A review of non-cognitive applications for neuromorphic computing

NEUROMORPHIC
Computing and Engineering

TOPICAL REVIEW

A review of non-cognitive applications for neuromorphic computing

James B Aimone[1], Prasanna Date[2], Gabriel A Fonseca-Guerra[3], Kathleen E Hamilton[2], Kyle Henke[4,5], Bill Kay[6], Garrett T Kenyon[4], Shruti R Kulkarni[2], Susan M Mniszewski[4], Maryam Parsa[7], Sumedh R Risbud[3], Catherine D Schuman[8,*], William Severa[1] and J Darby Smith[1]

[1] Neural Exploration and Research Laboratory, Sandia National Laboratories, Albuquerque, NM, United States of America
[2] Oak Ridge National Laboratory, Oak Ridge, TN, United States of America
[3] Intel Labs, Intel Corporation, Santa Clara, CA, United States of America
[4] Los Alamos National Laboratory, Los Alamos, NM, United States of America
[5] University of New Mexico, Albuquerque, NM, United States of America
[6] Pacific Northwest National Laboratory, Richland, WA, United States of America
[7] Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, United States of America
[8] Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: cschuman@utk.edu

Keywords: graph algorithms, optimization, spiking neural networks, neuromorphic computing

OPEN ACCESS

RECEIVED
7 April 2022

REVISED
23 June 2022

ACCEPTED FOR PUBLICATION
10 August 2022

PUBLISHED
2 September 2022

Abstract

Though neuromorphic computers have typically targeted applications in machine learning and neuroscience ('cognitive' applications), they have many computational characteristics that are attractive for a wide variety of computational problems. In this work, we review the current state-of-the-art for non-cognitive applications on neuromorphic computers, including simple computational kernels for composition, graph algorithms, constrained optimization, and signal processing. We discuss the advantages of using neuromorphic computers for these different applications, as well as the challenges that still remain. The ultimate goal of this work is to bring awareness to this class of problems for neuromorphic systems to the broader community, particularly to encourage further work in this area and to make sure that these applications are considered in the design of future neuromorphic systems.

1. Introduction

Neuromorphic computers are a non-von Neumann computing technology in which both the architecture and the operation of the computer is inspired by biological brains. Rather than conventional computers, which have processing and memory components, neuromorphic computers are composed of neurons and synapses, both of which perform processing and store some form of data. In this work, we focus on *spiking* neuromorphic computers, where the neural network type that is implemented in the hardware is a spiking neural network (SNN). In SNNs, both the neurons and synapses include time in their operation. In particular, it takes time for information to travel throughout these networks, and the timing of the arrival of events throughout the network influences how the network performs computation. Programming a spiking neuromorphic computer requires defining the appropriate SNN (structure and parameters), as well as how to communicate with the network with spikes, to perform a particular task.

With the rise of the success of artificial intelligence, computing systems that natively implement neural network-style computation such as neuromorphic computers have become increasingly attractive to academic, industry, and government groups who are seeking to implement these operations faster or more efficiently than can be realized in traditional computing systems. Additionally, the need to perform complex, large-scale neuroscience simulations has also driven research into neuromorphic hardware implementations. As such, much of the algorithmic and application work in neuromorphic computing over the last few decades has

been centered around cognitive applications, either for machine learning or for computational neuroscience purposes [1].

A growing area of interest, however, is the use of neuromorphic systems for a broader set of computational tasks. In particular, there has been a recognition that there are a variety of characteristics of neuromorphic computers that are attractive for a broader set of applications. Those characteristics include: inherent massively parallel computation, native scalability, sparsity of activity, event-driven computation, stochastic operations, collocated processing and memory, and very simple communication between components (usually in the form of spikes). By leveraging these characteristics of neuromorphic systems, an array of different types of computational tasks can be mapped onto these systems.

One significant catalyst in the growing attention towards non-cognitive applications of neuromorphic hardware has been the arrival of large-scale spiking architectures, such as Intel Loihi [2, 3], IBM TrueNorth [4, 5], SpiNNaker [6], and BrainScaleS [7]. These platforms not only provided the broader computing community with hardware suitable for algorithm exploration, but they also presented themselves as large-scale parallel architectures that provide many thousands of generally programmable neurons. As we will show, many algorithms can be represented as large graphs of neurons with spiking activity.

Another major catalyst in using neuromorphic systems for non-cognitive applications has been a focus on benchmarking in neuromorphic systems [8]. As new hardware implementations have been developed, it is critical to have benchmark applications that can be used to evaluate how they perform. Non-cognitive applications have been proposed as part of the benchmarking structure because they can evaluate different characteristics of neuromorphic hardware platforms that traditional cognitive applications may not require.

Here, we provide a review of the current state of non-cognitive tasks on neuromorphic computers. For each of these tasks, we provide a discussion of the importance of that particular computational task to the broader field of computing, why neuromorphic computing has or will have an advantage over other types of computational systems on the task, and the challenges that remain to be addressed in mapping that task successfully to neuromorphic systems. The goal of this work is to provide an understanding of the many different ways the native computational power of neuromorphic systems can be leveraged across many tasks in computing. We also hope to inspire others to consider mapping other non-cognitive applications to neuromorphic computers.

2. Graph algorithms

Many large-scale computing problems contain graphical elements or can be represented as connected networks. For example, analyzing and modeling connections in social networks is used to understand how sentiment and information is disseminated throughout communities. Another example is power grids; understanding vulnerable structures is imperative in order to protect against catastrophic failures such as blackouts. Neuromorphic architectures are readily modeled by (edge weighted) directed graphs, with vertices representing neurons, edges representing synapses, and edge weights representing synaptic delay. However, many real world networks are large, containing thousands of vertices. They are not guaranteed to be sparse (in that the number of connections may far exceed the number of vertices) and the connections may be time dependent. Networks at scale, dense networks, and dynamic networks all serve as obstacles to neuromorphic implementations of graph algorithms.

Neuromorphic platforms offer flexible, reprogrammable substrates that can be used for these kinds of analysis tasks. Neuromorphic computing uses spiking neurons and discrete pulses to execute algorithms. These networks of spiking neurons can be generalized as directed graphs where vertices (spiking neurons) are connected by weighted edges (synapses). In this section we discuss applications and algorithms designed to implement graph analysis via spiking neurons. These applications are set apart from the optimization applications discussed in section 3. While optimization problems can be formulated as graphs, when discussing graphical analysis we are focused on computing metric and quantities related to graphical structure and dynamical processes.

2.1. Neuromorphic co-processors for communication routing

One application of graph algorithms on neuromorphic systems is to provide co-processing units to future HPC systems. While graph algorithms do provide some computational tools (such as neuromorphically simulating walks in graphs to perform sparse matrix-vector multiplication [9]), it is unlikely that the ability of GPUs to perform many linear algebraic operations quickly will be off-boarded to neuromorphic processors any time soon. However, under the hood of high performance computing systems there are different network topologies [10] connecting various compute nodes. Each time multiple compute nodes need to pass messages through a network topology there is a bottleneck to parallelism. Optimal message passing protocols in the form of graph routing algorithms can cut down the cost of communication [11]. A neuromorphic co-processor could feasibly be used to compute the shortest path natively in parallel through a computer architecture based on

which nodes are actively being used, minimizing costly communication, or be used to offboard a number of graph algorithmic/combinatorial tasking which occur in the background for high performance computers in a low energy, massively parallel way.

2.2. Network dynamics and graph structure analysis

Neuromorphic platforms can emulate the spiking dynamics of neurons. This behavior can be adapted to analyze network dynamics and graph structure on simple and directed graphs, with weighted or unweighted edges [3, 12–18]. Many graph algorithms have been adapted for neuromorphic platforms, utilizing standard neuron models such as leaky-integrate and fire (LIF), and functionalities such as synaptic plasticity. With a small number of parameters (e.g. firing threshold v_{th}, refractory period t_{R}, synaptic weights w_{ij}), spike patterns can be generated, controlled and decoded in order to extract needed information about the underlying graphical structure.

The types of graph algorithms and network dynamics simulations can be generally sorted into three application groups: applications that characterize static graph properties (e.g. counting triangles, cycles, minimum spanning trees), applications that are used to understand dynamical graph properties (e.g. modeling flows and processes), and applications that are used to understand correlated structures on graphs (e.g. communities). The generation of spike trains may be done by driving single neurons, or groups of neurons. When the external stimulus is applied to a single neuron, the spike trains can be sorted according to a taxonomy that has been used to characterize non-spiking network dynamics based on network flows [19, 20]: radial or directional spread of spike patterns, and whether dynamics are allowed to revisit previously active neurons or synapses. The radial spread of spikes from a driven neuron reaches neurons in a breadth-first search (BFS) manner making neuromorphic architectures particularly useful in problems driven by greedy BFS search. When the external stimulus is applied to multiple neurons, then the relevant spike dynamics can be characterized by correlations and coherent behavior.

2.3. Graph algorithm implementations

Spiking neurons fire isotropically along all outward directed synapses but the synaptic weights will implement radial or directional spread and can be implemented by setting the synaptic weights of all outgoing synapses. If all synaptic connections are equally weighted, this creates radial spike pattern (flow), if certain synapses are suppressed (or strengthened) then this creates a spike pattern along directional flows. The refractory period of each neuron will determine if spikes can return to neurons which have previously fired. This can be useful to create spike trains which only visit unique neurons (in graph parlance, the spike trains spread along non-backtracking walks). Here, we describe some standard graph algorithms which have been implemented on neuromorphic systems.

Given a graph, finding the shortest path between two nodes has many applications in natural language processing, routing, and optimization. Shortest path algorithms have been simulated and deployed natively, and efficiency has been observed both empirically (energy measurements and spike counts) as well as provably [3, 12–14]. Dynamic programming is a branch of computer science where problems are broken into (overlapping) subproblems, and the subproblems are computed and chained together to find a solution to the main problem. Such examples are finding the longest path in a directed, acyclic graph and finding the longest monotonic subsequence. Dynamic programming has a broad spectrum of applications from optimization and operations research. Neuromorphic use-cases for constrained maximization, constraint satisfaction, number partitioning, and longest monotonic subsequence have seen neuromorphic implementations [21]. Given an edge weighted graph, finding a subgraph with no cycles which contains every vertex and has weights as small as possible is known as finding a minimum spanning tree. Many industrial problems (distribution of resources or utilities) can be framed as computing a minimum spanning tree where the edge weights are often represented by physical distances. The Ford–Fulkerson/Edmonds–Karp algorithms implement a greedy, BFS solution to the minimum spanning tree problem and have neuromorphic implementations [13]. Graph coloring is a rich field with applications in scheduling and telecommunication. The questions of whether or not a graph can be colored with two colors so that no adjacent pair receives the same color is closely connected to the cycle structure of the graph. Algorithms that detect cycles, odd cycles, and three cycles all have neuromorphic implementations [15, 22]. Flow networks are directed graphs in which each edge has a capacity, and some flow is assigned to each edge. The flow cannot exceed the capacity, and the flow into a node has to equal the flow out. Liquid through a series of pipes is a classic example of a network flow where the size of the pipes represents capacity and the quantity of liquid represents the flow. However, routing problems and communication among compute nodes are also readily modeled as a flow network. The standard algorithms for finding optimal flows by using augmenting paths can be deployed neuromorphically [15]. Community detection and graph clustering are ubiquitously used to identify similar nodes in large real world networks, and to partition graphs into clusters of nodes that are all highly interconnected. The use cases range from data compression to

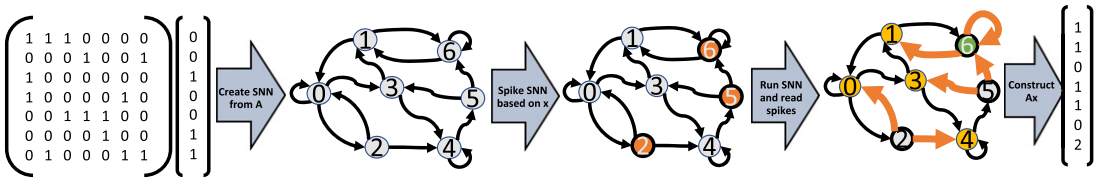

Figure 1. Depiction of neuromorphic walks being used to perform matrix-vector multiplication. Figure adapted from [9].

HPC communication tasks, and neuromorphic systems which analyze common spike patterns amongst neurons to identify communities based on the paradigm that neurons which are 'wired together fire together' have been used [16, 17, 23]. Communities are one important substructure of a graph, but other structures such as central or important vertices along with counts of specific subgraphs (triangles, notably) have neuromorphic implementations [22], and are important in analysis of critical infrastructures and computational chemistry. Neuromorphic (SEIR) models of disease spread have been used in conjunction with real life COVID data [18]. Lastly, neuromorphic walks on graphs have been used as a tool to compute sparse binary matrix-vector multiplication [9] (see figure 1).

2.4. Neuromorphic advantages and remaining challenges for graph algorithms

There has been some research to indicate that neuromorphic systems can offer a theoretical advantage over conventional computers in certain cases. In particular, Aimone *et al* [14] showed that neuromorphic systems provide a provable advantage over conventional algorithms on single-source shortest path problems. Davies *et al* [3] demonstrated an experimental advantage over CPU implementations of Dijkstra's algorithm on Loihi for large graph sizes.

However, there are still some obstacles and challenges for implementing graph algorithms on real world networks, which may contain millions of vertices. We identify two main challenges: embedding large networks and graphs into neuromorphic processors, and reducing computational runtime. Both of these challenges share a common challenge that they require low overhead for embedding graphs into SNN, driving the spike dynamics, and recording and decoding spike trains.

Near-term neuromorphic platforms can contain millions of neurons [24], but due to the sparse synapse network connectivity, implementing arbitrarily dense connections between neurons may not be possible. This presents one obstacle for implementing graph algorithms at scale: analyzing graphs with sizes (number of edges) that exceed the available number of synapses or analyzing graphs with orders (number of vertices) that exceed the number of available neurons on the neuromorphic platform. To address the challenge of graph sizes: one approach would be to use locally sparse implementation [17] or to use synaptic learning (STDP) to learn sparser representations [23]. The challenge of large graph orders can be addressed using distributed sub-routines on subgraphs, which are compiled into a single result using an approach similar to MapReduce [25].

Another challenge is reducing the runtime associated with graph analysis. With the exception of clique detection, the spike-based graph algorithms in [16–18, 22, 23] are implemented with single neuron driving: external stimulus is applied to one neuron of the network in order to initiate spike trains, and these spike trains may need to be generated for long times (e.g. until spike self-terminate). Serial execution of these routines (for example to rank all vertices in a graph according to a given centrality measure) will be inefficient as graph orders increase. The runtime can be reduced by running in parallel with multiple copies of the SNN. However, can these dynamics generate unique spike signals without requiring multiple copies of a SNN—can we use randomization to reduce embedding overhead?

Another obstacle is that analysis of neuromorphic algorithms is as-of-yet ill-defined. This is a particular hurdle in the arena of graph algorithms where a runtime based on number of vertices and edges is the de facto means by which algorithmic efficiency is compared [26]. Traditional algorithmic complexity is measured with respect to how many steps a Turing machine (or machine with traditional von Neumann architecture) can perform a task as a function of the input size [27]. For example, given the number of vertices and edges, how many steps will a computer take to count the number of triangles in a graph? Standard complexity analysis depends only on the size of the input problem and is agnostic to the size of the Turing machine/von Neumann architecture, while the size of a neuromorphic architecture and the number of steps it takes to set up is intrinsically part of a neuromorphic graph algorithm. Some early attempts to formalize neuromorphic complexity theory have been made [28, 29], but a comprehensive complexity analysis framework is still in the offing.

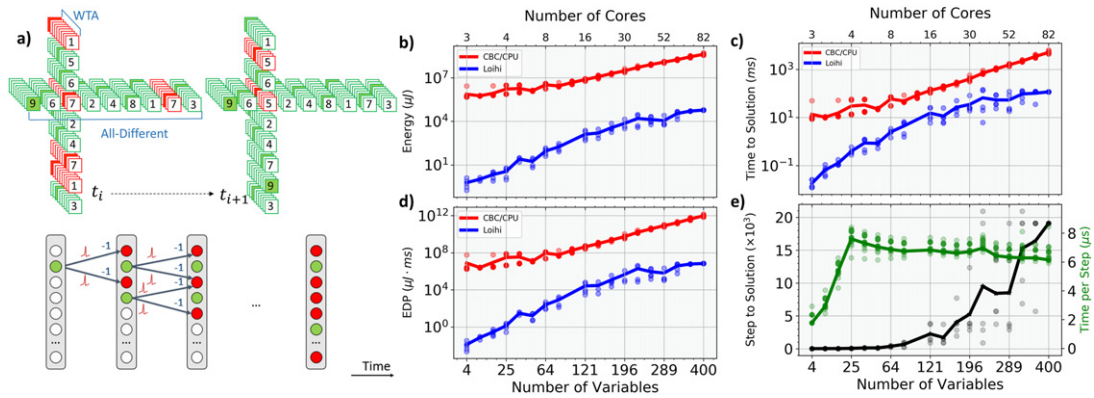

Figure 2. (Performance and energy benchmark results reproduced from [3]. Results may vary.) Benchmark of Loihi 1 against CPU for solving Latin squares of increased size up to 400 cells. (a) Top: problem encoding for a 9 × 9 Latin square. Every cell is represented by a winner-takes-all population of neurons, each neuron corresponds to a possible value of the cell. All-different constraints are encoded as inhibitory synapses among those populations according to the Latin squares uniqueness rules. Bottom: illustration of how spiking dynamics prunes the neurons state space while the stochastic search proceeds. A single sequence of events is shown for visualization purposes, in the solver multiple of these processes happen in parallel and compete to find the lowest-energy state. Subfigures (b)–(d) show the energy, time, and energy-delay-product advantage of Loihi 1 over the CBC solver on CPU. (d) Shows the break-down of time to solution into its components: steps to solution and time per time-step.

3. Constrained optimization

Constrained optimization problems are ubiquitous in real-world industry and academic settings, and despite widespread adoption of deep learning algorithms, non-DL optimization remains at the core of applications as diverse as warehouse logistics, molecular dynamics for drug discovery and robotic control.

Here, we present the state-of-the-art in integer and continuous neuromorphic optimization, expanding on two fundamental derivations of SNN dynamics, one for solving constraint satisfaction problems (CSPs) (neural sampling) and the other for solving a particular flavor of quadratic programming (spiking locally competitive algorithm (LCA) via proximal operator). Principled derivations for solving other optimization problems are desirable but still missing from the published literature. However, neural sampling and spiking LCA offer a basis to believe convergence proofs are possible for a larger family of problems. Despite the value of such formalisms, practical challenges usually follow when implementing and assessing the competitiveness of the neuromorphic solution. We also describe neuromorphic implementations of optimal control theory and Bayesian optimization (BO).

3.1. Optimization in integer domain—constraint satisfaction and QUBO problems

Neural approaches to integer optimization can be traced back as early as the 1980's [30–32]. Here, we focus on actual physical realizations of optimization solvers in event-driven neuromorphic hardware, the earliest of which was, to our knowledge, the solution to a 4 × 4 Sudoku in 2016 with an analog neuromorphic device [33]. Such implementations followed a theoretical framework for probabilistic sampling with networks of spiking neurons [34], and a few hardware proposals for event-driven highly-parallel hardware architectures [35], both of which were demonstrated to solve CSPs [34–36].

3.1.1. Constraint satisfaction

CSPs belong to the NP-complete class of computational complexity theory and are a crucial component of many scientific fields and technological applications. The notorious difficulty of these problems often requires approximation methods or heuristics as the best option for capturing 'good' solutions, because the existence of exact efficient algorithms remains an open question.

All published neuromorphic implementations for solving CSPs use some source of stochasticity. Analog hardware solvers, like those in DYNAP [37], BrainScales [38, 39] and other VLSI chips [33], harness the inherent variability in neural dynamics from fabrication device mismatch and thermal noise affecting analog circuits. Solvers in digital neuromorphic hardware like those in TrueNorth [40–43], SpiNNaker [44] and Loihi [3, 37, 45, 46] use pseudo-random number generators (PRNGs). Despite the remarkable energy efficiency of analog solvers, these struggle to scale up. Since 2016, the largest solved Sudoku remains of size 4 × 4 even when using very large analog systems like BrainScales [47]. Other CSPs that have been solved in analog hardware are the Tower of Hanoi task with three stacks and two discs, and the map coloring problem for the territories of Austria (7 territories) and Germany (16 states) [39]. The first digital neuromorphic solver [44] already solved

Table 1. Graph two-partitioning results.

Graph size(node count)	TrueNorthenergy	TrueNorthpart sizes	D-Waveenergy	D-Wavepart sizes
8	10	4/4	10	3/5
16	15	8/8	17	8/8
16	16	9/7		
16	17	8/8		
30	35	14/16	37	14/16
30	36	15/15		

9 × 9 Sudokus, the world map coloring (193 countries) and 1000-spins systems with further scaling only limited by the unoptimized SpiNNaker compiler. Since then Davies *et al.* [3] demonstrated Loihi solving Latin squares of up to 20 × 20 cells (see figure 2) and the same solver has been further scaled up to 32 × 32 Latin squares (unpublished data) while retaining the gains in performance published in [3]. Besides scaling up, the work in [3] demonstrated for the first time a competitive advantage of a neuromorphic solver over a classical state-of-the-art algorithm running on a modern CPU (10^3–10^5 better energy-delay product (EDP)). All previous works did not benchmark against classical solvers and the reported times to solution were 2–4 orders of magnitude worse than those on [3].

3.1.2. Neural sampling enables efficient state space exploration

Theoretical foundations provided by [36] demonstrated the noise and refractory periods in SNNs can be used as a computation feature for solving difficult CSP problems by allowing the system to reliably sample the complicated non-convex energy landscape by jumping into new random configuration and thus avoiding being trapped in local minima. These findings showed exponential convergence to the underlying probability distribution governing the energy landscape of CSPs and makes them drastically more effective at finding the global minimum compared to Boltzmann machines which are sampling from the same complicated landscape.

3.1.3. Quadratic unconstrained binary optimization (QUBO)

Many NP-hard combinatorial optimization problems can be framed as quadratic unconstrained binary optimization (QUBO) which can be solved classically or on specialized hardware such as the D-Wave quantum annealer. The following QUBO objective function is minimized

$$O(\mathbf{Q}, \mathbf{x}) = \sum_i Q_{ii} x_i + \sum_{i<j} Q_{ij} x_i x_j. \tag{1}$$

The Q matrix is symmetric with size $n \times n$ where Q_{ii} and Q_{ij} are the linear coefficients and quadratic coefficients that encode the problem Hamiltonian matrix. The problem variables $x_i \in \{0, 1\}$ encode the result values. The objective function can become a maximization problem by negating the Q matrix.

In [43], the QUBO formulation is mapped to the IBM TrueNorth neuromorphic hardware. The problem variables are represented by three sets of neurons—working neurons with stochastic leak, spontaneous stochastic neurons, and output neurons. Correspondingly, three sets of axons provide the inputs and weights between neurons—the initial guess, reinforcement feedback from the working neurons, and the spontaneous stochastic inputs. Driven by a simulated annealing (SA) metaheuristic, leakage or decay added to each neuron and the integrate-and-fire dynamics are used to search the input sampling space till convergence. The firing neurons provide feedback as input to neurons related by weights helping steer the search.

Spiking stochastic noise as a computational resource allowed for the exploration of the space of configurations as shown in [36, 44] for solving CSPs. This noise is the mechanism for exploring the larger sampling space (by adding or removing variables) which relies on the spontaneous stochastic leak neurons and stochastic leak/decay. The probability of firing or leak represents the temperature element of SA. The temperature initially starts out at a high value and is slowly reduced to a low value, based on an annealing schedule.

The same mathematical formulation developed for the D-Wave quantum annealing approach [48] is used for the TrueNorth neuromorphic hardware [43]. Due to the spiking nature of the TrueNorth a negated version of the QUBO matrix is used to solve for the highest energy solution (or maximization).

Graph partitioning is defined as splitting the vertices of a graph into k equal or near equal parts. The number of cut edges between the parts is minimized. The graph two-partitioning example is demonstrated in [48]. The QUBO includes penalty terms for creating balanced parts and minimizing the number of cut edges.

Results are shown in table 1, comparing partitioning of small graphs run on the TrueNorth and the D-wave, with size 8, 16, and 30. Energies and part sizes produced by both architectures are comparable.

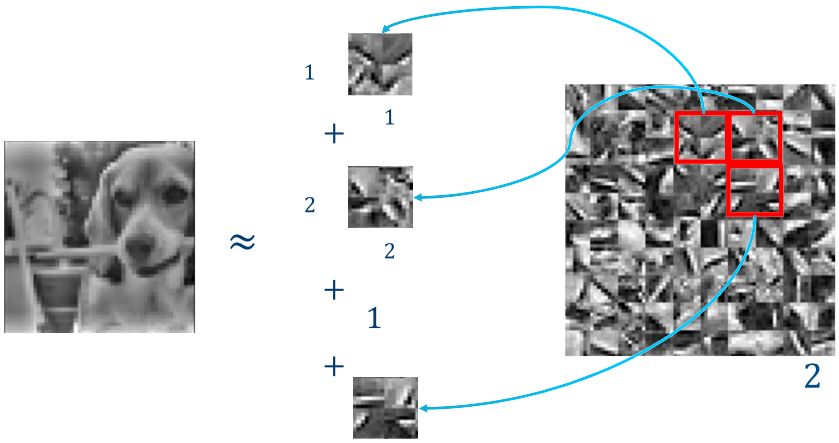

Figure 3. Sparse coding of an input image x using a dictionary $\mathbf{\Phi}$ of features ϕ_j. The number of features in $\mathbf{\Phi}$ is much bigger than the dimensions of the input image. The sparse code is the linear combination of features with coefficients a_j, most of which are zeros (sparsity of vector $\mathbf{a}$).

Future plans include implementation on other neuromorphic architectures, such as Intel Loihi. This QUBO approach on neuromorphic hardware is not limited to graph algorithms, but opens the door to solving a spectrum of NP-hard optimization problems.

New, unpublished work has made a similar comparison for solving QUBO problems with Loihi, classical and quantum annealing devices. Finding the lowest energy state of a QUBO is equivalent to settling into the lowest energy state of a Hamiltonian which governs a system of particles in a quantum system, an important scientific task. Preliminary results suggest SNNs can compete with current quantum annealing devices, which are designed to explicitly solve these problems, in terms of solution quality, but do so with less energy consumption and with the ability to scale up to larger problems than can fit on to existing quantum processors.

3.2. Optimization in continuous domain—sparse coding: neuromorphic solution of the LASSO problem

3.2.1. Locally competitive algorithm

The problem of sparse coding is encountered in the context of compressed sensing *inter alia* [49, 50]. The aim of sparse coding is to represent a dense input signal as a linear combination of a few features drawn from a high dimensional dictionary of features (i.e., an over-complete dictionary, see figure 3). In other words, sparse coding projects the input signal to a high dimensional space, such that very few coefficients in this high dimensional representation are non-zero.

Suppose the dense input is represented by $\mathbf{x}$, the over-complete dictionary is represented by $\mathbf{\Phi}$, and the sparse coefficients are represented by $\mathbf{a}$. In this notation, $\|\mathbf{x} - \mathbf{\Phi} \cdot \mathbf{a}\|_2$ is the ℓ_2-norm of the reconstruction error. Using the constraint that the reconstruction error should be zero, there are several ways in which sparse coding is posed as an optimization problem. Enforcing sparsity in $\mathbf{a}$ by minimizing its number of non-zero components translates to using the ℓ_0-norm to formulate the optimization problem [50]:

$$\min \|\mathbf{a}\|_0, \ \text{such that} \|\mathbf{x} - \mathbf{\Phi} \cdot \mathbf{a}\|_2 = 0.$$

It is known that the above optimization problem is NP-hard [51]. To make this problem tractable, ℓ_1-norm is used to enforce sparsity in $\mathbf{a}$:

$$\min \|\mathbf{a}\|_1, \ \text{such that} \|\mathbf{x} - \mathbf{\Phi} \cdot \mathbf{a}\|_2 = 0.$$

In the augmented Lagrangian form, the objective function for the above problem becomes the LASSO objective [52]:

$$\Omega(\mathbf{a}) = \underbrace{\frac{1}{2}\|\mathbf{x} - \mathbf{\Phi} \cdot \mathbf{a}\|_2^2}_{f(\mathbf{a})} + \underbrace{\lambda \|\mathbf{a}\|_1}_{g(\mathbf{a})}. \tag{2}$$

Then the sparse code is a solution of the optimization problem:

$$\mathbf{a}^* = \arg\min_{\mathbf{a}} \Omega(\mathbf{a}).$$

Rozell *et al* [50] proposed LCA, the first neuromorphic algorithm to minimize the LASSO objective (equation (2)) through temporal dynamics of a neural network:

$$\dot{\mathbf{u}}(t) = \frac{1}{\tau}\left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{x} - \mathbf{u}(t) - \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} - \mathbf{I}\right)\mathbf{a}(t)\right), \tag{3}$$
$$\mathbf{a}(t) = \mathcal{T}(\mathbf{u}(t); \lambda).$$

In equation (3), vector **u** represents the state of neurons in the network (e.g., their membrane voltage), which follows leaky integrator dynamics with a decay time-constant τ. The neurons are driven using a bias obtained by linearly projecting the dense input to the feature space $\mathbf{\Phi}^{\mathrm{T}}\mathbf{x}$. The neurons 'fire' with an activation **a** when their states cross a threshold λ according to the soft thresholding function $\mathcal{T}(\cdot)$. This activation is communicated between neurons through purely inhibitory weights given by $\left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} - \mathbf{I}\right)$. At steady state, the neural activity $\mathbf{a}_{\mathrm{ss}}$ corresponds to the minimizer of the LASSO objective function in equation (2).

3.2.2. LCA and ISTA are the same

Conventionally, the LASSO problem is solved using homotopy methods like least angle regression [53] or proximal gradient-based methods like iterative shrinkage thresholding algorithm (ISTA) [54], fast-ISTA/FISTA [55], etc.

When minimizing LASSO objective (equation (2)), kth iteration of the ISTA is given by [54, 55],

$$\begin{aligned}\mathbf{a}^{(k+1)} &= \mathcal{T}\left(\mathbf{a}^{(k)} - \nabla f(\mathbf{a}^{(k)}); \lambda\right),\\ &= \mathcal{T}\left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{x} - \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} - \mathbf{I}\right)\mathbf{a}^{(k)}; \lambda\right) \quad \ldots \quad \nabla f \text{ from equation (2).}\end{aligned}$$

If we define an auxiliary vector $\mathbf{u}^{(k+1)} = \left[-\mathbf{\Phi}^{\mathrm{T}}\mathbf{x} - \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} - \mathbf{I}\right)\mathbf{a}^{(k)}\right]$, then we get,

$$\begin{aligned}\mathbf{u}^{(k+1)} - \mathbf{u}^{(k)} &= -\mathbf{u}^{(k)} + \mathbf{\Phi}^{\mathrm{T}}\mathbf{x} - \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} - \mathbf{I}\right)\mathbf{a}^{(k)}, \text{ and}\\ \mathbf{a}^{(k+1)} &= \mathcal{T}\left(\mathbf{u}^{(k+1)}; \lambda\right).\end{aligned} \tag{4}$$

The dynamics in equation (4) is just a time-discretized version of continuous dynamics in equation (3), thus showing that neuromorphic LCA and ISTA are mathematically equivalent.

3.2.3. Spiking LCA

A spiking version of the locally competitive algorithm (spiking LCA) was derived by Tang *et al* [56]. In this work, they showed that a network of LIF neurons with binary spiking activation can perform sparse coding using LCA, such that the spiking rates of the LIF neurons are equivalent to the analog firing rates encoded by the activation vector **a** in equation (3), in the long time limit. In particular, they showed that if spiking response of pth LIF neuron in a network is modeled as a Dirac comb in time:

$$\sigma_p(t) = \sum_j \delta\left(t - t_j\right),$$

then the activation vector is

$$\mathbf{a}(t) = \frac{1}{t - t_0}\int_{t_0}^{t} \mathrm{d}s\, \boldsymbol{\sigma}(s).$$

Using the binary spiking formalism, the spiking LCA was implemented on Intel's Loihi chip and it was shown that the Loihi chip can generate a good approximate solution about an order of magnitude faster than FISTA for small problems [2]. Extending this work further, spiking LCA has been thoroughly benchmarked against FISTA for dimensionality of the sparse code spanning from a few hundred to ∼2 million in a recent work by Davies *et al* [3] (see figure 4).

3.3. Optimal control theory

Another interesting set of optimization problems targeted with neuromorphic computing are 'optimal control theory'. One approach is an iterative spiking adaptive dynamic programming (SADP) method to solve optimal impulsive control problems [57]. In many dynamic systems such as mathematical biology, information science, and engineering control, an impulsive behavior is observed where a sudden jump occurs at an instant during the dynamic process. This sudden jump significantly influences the performance and stability of the dynamic systems [58, 59]. In [58] a switching control strategy is used to analyze stability, and controllability of the system. In [59] a Hopfield neural network is used to define the global stability. For non-linear systems, non-analytical and approximate solutions are required as the hybrid Bellman equation is generally analytically unsolvable [60–62]. Adaptive dynamic programming (ADP) combines the advantages of dynamic programming, reinforcement learning, and function approximation in order to solve optimal control problems [63–65]. These traditional ADP approaches cannot solve impulse optimal control problems since they do not consider the impulse interval and amplitude [57]. Modified ADP techniques are introduced in the literature

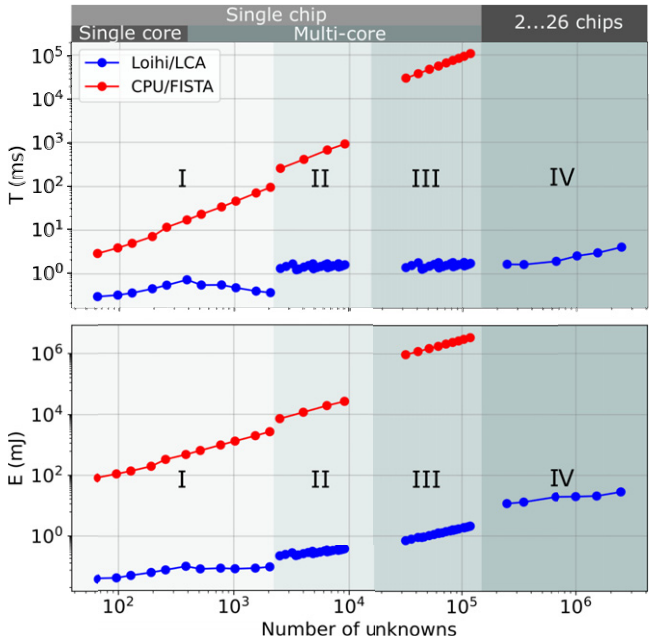

Figure 4. (Performance and energy benchmark results reproduced from [3]. Results may vary.) Comparison between FISTA algorithm running on CPU and spiking LCA running on Loihi: time-to-solution (top) and dynamic energy consumption (bottom). The shaded regions correspond to different regimes of problem sizes: region I corresponds to small LASSO problems (16×16 pixel input image, and dictionaries consisting of 32 to 1024 elements), regions II and III correspond to medium LASSO problems (24×24 and 76×76 pixel inputs, respectively, and dictionaries of 50–180 elements), region IV corresponds to large LASSO problems (input sizes between 130×130 and 300×300 pixels with dictionaries containing 120–250 elements). The problem sizes analyzed in region IV span multiple Loihi chips and are too large to benchmark on a CPU (up to 2 million unknowns).

to address this issue [66, 67]. An alternative approach for discrete-time nonlinear systems is to take inspiration from biology and solve the optimal impulse control with spike train (SADP) [57]. This SADP method is developed based on combining Poisson process and the maximum likelihood estimation and computing the three-tuple of state, spiking interval, and probability of Poisson distribution. The two-step SADP iterative process is summarized below.

Step 1. Compute the three-tuples $(x_k, \tau_k, p_{\tau_k})$**.** For given input, data is traversed to a single channel of spike trains with a fixed time interval of T. Interspike interval τ_k, and the firing rate λ_k is then calculated. The average firing rate is $\bar{\lambda} = \frac{1}{n}\sum_{k=1}^{n}\lambda_k$, and the probability p_{τ_k} in $[kT, (k+1)T$ is $p_{\tau_k} = ((\bar{\lambda}T)^{\tau_k}/\tau_k!)\exp(-(\bar{\lambda}T))$

Step 2. SADP algorithm based on Poisson process. For an initial state x_0, computation precision ϵ, and an arbitrary positive semi-definite function $\Psi(x)$, we obtain the three-tuple given in step 1, and compute iterative spiking control law $\nu_i(x_k)$ as

$$\nu_i(x_k) = \arg\min_{\nu_{k+\tau_k}} \left\{ U_{\tau_k}(x_k, \nu_{k+\tau_k}) + \sum_{j\in\Omega_x} p(j|x_k, \tau_k) V_i(j) \right\} \tag{5}$$

Finally the iterative spiking value function $V_{i+1}(x_k)$ can be computed as

$$V_{i+1}(x_k) = \min_{\nu_{k+\tau_k}} \left\{ U_{\tau_k}(x_k, \nu_{k+\tau_k}) + \sum_{j\in\Omega_x} p(j|x_k, \tau_k) V_i(j) \right\} \tag{6}$$

If $|V_{i+1}(x_k) - V_i(x_k) \leqslant \epsilon, \forall\, x_k \in \Omega_x$, then the optimal spiking control law is obtained, otherwise the iterative process will be continued [57].

3.4. Bayesian optimization

BO has been used to optimize performance of neuromorphic systems in terms of accuracy, energy efficiency, area efficiency, and latency [68, 69]. BO is well-suited for optimization problems where the objective function is unknown and/or expensive to evaluate. Equation (7) shows a problem of finding a global optimizer for an unknown objective function

$$x^* = \arg\max_{x\in X} f(x). \tag{7}$$

In equation (7), X is the entire design space, and f is the black-box objective function without simple closed form. As summarized by [70, 71], the best location x_{n+1} to observe y_{n+1} point is iteratively searched over X (the entire design space) in order to estimate f. After N iterations, the algorithm suggests the best estimation of the black-box function f. This sequential optimization is based on building a prior estimation over possible performance metrics, f (such as accuracy, latency, ...), and then iteratively re-estimating the prior model using the observations from updating the Bayesian posterior model. The posterior representations are the updated belief on the objective function(s) f we are optimizing. The design space, X, is explored and exploited by leveraging the inherent uncertainty of the posterior model and mathematically calculating a surrogate model, called the acquisition function α_n. The maximum point of the acquisition function α_n is the next candidate point to observe (x_{n+1}) and guides the search direction toward the true representation of the objective function f. The efficiency of the BO to estimate the global optimizer for the f with fewer evaluations lies on the ability of the Bayesian technique to learn from prior and posterior distributions on the problem and direct the observations by trading off exploration and exploitation of the design space X.

Instead of using BO to optimize neuromorphic systems, a new and unpublished work is aiming at using neuromorphic processors like Loihi to solve a BO problem. In this work, an event-driven acquisition function α_n is estimated iteratively enabling us to run BO on a neuromorphic processor efficiently with fewer evaluations of unknown objective function f.

3.5. Neuromorphic advantages and remaining challenges for constraint satisfaction

When compared with conventional computing architectures solving constrained optimization, the neuromorphic approach has shown orders-of-magnitude gains in EDP required to find a solution [3]. For example, Loihi neuromorphic platform has shown 10^3–10^6 EDP improvements over state-of-the-art conventional algorithms [3] in both convex (e.g., quadratic programming) as well as combinatorial (e.g., integer programming) problems. In the case of the latter (i.e., combinatorial problems), the EDP gains can be attributed to NP-complete/NP-hard complexity of the problems, which constrains the scalability and improvement in performance of von Neumann computing architectures. In the case of convex optimization problems like quadratic programming, conventional algorithms sometimes face parallelization bottlenecks, which are mitigated in the neuromorphic approach by the built-in fine-grained parallelism. In both cases, chips like Loihi have shown lower energy consumption compared to a CPU or GPU, when it is possible to run the same (or equivalent) algorithm on these architectures. The two main components for this advantage are the matching graph structure between problem encoding and the compute architecture and the temporal sparsity of the event-driven spike-based computation implementing the search algorithms (most of the power consumption in neuromorphic computers is due to spikes transfer across the communication interconnect).

The most relevant challenges to make neuromorphic optimization solvers broadly applicable in industry and academic settings are:

- Accuracy of solvers in continuous domain (e.g., quadratic programming solvers) is limited by the low-precision fixed-point architecture of digital neuromorphic hardware.
- Generalization of the current neuromorphic approach to solve combinatorial problems requires a rigorous theoretical foundation, such that hyper-parameter tuning can be performed in a principled manner for robustness of the solutions.
- Solving mixed integer problems presents algorithmic and theoretical challenges, especially when rigorous convergence proofs are desirable.

Therefore, in the context of constrained optimization problems, neuromorphic computing has the potential to carve out a niche and solve problems that require low latency, low power while being tolerant to low precision. For example, a control algorithm for a low-power robot interacting with the real-world through discrete actuators might be a better fit for a neuromorphic substrate, whereas a conventional CPU could be the ideal platform for a linear programming problem, where high precision representation is critical.

4. Random walks and partial differential equations solving

Stochastic spiking behavior is observed in biological neural systems, and neuromorphic computing hardware often has corresponding stochastic components. As examples, SpiNNaker allows for user-defined random draws; Intel Loihi and IBM TrueNorth both have PRNGs distributed throughout their chip layouts. The abundance of PRNGs allow for stochastic neuron updates, usually either within the membrane potential or in the integration of new spikes.

The massively parallel computation afforded by spiking neuromorphic systems combined with the availability of PRNGs has motivated research into neuromorphic random walk algorithms [72, 73]. Random walks, in this context, can be thought of as a time-series process where a point/walker moves among nodes on a

grid or graph, changing locations between discrete time steps with the probability of the next location of the point/walker depending only on the current location. This is also known as a discrete-time Markov chain. These neuromorphic random walk algorithms implement a random walk process by configuring neurons' stochastic behavior to reflect the intended distribution of transition between states.

Random walks can approximate a wide variety of stochastic processes, including those used in modeling neuronal activity [74], molecular motor protein motion [75], mathematical epidemiology [76], finance [77, 78], and Boltzmann transport [79]. Additionally, these random walks are useful in other Markov chain algorithms, including algorithms in industrial systems engineering [80] and recent work in modeling the spread of COVID-19 [81]. Regardless of the specific application, random walk algorithms on traditional processors are most commonly used to estimate integrals. However, challenging branching conditions in some domains (e.g. Boltzmann transport) lead to difficulties in parallelization. These challenges may be effectively overcome by implementing a massively parallel random walk on a neuromorphic platform.

4.1. Density-based random walks

For a random walk on a graph, a density-based random walk neuromorphic algorithm represents a graph node (rather than a walker) with a sub-circuit of neurons. With this method, walkers are represented by certain neurons' potential, and walkers transition from node to node by updating the corresponding potential. Functionally, the graph nodes represent various states or locations for a walker to assume. Spikes are treated as walkers and transition among the various nodes by means of a mutually exclusive probability draw [72]. This approach highlights the benefits of bespoke neuromorphic approaches as it exemplifies an algorithm that is uncharacteristic for traditional processors but efficient for neuromorphic processors.

4.2. Markov chains and PIDE solutions

Consider the initial value problem

$$\begin{aligned}\frac{\partial}{\partial t}u(t,x) &= \frac{1}{2}a^2(t,x)\frac{\partial^2}{\partial x^2}u(t,x) + b(t,x)\frac{\partial}{\partial x}u(t,x)\\ &\quad + \lambda(t,x)\int \big(u\big(t,x+h\big(t,x,q\big)\big) - u(t,x)\big)\phi_Q\big(q;t,x\big)\mathrm{d}q + c(t,x)u(t,x) + f(t,x),\\ u(0,x) &= g(x).\end{aligned} \tag{8}$$

With some mild conditions on the functions $a, b, h, \phi_Q, \lambda, c, f$, and g, if a solution to the initial value problem exists, then the solution may be written as

$$\begin{aligned}u(t,x) &= \mathbb{E}\left[\exp\left(\int_0^t c(s,X(s))\mathrm{d}s\right)g(X(t))\middle|X(0)=x\right]\\ &\quad + \mathbb{E}\left[\int_0^t f(s,X(s))\exp\left(\int_0^s c(\ell,X(\ell))\mathrm{d}\ell\right)\mathrm{d}s\middle|X(0)=x\right],\end{aligned} \tag{9}$$

where

$$\mathrm{d}X(t) = b(t,X(t))\mathrm{d}t + a(t,X(t))\mathrm{d}W(t) + h\big(t,X(t),q\big)\mathrm{d}P(t;Q,X(t),t). \tag{10}$$

Equation (10) is a stochastic differential equation describing the position of X over time. The process X has a drift term given by b and a diffusion term given by a. The process $W(t)$ is a mean-zero random process with variance t, called a Wiener process. X also may experience a non-local diffusion. This yields a diffusion increment according to h and occurs according to a Poisson process P with rate λ (that is, $\mathbb{E}\big[\mathrm{d}P(t;Q,X(t),t)|X(0)=x\big] = \lambda(t,x)\mathrm{d}t$). The jump-amplitude mark random variable for the Poisson process is Q and its probability density function (pdf) is given by ϕ_Q. Equation (9) is a probabilistic representation of the solution to the initial value problem, written as an expectation of this stochastic process X. The top line of the equation states that particles must start at the location x and contribute to the overall solution whenever they enter the support of the function g and that contribution is weighted according to some function c. If $c < 0$, then the exponential term could be interpreted as the survival probability of a particle reaching the support of g by time t. The second line allows the particles to contribute at a rate of f for the amount of time they remain in the support of f, again weighted by some function of c. These results are derived from such tools as Dynkin's formula and the Feynman–Kac formula. They can be extended to higher dimensional PDEs and boundary value problems, see [78, 82, 83].

A Markov chain approximating the process (10) can be created by first discretizing the equation using an appropriate scheme, such as Euler–Maruyama. This provides a pdf over a series of time bins. Then, the discrete states of the chain can be chosen by discretizing space. A transition matrix can be created by integrating the pdfs over these space bins (see [84]). In general, (10) translates nicely to such an interpretation. There is

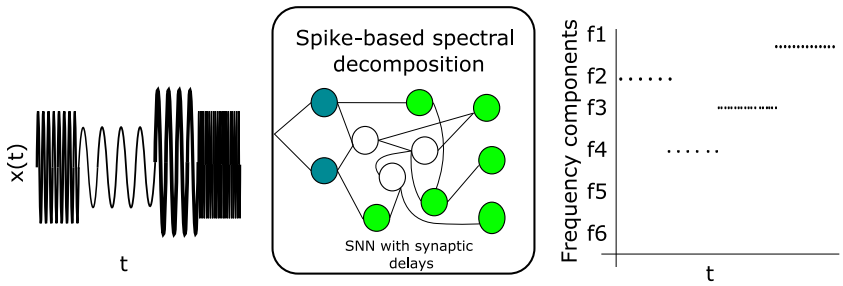

Figure 5. A possible way to determine the spectral components in the incoming analog signal with neuromorphic computing. Here, the light green colored circles represent the output neurons, and each correspond to different frequencies. The spike rate can be thought of as a representation of the power spectral density of each component in the signal.

some deterministic movement, given by the drift b. This is altered by some diffusion b. Finally, there is some probability aligning with a Poisson process P that an additional non-local jump h is added.

These Markov chains are well suited for the neuromorphic random walk implementation. As previously described, each spike is treated as a random walker, and the mutually exclusive probability draw is designed to reflect the transition matrix of the Markov chain. Samples drawn on neuromorphic can then be averaged according to (9) to obtain a solution to the PDE (8). Applications solved in this manner using neuromorphic samples include steady-state and time-dependent equations, and Boltzmann transport problems [85, 86].

4.3. Neuromorphic advantages and remaining challenges for random walks

Neuromorphic technologies are still new and it is reasonable to ask whether or not the stochastic process sampling accomplished by the graph embedded DTMC provides statistically reasonable data. In [84], it is shown that Loihi approximates a solution to a heat equation on a sphere as if it were sampling the underlying Markov chain with seven-bit limited transition probabilities. This suggests that in addition to approximation and application specific error accrued when constructing a DTMC from a stochastic process, there may be additional hardware and algorithmic specific error introduced.

Addressing whether or not the samples gathered from the density algorithm deployed on Loihi are statistically accurate [85], develops a methodology for comparing samples to the expected distribution by using measures of relative entropy as a hypothesis test. Using a common stochastic process as a test case, they find that even with the approximations needed in the algorithm and the hardware limited transition probabilities, that the samples generated sufficiently approximate the expected distribution.

5. Signal processing

Signal processing techniques are widely used in various sensory and signal acquisition devices and play an important role in several application domains such as audio, video and/or image, biomedical signal analysis, etc. These techniques rely on domain knowledge and are essential in extracting the relevant features from the input that would be used in the decision making process either via a machine learning based inference engine or to train a system to make the correct predictions. Signal processing computations, especially those involving spectral feature extractions are typically carried out on a dedicated digital signal processor (DSP), FPGAs, or a traditional CPU. Several neuromorphic solutions involving audio or image processing use a pre-processed input from acquired sensory signals and employ SNNs to generate the relevant output through neuromorphic machine learning algorithms.

5.1. Current neuromorphic approaches for signal processing

Several groups have proposed the idea of carrying out signal filtering based on its spectral information, a key aspect in most digital signal processing algorithms, using binary spikes. Various filters such as low pass, high pass and band-pass filters have been constructed and demonstrated in SNNs using spike rate encoding schemes [87–91]. Grzesiack and Meganck, have developed a mathematical formulation for carrying out spike signal processing and employ it in control systems and linear dynamical systems [89]. Some of the core signal processing algorithms such as Fourier transforms, have been demonstrated with SNN, and have been employed as a pre-processing step in object detection applications [92]. Orchard *et al*, have demonstrated performing equivalent of short time Fourier transforms, with resonate and fire (RF) neuron models, where different output neurons become active for different frequency components in the input signal [93]. Figure 5 shows a possible way to identify different spectral components in the incoming analog signal with spikes.

5.2. Signal processing application domains and opportunities

Different application domains have demonstrated spike based signal processing approaches. In the biomedical field, detection of different frequency components is an essential step in diagnosing various disorders. Sharifshazileh *et al*, designed an SNN on a low power neuromorphic hardware DYNAP-SE to detect high frequency oscillations in the brain to identify epileptic seizures [94]. Few works have also explored spike signal processing from the point of view of novel materials' research, e.g., Ganguly *et al*, have explored the use of a stochastic analog neuron based on spintronic materials in signal processing tasks such as channel equalization [95]. Neuromorphic approaches are also increasingly being used in assisted and autonomous driving applications to pre-process signals acquired from RADAR and LiDAR sensors [92, 96–98]. A great opportunity exists to take inspiration from the signal processing mechanisms in biology such as echolocation to design neuromorphic systems that are highly performant and accurate. Shalumov *et al*, have employed different bio-inspired models of spiking neurons [98]. Vogginger *et al*, have successfully employed RF neurons with temporal coding to carry out the Fourier transform of the radar signal to discern the location and speed of the target object [96]. SNNs have also been applied in processing the LiDAR data in real-time for object detection and collision avoidance in autonomous driving applications [97, 98]. Wang *et al*, achieved superior performance with SNNs for object detection task compared to deep learning models on the KITTI dataset (laser scanned real-world images taken while driving), transformed to an equivalent version of LiDAR sensor data [97, 99].

5.3. Neuromorphic advantages and remaining challenges for signal processing

The field of signal processing is often tied with the underlying hardware and sensory circuits, right from sampling the analog signal to encoding and computing the relevant features for a particular application. Such edge scenarios have low power budget and memory availability, and neuromorphic computing with its sparse event-driven computation and energy efficient hardware shows a promising way to design signal processing systems. Several research groups have also developed efficient sensory signal acquisition hardware systems mimicking the human cochlea, visual, tactile, and olfactory mechanisms as detailed in this review [100–102].

One of the key applications where neuromorphic signal processing can show benefits is in audio processing at the edge. There have been several demonstrations ranging from keyword spotting on Loihi and SpiNNaker platforms, on pre-processed audio signals using mel frequency cepstral coefficients [103, 104]. Some demonstrations have made use of the output from neuromorphic auditory sensor to carry out speech recognition with SNNs [87]. With an exponentially increasing number of smart devices being commercialised, designing an end-to-end neuromorphic audio processing system presents with unique opportunities. For example, using digital hardware for different stages of processing, employing hardware platforms which use the underlying physics of the materials (e.g. memristive technology-based platforms [95]), etc, are some of the key research areas in the field of neuromorphic computing. With the use of neuromorphic computing to realize some of these commonly used signal processing techniques there would be a significant reduction in the data communication costs between pre-processing on a DSP hardware and cognitive processing on a neuromorphic hardware. Hence, there is a great opportunity to explore neuromorphic computing for signal processing both from the perspective of building energy-efficient edge hardware and discovering novel computing motifs for processing temporal data with SNNs.

6. Composite algorithms through utility and numerical kernels

Functional programming and more recently computational graph methods have shown strong benefits for compositional approaches. In many ways mirroring traditional computing, there exists a new body of work investigating the composition of neuromorphic-compatible algorithms. However there are a wide range of approaches used in the neuromorphic field and linking any two arbitrary algorithms may be difficult. Many of the existing approaches have adopted specific conventions designed to be interoperable, but identifying more generally supported composite techniques is still an open question. Current demonstrations focus on simple logic operations (e.g., AND, OR, NOT) and numerical operations (e.g., maximum), which are critical to allow for composition of output to perform more complex algorithm operations. Though the extension to more sophisticated algorithms is certainly possible.

Unlike some other of the areas discussed, composite algorithms are not motivated by a single factor. Though common motivations include:

- Smaller, basic operations are easier to define, build and interpret.
- A composition of algorithms may avoid costly off-chip communication.
- Composable algorithms help contribute to a community of capability.

6.1. Binary operations, arithmetic, and encoding for composition

Plank *et al* [105] has demonstrated a variety of hand-constructed networks to perform binary operations such as AND, OR, and NOT using a variety of encoding schemes, including direct, population coding, rate coding, and temporal coding, as well as a variety of decoding schemes, including voting, rate coding, and temporal decoding. They constructed small networks that would perform conversions from one encoding scheme to another. With simple kernels such as these, multiple different networks performing a variety of tasks can be composed together onto a single neuromorphic chip, without relying on external, non-neuromorphic systems to perform conversions or compositions of outputs. Aimone *et al* [106] showed how to perform binary arithmetic using spiking neuromorphic implementations. The operations that they demonstrated included streaming adders, inversion, inequality, minimum and maximum, and subtraction, and scalar multiplication. Moreover, several of these operations were demonstrated on both Loihi and TrueNorth.

6.2. Numerical kernels

Lagorce and Benosman demonstrated that by leveraging precising spike timing, synaptic diversity, and temporal delay in neuromorphic systems, a complete computation framework could be realized [107]. They specifically realized values through the times between subsequent spikes, and they demonstrated networks that could realize different numerical calculations, including minimum, maximum, subtraction, linear combination, logarithm, exponential, multiplication, and integration. They demonstrated that using compositions of these approaches, they could implement linear and non-linear differential equations.

6.3. Neuromorphic advantages and remaining challenges for utility and numerical kernels

Traditional processors are very performant at these types of operations, and it is unlikely that a neuromorphic implementation would improve like-for-like. However, these operations may become critical in avoiding expensive off-chip communication. That is, if a neuromorphic process can perform these operations on chip, then less communication overhead is required to send results off of the neuromorphic chip for these calculations.

These small circuits tend to be easy to build, but it is difficult to know which functions will be useful and efficient. Modifications may be required for neuromorphic hardware deployment. Moreover, there remain challenges in deciding the correct level of granularity. Lastly, these circuits tend to be highly dependent on how values are represented. For example, the same function will require different implementations if using spike interval times versus binary representations.

7. Software and tools for numerical neuromorphic algorithms

Despite the growing theoretical evidence that neuromorphic computing can be used for non-cognitive applications, the mere existence of algorithms and hardware is not enough for such applications to proliferate in the community. A significant barrier is the availability of user-friendly programming paradigms and software to unlock the potential of neuromorphic computing for non-cognitive applications. Neural machine learning methods, such as deep learning, have been able to develop a broad user-base in large part due to the availability of a high quality software ecosystem to develop and deploy solutions. While these tools have continued to evolve (Caffe, Theano, Tensorflow, Keras, PyTorch, etc.), their presence has allowed a separation of algorithm development from hardware-specific optimizations.

While we can envision that these deep learning software tools can eventually compile onto neuromorphic hardware (see for example [108, 109]), most likely other tools will be required to develop the algorithms described here. What would constitute an effective framework for such applications remains an open research question on which the developers of a neuromorphic software framework like Lava are actively working (see below) [110].

The Fugu [111] is a software framework proposed to enable the progamming of complex numerical spiking neural algorithms. Fugu treats spiking algorithms as composable graphs of neurons. The graph algorithms outlined in section 2 and the mesh-based algorithms from section 4 benefit greatly from such graph-based circuit construction.

While Fugu generates intricate circuits from the bottom-up, other tools are likely more effective for circuits which are not as precisely defined, such as using populations of spiking neurons to provide statistical inference. These approaches to spiking neural algorithms are likely more naturally a fit to programming models originally developed for computational neuroscience. For instance, the PyNN programming model which is the entry-point to BrainScaleS and SpiNNaker originated as a tool for modeling biological neural circuits and can be readily used to define algorithms comprised of populations of spiking neurons. Similarly, tools such as Brian and GeNN offer similar population programmability.

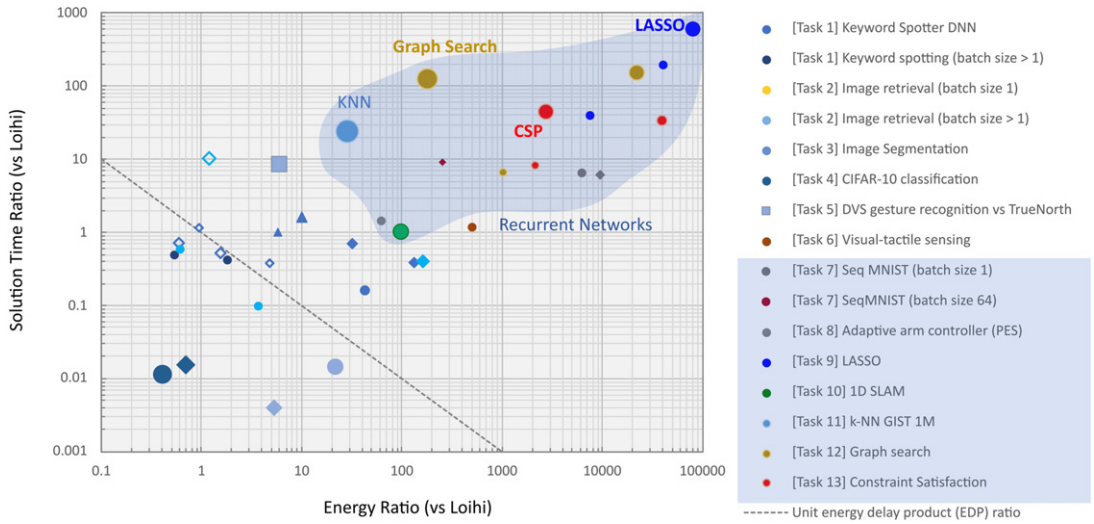

Figure 6. (Performance and energy benchmark results reproduced from [3]. Results may vary.) A comparison of variety of algorithms running on Loihi 1 vs classical state-of-the-art algorithms running on von Neumann architectures. These results demonstrate that neuromorphic graph and optimization algorithms based on recurrent SNNs offer the greatest energy-delay advantage, while several feed-forward DL applications perform equally or worse in Loihi 1. Modified from [3].

Another notable example is the neuromorphic simulator 'Nengo', which provides a software ecosystem that allows for the development of neural network solutions (both neuromorphic and conventional) inspired by the cognitive models in the brain, using Python [112]. At the core of Nengo is the neural engineering framework, which maps computational models needed for integrators, oscillators, etc. onto neuronal motifs [113]. Nengo ecosystem provides a convenient graphical front-end to a suite of different applications and supports various hardware platforms ranging from neuromorphic hardware (TrueNorth, Loihi, SpiNNaker) to GPUs [109, 114, 115]. There have been a few non-cognitive applications demonstrated with the use of Nengo, such as sparse distributed memory based on associative memory models [116], and bio-plausible solutions to the problem of semantic associations of ontology and inference [117].

Finally, Lava is a community-developed software framework launched by Intel Corporation in 2021 to address several of the limitations of current neuromorphic programming models [118]. Lava is inspired by the formal programming paradigm of *communicating sequential processes*, which considers computing primitives called *processes* that communicate with event-based message passing via interconnecting channels. Such a paradigm is compatible with the event-driven and finely-granular, parallel architectures of spiking neuromorphic hardware, while satisfying the locality constraint of their computing units (i.e., neurons). In Lava, the processes are fully generic and decoupled from their behavioural models [110]. A process is defined by its internal state variables as well as input and output ports allowing it to connect with other processes. In addition, the processes can be hierarchical, composed of other interconnected processes contained within them. A behavioural model for a process dictates how its internal states (or sub-processes within) are updated as function of time, how it consumes the asynchronous inputs received on the ports, and what messages, if any, are generated and communicated through its output ports. The decoupling of the behavioural model from the definition of the process enables Lava's processes to be executable on several hardware platforms, through a separate behavioural model written for each platform. This addresses one of the most important issues of software development for neuromorphic hardware, because it enables all components of the system to be programmed from a single coherent framework. Any algorithm defined through the time-dependent dynamics of nodes and communication between them via message passing, can be supported by Lava. Due to features like python interface, modular design, reusable behavioural models for processes, etc. we expect Lava's user experience to enable faster learning and diverse levels of programming across multiple platforms (CPUs, GPUs, neuromorphic chips). Although the framework is not confined to a particular hardware architecture or platform, the communicating sequential processes paradigm makes it inherently better suited for neuromorphic platforms. Intel's Loihi chips (Loihi 1 and Loihi 2) comprise the first of such neuromorphic platforms to be supported with Lava. As an open-source framework, the developers of non-cognitive applications for neuromorphic hardware will shape Lava in the future to meet their needs.

In the context of non-cognitive applications, the lava-optimization library is especially noteworthy. It is being developed as a suite of optimization solvers (linear and quadratic programming, constraint satisfaction, QUBO, etc), which can be used in a variety of applications.

8. Discussion and conclusion

Here, we have provided an overview of the current state of the art with respect to non-cognitive applications of neuromorphic systems. It is clear from these works that in many cases, there are clear advantages to using a neuromorphic system for certain tasks. For example, neuromorphic systems have been shown to outperform CPU implementations on tasks such as Dijkstra's algorithm and optimization (see figure 6). However, it is also clear that there are still a wide array of challenges in realizing these types of applications on neuromorphic systems.

It is important to note that spiking neuromorphic computers are often compared against neural hardware systems (i.e., those that realize traditional artificial neural network computation), particularly for cognitive applications. In this work, however, we demonstrate that there are a wide array of potential applications for spiking neuromorphic systems *beyond* just cognitive applications. Therefore, in a future compute landscape, particularly where a neuromorphic or neural processor may be included in a future heterogeneous compute system, this work demonstrates that neuromorphic implementations can potentially have utility across a wide array of applications.

It is also worth noting that for many of these applications, there are likely custom or application-specific hardware implementations that can outperform neuromorphic systems on these tasks. That is to say, neuromorphic systems are likely not the *most* performant of all possible computer architectures for these tasks. However, it is clear that in a compute environment where a diversity of applications are required, neuromorphic systems can take on a variety of roles, from cognitive applications to the many applications that have been described here.

One of the goals of this work is to inspire others to consider non-cognitive applications when they consider how neuromorphic systems will be used in the future. In particular, the design of future neuromorphic systems, from hardware to software, should take into account that neuromorphic systems are not likely to be used in a singular, cognitive way. Instead, we should allow these new types of applications to inform the design of our future neuromorphic systems alongside cognitive applications.

Acknowledgments

This material is based upon work supported in part 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. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under Award Number DE-SC0022566. This material is based upon work supported in part by the US Department of Energy Advanced Simulation and Computing program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy (Contract No. 89233218NCA000001). This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the view of the U.S. Department of Energy or the United States Government. This manuscript has been partially authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan). This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Data availability statement

No new data were created or analysed in this study.

ORCID iDs

James B Aimone https://orcid.org/0000-0002-7361-253X
Gabriel A Fonseca-Guerra https://orcid.org/0000-0001-5403-4634
Shruti R Kulkarni https://orcid.org/0000-0001-6894-9851
Susan M Mniszewski https://orcid.org/0000-0002-0077-0537
Sumedh R Risbud https://orcid.org/0000-0003-4777-1139
Catherine D Schuman https://orcid.org/0000-0002-4264-8097
William Severa https://orcid.org/0000-0002-8740-220X
J Darby Smith https://orcid.org/0000-0002-3646-0868

References

[1] Schuman C D, Potok T E, Patton R M, Birdwell J D, Dean M E, Rose G S and Plank J S 2017 A survey of neuromorphic computing and neural networks in hardware (arXiv:1705.06963)
[2] Davies M *et al* 2018 Loihi: a neuromorphic manycore processor with on-chip learning *IEEE Micro* **38** 82–99
[3] Davies M, Wild A, Orchard G, Sandamirskaya Y, Guerra G A F, Joshi P, Plank P and Risbud S R 2021 Advancing neuromorphic computing with Loihi: a survey of results and outlook *Proc. IEEE* **109** 911–34
[4] Merolla P, Arthur J, Filipp A, Imam N, Manohar R and Modha D S 2011 A digital neurosynaptic core using embedded crossbar memory with 45 pJ per spike in 45 nm *Proc. 2011 IEEE Custom Integrated Circuits Conf. (CICC)*
[5] DeBole M V *et al* 2019 TrueNorth: accelerating from zero to 64 million neurons in 10 years *Computer* **52** 20–9
[6] Sugiarto I, Liu G, Davidson S, Plana L A and Furber S B 2016 High performance computing on SpiNNaker neuromorphic platform: a case study for energy efficient image processing *2016 IEEE 35th Int. Performance Computing and Communications Conf. (IPCCC)* pp 1–8
[7] Grübl A, Billaudelle S, Cramer B, Karasenko V and Schemmel J 2020 Verification and design methods for the BrainScaleS neuromorphic hardware system *J. Signal Process. Syst.* **92** 1277–92
[8] Davies M 2019 Benchmarks for progress in neuromorphic computing *Nat. Mach. Intell.* **1** 386–8
[9] Schuman C D, Kay B, Date P, Kannan R, Sao P and Potok T E 2021 Sparse binary matrix-vector multiplication on neuromorphic computers *2021 IEEE Int. Parallel and Distributed Processing Symp. Workshops (IPDPSW)* (IEEE) pp 308–11
[10] Meador B 2008 *A Survey of Computer Network Topology and Analysis Examples* Washington University
[11] Gropp W, Gropp W D, Lusk E, Skjellum A and Lusk A D F E E 1999 *Using MPI: Portable Parallel Programming with the Message-Passing Interface* vol 1 (Cambridge, MA: MIT Press)
[12] Schuman C D, Hamilton K, Mintz T, Adnan M M, Ku B W, Lim S-K and Rose G S 2019 Shortest path and neighborhood subgraph extraction on a spiking memristive neuromorphic implementation *Proc. 7th Annual Neuro-Inspired Computational Elements Workshop* pp 1–6
[13] Kay B, Date P and Schuman C 2020 Neuromorphic graph algorithms: extracting longest shortest paths and minimum spanning trees *Proc. Neuro-Inspired Computational Elements Workshop* pp 1–6
[14] Aimone J B, Ho Y, Parekh O, Phillips C A, Pinar A, Severa W and Wang Y 2020 Provable neuromorphic advantages for computing shortest paths *Proc. 32nd ACM Symp. Parallelism in Algorithms and Architectures* pp 497–9
[15] Kay B, Schuman C, O'Connor J, Date P and Potok T 2021 Neuromorphic graph algorithms: cycle detection, odd cycle detection, and max flow *2021 Int. Conf. Neuromorphic Systems* pp 1–7
[16] Hamilton K E, Imam N and Humble T S 2017 Community detection with spiking neural networks for neuromorphic hardware *Proc. Neuromorphic Computing Symp.* pp 1–8
[17] Hamilton K E, Imam N and Humble T S 2018 Sparse hardware embedding of spiking neuron systems for community detection *ACM J. Emerg. Technol. Comput. Syst.* **14** 1–13
[18] Hamilton K, Date P, Kay B and Schuman C D 2020 Modeling epidemic spread with spike-based models *2020 Int. Conf. Neuromorphic Systems* pp 1–5
[19] Borgatti S P 2005 Centrality and network flow *Soc. Netw.* **27** 55–71
[20] Borgatti S P and Everett M G 2006 A graph-theoretic perspective on centrality *Soc. Netw.* **28** 466–84
[21] Aimone J B, Parekh O, Phillips C A, Pinar A, Severa W and Xu H 2019 Dynamic programming with spiking neural computing *Proc. Int. Conf. Neuromorphic Systems* pp 1–9
[22] Hamilton K, Mintz T, Date P and Schuman C D 2020 Spike-based graph centrality measures *2020 Int. Conf. Neuromorphic Systems* pp 1–8
[23] Hamilton K E and Schuman C D 2018 Towards adaptive spiking label propagation *Proc. Int. Conf. Neuromorphic Systems* pp 1–8
[24] Merolla P A *et al* 2014 A million spiking-neuron integrated circuit with a scalable communication network and interface *Science* **345** 668–73
[25] Dean J and Ghemawat S 2008 MapReduce: simplified data processing on large clusters *Commun. ACM* **51** 107–13
[26] Even S 2011 *Graph Algorithms* (Cambridge: Cambridge University Press)
[27] Arora S and Barak B 2009 *Computational Complexity: A Modern Approach* (Cambridge: Cambridge University Press)
[28] Date P, Kay B, Schuman C, Patton R and Potok T 2021 Computational complexity of neuromorphic algorithms *2021 Int. Conf. Neuromorphic Systems* pp 1–7
[29] Date P, Schuman C, Kay B and Potok T 2021 Neuromorphic computing is turing-complete (arXiv:2104.13983)
[30] Hopfield J J 1982 Neural networks and physical systems with emergent collective computational abilities *Proc. Natl Acad. Sci. USA* **79** 2554–8

[31] Hinton G E and Sejnowski T J 1983 Optimal perceptual inference *Proc. IEEE Conf. Computer Vision and Pattern Recognition* vol 448 (Citeseer) pp 448–53
[32] Hopfield J J and Tank D W 1985 'Neural' computation of decisions in optimization problems *Biol. Cybern.* **52** 141–52
[33] Binas J, Indiveri G and Pfeiffer M 2016 Spiking analog VLSI neuron assemblies as constraint satisfaction problem solvers *2016 IEEE Int. Symp. Circuits and Systems (ISCAS)* (IEEE) pp 2094–7
[34] Habenschuss S, Jonke Z and Maass W 2013 Stochastic computations in cortical microcircuit models *PLoS Comput. Biol.* **9** e1003311
[35] Mostafa H, Müller L K and Indiveri G 2015 An event-based architecture for solving constraint satisfaction problems *Nat. Commun.* **6** 8941
[36] Jonke Z, Habenschuss S and Maass W 2016 Solving constraint satisfaction problems with networks of spiking neurons *Front. Neurosci.* **10** 118
[37] Yakopcic C, Rahman N, Atahary T, Taha T M, Beigh A and Douglass S 2019 High speed cognitive domain ontologies for asset allocation using Loihi spiking neurons *2019 Int. Joint Conf. Neural Networks (IJCNN)* (IEEE) pp 1–8
[38] Kugele A and Meier K 2018 Solving the constraint satisfaction problem Sudoku on neuromorphic hardware *Masters Thesis* Heidelberg University
[39] Steidel J 2018 Solving map coloring problems on analog neuromorphic hardware *Bachelor Thesis* Kirchoff Institute for Physics, Heidelberg University
[40] Alom M Z, Van Essen B, Moody A T, Widemann D P and Taha T M 2017 Quadratic unconstrained binary optimization (QUBO) on neuromorphic computing system *2017 Int. Joint Conf. Neural Networks (IJCNN)* (IEEE) pp 3922–9
[41] Rahman N, Atahary T, Taha T and Douglass S 2017 A pattern matching approach to map cognitive domain ontologies to the IBM TrueNorth neurosynaptic system *2017 Cognitive Communications for Aerospace Applications Workshop (CCAA)* (IEEE) pp 1–4
[42] Corder K, Monaco J V and Vindiola M M 2018 Solving vertex cover via Ising model on a neuromorphic processor *2018 IEEE Int. Symp. Circuits and Systems (ISCAS)* (IEEE) pp 1–5
[43] Mniszewski S M 2019 Graph partitioning as quadratic unconstrained binary optimization (QUBO) on spiking neuromorphic hardware *Proc Int. Conf. Neuromorphic Systems (ICONS '19)* (ACM) pp 1–5
[44] Fonseca Guerra G A and Furber S B 2017 Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems *Front. Neurosci.* **11** 714
[45] Yakopcic C, Rahman N, Atahary T, Taha T M and Douglass S 2020 Leveraging the manycore architecture of the Loihi spiking processor to perform quasi-complete constraint satisfaction *2020 Int. Joint Conf. Neural Networks (IJCNN)* (IEEE) pp 1–8
[46] Yakopcic C, Rahman N, Atahary T, Taha T M and Douglass S 2020 Solving constraint satisfaction problems using the Loihi spiking neuromorphic processor *2020 Design, Automation & Test in Europe Conf. & Exhibition (DATE)* (IEEE) pp 1079–84
[47] Ostrau C, Klarhorst C, Thies M and Rückert U 2019 Comparing neuromorphic systems by solving Sudoku problems *2019 Int. Conf. High Performance Computing & Simulation (HPCS)* (IEEE) pp 521–7
[48] Ushijima H, Negre C F A and Mniszewski S M 2017 Graph partitioning using quantum annealing on the D-wave system *Proc. 2nd Int. Workshop Post-Moore's Era Supercomputing (PMES)* (ACM) pp 22–9
[49] Elad M 2010 *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing* vol 2 (Berlin: Springer)
[50] Rozell C J, Johnson D H, Baraniuk R G and Olshausen B A 2008 Sparse coding via thresholding and local competition in neural circuits *Neural Comput.* **20** 2526–63
[51] Natarajan B K 1995 Sparse approximate solutions to linear systems *SIAM J. Comput.* **24** 227–34
[52] Tibshirani R 1996 Regression shrinkage and selection via the LASSO *J. R. Stat. Soc.* B **58** 267–88
[53] Efron B, Hastie T, Johnstone I and Tibshirani R 2004 Least angle regression *Ann. Stat.* **32** 407–99
[54] Daubechies I, Defrise M and De Mol C 2004 An iterative thresholding algorithm for linear inverse problems with a sparsity constraint *Commun. Pure Appl. Math.* **57** 1413–57
[55] Beck A and Teboulle M 2009 A fast iterative shrinkage-thresholding algorithm for linear inverse problems *SIAM J. Imaging Sci.* **2** 183–202
[56] Tang P T P, Lin T-H and Davies M 2017 Sparse coding by spiking neural networks: convergence theory and computational results (arXiv:1705.05475)
[57] Wei Q, Han L and Zhang T 2021 Spiking adaptive dynamic programming based on Poisson process for discrete-time nonlinear systems *IEEE Trans. Neural Netw. Learn. Syst.* **33** 1846–56
[58] Yao J, Guan Z-H, Chen G and Ho D W C 2006 Stability, robust stabilization and control of singular-impulsive systems via switching control *Syst. Control Lett.* **55** 879–86
[59] Zhang X, Li C and Huang T 2017 Hybrid impulsive and switching Hopfield neural networks with state-dependent impulses *Neural Netw.* **93** 176–84
[60] Fan B, Yang Q, Tang X and Sun Y 2018 Robust ADP design for continuous-time nonlinear systems with output constraints *IEEE Trans. Neural Netw. Learn. Syst.* **29** 2127–38
[61] Huang X, Khalil H K and Song Y 2018 Regulation of nonminimum-phase nonlinear systems using slow integrators and high-gain feedback *IEEE Trans. Autom. Control* **64** 640–53
[62] Zhao K, Song Y and Shen Z 2016 Neuroadaptive fault-tolerant control of nonlinear systems under output constraints and actuation faults *IEEE Trans. Neural Netw. Learn. Syst.* **29** 286–98
[63] Wei Q, Li H, Yang X and He H 2020 Continuous-time distributed policy iteration for multicontroller nonlinear systems *IEEE Trans. Cybern.* **51** 2372–83
[64] Wei Q, Wang L, Liu Y and Polycarpou M M 2020 Optimal elevator group control via deep asynchronous actor–critic learning *IEEE Trans. Neural Netw. Learn. Syst.* **31** 5245–56
[65] Chen C, Modares H, Xie K, Lewis F L, Wan Y and Xie S 2019 Reinforcement learning-based adaptive optimal exponential tracking control of linear systems with unknown dynamics *IEEE Trans. Autom. Control* **64** 4423–38
[66] Heydari A 2020 Optimal impulsive control using adaptive dynamic programming and its application in spacecraft rendezvous *IEEE Trans. Neural Netw. Learn. Syst.* **32** 4544–52
[67] Wang D, Ha M and Qiao J 2019 Self-learning optimal regulation for discrete-time nonlinear systems under event-driven formulation *IEEE Trans. Autom. Control* **65** 1272–9
[68] Parsa M, Mitchell J P, Schuman C D, Patton R M, Potok T E and Roy K 2020 Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design *Front. Neurosci.* **14** 667

[69] Parsa M *et al* 2021 Accurate and accelerated neuromorphic network design leveraging a Bayesian hyperparameter pareto optimization approach *2021 Int. Conf. Neuromorphic Systems* pp 1–8
[70] Shahriari B, Swersky K, Wang Z, Adams R P and De Freitas N 2015 Taking the human out of the loop: a review of Bayesian optimization *Proc. IEEE* **104** 148–75
[71] Parsa M, Ankit A, Ziabari A and Roy K 2019 PABO: pseudo agent-based multi-objective Bayesian hyperparameter optimization for efficient neural accelerator design *2019 IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)* pp 1–8
[72] Severa W, Lehoucq R, Parekh O and Aimone J B 2018 Spiking neural algorithms for Markov process random walk *2018 Int. Joint Conf. Neural Networks (IJCNN)* (IEEE) pp 1–8
[73] Iaroshenko O and Sornborger A T 2021 Binary operations on neuromorphic hardware with application to linear algebraic operations and stochastic equations (arXiv:2103.09198)
[74] Ricciardi L M and Sacerdote L 1979 The Ornstein–Uhlenbeck process as a model for neuronal activity *Biol. Cybern.* **35** 1–9
[75] Smith J D and McKinley S A 2018 Assessing the impact of electrostatic drag on processive molecular motor transport *Bull. Math. Biol.* **80** 2088–123
[76] Wang W, Cai Y, Ding Z and Gui Z 2018 A stochastic differential equation SIS epidemic model incorporating Ornstein–Uhlenbeck process *Physica* A **509** 921–36
[77] Nicolato E and Venardos E 2003 Option pricing in stochastic volatility models of the Ornstein–Uhlenbeck type *Math. Finance* **13** 445–66
[78] Bossy M and Champagnat N 2010 Markov processes and parabolic partial differential equations *Encyclopedia of Quantitative Finance* (New York: Wiley) pp 1142–59
[79] Dupree S A and Fraley S K 2002 *A Monte Carlo Primer: A Practical Approach to Radiation Transport* (Berlin: Springer)
[80] Jensen P A and Bard J F 2003 *Operations Research Models and Methods* (New York: Wiley)
[81] Saez M, Tobias A, Varga D and Barceló M A 2020 Effectiveness of the measures to flatten the epidemic curve of COVID-19. The case of Spain *Sci. Total Environ.* **727** 138761
[82] Grigoriu M 2013 *Stochastic Calculus: Applications in Science and Engineering* (Berlin: Springer)
[83] Hanson F B 2007 *Applied Stochastic Processes and Control for Jump-Diffusions: Modeling, Analysis and Computation* (Philadelphia, PA: SIAM)
[84] Smith J D, Hill A J, Reeder L E, Franke B C, Lehoucq R B, Parekh O, Severa W and Aimone J B 2022 Neuromorphic scaling advantages for energy-efficient random walk computations *Nat. Electron.* **5** 102–12
[85] Aimone J B, Lehoucq R, Severa W and Smith J D 2021 Assessing a neuromorphic platform for use in scientific stochastic sampling *2021 Int. Conf. Rebooting Computing (ICRC)* (IEEE)
[86] Smith J D, Severa W, Hill A J, Reeder L, Franke B, Lehoucq R B, Parekh O D and Aimone J B 2020 Solving a steady-state PDE using spiking networks and neuromorphic hardware *2020 Int. Conf. Neuromorphic Systems 2020* pp 1–8
[87] Dominguez-Morales J P, Liu Q, James R, Gutierrez-Galan D, Jimenez-Fernandez A, Davidson S and Furber S 2018 Deep spiking neural network model for time-variant signals classification: a real-time speech recognition approach *2018 Int. Joint Conf. Neural Networks (IJCNN)* (IEEE) pp 1–8
[88] Jiménez-Fernández A, Cerezuela-Escudero E, Miró-Amarante L, Domínguez-Morales M J, de Asís Gómez-Rodríguez F, Linares-Barranco A and Jiménez-Moreno G 2016 A binaural neuromorphic auditory sensor for FPGA: a spike signal processing approach *IEEE Trans. Neural Networks Learn. Syst.* **28** 804–18
[89] Grzesiak L M and Meganck V 2018 Spiking signal processing: principle and applications in control system *Neurocomputing* **308** 31–48
[90] Domínguez-Morales M, Jimenez-Fernandez A, Cerezuela-Escudero E, Paz-Vicente R, Linares-Barranco A and Jimenez G 2011 On the designing of spikes band-pass filters for FPGA *Int. Conf. Artificial Neural Networks* (Springer) pp 389–96
[91] Severa W, Parekh O, Carlson K D, James C D and Aimone J B 2016 Spiking network algorithms for scientific computing *2016 IEEE Int. Conf. Rebooting Computing (ICRC)* (IEEE) pp 1–8
[92] López-Randulfe J, Duswald T, Bing Z and Knoll A 2021 Spiking neural network for Fourier transform and object detection for automotive radar *Front. Neurorob.* **15** 69
[93] Orchard G, Frady E P, Rubin D B D, Sanborn S, Shrestha S B, Sommer F T and Davies M 2021 Efficient neuromorphic signal processing with Loihi 2 *2021 IEEE Workshop Signal Processing Systems (SiPS)* (IEEE) pp 254–9
[94] Sharifshazileh M, Burelo K, Sarnthein J and Indiveri G 2021 An electronic neuromorphic system for real-time detection of high frequency oscillations (HFO) in intracranial EEG *Nat. Commun.* **12** 3095
[95] Ganguly S, Camsari K Y and Ghosh A W 2021 Analog signal processing using stochastic magnets *IEEE Access* **9** 92640–50
[96] Vogginger B *et al* 2022 Automotive radar processing with spiking neural networks: concepts and challenges *Front. Neurosci.* **16** 851774
[97] Wang W, Zhou S, Li J, Li X, Yuan J and Jin Z 2020 Temporal pulses driven spiking neural network for fast object recognition in autonomous driving (arXiv:2001.09220)
[98] Shalumov A, Halaly R and Tsur E E 2021 LiDAR-driven spiking neural network for collision avoidance in autonomous driving *Bioinsp. Biomim.* **16** 066016
[99] Geiger A, Lenz P and Urtasun R 2012 Are we ready for autonomous driving? The KITTI vision benchmark suite *2012 IEEE Conf. Computer Vision and Pattern Recognition* (IEEE) pp 3354–61
[100] Vanarse A, Osseiran A and Rassau A 2016 A review of current neuromorphic approaches for vision, auditory, and olfactory sensors *Front. Neurosci.* **10** 115
[101] Wu C, Kim T W, Park J H, Koo B, Sung S, Shao J, Zhang C and Wang Z L 2019 Self-powered tactile sensor with learning and memory *ACS Nano* **14** 1390–8
[102] Etienne-Cummings R and Van der Spiegel J 1996 Neuromorphic vision sensors *Sensors Actuators* A **56** 19–29
[103] Blouw P, Choo X, Hunsberger E and Eliasmith C 2019 Benchmarking keyword spotting efficiency on neuromorphic hardware *Proc. 7th Annual Neuro-Inspired Computational Elements Workshop* pp 1–8
[104] Yan Y *et al* 2021 Comparing loihi with a spinnaker 2 prototype on low-latency keyword spotting and adaptive robotic control *Neuromorphic Comput. Eng.* **1** 014002
[105] Plank J, Zheng C, Schuman C and Dean C 2021 Spiking neuromorphic networks for binary tasks *2021 Int. Conf. Neuromorphic Systems* pp 1–9
[106] Aimone J B, Hill A J, Severa W M and Vineyard C M 2021 Spiking neural streaming binary arthimetic *IEEE Int. Conf. Rebooting Computing*

[107] Lagorce X and Benosman R 2015 Stick: spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony *Neural Comput.* **27** 2261–317
[108] Rueckauer B, Bybee C, Goettsche R, Singh Y, Mishra J and Wild A 2022 NXTF: an api and compiler for deep spiking neural networks on Intel Loihi *ACM J. Emerg. Technol. Comput. Syst.* **18** 1–22
[109] DeWolf T, Jaworski P and Eliasmith C 2020 Nengo and low-power AI hardware for robust, embedded neurorobotics *Front. Neurorob.* **73** 568359
[110] Lava 2021 A software framework for neuromorphic computing https://lava-nc.org/
[111] Aimone J B, Severa W and Vineyard C M 2019 Composing neural algorithms with FUGU *Proc. Int. Conf. Neuromorphic Systems* pp 1–8
[112] Bekolay T, Bergstra J, Hunsberger E, DeWolf T, Stewart T C, Rasmussen D, Choo X, Voelker A R and Eliasmith C 2014 Nengo: a Python tool for building large-scale functional brain models *Front. Neuroinf.* **7** 48
[113] Stewart T C 2012 *A technical overview of the neural engineering framework* vol **110** University of Waterloo
[114] Fischl K D, Andreou A G, Stewart T C and Fair K 2018 Implementation of the neural engineering framework on the TrueNorth neurosynaptic system *2018 IEEE Biomedical Circuits and Systems Conf. (BioCAS)* (IEEE) pp 1–4
[115] Mundy A, Knight J, Stewart T C and Furber S 2015 An efficient SpiNNaker implementation of the neural engineering framework *2015 Int. Joint Conf.Neural Networks (IJCNN)* (IEEE) pp 1–8
[116] Ajwani R D, Lalan A, Bhattacharya B S and Bose J 2021 Sparse distributed memory using spiking neural networks on Nengo (arXiv:2109.03111)
[117] Mercier C, Chateau-Laurent H, Alexandre F and Viéville T 2021 Ontology as neuronal-space manifold: towards symbolic and numerical artificial embedding *KRHCAI 2021 Workshop on Knowledge Representation for Hybrid & Compositional AI@ KR2021*
[118] Intel Corporation 2021 *Taking Neuromorphic Computing to the Next Level with Loihi 2* https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf