryam Parsa[1,3], J. Parker Mitchell[1], Prasanna Date[1] and Bill Kay[1]

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.

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. The term neuromorphic was coined by Carver Mead in the late 1980s[1,2], and at that time primarily referred to mixed analogue–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 separate CPUs and memory units, where data and instructions are stored in the latter. In a neuromorphic computer, on the other hand, both processing and memory are governed by the neurons and the synapses. Programs in neuromorphic computers are defined by the structure of the neural network and its parameters, rather than by explicit instructions as in a von Neumann computer. In addition, while von Neumann computers encode information as numerical values represented by binary values, neuromorphic computers 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[3].

Given the aforementioned contrasting characteristics between the two architectures (Fig. 1), neuromorphic computers present some fundamental operational differences:

- Highly parallel operation: neuromorphic computers are inherently 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 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 regarding the processor/memory separation, which causes a slowdown in the maximum throughput that can be achieved. In addition, this collocation helps avoid data accesses from main memory, as in conventional computing systems, which consume a considerable amount of energy compared with the compute energy[4].
- Inherent scalability: neuromorphic computers are meant to be inherently scalable as adding additional neuromorphic chips entails 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[5,6] and Loihi[7].
- Event-driven computation: neuromorphic computers leverage event-driven computation (meaning, computing only when data are available) and temporally sparse activity to allow for extremely efficient computation[8,9]. 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 computers can include a notion of randomness, such as in the firing of neurons, to allow for noise.

The features of a neuromorphic computer are well noted in the literature and are given as motivators for implementing and using them[10–14]. One of the most attractive features of neuromorphic computers 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 while the rest is idle. Due to the increasing energy cost of computing, as well as the increasing number of 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 computers. Furthermore, as they inherently

[1]Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA. [2]Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA. [3]Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA. ✉e-mail: cschuman@utk.edu

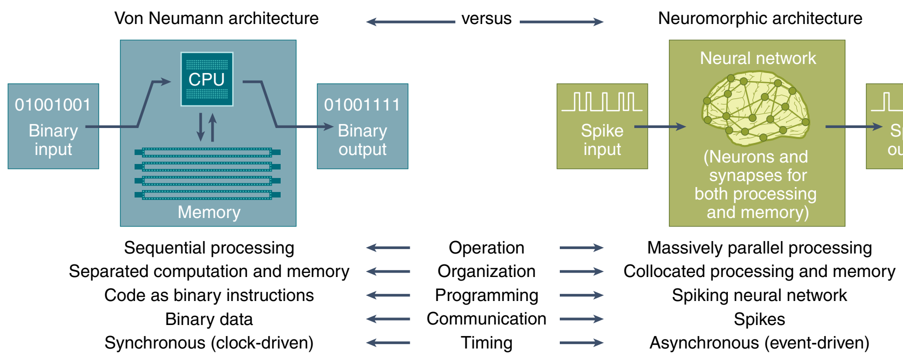

Von Neumann architecture
versus
Neuromorphic architecture
CPU
01001001 Binary input
01001111 Binary output
Memory
Neural network
Spike input
Spike output
(Neurons and synapses for both processing and memory)
Sequential processing ← Operation → Massively parallel processing
Separated computation and memory ← Organization → Collocated processing and memory
Code as binary instructions ← Programming → Spiking neural network
Binary data ← Communication → Spikes
Synchronous (clock-driven) ← Timing → Asynchronous (event-driven)

Fig. 1 | Comparison of the von Neumann architecture with the neuromorphic architecture. These two architectures have some fundamental differences when it comes to operation, organization, programming, communication, and timing, as depicted here.

implement neural network-style computation, neuromorphic computers 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[15].

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, although 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[16,17]. Moreover, neurons and synapses have been a convenient level of abstraction for neuromorphic computers, but whether they are the most appropriate level of abstraction is still an open question[18].

Unlike some of the 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 computers have been developed with a variety of approaches and goals[19]. The European Union's Human Brain Project sponsored the development of SpiNNaker[6] and BrainScaleS[20] to enable neuroscience simulations at scale. An optimized digital neuromorphic processor called the online-learning digital spiking neuromorphic (ODIN) has also been proposed[21], 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[22]. Both industry and academia have taken an interest in neuromorphic systems: in industry, some examples include IBM's TrueNorth[23] and Intel's Loihi[7], and there are also a variety of academic efforts, including DYNAPs[24], Neurogrid[25], IFAT[13] and BrainScales-2[26]. The usefulness of neuromorphic hardware such as BrainScales-2 has been demonstrated in carrying out optimizations for learning to learn scenarios (meaning, where an optimization process is used to define how learning occurs) for spiking neural networks, running at a much accelerated timescales compared to biological timescales[27].

All of the aforementioned large-scale neuromorphic computers are silicon-based and implemented using conventional 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, non-filamentary, topological insulators or channel-doped biomembranes[28–30]. One popular approach in the literature is using memristors as the fundamental device to have resistive memory to collocate processing and memory[31,32], but other types of devices have also been used to implement neuromorphic computers, including optoelectronic devices[10]. Each device and material used to implement neuromorphic computers 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 of 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 of the opportunities for the future of neuromorphic computing in computer science and computational science. 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 analogue-digital implementations. In this work, we consider all types of hardware implementations (digital, mixed analogue-digital, analogue) as neuromorphic, but we restrict our attention to spiking neuromorphic computers, that is, those that implement spiking neural networks.

Neuromorphic algorithms and applications

Programming a neuromorphic computer often entails creating a spiking neural network (SNN) that can be deployed to that neuromorphic computer (see Box 1). SNNs take an additional level of inspiration from biological neural systems in the way that they perform computation; in particular, neurons and synapses in SNNs include notions of time within most neuromorphic computers. 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.

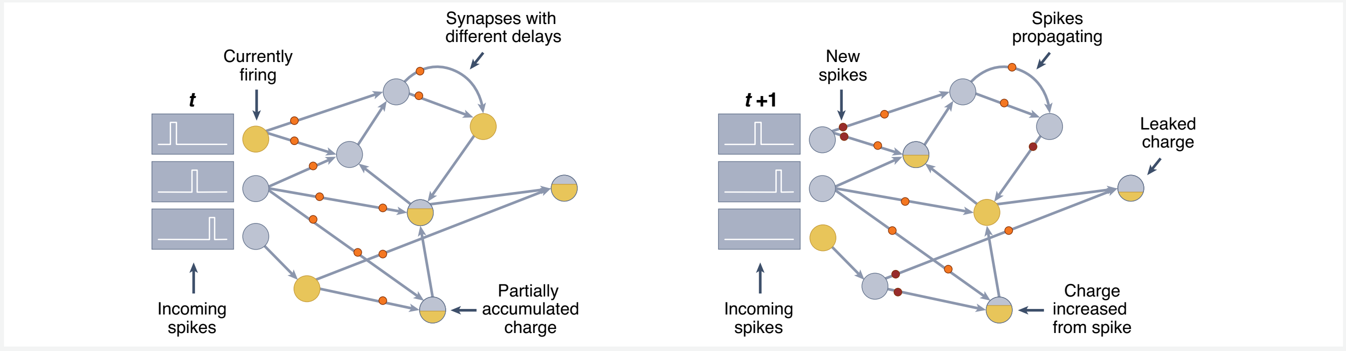
Algorithms for neuromorphic implementations often entail how to define an SNN for a given application. There are a wide variety of algorithmic approaches for neuromorphic computing systems that fall into two broad categories: (1) algorithms for training or learning an SNN to be deployed to a neuromorphic computer (Fig. 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

Box 1 | Spiking neural networks

Spiking neural networks 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 multilayer 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 simple 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[10]. 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 leakage, where the accumulated charge that is not above the threshold dissipates as time passes. Furthermore, 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, which 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 computers often realize a particular fabric of connectivity, but the synapses may be turned on and off to realize a network structure within that connectivity. Furthermore, 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 traditional artificial neural networks, in which 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 into layers, as the delays on each synapse and neuron may be different, information is propagated asynchronously throughout the network, arriving at different times; this is beneficial for realizing SNNs on 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 depicted 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 depicted, to show how the network's state changes with time.



the parameters of an SNN (typically the synaptic weights) for a particular problem.

Machine learning algorithms. *Spike-based quasi-backpropagation.* Backpropagation and stochastic gradient descent have shown impressive performance in the field of deep learning; however, these approaches do not map directly to SNNs as spiking neurons do not have differentiable activation functions (that is, many spiking neurons use a threshold function, which is not directly differentiable). Furthermore, 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 must be adapted to work with SNNs (Fig. 2a), and these adaptations can reduce the accuracy of the SNN compared with a similar artificial neural network[33–36].

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[19,21]. There have also been a few demonstrations on computing the spike error gradient[37–39] 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[40], 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[41].

Mapping a pre-trained deep neural network. As 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 (Fig. 2b). Most of these approaches have yielded near state-of-the-art performance with potential for substantial energy reduction due to the use of only accumulate

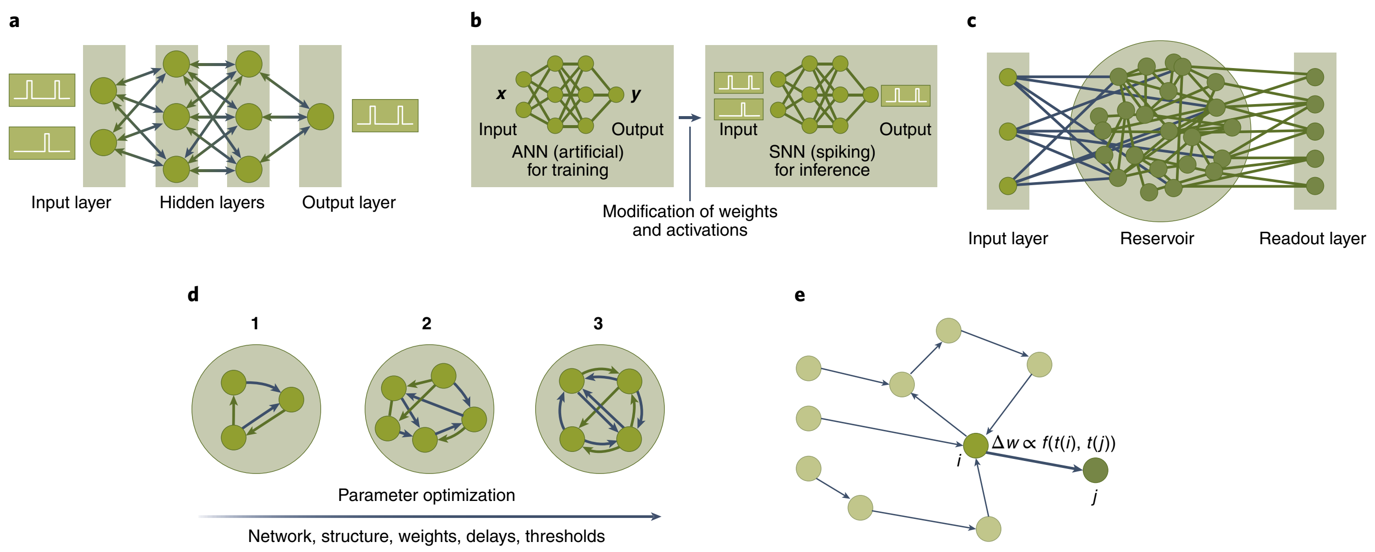


Fig. 2 | Common training approaches for SNNs. a, The structure of a network for a spike-based quasi-backpropagation is depicted. In this case, the training approach is performed directly on the SNN. **b**, The procedure for mapping approaches, where a traditional artificial neural network (ANN) is trained and then mapped into an SNN. **c**, The structure of a typical reservoir computing solution, including the input layer, the reservoir and the readout layer. **d**, In an evolutionary approach, the structures and parameters of an SNN evolve over time. **e**, The approach taken in spike-timing-dependent plasticity, which is a synaptic plasticity mechanism where the weights (Δw) are adjusted with a function f based on relative spike timings (t) from pre- and post-synaptic neurons (i and j, respectively).

computations over multiply and accumulate computations in DNNs on several commonly employed datasets, such as MNIST, Canadian Institute for Advanced Research (CIFAR)-10, and ImageNet[42–45]. Most initial conversion techniques used weight normalization or activation normalization, or employed average pooling instead of max pooling[42,44,46]. Other approaches involved training DNNs in a constrained manner so that the neuron's activation function iteratively starts resembling that of a spiking neuron[43,45]. Stockl and colleagues have proposed a new mapping strategy where SNNs make use of Few Spikes neuron model (FS-neuron), which can represent complex activation functions temporally with at most two spikes[47]. They have shown close to deep neural network accuracies on benchmark image classification datasets with significantly fewer time-steps per inference compared with 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[48–50].

It is worth noting that training a conventional DNN and then mapping it to neuromorphic hardware, especially emerging hardware systems, can result in 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. In addition, 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 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.

Reservoir computing. Another common algorithm used in SNNs is reservoir computing or liquid state machines (Fig. 2c). 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, that is trained to recognize the output of the reservoir. The key advantage of reservoir computing is that it 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[51]. Several demonstrations of spike-based reservoir computing have shown their effectiveness at processing temporally varying signals[52–54]. Variants of this computing framework have ranged from simple reservoir networks for bio-signal processing and prosthetic control applications[52] to using hierarchical layers of liquid state machines—a type of reservoir network—interconnected with layers trained in supervised mode for video[55] and audio signal processing applications[54].

Evolutionary approaches. Evolutionary approaches for training or designing SNNs (Fig. 2d) have also been used[56–58]. 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 is 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 (for instance, feed-forward and recurrent). They can also 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[59] and autonomous robot navigation[57,60].

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[61]. Spike-timing-dependent plasticity (STDP)—which operates on the underlying principle of adjusting the weights on the basis of relative spike timings from pre- and post-synaptic neurons (Fig. 2e)—is the most commonly implemented synaptic plasticity mechanism in neuromorphic literature[10]. Several different mathematical formulations of this rule have been demonstrated on the MNIST, CIFAR-10 and ImageNet datasets[62–67]. Shrestha et al. presented 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[62]. STDP-style rules have also been shown to approximate several machine learning approaches such as clustering and Bayesian inference[68,69]. STDP as a clustering mechanism has been demonstrated as a spike sorter in brain machine interface applications[68]. Combinations of spiking reservoirs and STDP have also been employed in an SNN approach called NeuCube[70], which has been used to process electroencephalograms and functional magnetic resonance imaging signals in applications such as sleep state detection and prosthetic controllers[70–72].

A much broader class of SNNs for modelling dynamical systems are the recurrent networks with delays and synaptic plasticity. One such class of networks are the polychronization networks[73], which have been employed for different spatio-temporal classification tasks[74]. Alemi et al. demonstrated a local learning rule with recurrent SNNs with fewer spikes to realize non-linear dynamical systems[75]. Such recurrent SNNs have shown greater classification ability with winner-take-all models[76–78]. To leverage the temporal dimension of SNN, some learning algorithms aim to generate single or multiple spikes at desired times, which have been applied in classification tasks[79–83]. 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[3,84].

Non-machine learning algorithms. The typical use cases for neuromorphic computing have been mainly machine learning-related, but neuromorphic computers have also been recently considered for non-machine learning algorithms. One common class of algorithms that have been mapped onto neuromorphic implementations comes from graph theory[85–88]. The underlying architecture of a neuromorphic computer is a directed graph, and thus 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[89]. During the COVID-19 pandemic, neuromorphic computing was coupled with graph theory as a tool for analysing the spread of disease[90].

Random walks have also been implemented within neuromorphic computers. 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 for the sequential operation and does not always benefit from parallel architectures, such as GPUs. Severa and colleagues[91] showed that in certain settings, random walks could be studied in low-energy neuromorphic settings and that the analysis can be performed in an inherently parallel fashion. Smith and co-workers[92] 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[93] on relational structures has proven to be compatible with neuromorphic hardware, finding application to learning in cortical networks[94] and unsupervised learning tasks[95].

Neuromorphic computing has also been used to find approximate solutions to NP-complete problems: several studies have shown that neuromorphic systems can achieve a similar performance in terms of time-to-solution and solution accuracy when compared with other conventional approaches, which use CPUs and GPUs to approximately solve NP-complete problems. For instance, Alom and co-workers used the IBM TrueNorth Neurosynaptic system to approximately solve the quadratic unconstrained binary optimization (QUBO) problem[96]. Mniszewski[97] converted the NP-complete graph partitioning problem to the QUBO problem and used the IBM TrueNorth system to solve it approximately: in some cases, neuromorphic solutions were more accurate than the solutions returned by the D-Wave quantum computer. Yakopcic et al. leveraged Intel Loihi to approximately solve the boolean satisfiablity (SAT) problem[98]. Earlier work of Mostafa et al. developed neural network techniques for approximately solving many constraint SAT problems[8,99]. Fonseca and Furber[100] developed a software framework for solving NP-complete constraint SAT problems on the SpiNNaker architecture. Pecevski et al.[101] used neuromorphic hardware to perform inference and sampling on general graphical structures, such as Bayes's nets, which is NP complete for random variables with probabilities not bounded away from zero[102].

Closing the gap between expectations and reality

Although 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 computers 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.

Widening algorithmic focus. There has yet to come a machine learning algorithm/application combination for which neuromorphic computing substantially outperforms deep learning approaches in terms of accuracy, although 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[48]. This has led to the argument that neuromorphic computers are primarily interesting because of their low power computing abilities; however, we believe that there is a tremendous algorithmic opportunity for neuromorphic computers 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 are more opportunities to develop approaches that utilize the inherent features of spiking neuromorphic systems, such as evolutionary algorithms or neuroscience-inspired approaches.

At the same time, although these approaches have also been iterated on for decades, they similarly have not achieved state-of-the-art results. As 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. As performant neuromorphic computers have only recently become available to the research community, now is the time to investigate on-chip learning and training for more efficient computation of these types of algorithms.

Widening usability and access to hardware and simulators. One key issue that inhibits algorithmic and application development for neuromorphic computers is the lack of readily accessible and usable software and hardware systems for the entire computational and computer science communities. Several different neuromorphic implementations are available; however, there are a limited number of each of these implementations and they are typically only available via restricted 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 (for example, SpiNNaker[103]). Although simulators such as NEST[104], Brian[105] and Nengo[106] are available, 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]. As 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[108]. With the current data explosion comes the need to process data quickly enough to keep up with data generation speeds, hence emphasizing 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 with non-spiking approaches have limited the usage of neuromorphic solutions to real-world applications, which actively needs to be addressed. Furthermore, as 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 that can be used when hardware is difficult or impossible to access.

Enabling more diverse computing environments. 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. Due to performance constraints (for example, energy usage or processing speed) in existing hardware, emergent hardware systems, such as neuromorphic and quantum computers, will increasingly be included in the computing landscape to accelerate particular types of computation. 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[109].

Neuromorphic computers are heavily reliant on conventional host machines for defining the software structure that is deployed to the neuromorphic computer and often for communication to and from the outside world (that is, interfacing with sensors and actuators for real-world applications). This reliance can have a considerable 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[110]. A key challenge moving forward is how to minimize this reliance on traditional computers, as well as to optimize communication between them.

Defining 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 that field[111]. 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 temporal dimension specifically for benchmarking neuromorphic training algorithms, such as the neuromorphic MNIST[112], DVS Gesture[9] and the Spiking Heidelberg audio datasets[41]; however, 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 computers, and as such, do not showcase the full capabilities of neuromorphic computers.

Although 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[113]). Due to the wide variety of algorithms and applications of neuromorphic computers as detailed in the previous sections, we propose that, instead of a single benchmark or challenge problem, there should be a suite of challenge problems, drawing from both machine learning and non-machine learning use cases.

Defining 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. These approaches currently 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 also very time consuming and error prone. It is no coincidence that many of the non-machine learning algorithms for neuromorphic are centred on graph algorithms, as there is a very clear approach for mapping a graph into a network (that is, 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 (DNFs)[114]. However, these are often restricted to specific use cases and algorithms, such as biologically plausible neural models in the case of NEFs and modelling-embodied cognition for DNFs. We believe both the NEF and DNFs are important abstractions for the field, but we also believe that there is still a gap in defining abstractions for using neuromorphic computers more broadly.

One possible approach 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

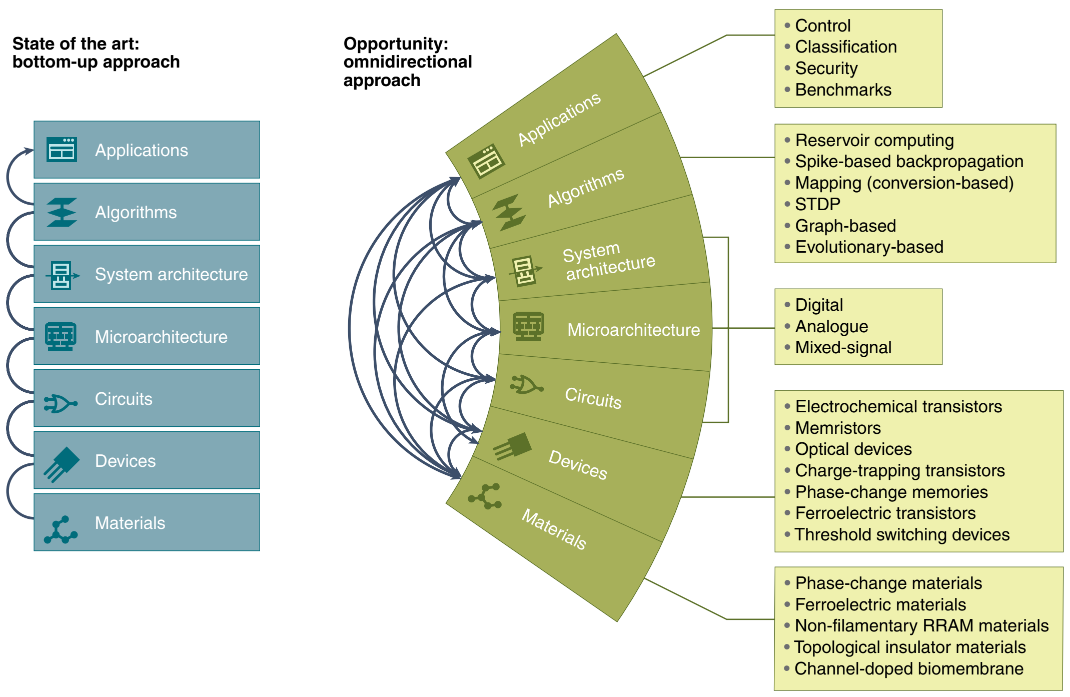


Fig. 3 | Opportunity for full compute stack co-design in neuromorphic computers. The current approach (shown on the left) is a bottom-up approach, where materials and devices are defined first, and those inform the architectures, algorithms and applications sequentially. The opportunity for a future co-design approach (shown on the right) is for all aspects of the design stack to influence other components directly; for example, for applications to directly influence the materials chosen or for the algorithms to directly influence the circuits used. RRAM, resistive random-access memory.

complex tasks. For instance, Plank et al. described subnetworks that perform basic tasks such as AND, OR and XOR using different spike encoding schemes[115], but 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 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: not every problem is likely to benefit from the computational characteristics of neuromorphic computers described in the first section. 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.

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, due to their low power consumption, neuromorphic processors will be indispensable for edge-computing applications such as autonomous systems (for example, vehicles and drones), robotics, remote sensing, wearable technology and the Internet of Things. 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 computers can help realize those operations with potentially orders of magnitude less power than today's accelerators. Finally, due to their ability to perform certain non-machine learning computations, we envision that neuromorphic computers will be added on as co-processors in next-generation heterogeneous high-performance computing systems. In this scenario, neuromorphic computers would be expected to enable spike-based simulations[90], run graph algorithms[85,87], solve differential equations[116] and efficiently approximate NP-complete problems[97]. 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 computers 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 computers for broader types of computations in an high-performance computing setting will likely have a focus on enabling reconfigurability and training acceleration. Although neuromorphic computers are not currently present in these use cases, we do expect that they will begin to emerge in these technologies in the future, first probably in the edge computing space as specialized processors and later in future heterogeneous computers.

Several large-scale neuromorphic hardware systems are already available to the research community, and 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 co-design process in the development of neuromorphic hardware[117]. Most neuromorphic hardware design currently begins from the bottom of the compute stack (that is, the materials and devices) and then goes up to the algorithms and applications; that is, the hardware substrate is defined first, and the onus is then on the algorithm and application developers to map them onto that particular hardware implementation. However, there is tremendous opportunity to engage in codesign all across the compute stack so that the algorithms and applications can influence the underlying hardware design (Fig. 3), and 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 analogue, approximate and mixed-signal computing[118], as biological neural computation itself is inherently analogue and stochastic. Among several approaches proposed in the literature on software–hardware co-design, 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[119–121]. 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, 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, it is then implemented and demonstrated on emerging devices (top-down co-design), and then finally the specific plasticity algorithm is adapted to match how 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 behaviours of other devices and materials to inform new neuromorphic algorithms[122].

The potential of neuromorphic computers in the future of computer 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 computers. 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 computational science community to begin considering neuromorphic computers a part of the greater computing landscape.

Received: 30 April 2021; Accepted: 7 December 2021;
Published online: 31 January 2022

References

1. Mead, C. Neuromorphic electronic systems. *Proc. IEEE* **78**, 1629–1636 (1990).
2. Mead, C. How we created neuromorphic engineering. *Nat. Electron.* **3**, 434–435 (2020).
3. Schuman, C. D., Plank, J. S., Bruer, G. & Anantharaj, J. Non-traditional input encoding schemes for spiking neuromorphic systems. In *2019 International Joint Conference on Neural Networks (IJCNN)* 1–10 (IEEE, 2019).
4. Sze, V., Chen, Y.-H., Emer, J., Suleiman, A. & Zhang, Z. Hardware for machine learning: challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)* 1–8 (IEEE, 2017).
5. Mayr, C., Hoeppner, S. & Furber, S. SpiNNaker 2: a 10 million core processor system for brain simulation and machine learning. Preprint at https://arxiv.org/abs/1911.02385 (2019).
6. Furber, S. B., Galluppi, F., Temple, S. & Plana, L. A. The SpiNNaker project. *Proc. IEEE* **102**, 652–665 (2014).
7. Davies, M. et al. Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* **38**, 82–99 (2018).
8. Mostafa, H., Müller, L. K. & Indiveri, G. An event-based architecture for solving constraint satisfaction problems. *Nat. Commun.* **6**, 1–10 (2015).
9. Amir, A. et al. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 7388–7397 (IEEE, 2017).
10. Schuman, C. D. et al. A survey of neuromorphic computing and neural networks in hardware. Preprint at https://arxiv.org/abs/1705.06963 (2017).
11. James, C. D. et al. A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications. *Biol. Inspired Cogn. Archit.* **19**, 49–64 (2017).
12. Strukov, D., Indiveri, G., Grollier, J. & Fusi, S. Building brain-inspired computing. *Nat. Commun.* **10**, 4838–2019 (2019).
13. Thakur, C. S. et al. Large-scale neuromorphic spiking array processors: a quest to mimic the brain. *Front. Neurosci.* **12**, 891 (2018).
14. Davies, M. et al. Advancing neuromorphic computing with Loihi: a survey of results and outlook. *Proc. IEEE* **109**, 911–934 (2021).
15. Aimone, J. B. et al. Non-neural network applications for spiking neuromorphic hardware. In *Proc. 3rd International Workshop on Post Moores Era Supercomputing* 24–26 (PMES, 2018).
16. Polykretis, I., Tang, G. & Michmizos, K. P. An astrocyte-modulated neuromorphic central pattern generator for hexapod robot locomotion on intel's Loihi. In *International Conference on Neuromorphic Systems 2020* 1–9 (ACM, 2020).
17. Irizarry-Valle, Y. & Parker, A. C. An astrocyte neuromorphic circuit that influences neuronal phase synchrony. *IEEE Trans. Biomed. circuits Syst.* **9**, 175–187 (2015).
18. Potok, T., Schuman, C., Patton, R. & Li, H. *Neuromorphic Computing, Architectures, Models, and Applications. A Beyond-CMOS Approach to Future Computing* (US Department of Energy, 2016).
19. Yin, S. et al. 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)* 1–5 (IEEE, 2017).
20. Schemmel, J. et al. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)* 1947–1950 (IEEE, 2010).
21. Neftci, E. O., Mostafa, H. & Zenke, F. Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* **36**, 51–63 (2019).
22. Pei, J. et al. Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature* **572**, 106–111 (2019).
23. Merolla, P. A. et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **345**, 668–673 (2014).
24. Moradi, S., Qiao, N., Stefanini, F. & Indiveri, G. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Trans. Biomed. Circuits Syst.* **12**, 106–122 (2017).
25. Benjamin, B. V. et al. Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* **102**, 699–716 (2014).
26. Schemmel, J., Billaudelle, S., Dauer, P. & Weis, J. Accelerated analog neuromorphic computing. Preprint at https://arxiv.org/abs/2003.11996 (2020).
27. Bohnstingl, T., Scherr, F., Pehle, C., Meier, K. & Maass, W. Neuromorphic hardware learns to learn. *Front. Neurosci.* **13**, 483 (2019).
28. Islam, R. et al. Device and materials requirements for neuromorphic computing. *J. Phys. D* **52**, 113001 (2019).
29. Nandakumar, S., Kulkarni, S. R., Babu, A. V. & Rajendran, B. Building brain-inspired computing systems: examining the role of nanoscale devices. *IEEE Nanotechnol. Mag.* **12**, 19–35 (2018).
30. Najem, J. S. et al. Memristive ion channel-doped biomembranes as synaptic mimics. *ACS Nano* **12**, 4702–4711 (2018).
31. Jo, S. H. et al. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* **10**, 1297–1301 (2010).
32. Li, Y., Wang, Z., Midya, R., Xia, Q. & Yang, J. J. Review of memristor devices in neuromorphic computing: materials sciences and device challenges. *J. Phys. D* **51**, 503002 (2018).
33. Wu, Y., Deng, L., Li, G., Zhu, J. & Shi, L. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* **12**, 331 (2018).

34. Kulkarni, S. R. & Rajendran, B. Spiking neural networks for handwritten digit recognition–supervised learning and network optimization. *Neural Netw.* **103**, 118–127 (2018).
35. Anwani, N. & Rajendran, B. Training multi-layer spiking neural networks using normad based spatio-temporal error backpropagation. *Neurocomputing* **380**, 67–77 (2020).
36. Bagheri, A., Simeone, O. & Rajendran, B. Training probabilistic spiking neural networks with first-to-spike decoding. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2986–2990 (IEEE, 2018).
37. Göltz, J. et al. Fast and deep neuromorphic learning with time-to-first-spike coding. Preprint at https://arxiv.org/abs/1912.11443 (2019).
38. Lee, J. H., Delbruck, T. & Pfeiffer, M. Training deep spiking neural networks using backpropagation. *Front. Neurosci.* **10**, 508 (2016).
39. Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G. & Roy, K. Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* https://doi.org/10.3389/fnins.2020.00119 (2020).
40. Zenke, F. & Neftci, E. O. Brain-inspired learning on neuromorphic substrates. *Proc. IEEE* **109**, 935–950 (2021).
41. Cramer, B., Stradmann, Y., Schemmel, J. & Zenke, F. The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* https://doi.org/10.1109/TNNLS.2020.3044364 (2020).
42. Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U. & Neftci, E. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)* 1–8 (IEEE, 2016).
43. Hunsberger, E. & Eliasmith, C. Training spiking deep networks for neuromorphic hardware. Preprint at https://arxiv.org/abs/1611.05141 (2016).
44. Sengupta, A., Ye, Y., Wang, R., Liu, C. & Roy, K. Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* **13**, 95 (2019).
45. Severa, W., Vineyard, C. M., Dellana, R., Verzi, S. J. & Aimone, J. B. Training deep neural networks for binary communication with the whetstone method. *Nat. Mach. Intell.* **1**, 86–94 (2019).
46. Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M. & Liu, S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* **11**, 682 (2017).
47. Stöckl, C. & Maass, W. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nat. Mach. Intell.* **3**, 230–238 (2021).
48. Blouw, P., Choo, X., Hunsberger, E. & Eliasmith, C. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *NICE '19: Proc. 7th Annual Neuro-inspired Computational Elements Workshop* 1–8 (ACM, 2019).
49. Getty, N., Brettin, T., Jin, D., Stevens, R. & Xia, F. Deep medical image analysis with representation learning and neuromorphic computing. *Interface Focus* **11**, 20190122 (2021).
50. Shukla, R., Lipasti, M., Van Essen, B., Moody, A. & Maruyama, N. Remodel: rethinking deep CNN models to detect and count on a neurosynaptic system. *Front. Neurosci.* **13**, 4 (2019).
51. Tanaka, G. et al. Recent advances in physical reservoir computing: a review. *Neural Netw.* **115**, 100–123 (2019).
52. Kudithipudi, D., Saleh, Q., Merkel, C., Thesing, J. & Wysocki, B. Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing. *Front. Neurosci.* **9**, 502 (2016).
53. Du, C. et al. Reservoir computing using dynamic memristors for temporal information processing. *Nat. Commun.* **8**, 1–10 (2017).
54. Wijesinghe, P., Srinivasan, G., Panda, P. & Roy, K. Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines. *Front. Neurosci.* **13**, 504 (2019).
55. Soures, N. & Kudithipudi, D. Deep liquid state machines with neural plasticity for video activity recognition. *Front. Neurosci.* **13**, 686 (2019).
56. Schuman, C. D., Mitchell, J. P., Patton, R. M., Potok, T. E. & Plank, J. S. Evolutionary optimization for neuromorphic systems. In *Proc. Neuro-inspired Computational Elements Workshop* 1–9 (ACM, 2020).
57. Schaffer, J. D. Evolving spiking neural networks for robot sensory-motor decision tasks of varying difficulty. In *Proc. Neuro-inspired Computational Elements Workshop* 1–7 (ACM, 2020).
58. Schliebs, S. & Kasabov, N. Evolving spiking neural network–a survey. *Evol. Syst.* **4**, 87–98 (2013).
59. Plank, J. S. et al. The TENNLab suite of LIDAR-based control applications for recurrent, spiking, neuromorphic systems. In *44th Annual GOMACTech Conference* (GOMAC Tech, 2019); http://neuromorphic.eecs.utk.edu/raw/files/publications/2019-Plank-Gomac.pdf
60. Mitchell, J. P. et al. Neon: neuromorphic control for autonomous robotic navigation. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)* 136–142 (IEEE, 2017).
61. Bi, G.-q & Poo, M.-m Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* **18**, 10464–10472 (1998).
62. Shrestha, A., Ahmed, K., Wang, Y. & Qiu, Q. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 International Joint Conference on Neural Networks (IJCNN)* 1999–2006 (IEEE, 2017).
63. Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A. & Ganjtabesh, M. First-spike-based visual categorization using reward-modulated stdp. *IEEE Trans. Neural Netw. Learn. Syst.* **29**, 6178–6190 (2018).
64. Lee, C., Panda, P., Srinivasan, G. & Roy, K. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* **12**, 435 (2018).
65. Kaiser, J., Mostafa, H. & Neftci, E. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Front. Neurosci.* **14**, 424 (2020).
66. Bellec, G. et al. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* **11**, 1–15 (2020).
67. Martin, E. et al. EqSpike: spike-driven equilibrium propagation for neuromorphic implementations. *iScience* **24**, 102222 (2021).
68. Mukhopadhyay, A. K., Sharma, A., Chakrabarti, I., Basu, A. & Sharad, M. Power-efficient spike sorting scheme using analog spiking neural network classifier. *ACM J. Emerg. Technol. Comput. Syst.* **17**, 1–29 (2021).
69. Nessler, B., Pfeiffer, M., Buesing, L. & Maass, W. Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Comput. Biol.* **9**, e1003037 (2013).
70. Kasabov, N. K. NeuCube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Netw.* **52**, 62–76 (2014).
71. Budhraja, S. et al. Sleep stage classification using neucube on spinnaker: a preliminary study. In *2020 International Joint Conference on Neural Networks (IJCNN)* 1–8 (IEEE, 2020).
72. Kumarasinghe, K., Owen, M., Taylor, D., Kasabov, N. & Kit, C. 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)* 4465–4472 (IEEE, 2018).
73. Izhikevich, E. M. Polychronization: computation with spikes. *Neural Comput.* **18**, 245–282 (2006).
74. Wang, F., Severa, W. M. & Rothganger, F. Acquisition and representation of spatio-temporal signals in polychronizing spiking neural networks. In *Proc. 7th Annual Neuro-inspired Computational Elements Workshop* 1–5 (ACM, 2019).
75. Alemi, A., Machens, C., Deneve, S. & Slotine, J.-J. Learning nonlinear dynamics in efficient, balanced spiking networks using local plasticity rules. In *Proc. AAAI Conference on Artificial Intelligence* Vol. 32 (AAAI, 2018).
76. Maass, W. On the computational power of winner-take-all. *Neural Comput.* **12**, 2519–2535 (2000).
77. Oster, M., Douglas, R. & Liu, S.-C. Computation with spikes in a winner-take-all network. *Neural Comput.* **21**, 2437–2465 (2009).
78. Kappel, D., Nessler, B. & Maass, W. STDP installs in winner-take-all circuits an online approximation to hidden Markov model learning. *PLoS Comput. Biol.* **10**, e1003511 (2014).
79. Gütig, R. & Sompolinsky, H. The tempotron: a neuron that learns spike timing–based decisions. *Nat. Neurosci.* **9**, 420–428 (2006).
80. Bohte, S. M., Kok, J. N. & La Poutre, H. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* **48**, 17–37 (2002).
81. Wang, Q., Rothkopf, C. A. & Triesch, J. A model of human motor sequence learning explains facilitation and interference effects based on spike-timing dependent plasticity. *PLoS Comput. Biol.* **13**, e1005632 (2017).
82. Li, S. & Yu, Q. New efficient multi-spike learning for fast processing and robust learning. In *Proc. AAAI Conference on Artificial Intelligence* Vol. 34, 4650–4657 (AAAI, 2020).
83. Zenke, F. & Ganguli, S. Superspike: supervised learning in multilayer spiking neural networks. *Neural Comput.* **30**, 1514–1541 (2018).
84. Petro, B., Kasabov, N. & Kiss, R. M. Selection and optimization of temporal spike encoding methods for spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **31**, 358–370 (2019).
85. Hamilton, K. E., Mintz, T. M. & Schuman, C. D. Spike-based primitives for graph algorithms. Preprint at https://arxiv.org/abs/1903.10574 (2019).
86. Corder, K., Monaco, J. V. & Vindiola, M. M. Solving vertex cover via ising model on a neuromorphic processor. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5 (IEEE, 2018).
87. Kay, B., Date, P. & Schuman, C. Neuromorphic graph algorithms: extracting longest shortest paths and minimum spanning trees. In *Proc. Neuro-Inspired Computational Elements Workshop* 1–6 (ACM, 2020).
88. Ali, A. & Kwisthout, J. A spiking neural algorithm for the network flow problem. Preprint at https://arxiv.org/abs/1911.13097 (2019).
89. Aimone, J. B. et al. Provable neuromorphic advantages for computing shortest paths. In *Proc. 32nd ACM Symposium on Parallelism in Algorithms and Architectures* 497–499 (ACM, 2020).

90. Hamilton, K., Date, P., Kay, B. & Schuman D, C. Modeling epidemic spread with spike-based models. In *International Conference on Neuromorphic Systems 2020* 1–5 (ACM, 2020).
91. Severa, W., Lehoucq, R., Parekh, O. & Aimone, J. B. Spiking neural algorithms for Markov process random walk. In *2018 International Joint Conference on Neural Networks (IJCNN)* 1–8 (IEEE, 2018).
92. Smith, J. D. et al. Neuromorphic scaling advantages for energy-efficient random walk computations. Preprint at https://arxiv.org/abs/2107.13057 (2021).
93. Cook, M. *Networks of Relations* (California Institute of Technology, 2005).
94. Diehl, P. U. & Cook, M. Learning and inferring relations in cortical networks. Preprint at https://arxiv.org/abs/1608.08267 (2016).
95. Diehl, P. U. & Cook, M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* **9**, 99 (2015).
96. Alom, M. Z., Van Essen, B., Moody, A. T., Widemann, D. P. & Taha, T. M. Quadratic unconstrained binary optimization (QUBO) on neuromorphic computing system. In *2017 International Joint Conference on Neural Networks (IJCNN)* 3922–3929 (IEEE, 2017).
97. Mniszewski, S. M. Graph partitioning as quadratic unconstrained binary optimization (QUBO) on spiking neuromorphic hardware. In *Proc. International Conference on Neuromorphic Systems* 1–5 (ACM, 2019).
98. Yakopcic, C., Rahman, N., Atahary, T., Taha, T. M. & Douglass, S. Solving constraint satisfaction problems using the Loihi spiking neuromorphic processor. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* 1079–1084 (IEEE, 2020).
99. Mostafa, H., Müller, L. K. & Indiveri, G. Rhythmic inhibition allows neural networks to search for maximally consistent states. *Neural Comput.* **27**, 2510–2547 (2015).
100. Fonseca Guerra, G. A. & Furber, S. B. Using stochastic spiking neural networks on SpiNNaker to solve constraint satisfaction problems. *Front. Neurosci.* **11**, 714 (2017).
101. Pecevski, D., Buesing, L. & Maass, W. Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons. *PLoS Comput. Biol.* **7**, e1002294 (2011).
102. Dagum, P. & Luby, M. An optimal approximation algorithm for Bayesian inference. *Artif. Intell.* **93**, 1–27 (1997).
103. Knight, J. C. & Nowotny, T. GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Front. Neurosci.* **12**, 941 (2018).
104. Gewaltig, M.-O. & Diesmann, M. NEST (Neural Simulation Tool). *Scholarpedia* **2**, 1430 (2007).
105. Goodman, D. F. & Brette, R. Brian: a simulator for spiking neural networks in python. *Front. Neuroinform.* **2**, 5 (2008).
106. Bekolay, T. et al. Nengo: a python tool for building large-scale functional brain models. *Front. Neuroinform.* **7**, 48 (2014).
107. Stewart, T. C. *A Technical Overview of the Neural Engineering Framework* (University of Waterloo, 2012).
108. Kulkarni, S. R., Parsa, M., Mitchell, J. P. & Schuman, C. D. Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing* **447**, 145–160 (2021).
109. Vetter, J. S. et al. *Extreme Heterogeneity 2018-Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity* Technical Report (US Department of Energy, 2018).
110. Diamond, A., Nowotny, T. & Schmuker, M. Comparing neuromorphic solutions in action: implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms. *Front. Neurosci.* **9**, 491 (2016).
111. Mishkin, D., Sergievskiy, N. & Matas, J. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vis. Image Underst.* **161**, 11–19 (2017).
112. Orchard, G., Jayawant, A., Cohen, G. K. & Thakor, N. Converting static image datasets to spiking neuromorphic datasets using Saccades. *Front. Neurosci.* **9**, 437 (2015).
113. Tuggener, L., Schmidhuber, J. & Stadelmann, T. Is it enough to optimize CNN architectures on ImageNet? Preprint at https://arxiv.org/abs/2103.09108 (2021).
114. Sandamirskaya, Y. Dynamic neural fields as a step toward cognitive neuromorphic architectures. *Front. Neurosci.* **7**, 276 (2014).
115. Plank, J. S., Zheng, C., Schumann, C. D. & Dean, C. Spiking neuromorphic networks for binary tasks. In *International Conference on Neuromorphic Computing Systems (ICONS)* 1–8 (ACM, 2021).
116. Smith, J. D. et al. Solving a steady-state PDE using spiking networks and neuromorphic hardware. In *International Conference on Neuromorphic Systems 2020* 1–8 (ACM, 2020).
117. Aimone, J. B. A roadmap for reaching the potential of brain-derived computing. *Adv. Intell. Syst.* **3**, 2000191 (2021).
118. Douglas, R., Mahowald, M. & Mead, C. Neuromorphic analogue VLSI. *Annu. Rev. Neurosci.* **18**, 255–281 (1995).
119. Parsa, M. et al. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Front. Neurosci.* **14**, 667 (2020).
120. Parsa, M., Ankit, A., Ziabari, A. & Roy, K. 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)* 1–8 (IEEE, 2019).
121. Parsa, M. et al. Bayesian-based hyperparameter optimization for spiking neuromorphic systems. In *2019 IEEE International Conference on Big Data (Big Data)* 4472–4478 (IEEE, 2019).
122. Indiveri, G. et al. Neuromorphic silicon neuron circuits. *Front. Neurosci.* **5**, 73 (2011).

Acknowledgements
This material is based on work supported by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, R. Pino, program manager, under contract number DE-AC05-00OR22725. We would like to thank N. Armistead for his aid in creating the graphics for this manuscript.

Author contributions
C.D.S. lead the preparation, writing and editing of this manuscript. S.R.K., M.P. and J.P.M. contributed to the neuromorphic hardware description. C.D.S., S.R.K. and M.P. contributed to the machine learning algorithms section and S.R.K., P.D. and B.K. contributed to the non-machine learning section. All authors contributed to the sections on closing the gap between expectations and reality and outlook.

Competing interests
The authors declare no competing interests.

Additional information
Correspondence should be addressed to Catherine D. Schuman.

Peer review information *Nature Computational Science* thanks James Aimone, Giacomo Indiveri and Thomas Nowotny for their contribution to the peer review of this work. Handling editor: Fernando Chirigati, in collaboration with the *Nature Computational Science* team.