INVITED PAPER

# Brain-Inspired Learning on Neuromorphic Substrates

*This article provides a mathematical framework for the design of practical online learning algorithms for neuromorphic substrates.*

By Friedemann Zenke and Emre O. Neftci, *Member IEEE*

**ABSTRACT** | Neuromorphic hardware strives to emulate brain-like neural networks and thus holds the promise for scalable, low-power information processing on temporal data streams. Yet, to solve real-world problems, these networks need to be trained. However, training on neuromorphic substrates creates significant challenges due to the offline character and the required nonlocal computations of gradient-based learning algorithms. This article provides a mathematical framework for the design of practical online learning algorithms for neuromorphic substrates. Specifically, we show a direct connection between real-time recurrent learning (RTRL), an online algorithm for computing gradients in conventional recurrent neural networks (RNNs), and biologically plausible learning rules for training spiking neural networks (SNNs). Furthermore, we motivate a sparse approximation based on block-diagonal Jacobians, which reduces the algorithm's computational complexity, diminishes the nonlocal information requirements, and empirically leads to good learning performance, thereby improving its applicability to neuromorphic substrates. In summary, our framework bridges the gap between synaptic plasticity and gradient-based approaches from deep learning and lays the foundations for powerful information processing on future neuromorphic hardware systems.

**KEYWORDS** | Artificial neural networks; biological neural networks; learning systems; machine learning; neural network hardware; neuromorphic engineering; recurrent neural networks (RNNs).

Manuscript received August 1, 2020; revised October 22, 2020; accepted December 6, 2020. Date of publication January 8, 2021; date of current version April 30, 2021. This work was supported in part by the National Science Foundation under Grant 1652159 and Grant 1823366 (EN) and in part by the Novartis Research Foundation (FZ). *(Friedemann Zenke and Emre O. Neftci contributed equally to this work.) (Corresponding author: Emre O. Neftci.)*

**Friedemann Zenke** is with the Friedrich Miescher Institute for Biomedical Research, 4058 Basel, Switzerland.

**Emre O. Neftci** is with the Department of Cognitive Sciences and the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA. (e-mail: eneftci@uci.edu).

Digital Object Identifier 10.1109/JPROC.2020.3045625

## I. INTRODUCTION

Our brain simultaneously processes various streams of temporal information, allowing us to solve challenging real-world problems that ensure our survival. Importantly, brains do this with an aptitude that dwarfs existing computer technologies while only consuming a meek 25 W of power.

Neuromorphic engineering has taken on the challenge of approaching such efficiency by building scalable, low-power systems that mirror the brain's essential architectural features [1]–[3]. Decades of research helped overcome major engineering challenges toward this goal, resulting in an increasing number of neuromorphic substrates available today [4]–[6] and allowing the efficient emulation of brain-inspired neural networks. One key challenge that remains and prevents the widespread application of neuromorphic systems is the lack of practical algorithms that run on such hardware and equip it with complex functionality.

Deep learning provides algorithmic blueprints to organize large neural networks into suitable function approximators that flexibly solve diverse real-world problems [7]. To achieve this feat, deep learning optimizes loss functions with gradient descent, which can be computed efficiently with the back-propagation (BP) algorithm. This efficiency, however, rests on the von Neumann computer architecture. In contrast, gradient BP is difficult to implement on non-von Neumann neuromorphic substrates. These difficulties mainly arise from limitations in their ability to communicate neural activities and weight values between different network elements, similar to the architectural constraints of biological neural networks [8]. For instance, synaptic plasticity implemented at a biological synapse may have access to the activity of the two neurons that it connects but not to the activity of other neurons that it is not physically connected to. This notion is often expressed by saying that plasticity in the brain is local. Local learning rules have been extensively studied in computational neuroscience, typically based on experimental data. However, these rules often lack

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

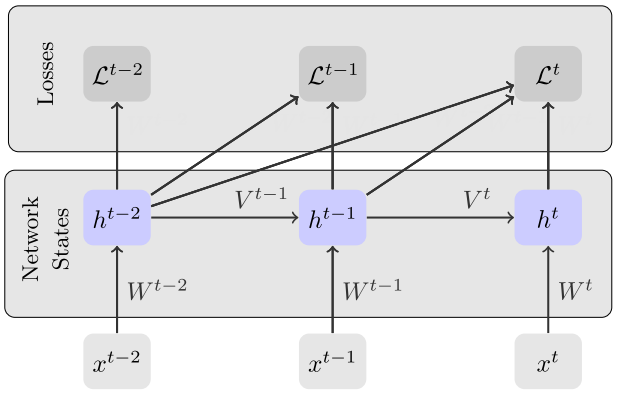


**Fig. 1.** *Computational graph of a recurrent neural network (RNN). The bottom row represents the inputs $x^t$. The middle row represents the network states of a network with parameters $\theta = \{W, V\}$. Here, arrows indicate operations on the nodes, and the arrow labels indicate the parameters involved in the computations. Note that, to distinguish between direct and indirect influences, the parameters have superscripts with time indices even though these parameters are tied across time. The top row shows the computations of the output loss in each time step.*

the normative foundations of BP and hence the ability to instantiate complex functional neural networks. However, top-down-driven synaptic plasticity rules can also be derived from gradient descent both in the case of single biologically inspired spiking neurons [9]–[11], thereby establishing a link to the Perceptron [12], and in more complex multilayer networks [13]–[21]. Remarkably, many normative approaches do result in learning rules largely consistent with models of cortical neurons and synapses [17], [22]–[27].

In this article, we discuss the remarkable commonalities across machine-learning and computational neuroscience learning algorithms from the standpoint of neuromorphic engineering. To this end, we rely on the conceptual framework provided by deep learning, allowing us to focus on three distinct aspects of building functional artificial neural networks: architecture, learning rules, and loss functions (see Fig. 1) [28]. Within this framework, we focus on online learning rules (see Sections II and III) and loss functions (see Section IV) due to their specific relevance for neuromorphic engineering. This relevance largely derives from the use of non-von Neumann architectures in this field, which we introduce next.

### A. von Neumann Computers and Biological Brains

Conceptually, computation requires memory, communication, and arithmetic. Computation is a physical process in which these three components come together in the same place and at the same time. Doing so requires space, time, and energy. The number of resources used determines the overall cost of computation and varies depending on both the computation itself and the architecture on which it runs.

The von Neumann architecture, which underlies virtually all human-made computers, posits a physical separation between processing units for arithmetic, program flow control, and memory. This separation induces a communication channel between the two. A significant design limitation of this architecture is that, when the amount of data becomes too large, communication between the processing and the memory unit becomes a bottleneck. Thus, for memory-intensive computations, this so-called von Neumann bottleneck impacts both latency and power efficiency. The impact is especially noticeable in deep learning applications that require large amounts of data to first propagate through immense deep neural network models. Then, gradient information propagates back through the same nodes to compute gradients for learning. Since both the data and the model parameters reside in memory while the operations that act on them take place in the processing units, the communication requirements become substantial. Although modern computer architectures use diverse strategies to mitigate this bottleneck, for example, caching, branch prediction, and parallelism, all these measures come at the expense of higher energy cost and larger on-chip space requirements. Hence, they eventually run into the same bottleneck. With the looming end of Moore's law, the communication bandwidth across processing and memory units is reaching a plateau. While the continued development of deep learning hinges, in large part, on ever faster and larger-scale hardware, some emerging neuromorphic developments seek to provide alternatives to the von Neumann architecture [29].

While von Neumann architectures simulate deep neural networks, neuromorphic solutions attempt to emulate them on a physical substrate, inspired by the brain. In the brain, neurons integrate inputs in an analog manner, apply nonlinear transformations to them, and communicate asynchronously through digital spikes or action potentials. Spikes are binary events in continuous time that evoke electric currents in receiving neurons whose synaptic strength is determined by their connecting synapses. Importantly, these connections are plastic and can change dynamically in an experience-dependent manner to implement learning and memory. Together, neurons and their intricate synaptic connectivity achieve neural processing and long-term memory storage in a completely parallel and time-continuous way. Since memory and arithmetic are physically colocalized and distributed within the same physical network, brains are fundamentally non-von Neumann architectures.

Sparse, event-based communications and physically colocalized memory and computation are two defining principles of brain-inspired computing and neuromorphic engineering [29], [30]. Neuromorphic engineers recognized the benefits of this approach and developed large networks of VLSI neurons and synapses communicating asynchronously through address event representation (AER) [31], [32]. Recent digital hardware, such as Google's TPU [33], Graphcore's intelligence processing unit (IPU), and Cerebras' wafer-scale CS-1, have also embraced some form of memory computation

colocalization to improve the energy and performance metrics of scientific computing and machine-learning workloads. The development of these brain-inspired architectures is, in large part, motivated by the need to process the ever-increasing deluge of data generated by the pervasive sensors in everyday life.

### B. Processing and Learning From Temporal Data Streams

A large fraction of sensor data are temporal and require real-time processing, thus warranting dedicated architectures. For example, autonomous vehicles need continuous monitoring and processing of time-series data from their sensors to navigate safely. Similarly, Internet-of-Things (IoT) devices have to continuously monitor their environment to respond to speech commands, detect anomalies online from biosensor data (e.g., a pacemaker implant), and learn continually [34]–[36]. While data streams induce specific challenges, most of which we will discuss in Section IV, we first focus on the essential fact that most real-world data, whether streaming or not, are also temporal data that need to be processed in real time. In the following, we briefly review RNNs as the de facto standard for processing temporal data before focusing specifically on neuromorphic implementations thereof and how we can accomplish gradient-based optimization in real time.

The brain is an inherently time-dependent dynamical system [37], [38] that relies on biophysical processes, recurrence, and feedback of its physical substrate for computation [30], [39]. These dynamics are different from the majority of deep neural networks, which are often strictly feedforward, and lack the fine temporal dynamics of brains. From a technological point of view, emulating neural dynamics on a physical substrate has the advantage of operating much more efficiently compared to simulations [40]. However, this comes with a key challenge in which "time represents itself." This implies that all computational processes occur at the timescales of the physical system. In VLSI technologies, this is achieved by operating CMOS transistors in their subthreshold regime, such that currents and, consequently, the time constants are matched to those of the brain [40], [41]. As such, these systems are both online and streaming. Other technologies run in accelerated time [42], which can create a mismatch of timescales in certain real-world applications.

RNNs have proved highly effective for sequential processing, such as keyword spotting, object recognition, and time-series forecasting [7], [43]. Neuromorphic processors generally implement spiking neural networks (SNNs) [6], which can be viewed as a special class of RNNs inspired by biology. SNNs are particularly suited for energy-efficient processing due to their rich local dynamics but spatiotemporally sparse communication via spikes (events). This is in contrast to most RNNs used in machine learning, which rely on dense and analog-valued communications. Moreover, biological neurons presumably carry specific inductive biases in their internal dynamics that are potentially advantageous for real-world information processing.

Like all neural networks, SNNs can be trained to find suitable connection weights. Because SNNs are RNNs, they can be trained with similar gradient-based methods that only need to be modified slightly to accommodate the binary activation functions underlying action potentials [20]. Gradient descent on a loss function therefore automatically adjusts neuron and synapse parameters in the hidden layers of the network to reduce a scalar training loss. In the following, we review two common algorithms underlying gradient computation in RNNs and highlight their limitations for gradient-based learning on neuromorphic substrates, before discussing a range of possible remedies.

## II. GRADIENT-BASED LEARNING IN RECURRENT NEURAL NETWORKS

Training RNNs requires computing objective function gradients with respect to the network parameters. To gain a better understanding of why specific challenges arise when computing gradients for RNNs, we consider a simple RNN

$$h^t = f_\theta(h^{t-1}, x^t)$$

with the network state $h^t \in \mathbb{R}^k$, the input $x^t \in \mathbb{R}^n$, and the parameters $\theta \in \mathbb{R}^p$, where $p$ is the number of parameters. We further define the output $y^t = g_\theta(h^t)$, the associated target $y^{t*}$, and the loss function $\mathcal{L} = \sum_t \mathcal{L}^t(y^t, y^{t*})$. Training an RNN requires computing the gradients of $\mathcal{L}$ with respect to all parameters $\theta$. Following the steps of Marschall *et al.* [44], we define $\theta^t$ as the application of the parameter at time $t$ to distinguish between their direct and indirect influence, and bear in mind that all parameters $\theta^t$ are tied across all timesteps $t$. The gradient is given by

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_t \frac{\partial \mathcal{L}^t}{\partial \theta} = \sum_t \frac{\partial \mathcal{L}^t}{\partial h^t}\frac{\partial h^t}{\partial \theta} = \sum_t \frac{\partial \mathcal{L}^t}{\partial h^t}\sum_{s\le t}\frac{\partial h^t}{\partial \theta^s}. \quad (1)$$

Two different temporal summations appear in this expression. To be able to use (1) for online learning, we have to first evaluate the sum over $s$, which underlies $(\partial h^t/\partial \theta)$. Fortunately, it is possible to compute $(\partial h^t/\partial \theta)$ with the help of a simple recursion relationship. To write this relationship compactly, we define the influence $G^t := (\partial h^t/\partial \theta) \in \mathbb{R}^{k\times p}$, the immediate influence $F^t := (\partial h^t/\partial \theta^t) \in \mathbb{R}^{k\times p}$, and the RNN's dynamics $H^t := (\partial h^t/\partial h^{t-1}) \in \mathbb{R}^{k\times k}$. By further assuming that $G^0 = 0$, $G^t$ is given as the recursive product of Jacobians

$$G^t = H^t G^{t-1} + F^t \quad (2)$$

which can be computed going forward in time. For a detailed derivation of the above expressions, see [44]. By inserting $G^t$ back into (1) and assuming a small learn-

ing rate, the summation over $t$ can be implemented as an online learning algorithm. This causal learning algorithm is called real-time recurrent learning (RTRL) [44], [45].

However, a more common method to evaluate the gradient is back-propagation-through-time (BPTT). The BPTT algorithm takes an acausal approach to evaluate the same Jacobian matrix products: It computes the product of Jacobians starting at the end and works its way backward through time, hence its name. To make this relationship explicit, we define the credit assignment vector $C^t = (\partial \mathcal{L}/\partial h^t) \in \mathbb{R}^k$ and the instantaneous credit vector $D^t = (\partial \mathcal{L}^t/\partial h^t) \in \mathbb{R}^k$. The recursion relation underlying BPTT is then given by

$$C^t = C^{t+1} H^{t+1} + D^t. \tag{3}$$

This expression needs to be computed in an acausal manner [44], which precludes its use as an online learning algorithm. Although, for a given input sequence, the gradients resulting from BPTT and RTRL are the same, BPTT remains the gold standard for training RNNs on von Neumann hardware. The reason is that the different implementations have specific advantages. To understand the origin of these differences, we now analyze the computational costs of BPTT and RTRL.

### A. Cost Analysis of BPTT

The first term in (3) is the $k$-vector derivative of a scalar loss function and is thus a row vector. The factorization afforded by the chain rule means that all products are vector Jacobian products of dimensions $k$ and $k \times k$, respectively, and can be computed in $k^2$ evaluations. To evaluate the gradient in the reverse mode, it is necessary to record all evaluations in the forward phase. Thus, the full activation history needs to be stored in memory, and in the case of RNNs, memory complexity scales as $O(kT)$. The time complexity of each timestep is dominated by the number of scalar multiplication operations underlying the product of the Jacobians, which is $O(k^2\ T)$, or $O(k^2)$ if a single update is performed at the end of the sequence [44]. (see Fig. 4). The memory dependence on $T$ restricts BPTT to temporal inputs of limited duration and to substrates that offer enough memory to store the activation history. Both requirements are major shortcomings when we want to process continuous streaming data on non-von Neumann architectures on which locally accessible memory is limited. To reduce the complexity of BPTT, gradient propagation is generally truncated at a number of steps smaller than $T$. This temporal restriction reduces memory complexity but severely restricts learning performance on tasks that require long time horizons. Truncation is even suspected to render RNNs trained with BPTT to what is effectively a feed-forward (FF) network [46].

### B. Cost Analysis of RTRL

The computational cost of RTRL is determined by evaluating $G^t$ as a product of Jacobians of the shape $\mathbb{R}^{k \times k}$ and $\mathbb{R}^{k \times p}$, which requires $k^2\ p$ scalar multiplications. Since $p$ is the number of parameters, the cost is sizable. RTRL requires $O(kp)$ memory to store $G^t$, a $\mathbb{R}^{k \times p}$ matrix. For instance, in a fully connected network, we have $k(n + k)$ parameters for the feedforward and recurrent connections. Assuming that $n \sim k$, the overall memory complexity scales cubically with the number of neurons $[O(k^3)]$, whereas time scales as $O(k^4)$ per time step.

Nevertheless, one decisive advantage of RTRL is that, although $G^t$ is a potentially large matrix, it can be discarded after each update. In other words, RTRL's complexity is independent of $T$, which makes it an interesting contender for training on streaming data using neuromorphic substrates. Despite the benefit of the online evaluation, RTRL's high computational burden $O(k^4)$, compared with $O(k^2\ T)$ for BPTT, makes it prohibitive for any practical RNN implementation.

### C. Reducing the Cost of Online Learning Through Sparseness

Fortunately, there are multiple ways to reduce the computational load of RTRL while, at the same time, retaining most of its efficiency. One way of lowering RTRL's high computational burden is to approximate the computation of the influence matrix $G^t$, for instance, by decomposing it in a product of lower order tensors [47], [48] or Kronecker factors [49] (see [44] for a comprehensive review). Another way of reducing RTRL's complexity is to consider sparse approximations of the influence matrix $G^t$ [50]. This situation arises naturally when the network connectivity is either sparse [51]–[55] or approximately sparse. Approximately sparse here means there are a few strong connections that dominate the temporal dynamics and thus the gradients.

Sparsity has additional benefits for neuromorphic substrates because it alleviates two of their inherent limitations. First, it caters to limited on-device storage by requiring less memory for model parameters. Second, sparseness can help overcome communication bottlenecks that result from the need of communicating nonlocal but relevant information for learning to where it is needed. In both BPTT and RTRL, the update of a single network parameter indirectly depends on all other network parameters, thus rendering learning nonlocal.

Due to the relevance of sparseness and local learning rules to neuromorphic substrates, we dedicate the remainder of this article to efficient biologically inspired approximations of RTRL and describe how they empower neuromorphic devices to learn from streaming data. As we will see, the dynamics of biological neuron models naturally admit a formulation with sparse block Jacobians that are both sparse and obey locality principles by tying gradient propagation to diagonal blocks that implement the underlying neural and synaptic dynamics. However, before we can fully appreciate the algorithmic importance of sparse Jacobians, we will briefly review the notion

of autodifferentiation, which most modern gradient-based learning algorithms rely on.

### D. Autodifferentiation Strategies in Practice

Automatic differentiation (AD) is a type of differentiable programming that allows us to compute the gradients across entire computational pipelines [56]–[58]. automatic differentiation (AD) has been used extensively in scientific computing before it was applied to neural networks [59], but it has been popularized with recent libraries, such as Theano [60], Tensorflow [61], and PyTorch [62]. The key principle underlying AD is that numerical computations are compositions of a finite set of elementary operations for which derivatives can be defined. By combining the derivatives of the operations through the chain rule of derivatives, the gradient of the overall composition is systematically computed.

Just as in (2) and (3), gradients in AD can be accumulated according to different modes. The two main modes are forward and reverse. For RNNs, forward-mode accumulation is equivalent to RTRL, and backward-mode accumulation corresponds to BPTT. These two modes are two extremes of gradient accumulation. In principle, it is possible to mix forward and backward within the same algorithm [63], [64]. However, for reasons of its superior time and space complexity, most machine-learning frameworks use pure reverse-mode accumulation. Later in this article, we discuss an example of mixed-mode AD that is computationally advantageous for neuromorphic hardware.

## III. APPLYING GRADIENT-BASED LEARNING TO BIOLOGICALLY INSPIRED NEURAL NETWORKS

Biological neural networks are RNNs, but there are two defining characteristics that set them apart from the generic RNN models discussed in Section II. First, biological neurons possess internal dynamics on different timescales due to a plethora of biochemical processes that interact with the membrane dynamics [30], [65]. Because these dynamics play an important role for the approximations of the Jacobians $H^t$ and $F^t$, which are the basis for the efficient online learning algorithms, we will discuss them in detail in Section III-B. Second, most biological neurons communicate through action potentials or spikes. Since spikes are binary neuronal outputs, they render SNNs nondifferentiable, which poses a problem for standard gradient-based optimization algorithms. Hence, special training methods are required, which we briefly review in the following.

### A. Training Spiking Neural Networks

Several learning schemes have been developed to overcome the nondifferential nature of spiking neurons [20], [66]–[68]. The most commonly used gradient-based SNN learning paradigms are network translation [69]–[72], variational learning with stochastic neuron models [13], [23], [73], [74], and surrogate gradients in combination with deterministic integrate and fire (LIF) neurons [15]–[17], [20], [21], [75]–[78].

While translation approaches require the training of a nonspiking proxy network whose connectivity is later translated into a spiking network, the other methods operate directly on SNNs. Variational learning approaches attempt to change the distribution of the network output toward a given target distribution by minimizing an upper bound on the Kullback–Leibler divergence between the two. To this end, these models employ stochastic neuron models, typically formulated within the scope of the spike response model (SRM) with escape noise or a stochastic firing threshold [79]. Variational methods can learn useful representations in hidden neurons, and importantly, the resulting learning rules often have natural interpretations as local learning rules with a global modulatory factor [13], [23], [80]. However, both low-dimensional feedback [81] and stochasticity [13] are known to result in noisy gradient estimates, which can lead to slow convergence and can render learning practically impossible.

Surrogate gradient learning avoids such problems by using neuron-specific feedback signals as in standard BP and dispensing with stochasticity in the forward pass. Nevertheless, gradients are computed as if the noise was present to smooth out the nondifferentiable binary nonlinearities of spiking neurons. While a rigorous theoretical formulation of this interpretation still needs to be worked out, it could provide a compelling explanation of why surrogate gradient learning is robust to the choice of surrogate derivative [78]. Within such a framework, different functional shapes of surrogate derivatives may simply correspond to different choices of neuronal noise distributions.

Irrespective of the underlying explanation, a host of recent studies has established the effectiveness of surrogate gradient learning at scale on various deep SNN architectures and diverse tasks and data sets [15], [16], [20], [21], [27], [77], [78]. To this end, surrogate derivatives have been used in combination with variants of both RTRL or BPTT.

### B. Spiking Neuron Models Have Implicit Recurrence

To capture the internal dynamics of real neurons, spiking neuron models possess implicit recurrence [20]. To understand this concept, we consider one of the simplest and most widely used spiking neuron models: the linear integrate and fire (LIF) neuron [79]. The LIF neurons capture key electrophysiological properties of biological neurons while being abstract, analytically tractable, and easy to simulate. The state of a single-compartment LIF neuron $i$ is described by its membrane potential $U_i$, which

obeys the following temporal dynamics:

$$\tau_m \frac{dU_i}{dt} = -(U_i - U_{\text{rest}}) + R\sum_j W_{ij} S_{j,\text{in}}(t) + R\sum_k V_{ik} S_k(t) \quad (4)$$

where we have introduced the resting potential $U_{\text{rest}}$, the membrane time constant $\tau_m$, and the input resistance $R$ [79]. The membrane potential $U_i$ acts as a leaky integrator of the input spike trains $S_{j,\text{in}}(t) = \sum_{k\in\zeta_j} \delta(t_j^k - t)$, where $\delta$ is the Dirac delta and the sum runs over all firing times $\zeta_j$ of neuron $j$. The output spike trains $S_k(t)$ are defined similarly. Matrices $W$ and $V$ here define feedforward and recurrent weight matrices. On a digital computer, (4) is commonly integrated using an Euler numerical integration scheme[1] on a discrete-time grid as follows:

$$U_i^{t+1} = \beta U_i^t + (1-\beta)\left(-U_{\text{rest}} + R\sum_j W_{ij} S_{j,\text{in}}^t + R\sum_k V_{ik} S_k^t\right) \quad (5)$$

where we further introduced the decay factor $\beta := \exp(-(\Delta/\tau_m))$ with timestep $\Delta$ and the discrete spike trains $S_{j,\text{in}}^t$, $S_j^t$, which are equal to 1 if the respective neurons $j$ spiked in timestep $t$ and zero otherwise. Equation (5) makes the formal connection to our RNN example above more obvious. Similar to a long short-term memory (LSTM) cell, the spiking neuron's leaky membrane potential maintains an internal state through implicit recurrence via the decay constant $0 < \beta < 1$. The LIF [see (5)] model is related to the RNN and its learning dynamics in (2), by replacing $h^t$ with the membrane potential state $U^t$.

To understand the implications of this property for gradient computation and use it for efficient algorithmic learning implementations, we distinguish between two types of recurrence: explicit and implicit. In keeping with our convention [see (2)], we define the following notation:

$$G^t = \frac{\partial U^t}{\partial \theta}, \quad F^t = \frac{\partial U^t}{\partial \theta^t}, \; H^t = \underbrace{\frac{\partial U^t}{\partial U^{t-1}}}_{H^{I,t}} + \underbrace{\frac{\partial U^t}{\partial S^{t-1}}\frac{\partial S^{t-1}}{\partial U^{t-1}}}_{H^{E,t}}$$

where $H^{I,t}$ and $H^{E,t}$ denote the implicit and explicit recurrent dynamics, respectively. This decomposition is motivated by the elementwise nature of the computations inside the network elements. Implicit recurrence captures the sensitivity to perturbations of the internal neuronal dynamics, such as membrane dynamics. Explicit recurrence is due to interneuronal synaptic connections, such as in any vanilla RNN model. With these definitions, the forward-mode recursion takes the familiar RTRL form that exhibits the contributions of the two types of recurrences

$$G^t = \underbrace{H^{I,t}G^{t-1}}_{\text{Implicit}} + \underbrace{H^{E,t}G^{t-1}}_{\text{Explicit}} + F^t \quad (6)$$

with initial states $G^0 = 0$.

A consequence of elementwise operations within the neuron is that $H^{I,t}$ and $(\partial S^t/\partial U^t)$ are block sparse Jacobian matrices with a diagonal structure (see Fig. 2).

The model descriptions above represent SNNs in the same form as artificial RNNs. However, compared with RNNs, the timestep used for SNN integration is much finer in practice to account for the internal neuronal dynamics. Furthermore, a small time step provides a more accurate integration with respect to the continuous-time dynamics and thus the modeled neuromorphic substrate. Since this timestep is determined by the shortest timescale in the dynamical system and thus independent of the input data, many SNNs are trained over hundreds of steps. This results in as many network instances in memory as there are time steps, which becomes cumbersome for networks of more than a few thousand neurons. As a result, in recent works, the size of SNNs trainable by BPTT has remained severely limited by the available GPU memory [15]. Hence, pure reverse-mode AD, that is, BPTT, is not suitable to train large SNNs. The invariance of RTRL's memory complexity to the number of time steps $T$ offers a potential solution to this problem, provided that the unfavorable $O(k^3)$ space and $O(k^4)$ time scaling can be mitigated. But how can one achieve a reduction in complexity?

Let us, for a moment, ignore the explicit recurrences when computing $G^t$. In this case, if $H^I$ and $G^t$ have the same block structure,[2] the resulting product $H^{I,t}G^{t-1}$ is also block diagonal. Thus, only $O(k^2)$ operations are necessary to compute the recursion of the derivative, and $O(k^2)$ memory is required to store the nonzero values of $G^t$. Therefore, maintaining the sparseness of the $G^t$ recursion has the potential of reducing the complexity of RTRL. Here, we focus on two solutions to maintain $G^t$ sparse.

The first solution is to work with sparse $H^E$ matrices, meaning, in the case of a single-layer RNN, a sparse matrix $V$ with few nonzero entries. Sparse connectivity patterns are pervasive in the brain as biological neural networks tend to be locally dense but globally sparse [84]. While a sparse $V$ does not indefinitely maintain the $G^t$ sparse, it does so for a certain number of steps. This opens up possibilities for sparse $n$-step approximations, allowing to save computation by a factor of the sparsity squared [50]. Moreover, sparse connectivity caters to the fact that neuromorphic hardware often relies on sparse connectivity for better memory efficiency [85], which is

[1] Due to the chaotic nature of many SNN models and the dominant discretization error introduced by the simulation time grid, most simulators rely on the Euler integration [82], [83].

[2] $G$ is not square, in general, but the same arguments hold for block diagonal matrices shown in Fig. 2.

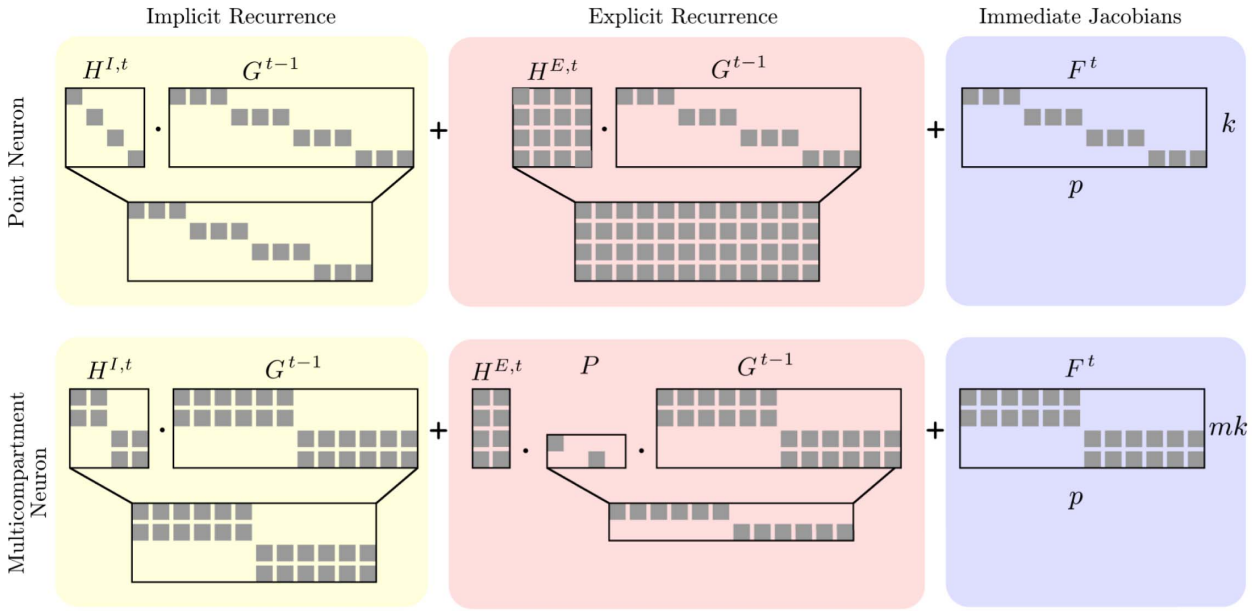


**Fig. 2.** *Illustration of the Jacobians involved in the RTRL recursion [see (6)] for four point neurons $n = 3, k = 4$, and $p = nk$ (top) and two 2-compartment neurons $n = 3, k = 2$, $m = 2$, and $p = nkm$ (bottom). To distinguish input and output dimensions, the recursion is shown here for the parameters of W only. For the parameters V, this illustration would look similar, but with $n = k$. Top: single-compartment neuron. For both cases, it was assumed that $G^{t-1}$ is a stepwise diagonal matrix. This shows the "mixing" effect of $H^{E,t}$. By virtue of block matrix algebra, the implicit recurrence preserves the (block) diagonal structure of $F^t$. Hence, $H^{I,t}$ is referred to as implicit recurrence (yellow shaded area). On the other hand, the off-diagonal terms of $H^{E,t}$ destroy the (block) diagonal property (red shaded area).*

also mirrored in the hierarchical communication fabric [5], [86], [87]. Thus, from an implementation standpoint, sparse connectivity matrices are preferable on neuromorphic hardware. From a performance standpoint, the suitability of sparsely connected networks varies from case to case and is an intensely studied topic, both in terms of implementation [88] and algorithms. Randomly pruning network weights typically impairs overall network performance unless special care is taken, such as intelligent sparse initialization schemes [52], [53], [55], [89] or dynamic rewiring during the training [51], [54]. Moving forward, the "lottery ticket hypothesis," which posits the existence of sparse, trainable strictly feed-forward networks without loss in accuracy [52], is likely to spur further research on sparse RNNs.

The second solution is to approximate the gradient computation by assuming that certain connections contribute more to the gradient than others. For SNNs, a simple way of doing so is to ignore all contributions of $H^E$ to the gradient which amounts to assuming that most relevant temporal information is carried forward in time through implicit recurrence, that is, $H^I$. All the while, the recurrent connections remain in place in the network and contribute to the dynamics. But how well do such approximations work?

## C. Implicit Recurrence Induces Approximate, Local, and Efficient Learning Rules

Although ignoring $H^E$ seems like a drastic simplification, several studies have used it to construct biologically plausible online learning rules as local approximations of RTRL. Empirically, these rules perform well on a number of complex problems either without recurrent connections, such as in the case of SuperSpike [17], or by ignoring gradient flow through the recurrent synaptic connectivity as done in e-Prop [27], RFLO [92], and DECOLLE [19]. These findings suggest that explicit recurrence is either not necessary for many problems or that ignoring explicit recurrence in gradient computations does not create a major impediment for successful learning, even when such recurrent connections are present.

To disambiguate between these different possibilities, we extended previous work [78] by running additional simulations in which we either ignored or included the contribution of $H^E$ during SNN training. We then systematically compared the resulting network performance of the two approaches to networks without any explicit recurrent connections (see Fig. 3). Since these results may be dataset-dependent, we repeated this analysis for a range of different classification datasets that required different levels of temporal memory. The tasks can be coarsely divided according to the temporal duration of their inputs. For the Randman and MNIST datasets, all input spikes arrive within a short temporal window ($< 50$ ms), whereas for the speech processing problems, we considered individual inputs with a duration of $\sim 1$ s.

Not surprisingly, the addition of explicit recurrent connections to a given SNN results in the reduction of error over strictly feed-forward synaptic connectivity, in most cases (compare RC and FF in Fig. 3). Note that we did not correct for the larger parameter count of the recurrently

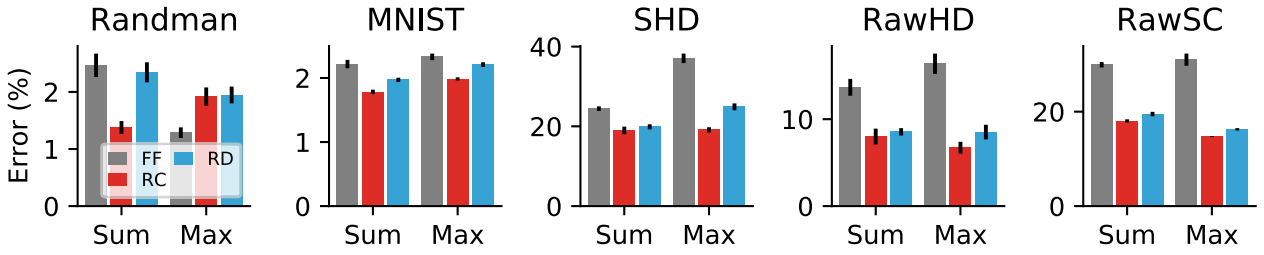


**Fig. 3.** *Classification error of SNNs trained on various classification tasks, as in [78]. Randman: synthetic spike-timing-dependent task based on smooth random manifolds. MNIST: time-to-first spike latency encoded version of the MNIST data set. SHD: Spiking Heidelberg Digits data set [90]. RawHD: Heidelberg Digits data set, but using analog current-based input instead of spikes. RawSC: same current-based input, but using the Speech Commands data set [91]. We distinguish between the following types of synaptic connectivity and training approaches. FF (gray): Strictly feed-forward SNNs. RC (red): explicitly recurrent SNNs trained with surrogate gradients. RD (blue): "recurrent detached" networks that are architecturally identical to recurrent synaptic connections (RCs), in which we ignored explicit recurrence for gradient computation (see $H^E$). Finally, we used two different readout configurations to compute the logits for the softmax cross-entropy loss. We either summed up the readout unit activation over all timesteps (Sum) or computed the Max over all timesteps. Due to the high computational cost of full RTRL and the absence of efficient training libraries, we performed all simulations using BPTT and determined optimal hyperparameters with a grid search comprising more than 8000 simulations. For each configuration, we selected the ten best models using held-out validation data and computed the mean error and SEM on separate test data.*

connected models. This observed difference was generally larger for tasks with increased complexity and temporally longer stimuli, consistent with the idea that longer stimuli require longer memory timescales. However, when the same SNNs were trained while ignoring gradient contributions from explicit recurrence through $H^E$, error rates increased mildly on most data sets. Importantly, however, the errors remained lower than for the networks without RCs. The effect was largest on tasks that required more temporal memory.

Thus, in the scenarios that we tested, the cost of ignoring the contribution of $H^E$ to gradients is small; all the while, the algorithmic benefits are substantial. As illustrated earlier, ignoring $H^E$ yields local online learning rules [see (6)], which are better suited for hardware implementations. Several online learning approaches, therefore, make use of this or similar approximations [19], [27], [93].

The LIF neuron model used for the simulations in Fig. 3 had a 2-D state. One variable was used to model the membrane potential, whereas the other described the time course of exponentially decaying synaptic currents. However, the scaling properties of the online learning algorithm are not affected by extending it to more complex multicompartment neuron models, for instance, by incorporating additional slow dynamical variables [27], [75], [77], [94].

## D. Implicit Recurrence of Multicompartment Neurons Can Increase Computational Power

We now illustrate that RTRL applied to multicompartment neurons results in Jacobians $H^{I,t}$ that are block diagonal and, hence, efficient to train using approximate online algorithms. This can be formalized by extending the domain of $U^t$ to $\mathbb{R}^{mk}$, where $m$ is the total number of compartments per neuron. Note that, in this formalism, synaptic dynamical variables also count as compartments, for instance, the ones that are typically used to implement exponentially decaying conductance or current variables as for the experiments shown in (see Fig. 3 and Appendix A). In a typical multicompartment model, $S^t \in \mathbb{R}^k$ is a function of only one compartment per neuron, which can be written as

$$S^t = \Theta(PU^t)$$

where $P$ is a binary $k \times mk$ matrix that selects the spiking compartment. Without loss of generality, we can assume that the first compartment is the spiking one, resulting in the following matrix $P$:

$$P_{ij} = \begin{cases} 1, & \text{if } j = m \cdot i \\ 0, & \text{otherwise.} \end{cases}$$

Consequently, the dynamics are given by

$$H^t = \underbrace{\frac{\partial U^t}{\partial U^{t-1}}}_{H^{I,t}} + \underbrace{\frac{\partial U^t}{\partial S^{t-1}} \frac{\partial S^{t-1}}{\partial P U^{t-1}}}_{H^{E,t}} P. \tag{7}$$

Since only spiking compartments are assumed to be connected with other compartments through explicit recurrence, the weights $V$ are matrices defined in $\mathbb{R}^{mk \times k}$. Furthermore, we have that $G^t \in \mathbb{R}^{mk \times p}$. Following the approximate gradient computation where $H^E$ is ignored, $G^t$ becomes block diagonal (see Fig. 2), resulting in $pm$ nonzero entries.

Thus, adding neuronal complexity by widening the neuronal state space does not change the time complexity of the learning algorithms and does not preclude the use of efficient online learning algorithms. However, such changes can have dramatic effects on the computational expressivity of the resulting network models and are reflected in the corresponding approximate learning rules [27]. These insights may partially explain why neurobiology uses a diversity of different neuron types with

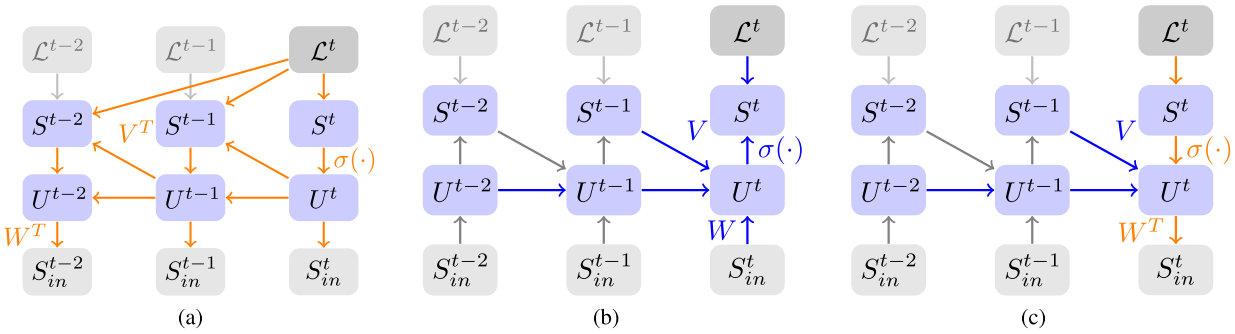


**Fig. 4.** *Computational graphs for the gradient using three distinct AD modes. Edge colors emphasize gradient computations in reverse (orange) or forward (blue). Gray edges indicate that gradients are not involved. For clarity, only the computations relating to $L^t$ are emphasized. (a) Full reverse mode (BPTT). (b) Approximate forward mode (RTRL). (c) Mixed mode.*

distinct internal dynamics and thus opens up new vistas for exciting future research.

## E. Different Modes of Autodifferentiation in Biologically Inspired Learning

To train biologically inspired SNNs and study the dynamical properties of biological networks discussed above in functional networks, many researchers rely on autodifferentiation frameworks. However, because of the shortcomings of BP in the context of online learning for neuromorphic applications, increasing attention is given to the approximate online learning methods discussed in Sections III-C and III-D. In the following, we give concrete examples of recent work on SNN training. However, it is important to bear in mind that the same algorithms also apply to nonspiking RNNs when the spiking nonlinearity is replaced with a smooth differentiable function.

Specifically, we will show one case of reverse mode AD (BPTT), one case of forward-mode AD (RTRL), and one example of mixed-mode AD (see Fig. 4). For all three examples, we consider a network consisting of LIF neurons that evolve according to the dynamics in (5). To simplify the mathematical expressions and without loss of generality, we consider $U_{\text{rest}} = 0$ and $R = 1$. We write the LIF dynamics in matrix form as follows:

$$U^{t+1} = \beta U^t + (1-\beta)(W S^t_{\text{in}} + V S^t), \quad S^t = \Theta(U^t). \tag{8}$$

Here, we assume $n$ input and $k$ output neurons and $S^t_{\text{in}}$ to be the input spike train.

*1) Training With BPTT [see Fig. 4(a)]:* We first analyze the case of training this network with BPTT. The hidden state is $U^t$, and the dynamics and immediate Jacobian take the form

$$H^t := \frac{\partial U^t}{\partial U^{t-1}} = \beta + (1-\beta) V \sigma'(U^{t-1})$$
$$D^t := \frac{\partial \mathcal{L}^t}{\partial U^t} = \sigma'(U^t)$$

where we have assumed the smooth surrogate function $\sigma$. Using the BPTT recursive expression for $C^t$ [which includes dynamics $H^t$; see (3)], the gradients for $W$ and $V$ become

$$\frac{\partial \mathcal{L}^t}{\partial W} = C^t S^t_{\text{in}}, \quad \frac{\partial \mathcal{L}^t}{\partial V} = C^t S^t. \tag{9}$$

Due to the inherent support of reverse-mode AD in machine-learning frameworks, this spiking neuron model can be implemented and differentiated like RNNs. This approach has been applied successfully to train general SNNs models [15], [18], [76], [77], [90], [95], [96].

The BPTT has the advantage that it does not restrict the dynamics, connectivity patterns, and loss function. However, these advantages come at the cost of a large memory footprint and temporal nonlocality. Thus, for applications using small networks of some thousands of neurons that do not require online learning, BPTT is the method of choice. For larger networks, the memory overhead becomes impractical, and other modes of gradient computation described in the following may be necessary.

*2) Training With Sparse RTRL [see Fig. 4(b)]:* As discussed above, it is possible to reduce the complexity of RTRL by keeping $G^t$ sparse. We demonstrate this in the case of the LIF neuron. Using the RTRL recursion [see (2)], the gradient of $W$ obeys

$$G^t_W = (\beta + \underbrace{(1-\beta) V \sigma'(U^{t-1})}_{\text{Explicit rec.}}) G^{t-1}_W + F^t. \tag{10}$$

If we neglect explicit recurrence in the above expression, all involved matrices remain block sparse as shown in Fig. 2. The recursion can be compactly written in vector form by defining the following $k$-vector trace:

$$Q^{t+1}_{\text{in}} = \beta Q^t_{\text{in}} + S^t_{\text{in}} \tag{11}$$

where we used a different variable $Q_{\text{in}}$ to emphasize that it represents a vector, rather than a tensor. Since the $Q_{\text{in}} \in \mathbb{R}^n$ terms are dense $n$-vectors, for gradients with

respect to $W$, the RTRL recursion is simplified from $O(np)$ memory to $O(n)$ memory. Gradients are then computed in the familiar three-factor form:

$$\frac{\partial \mathcal{L}^t}{\partial W} = \frac{\partial \mathcal{L}^t}{\partial S^t} \sigma'(U^t)\, Q^t_{\text{in}} \tag{12}$$

and similarly for $V$ gradients

$$\frac{\partial \mathcal{L}^t}{\partial V} = \frac{\partial \mathcal{L}^t}{\partial S^t} \sigma'(U^t)\, Q^t$$

where $Q^t \in \mathbb{R}^k$ is defined in analogy to $Q^t_{\text{in}}$, but replacing $S^t_{\text{in}}$ with $S^t$. Note that the above factorization to the dense $k$-vector form is only valid for linear LIF neurons with instantaneous approximations of the loss function [19]. However, in the case of nonlinear neuronal dynamics, such as spike-triggered adaptation or for certain loss functions (see Section IV), $O(k^2)$ traces may be necessary, leading to quadratic scaling with the number of neurons [17], [27].

*3) Mixed Mode:* It is possible to combine elements of both BPTT and RTRL for training RNNs and SNNs. We call this situation mixed-mode AD [see Fig. 4(c)]. The portion of the graph from $S^t_{\text{in}}$ to $\mathcal{L}^t$ in (see Fig. 4) may involve several steps. Such cases can occur, for example, in convolutional networks with pooling layers, linear readout layers, and multiple recurrent layers [97]. In [19], for example, loss functions were defined based on random combinations of max-pooled outputs of a convolutional layer consisting of spiking neurons. If these steps are instantaneous, that is, they do not explicitly depend on past states, the immediate Jacobians $F^t_W$ and $F^t_V$ can be computed online using BP within a single timestep. All other gradients can be accumulated over time by virtue of the RTRL recursion. Up to rounding errors, the numerical result will be exactly the same as sparse RTRL discussed earlier, but the memory footprint of the implementation differs and may be more favorable. The recent deep continuous local learning (DECOLLE) [19] is one example of a mixed-mode AD implementation using spatially and temporally local loss functions. It combines the forward-mode AD, to achieve temporal credit assignment, with the reverse-mode AD for spatial credit assignment. This allows the convenient use of existing autodifferentiation tools while combining them with the more favorable scaling properties of BP in space. Bohnstingl *et al.* [97] laid the ground to extend this idea to multiple recurrent layers and found that estimating gradients for layer $l$ using solely the recurrence relation of that layer and ignoring others result in good learning performance and low complexity.

## IV. LOSS FUNCTIONS FOR ONLINE LEARNING

So far, we have focused on learning algorithms that permit efficiently computing loss gradients in an online manner to learn from temporal data. The applicability of these algorithms, however, hinges on the ability to also compute the loss functions in an online manner, which is not always possible. In the following, we discuss the desiderata of loss functions for online real-time learning and point out directions where future research is required.

Since loss functions are defined at the output of a network, let us briefly review what input–output (IO) paradigms exist for RNNs. We can coarsely divide neural network processing by specifying whether they require inputs at every timestep or only once. Similarly, we can distinguish between networks that yield one output at the end of processing as opposed to networks that produce outputs in every timestep. Networks that receive a single input at the beginning produce a single output at the end are very similar to FF neural networks in that they provide a one-to-one mapping [see Fig. 5(a)]. Similarly, we can construct networks that output a trajectory in response to a single command input, which corresponds to a one-to-many mapping. For streaming data, networks must be able to consume a temporal sequence of inputs. Thus, these networks have many inputs and can be further separated into many-to-many or many-to-one mappings.

For our above derivations pertaining to RTRL, we assumed a gradient of a loss function $\mathcal{L}$ defined as a sum over time $\mathcal{L} = \sum_t \mathcal{L}^t$ of temporally localized loss functions $\mathcal{L}^t$. We assumed further that these localized loss functions can be computed immediately after the corresponding network timestep is computed, that is, online. It is easy to see how such local functions can be defined on the output of a many-to-many network [see Fig. 5(b)]. Similarly, this also covers the many-to-one case since we can simply set all losses for $t < T$ to zero ($\mathcal{L}^t = 0$) and retain only one late value $\mathcal{L}^T$.

In the context of SNNs, several studies have focused on the many-to-many setting in which the network has to learn a predefined output trajectory. A classic example is FORCE learning that applies the recursive least-squares algorithm that can be done forward in time and online [98], [99]. A related approach is FOLLOW learning [100] that is another online learning approach for SNNs that relies on local learning rules to learn a forward model of arbitrary dynamical systems.

However, not all learning tasks require learning an output trajectory, and hence, the temporal locality of the loss function is not the rule, but rather the exception. To solve classification problems, for example, one often considers a nonlocal loss that takes into account extended periods of network activity [see Fig. 5(b)]. For instance, in seminal work, Gütig and Sompolinsky [9] and Gütig [101] trained a spiking neuron, the Tempotron, as a binary classifier on input spike trains with a sparse temporal code. Crucially, the Tempotron can freely decide when to spike within a given temporal window. This has the advantage that targets or labels do not have to be given to a learning system with temporal acuity. To achieve this, the authors introduced a loss that depends on the maximum of the

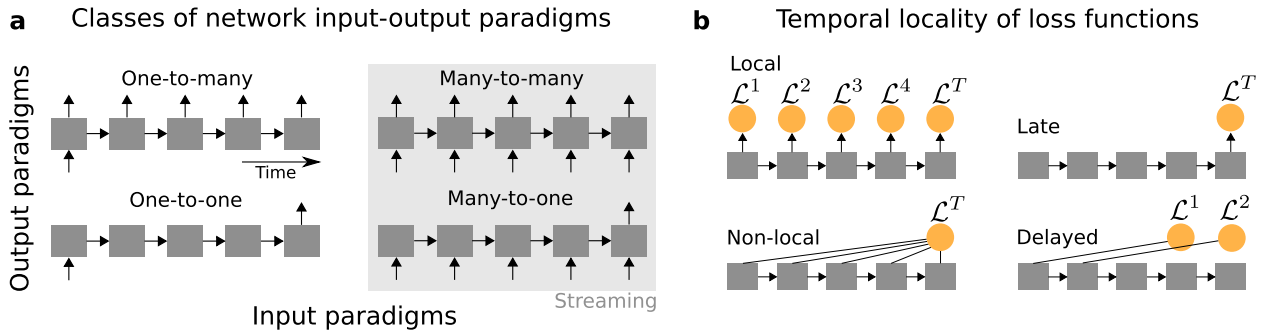


**Fig. 5.** *(a) Schematic of different computational graphs of RNNs, illustrating the main different IO paradigms that typically exist in RNNs. For streaming data purposes, the many-to-one and many-to-many mappings in the gray shaded box are most relevant. (b) Illustration of different computational graphs used for computing loss functions. Online learning algorithms, such as RTRL, require temporally localized loss functions that can be evaluated at every timestep, which trivially generalizes to the regime in which many local losses are zero, for example, $\mathcal{L}^t$ for $t < T$. However, nonlocal loss functions, such as the sum or the maximum over network outputs and delayed arrival of labels that cause the loss evaluation to be delayed, pose challenges for online learning.*

neural membrane potential during an entire trial. Hence, the loss function can only be evaluated once the trial is completed.

A similar approach was later extended to train multilayer SNNs by either computing the maximum over time or the sum over time of dedicated readout units [78], [90] or by simply summing over the number of output spikes [15], [18], [102]. Although this approach can learn powerful classification models, nonlocal loss functions are not suitable for online learning since their gradient can only be evaluated after a trial has completed. Although similar to the late loss case, they are different in that their value depends explicitly on all timesteps [see Fig. 5(b)].

This temporal nonlocality has far-reaching consequences for all classification and pattern detection tasks because it means that all network activities have to be stored until the loss function can be evaluated. Thus, gradient computation is locked until the last timestep of a given trial is evaluated. A similar situation arises when targets or labels are delayed because this also affects the corresponding loss function evaluations [see Fig. 5(b)]. While locking is the norm in the context of BPTT, online algorithms, such as RTRL, may lose their real-time character due to it. Therefore, loss functions that can be evaluated online are crucial to successful online learning. Although research on this issue is still limited, one possible remedy to avoiding delays and locking is to fold the nonlocality back into the network and rely on a loss function that can be written as a sum of temporally localized losses.

One example builds on the van Rossum distance that was developed as a distance metric for spike trains [103]. As a spike train distance metric, it was recognized early on as a suitable loss function for training SNNs to produce precisely timed output spikes [104], but, as we will see in a moment, it is not limited to precisely timed output spike trains. For a given spike train $S(t) = \sum_k \delta(t_k - t)$ and an associated target spike train $S^*(t)$, the van Rossum distance is given defined as

$$\mathcal{L} = \frac{1}{2}\int_{-\infty}^{\infty} dt(\epsilon * S(t) - \epsilon * S^*(t))^2 \tag{13}$$

where $*$ denotes the temporal convolution of the spike trains with the kernel function $\epsilon$. The trick is now to define a filtered spike train $Y = \epsilon * S(t)$ as the network output and similarly a target network $Y^* = \epsilon * S^*(t)$. The kernel $\epsilon$ can be arbitrary, but choosing a causal kernel is critical for all online implementations. In practice, we choose a kernel that can be easily implemented as a simple dynamical system [17], [104] that allows the online evaluation of the term in parenthesis [see (13)]. Canonical choices are exponential or double-exponential functions that are straightforward to implement with one or two additional ordinary differential equations for each output, for example, $\tau(\partial z/\partial t) = -z + S(t)$ with the kernel timescale $\tau$. These manipulations allow us to write $\mathcal{L} = (1/2)\int_{-\infty}^{\infty} dt(Y(t) - Y^*(t))^2$. Finally, by further reducing the integral to a sum over discrete timesteps, this expression can be expressed as $\mathcal{L} = (1/2)\sum_t (Y^t - Y^{*,t})^2 = \sum_t \mathcal{L}^t$, a sum of local losses. Finally, it is not necessary to provide a target spike train. Instead, it suffices to simply provide a target time-series $Y^{*,t}$ that can be interpreted as an instantaneous target firing rate.

As we already alluded to earlier, the van Rossum distance does not necessarily learn precisely timed spikes. Rather, the extent to which the van Rossum distance punishes temporal misalignment can be smoothly adjusted by the time horizon of the $\epsilon$ kernel. For instance, if one chooses a small timescale $\tau$ for the exponential kernel function $\epsilon$, this allows for learning precisely timed output spike trains. However, a large choice of $\tau$ will result in a reduction of temporal spike alignment of individual spikes with their targets. The distance then smoothly interpolates between the spike-timing code and rate codes. Another interesting property of the van Rossum distance, in particular, when used with causal kernels, is that causal

kernels induce an effective delay, which acts as an eligibility trace [17]. Eligibility traces are found in neurobiology [105], and importantly, they solve the distal reward problem by bridging the delay between network output and the error feedback that arrives later in time [106].

Thus, at the expense of temporal precision, the van Rossum distance can introduce temporal memory and accommodate label delay during learning [17] while, at the same time, allowing to compute temporally localized losses. Finally, computing simple kernels, such as exponential or double-exponential, for each network output corresponds to adding additional filtering layers to the network. This layer implies another set of diagonal Jacobians for which we have already seen how their gradients can be computed efficiently using RTRL.

While the van Rossum distance solves some challenges in the streaming data setting in which labels can only be assigned coarsely in time, a number of important issues remain open. Here, future work, possibly building on aggregate label losses [101], [107] or connectionist temporal classification (CTC) losses [43], provided that these approaches can be made online-capable, may offer possible solutions.

In summary, online gradient algorithms based on RTRL require temporally localized losses. While such losses are the standard for learning output trajectories, there are situations in which temporally nonlocal loss functions are more natural choices (e.g., classification problems). Fortunately, a conversion to online-enabled losses is possible for some nonlocal loss functions, as we have illustrated in the example of the van Rossum distance. However, future research efforts are required to firmly establish online-capable loss functions for situations in which labels are only loosely aligned with streaming input.

## V. IMPLEMENTATION STRATEGIES IN NEUROMORPHIC HARDWARE

How can the gradient-based learning strategies discussed in this article guide the development of a neuromorphic learning substrate? Memory is generally a limiting factor in neuromorphic hardware. In SNNs, the need for online learning in combination with such capacity limits makes forward-mode accumulation particularly compelling. Sparse RTRL equates to one trace per parameter, which can be realized in hardware by replicating the circuits and storage for weight updates. For instance, for each connection in the Intel Loihi Research Chip, up to two states that evolve as functions of the postsynaptic and presynaptic states are possible [5]. These states and their dynamics are implemented similar to the weight update dynamics. Thus, Intel Loihi is, in principle, compatible with a sparse forward-mode AD learning scheme. However, several significant open challenges remain for its implementation. First, unlike weight (parameter) updates that can be carried out in an event-based fashion, traces are dynamical states per connection that must be updated at every time step. Because scale in neuromorphic hardware and software simulations is often achieved by storing the synaptic traces at the neuronal level [108], computing traces per connection would significantly increase the computational burden. Second, the memory overhead becomes significant for large networks with shared weights, for instance, in convolutional architectures, because weight sharing does not imply trace sharing. Note that although the idea of weight sharing is counterintuitive to a neuromorphic implementation, digital neuromorphic hardware can take advantage of it [4], [5].

One solution to the above problems is to use the sparse approximation described above [see (11)], previously implemented as part of DECOLLE [19] and, subsequently, for "LIF neurons" in [27], resulting in one trace per axon. In this approximation, synaptic traces use the same state as the synaptic currents for learning. This implies only constant [$O(1)$] memory overhead for learning, which significantly simplifies the neuromorphic hardware implementation. Payvand *et al.* [109] described a DECOLLE crossbar implementation that leverages the sharing of the learning and inference signals while eliciting updates in a temporally sparse, error-driven fashion. Furthermore, the learning dynamics are potentially immune to the mismatch in the synaptic dynamics since the same signal is used for computing the forward pass and gradient dynamics. The gradient-based learning of SNNs induces a three-factor rule [see (12)], comprising one term to compute the loss gradient ($\partial\mathcal{L}/\partial S^t$) and two terms for the network states ($\partial S^t/\partial\theta$). Payvand *et al.* [109] exploited this factorization in a neuromorphic design comprising two types of cores: processing cores and neuromorphic cores. Processing cores are general-purpose processors that compute the loss gradients and neuromorphic cores compute the network states and their gradients. A similar strategy was demonstrated on the Intel Loihi using the on-chip three Lakemont $\times 86$ cores [102]. This separation imparts significant flexibility to the hardware, as loss functions are often task-dependent, but network architectures tend to be generic.

Despite the advantages of online algorithms, reverse-mode AD remains an important reference and a tool for training SNNs offline and offchip. One strategy for digital neuromorphic chips is to use a functional simulator of the dynamics [110] and train it using conventional deep learning techniques (GPUs and BPTT). Functional simulators and BPTT were also used to pretrain networks for subsequent learning on-chip [102]. Due to device-to-device variation, functional simulators of mixed-signal hardware require calibration, for example, by system identification [111], [112]. This calibration scheme is costly and often imprecise because the limited access to the chips' internal states dictates several approximations.

Hardware-in-the-loop approaches can partially overcome this limitation by using the hardware substrate to compute the forward pass and computing synaptic updates in software. Recent work demonstrated successful

learning on an accelerated neuromorphic VLSI substrate using a chip-in-the-loop approach [113], as well as in phase-change memory [21]. While this approach can self-correct for any remaining device mismatch, it requires dedicated on-chip circuitry, for example, sampling ADCs, to read out the internal voltages. In addition, the external computation of updates poses a significant performance bottleneck that renders this strategy often too slow for real-time or accelerated learning. One solution that has not been fully explored is to pretrain networks on an approximate functional simulator of a mixed-signal chip and fine-tune on the chip, as in [102]. Regardless of which training strategy is employed, the methods based on reverse-mode AD are generally limited by memory. Hence, mixed-mode AD and other advanced AD methods will presumably play a central role in reducing the memory footprint, thereby improving the performance and applicability of SNNs training to dedicated neuromorphic substrates.

## VI. DISCUSSION

This article has reviewed common bottlenecks encountered when applying gradient-based learning to neuromorphic architectures that require online computation of gradients. Using the example of SNNs, we have shown that many recently proposed learning algorithms for online learning are approximate variants of RTRL. In its exact form, RTRL is an online algorithm to compute gradients in RNNs, but it is computationally expensive. However, when used in combination with temporally local losses and biologically inspired neural architectures, such as LIF neurons, it is possible to find effective approximations that reduce RTRL's computational cost substantially while retaining good learning performance. We have shown that such approximations are inspired by biological neurons whose implicit recurrence structure induces block sparse or approximately block sparse Jacobians, allowing speeding up gradient computation while simultaneously reducing the memory footprint. These conceptual links expose a clear path forward toward building more efficient online learning algorithms for neuromorphic devices.

We further elaborated on the relationship between gradient-based learning in SNNs and the different modes of automatic differentiation. Due to sufficient memory and the lower computational cost on the von Neumann computers, deep learning has primarily focused on reverse-mode accumulation or BPTT. However, when combining appropriate architectures with approximate RTRL, we expect a renaissance of forward-mode AD to empower non-von Neumann computers and streaming applications requiring online learning. Mixed-mode AD is a particularly exciting direction for implementing learning in biological neural networks as it efficiently reduces complexity by exploiting the Jacobians' sparseness. This idea resonates with a widespread trend in deep learning accelerators: To tradeoff compute against memory, as evidenced by advanced AD techniques on manycore processors the Cerebras CS-1 and the Graphcore IPU.

### A. Spatial and Temporal Scales in Gradient-Based Learning

Computation is a physical process that extends across multiple spatial and temporal scales. Practical learning algorithms have to take this multiscale behavior into account. The finite truncation length in BPTT defines the time span (memory) over which the learning algorithm can efficiently navigate between timescales, even when the network itself supports such dynamics (e.g., through working memory or gating mechanisms in LSTMs [114]). Thus, prematurely truncated gradients can be harmful. Memory and stability go hand-in-hand in dynamical systems, including RNNs. Miller and Hardt [46] argued that RNNs are trained to operate in the stable regime in which gradients vanish for stable learning. However, stability can often be at odds with flexible long-term memory [115]. Therefore, many working memory models in neuroscience rely on multistability and attractor states to implement long-term memory [116]–[118], with regions of state space in which gradients either vanish [119] or become very large [120]. The LSTM networks explicitly overcome this shortcoming by including long timescales in their architecture, thereby avoiding vanishing gradients [115]. This twist, however, requires training procedures that can similarly bridge such long time horizons, which is where truncated BPTT can easily reach its limits. An interesting question is whether training with approximate forms of RTRL can offer advantages in training networks with such longer short-term memory. Thus, if remaining issues related to the bias of approximate variants of RTRL and stability can be addressed at scale, these findings will open new vistas in computing with the non-von Neumann computers.

### B. Solutions to the Weight Transport Problem

Another issue that plagues neuromorphic implementations of BP is that reverse accumulation requires access to the transposed weight matrices $W$ and $V$. This requirement has also been referred to as the weight transport problem [121]. The weight transport problem implies a bidirectional flow of information, which is biologically implausible and crucially difficult to realize in any physical system [122], [123]. The problem is that, to compute the gradient, error information needs to flow backward through the same connections that are used in the forward pass. This creates a major impediment for any physical system, be it biological or neuromorphic, in which information flow is directed. Overcoming this limitation often requires an explicit learning channel that implements these backward weights or weight transposes [122], [123] and ideally keeps them synchronized across nodes or different physical locations of physical network implementation. A number of studies have shown that such synchronization

can be achieved, for instance, by learning the backward weights [124]–[126].

In any case, the mere existence of such backward connections for spatial credit assignment still has a real physical price as they require space on the chip and consume energy, thus raising the question of whether one could dispense with them entirely. Interestingly, RTRL and its approximate variants do suggest ways forward toward viable alternatives. Since RTRL preserves causality with respect to the forward dynamics, the temporal weight transport problem does not apply. Unfortunately, exact RTRL still requires synapses to know the state of all other neurons and synapses in the network, thus posing a state transport problem. The block sparse Jacobian approximations that we discussed in this article are not straightforward to extend to learning across multiple layers. Nevertheless, the inherently causal nature of RTRL and the recent success of local loss functions for training SNNs [19], [97], [127], [128] provide exciting avenues of future research to spatially assign credit and circumvent the weight transport problem in multilayer networks.

While a plethora of neuromorphic platforms, which mimic the brain's computational substrate, have matured over the years, seeing these solutions thrive in real-world applications will require them to learn. To tackle this challenge, we should turn once more to biology. Taking inspiration from the brain will allow us to develop neuromorphic learning algorithms toward tomorrow's neuromorphic computers.

## APPENDIX A
## DERIVATION OF LOCAL LEARNING APPROXIMATIONS FOR SNNs FROM RTRL

RTRL is a special case of forward-mode AD applied to RNNs. For a population of spiking LIF neurons with currently available exponential synapses, the discrete-time dynamics are given by

$$\begin{aligned} U_i^{(l)}[t+1] &= \beta U_i^{(l)}[t] + I_i^l[t] \\ I_i^{(l)}[t+1] &= \sum_j W_{ij}^{(l)} S_j^{(l-1)}[t] + \sum_j V_{ij} S_j^{(l)}[t]. \end{aligned} \tag{14}$$

The output of neuron $U_i^{(l)}$ is given as the spike train $S_i^{(l)} = \Theta(U_i^{(l)})$. To compute gradients with respect to this spike train output, we first define

$$P_{ijk}^{(l,m)}[t] = \frac{\partial}{\partial W_{ij}^{(m)}} U_k^{(l)}[t] \tag{15}$$

and perform gradient descent on the global loss function $\mathcal{L}$ with respect to the parameters

$$\begin{aligned} \Delta W_{ij}^{(m)} &\propto -\frac{\partial \mathcal{L}[t]}{\partial W_{ij}^{(m)}} \\ \frac{\partial \mathcal{L}[t]}{\partial W_{ij}^{(m)}} &= \sum_k \frac{\partial \mathcal{L}[t]}{\partial S_k^{(L)}[t]} \sigma'(U_k^{(L)}[t]) P_{ijk}^{(L,m)}[t] \\ P_{ijk}^{(l,m)}[t+1] &= \left(\beta P_{ijk}^{(l,m)}[t] + \frac{\partial}{\partial W_{ij}^{(m)}} I_k^{(l)}[t]\right) \\ \frac{\partial}{\partial W_{ij}^{(m)}} I_k^{(l)}[t+1] &= \sum_{j'} V_{ij'}^{(l)} \sigma'(U_{j'}^{(l)}[t]) P_{ijj'}^{(l,m)}[t] \\ &\quad + \underbrace{\sum_{j'} W_{ij'}^{(l)} \sigma'(I_{j'}^{(l)}[t]) P_{ijj'}^{(l-1,m)}[t]}_{\text{explicit}} \\ &\quad + \delta_{lm} S_j^{(l-1)}[t]. \end{aligned} \tag{16}$$

Similar equations can be obtained for $\Delta V_{ij}^{(m)}$. The terms underwritten with "explicit" introduce nonlocality to the learning that is difficult to compute since it depends on the history of all other neurons in the network. If these terms are dropped, the indices $i$ and $k$ from $P$ become unnecessary since no term on the right-hand side depends on those indices. Thus, for purposes of gradient computation, the indirect influence of all these interactions is ignored, whereas, in interactions through the implicit recurrence, the memory within the neuron is maintained. The price of performing such an approximation on task performance is often surprisingly low, and it seems that neural networks can, nevertheless, take advantage of their explicit recurrent connectivity (see Fig. 3). ■

## REFERENCES


[1] C. Mead, *Analog VLSI and Neural Systems*. Reading, MA, USA: Addison-Wesley, 1989.

[2] C. S. Thakur *et al.*, "Large-scale neuromorphic spiking array processors: A quest to mimic the brain," *Frontiers Neurosci.*, vol. 12, p. 891, Dec. 2018.

[3] M. Davies, "Benchmarks for progress in neuromorphic computing," *Nature Mach. Intell.*, vol. 1, no. 9, pp. 386–388, Sep. 2019.

[4] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[5] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[6] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers Neurosci.*, vol. 5, pp. 1–23, 2011.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[8] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Rev. Neurosci.*, vol. 21, pp. 335–346, 2020.

[9] R. Gütig and H. Sompolinsky, "The tempotron: A neuron that learns spike timing-based decisions," *Nat Neurosci*, vol. 9, no. 3, pp. 420–428, Mar. 2006.

[10] S. M. Bohte, J. N. Kok, and J. A. La Poutré, "Spikeprop: Backpropagation for networks of spiking neurons," in *Proc. ESANN*, 2000, pp. 419–424.

[11] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning," *Neural Comput.*, vol. 18, no. 6, pp. 1318–1348, Jun. 2006.

[12] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958.

[13] D. Jimenez Rezende and W. Gerstner, "Stochastic variational learning in recurrent spiking networks," *Frontiers Comput. Neurosci.*, vol. 8, p. 38, Apr. 2014.

[14] B. Gardner, I. Sporea, and A. Grüning, "Learning spatiotemporally encoded pattern transformations in structured spiking neural networks," *Neural Comput.*, vol. 27, no. 12, pp. 2548–2586, Dec. 2015.

[15] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1412–1421.

[16] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers Neurosci.*, vol. 10, p. 508, Nov. 2016.

[17] F. Zenke and S. Ganguli, "SuperSpike: Supervised learning in multilayer spiking neural networks," *Neural Comput.*, vol. 30, no. 6, pp. 1514–1541, Jun. 2018.

[18] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 795–805.

[19] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (DECOLLE)," *Frontiers Neurosci.*, vol. 14, p. 424, May 2020.

[20] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51–63, Nov. 2019.

[21] S. Woźniak, A. Pantazi, T. Bohnstingl, and E. Eleftheriou, "Deep learning incorporating biologically inspired neural dynamics and in-memory computing," *Nature Mach. Intell.*, vol. 2, no. 6, pp. 325–336, Jun. 2020.

[22] J. Sacramento, R. P. Costa, Y. Bengio, and W. Senn, "Dendritic cortical microcircuits approximate the backpropagation algorithm," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8721–8732.

[23] J. Brea, W. Senn, and J.-P. Pfister, "Matching recall and storage in sequence learning with spiking neural networks," *J. Neurosci.*, vol. 33, no. 23, pp. 9565–9575, Jun. 2013.

[24] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers Neural Circuits*, vol. 9, p. 85, Jan. 2016.

[25] Ł. Kušmierz, T. Isomura, and T. Toyoizumi, "Learning with three factors: Modulating Hebbian plasticity with errors," *Current Opinion Neurobiol.*, vol. 46, pp. 170–177, Oct. 2017.

[26] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, "Towards deep learning with segregated dendrites," *eLife*, vol. 6, Dec. 2017, Art. no. e22901.

[27] G. Bellec *et al.*, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Commun.*, vol. 11, no. 1, p. 3625, Jul. 2020.

[28] B. A. Richards *et al.*, "A deep learning framework for neuroscience," *Nature Neurosci.*, vol. 22, no. 11, pp. 1761–1770, Nov. 2019.

[29] E. O. Neftci, "Data and power efficient intelligence with neuromorphic learning machines," *iScience*, vol. 5, pp. 52–68, Jul. 2018.

[30] P. Sterling and S. Laughlin, *Principles of Neural Design*. Cambridge, MA, USA: MIT Press, Jun. 2017.

[31] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, "Silicon auditory processors as computer peripherals," *IEEE Trans. Neural Netw.*, vol. 4, no. 3, pp. 523–528, May 1993.

[32] S. Deiss, R. Douglas, and A. Whatley, "A pulse-coded communications infrastructure for neuromorphic systems," in *Pulsed Neural Networks*, W. Maass and C. Bishop, Eds. Cambridge, MA, USA: MIT Press, 1998, ch. 6, pp. 78–157.

[33] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.

[34] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, Mar. 2017, Art. no. 201611835.

[35] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," 2017, *arXiv:1703.04200*. [Online]. Available: http://arxiv.org/abs/1703.04200

[36] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," Feb. 2018, *arXiv:1802.07569*. [Online]. Available: http://arxiv.org/abs/1802.07569

[37] G. Drion, T. O'Leary, J. Dethier, A. Franci, and R. Sepulchre, "Neuronal behaviors: A control perspective," in *Proc. 54th IEEE Conf. Decis. Control (CDC)*, Dec. 2015, pp. 1923–1944.

[38] E. M. Izhikevich, *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. Cambridge, MA, USA: MIT Press, 2007.

[39] U. S. Bhalla, "Molecular computation in neurons: A modeling perspective," *Current Opinion Neurobiol.*, vol. 25, pp. 31–37, Apr. 2014.

[40] C. Mead, "Neuromorphic electronic systems," *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct. 1990.

[41] P. Livi and G. Indiveri, "A current-mode conductance-based silicon neuron for address-event neuromorphic systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2009, pp. 2898–2901.

[42] J. Schemmel, L. Kriener, P. Müller, and K. Meier, "An accelerated analog neuromorphic hardware system emulating NMDA-and calcium-based non-linear dendrites," Mar. 2017, *arXiv:1703.07286*. [Online]. Available: http://arxiv.org/abs/1703.07286

[43] A. Graves, "Supervised sequence labelling," in *Supervised Sequence Labelling With Recurrent Neural Networks*. Berlin, Germany: Springer, 2012, pp. 5–13.

[44] O. Marschall, K. Cho, and C. Savin, "A unified framework of online learning algorithms for training recurrent neural networks," *J. Mach. Learn. Res.*, vol. 21, no. 135, pp. 1–34, 2019.

[45] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, Jun. 1989.

[46] J. Miller and M. Hardt, "Stable recurrent models," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–28.

[47] C. Tallec and Y. Ollivier, "Unbiased online recurrent optimization," Feb. 2017, *arXiv:1702.05043*. [Online]. Available: http://arxiv.org/abs/1702.05043

[48] T. Cooijmans and J. Martens, "On the variance of unbiased online recurrent optimization," Feb. 2019, *arXiv:1902.02405*. [Online]. Available: http://arxiv.org/abs/1902.02405

[49] A. Mujika, F. Meier, and A. Steger, "Approximating real-time recurrent learning with random kronecker factors," May 2018, *arXiv:1805.10842*. [Online]. Available: http://arxiv.org/abs/1805.10842

[50] J. Menick, E. Elsen, U. Evci, S. Osindero, K. Simonyan, and A. Graves, "A practical sparse approximation for real time recurrent learning," Jun. 2020, *arXiv:2006.07232*. [Online]. Available: http://arxiv.org/abs/2006.07232

[51] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks," Nov. 2017, *arXiv:1711.05136*. [Online]. Available: http://arxiv.org/abs/1711.05136

[52] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," Mar. 2018, *arXiv:1803.03635*. [Online]. Available: http://arxiv.org/abs/1803.03635

[53] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," Jun. 2020, *arXiv:2006.05467*. [Online]. Available: http://arxiv.org/abs/2006.05467

[54] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," Nov. 2019, *arXiv:1911.11134*. [Online]. Available: http://arxiv.org/abs/1911.11134

[55] T. Liu and F. Zenke, "Finding trainable sparse networks through neural tangent transfer," in *Proc. Int. Conf. Mach. Learn.*, vol. 1, 2020, pp. 6336–6347.

[56] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia, PA, USA: SIAM, 2008.

[57] U. Naumann, *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. Philadelphia, PA, USA: SIAM, 2011.

[58] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 5595–5637, 2017.

[59] L. Bottou and Y. LeCun, "SN: A simulator for connectionist models," in *Proc. NeuroNimes*, Nimes, France, vol. 88, 1988.

[60] R. Al-Rfou, "Theano: A Python framework for fast computation of mathematical expressions," May 2016, *arXiv:1605.02688*. [Online]. Available: https://arxiv.org/abs/1605.02688

[61] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://www.tensorflow.org

[62] A. Paszke *et al.*, "Automatic differentiation in Pytorch," in *Proc. NIPS Autodiff Workshop*, Long Beach, CA, USA, 2017.

[63] J. Revels, T. Besard, V. Churavy, B. De Sutter, and J. P. Vielma, "Dynamic automatic differentiation of GPU broadcast kernels," 2018, *arXiv:1810.08297*. [Online]. Available: http://arxiv.org/abs/1810.08297

[64] U. Naumann, "Optimal jacobian accumulation is NP-complete," *Math. Program.*, vol. 112, no. 2, pp. 427–441, Nov. 2007.

[65] C. Pozzorini, R. Naud, S. Mensi, and W. Gerstner, "Temporal whitening by power-law adaptation in neocortical neurons," *Nature Neurosci.*, vol. 16, no. 7, pp. 942–948, Jul. 2013.

[66] L. F. Abbott, B. DePasquale, and R.-M. Memmesheimer, "Building functional networks of spiking model neurons," *Nature Neurosci.*, vol. 19, no. 3, pp. 350–355, Mar. 2016.

[67] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers Neurosci.*, vol. 12, p. 774, Oct. 2018.

[68] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Netw.*, vol. 111, pp. 47–63, Mar. 2019.

[69] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers Neurosci.*, vol. 11, p. 682, Dec. 2017.

[70] D. Zambrano, R. Nusselder, H. Steven Scholte, and S. Bohte, "Efficient computation in adaptive artificial spiking neural networks," Oct. 2017, *arXiv:1710.04838*. [Online]. Available: http://arxiv.org/abs/1710.04838

[71] R. Kim, Y. Li, and T. J. Sejnowski, "Simple framework for constructing functional spiking recurrent neural networks," *PNAS*, vol. 116, pp. 22811–22820, 2019.

[72] C. Stöckl and W. Maass, "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes," Jan. 2020, *arXiv:2002.00860*. [Online]. Available: http://arxiv.org/abs/2002.00860

[73] D. Ackley, G. Hinton, and T. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognit. Sci., Multidisciplinary J.*, vol. 9, no. 1, pp. 147–169, 1985.

[74] H. Jang, O. Simeone, B. Gardner, and A. Gruning, "An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 64–77, Nov. 2019.

[75] G. Bellec, M. Galtier, R. Brette, and P. Yger, "Slow feature analysis with spiking neurons and its application to audio stimuli," *J. Comput. Neurosci.*, vol. 40, no. 3, pp. 317–329, Jun. 2016.

[76] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1440–1450.

[77] B. Yin, F. Corradi, and S. M. Bohté, "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," Jun. 2020, *arXiv:2005.11633*. [Online]. Available: http://arxiv.org/abs/2005.11633

[78] F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural

networks," *bioRxiv*, Jun. 2020, doi: 10.1101/2020.06.29.176925.
[79] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[80] H. Jang, N. Skatchkovsky, and O. Simeone, "VOWEL: A local online learning rule for recurrent networks of probabilistic spiking winner-take-all circuits," Apr. 2020, *arXiv:2004.09416*. [Online]. Available: http://arxiv.org/abs/2004.09416
[81] J. Werfel, X. Xie, and H. S. Seung, "Learning curves for stochastic gradient descent in linear feedforward networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 1197–1204.
[82] D. Goodman and R. Brette, "Brian: A simulator for spiking neural networks in Python," *Frontiers Neuroinform.*, vol. 2, p. 5, Nov. 2008.
[83] F. Zenke and W. Gerstner, "Limits to high-speed simulations of spiking neural networks using general-purpose computers," *Frontiers Neuroinform.*, vol. 8, p. 76, Sep. 2014.
[84] M. Ercsey-Ravasz *et al.*, "A predictive network model of cerebral cortical connectivity based on a distance rule," *Neuron*, vol. 80, no. 1, pp. 184–197, Oct. 2013.
[85] B. U. Pedroni *et al.*, "Memory-efficient synaptic connectivity for spike-timing-dependent plasticity," *Frontiers Neurosci.*, vol. 13, p. 357, Apr. 2019.
[86] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.
[87] J. Park, T. Yu, S. Joshi, C. Maier, and G. Cauwenberghs, "Hierarchical address event routing for reconfigurable large-scale neuromorphic systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2408–2422, Oct. 2017.
[88] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger, "CondenseNet: An efficient DenseNet using learned group convolutions," vol. 3, 2017, *arXiv:1711.09224*. [Online]. Available: http://arxiv.org/abs/1711.09224
[89] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," Feb. 2019, *arXiv:1810.02340*. [Online]. Available: http://arxiv.org/abs/1810.02340
[90] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg spiking datasets for the systematic evaluation of spiking neural networks," Dec. 2019, *arXiv:1910.07407*. [Online]. Available: http://arxiv.org/abs/1910.07407
[91] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," Apr. 2018, *arXiv:1804.03209*. [Online]. Available: http://arxiv.org/abs/1804.03209
[92] J. M. Murray, "Local online learning in recurrent networks with random feedback," *eLife*, vol. 8, May 2019, Art. no. e43299.
[93] J. M. Murray, "Local online learning in recurrent networks with random feedback," *ELife*, vol. 8, p. e43299, 2019.
[94] N. Kalchbrenner *et al.*, "Efficient neural audio synthesis," Jun. 2018, *arXiv:1802.08435*. [Online]. Available: http://arxiv.org/abs/1802.08435
[95] J. Christian Thiele, O. Bichler, and A. Dupret, "SpikeGrad: An ANN-equivalent computation model for implementing backpropagation with spikes," 2019, *arXiv:1906.00851*. [Online]. Available: http://arxiv.org/abs/1906.00851
[96] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers Neurosci.*, vol. 14, p. 119, Feb. 2020.
[97] T. Bohnstingl, S. Woźniak, W. Maass, A. Pantazi, and E. Eleftheriou, "Online spatio-temporal learning in deep neural networks," 2020, *arXiv:2007.12723*. [Online]. Available: http://arxiv.org/abs/2007.12723
[98] D. Sussillo and L. F. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, no. 4, pp. 544–557, Aug. 2009.
[99] W. Nicola and C. Clopath, "Supervised learning in spiking neural networks with FORCE training," *Nature Commun.*, vol. 8, no. 1, p. 2208, Dec. 2017.
[100] A. Gilra and W. Gerstner, "Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network," *eLife*, vol. 6, Nov. 2017, Art. no. e28295.
[101] R. Gütig, "Spiking neurons can discover predictive features by aggregate-label learning," *Science*, vol. 351, no. 6277, Mar. 2016, Art. no. aab4113.
[102] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci, "Online few-shot gesture learning on a neuromorphic processor," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 4, pp. 512–521, Dec. 2020.
[103] M. C. W. V. Rossum, "A novel spike distance," *Neural Comput.*, vol. 13, no. 4, pp. 751–763, Apr. 2001.
[104] B. Gardner and A. Grüning, "Supervised learning in spiking neural networks for precise temporal encoding," *PLoS ONE*, vol. 11, no. 8, Aug. 2016, Art. no. e0161335.
[105] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, "Eligibility traces and plasticity on behavioral time scales: Experimental support of neoHebbian three-factor learning rules," Jan. 2018, *arXiv:1801.05219*. [Online]. Available: http://arxiv.org/abs/1801.05219
[106] E. M. Izhikevich, "Solving the distal reward problem through linkage of STDP and dopamine signaling," *Cerebral Cortex*, vol. 17, no. 10, pp. 2443–2452, Oct. 2007.
[107] Z. Pan, Y. Chua, J. Wu, M. Zhang, H. Li, and E. Ambikairajah, "An efficient and perceptually motivated auditory neural encoding and decoding algorithm for spiking neural networks," Sep. 2019, *arXiv:1909.01302*. [Online]. Available: http://arxiv.org/abs/1909.01302
[108] R. Brette *et al.*, "Simulation of networks of spiking neurons: A review of tools and strategies," *J. Comput. Neurosci.*, vol. 23, no. 3, pp. 349–398, Dec. 2007.
[109] M. Payvand, M. E. Fouda, F. Kurdahi, A. Eltawil, and E. O. Neftci, "Error-triggered three-factor learning dynamics for crossbar arrays," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 218–222.
[110] S. K. Esser *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *PNAS*, vol. 113, no. 41, pp. 11441–11446, 2016.
[111] E. Neftci, E. Chicca, G. Indiveri, and R. Douglas, "A systematic method for configuring VLSI networks of spiking neurons," *Neural Comput.*, vol. 23, no. 10, pp. 2457–2497, Oct. 2011.
[112] D. Brüderle *et al.*, "A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems," *Biol. Cybern.*, vol. 104, nos. 4–5, pp. 263–296, May 2011.
[113] B. Cramer *et al.*, "Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate," 2020, *arXiv:2006.07239*. [Online]. Available: http://arxiv.org/abs/2006.07239
[114] R. C. O'Reilly and M. J. Frank, "Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia," *Neural Comput.*, vol. 18, no. 2, pp. 283–328, Feb. 2006.
[115] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[116] D. Amit, *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1992.
[117] A. Litwin-Kumar and B. Doiron, "Formation and maintenance of neuronal assemblies through synaptic plasticity," *Nature Commun.*, vol. 5, no. 1, pp. 1–12, Nov. 2014.
[118] F. Zenke, E. J. Agnes, and W. Gerstner, "Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks," *Nature Commun.*, vol. 6, no. 1, p. 6922, Apr. 2015.
[119] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[120] U. Rutishauser, R. J. Douglas, and J.-J. Slotine, "Collective stability of networks of winner-take-all circuits," *Neural Comput.*, vol. 23, no. 3, pp. 735–773, Mar. 2011.
[121] S. Grossberg, *The Adaptive Brain I*. Amsterdam, The Netherlands: Elsevier, 1987.
[122] P. Baldi, P. Sadowski, and Z. Lu, "Learning in the machine: Random backpropagation and the deep learning channel," *Artif. Intell.*, vol. 260, pp. 1–35, Jul. 2018.
[123] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Commun.*, vol. 7, no. 1, p. 13276, Dec. 2016.
[124] J. F. Kolen and J. B. Pollack, "Backpropagation without weight transport," in *Proc. IEEE Int. Conf. Neural Netw. (ICNN)*, vol. 3, Jun. 1994, pp. 1375–1380.
[125] M. Akrout, C. Wilson, P. Humphreys, T. Lillicrap, and D. B. Tweed, "Deep learning without weight transport," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. D. Alché-Buc, E. Fox, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2019, pp. 976–984.
[126] Y. Amit, "Deep learning with asymmetric connections and Hebbian updates," *Frontiers Comput. Neurosci.*, vol. 13, p. 18, Apr. 2019.
[127] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," Jan. 2019, *arXiv:1901.06656*. [Online]. Available: https://arxiv.org/abs/1901.06656
[128] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers Neurosci.*, vol. 12, p. 608, Aug. 2018.