CONTRIBUTED PAPER

Training Spiking Neural Networks Using Lessons From Deep Learning

This article serves as a tutorial and perspective showing how to apply the lessons learned from several decades of research in deep learning, gradient descent, backpropagation, and neuroscience to biologically plausible spiking neural networks.

By Jason K. Eshraghian, *Member IEEE*, Max Ward, Emre O. Neftci, *Member IEEE*, Xinxin Wang, *Student Member IEEE*, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, *Senior Member IEEE*, Doo Seok Jeong, *Member IEEE*, and Wei D. Lu, *Fellow IEEE*

ABSTRACT | The brain is the perfect place to look for inspiration to develop more efficient neural networks. The inner workings of our synapses and neurons provide a glimpse at what the future of deep learning might look like. This article serves as a tutorial and perspective showing how to apply the lessons learned from several decades of research in deep learning, gradient descent, backpropagation, and neuroscience to biologically plausible spiking neural networks (SNNs). We also explore the delicate interplay between encoding data as spikes and the learning process; the challenges and solutions of applying gradient-based learning to SNNs; the subtle link between temporal backpropagation and spike timing-dependent plasticity; and how deep learning might move toward biologically plausible online learning. Some ideas are well accepted and commonly used among the neuromorphic engineering community, while others are presented or justified for the first time here. A series of companion interactive tutorials complementary to this article using our Python package, *snnTorch*, are also made available: https://snntorch.readthedocs.io/en/latest/tutorials/index.html.

KEYWORDS | Deep learning; neural code; neuromorphic; online learning; spiking neural networks (SNNs).

Manuscript received 9 November 2022; revised 13 August 2023; accepted 15 August 2023. Date of publication 6 September 2023; date of current version 15 September 2023. The work of Wei D. Lu was supported by the National Science Foundation under Award ECCS-1915550. *(Corresponding author: Jason K. Eshraghian.)*

Jason K. Eshraghian was with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA. He is now with the Department of Electrical and Computer Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: Jeshragh@ucsc.edu).

Max Ward was with the Department of Molecular and Cellular Biology, Harvard University, Cambridge, MA 02138 USA. He is now with the Department of Computer Science and Software Engineering, The University of Western Australia, Crawley, WA 6009, Australia.

Emre O. Neftci is with the Department of Computer Science and the Department of Cognitive Sciences, University of California at Irvine (UC Irvine), Irvine, CA 92697 USA, with the Peter Grünberg Institute—Neuromorphic Software Ecosystems, Forschungszentrum Jülich, 52428 Jülich, Germany, and also with the Faculty of Electrical Engineering and Information Technology, RWTH Aachen University, 52062 Aachen, Germany.

Xinxin Wang and **Wei D. Lu** are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA.

Gregor Lenz is with SynSense AG, 8050 Zürich, Switzerland.

Girish Dwivedi is with the School of Medicine, The University of Western Australia, Crawley, WA 6009, Australia.

Mohammed Bennamoun is with the Department of Computer Science and Software Engineering, The University of Western Australia, Crawley, WA 6009, Australia.

Doo Seok Jeong is with the Division of Materials Science and Engineering, Hanyang University, Seoul 04763, South Korea.

Digital Object Identifier 10.1109/JPROC.2023.3308088

I. INTRODUCTION

Deep learning has solved numerous problems in computer vision [1], [2], [3], [4], [5], [6], speech recognition [7], [8], [9], and natural language processing [10], [11], [12], [13], [14]. Neural networks have been instrumental in outperforming world champions in a diverse range of games from Go to Starcraft [15], [16], and they are now surpassing the diagnostic capability of clinical specialists in numerous medical tasks [17], [18], [19], [20], [21]. However, for all the state-of-the-art models designed every day, a Kaggle [22] contest for state-of-the-art energy efficiency would go to the brain, every time. A new generation of brain-inspired spiking neural networks (SNNs) is poised to bridge this efficiency gap.

The amount of computational power required to run top-performing deep learning models has increased at a rate of 10× per year from 2012 to 2019 [23], [24]. The rate of data generation is likewise increasing at an exponential rate. OpenAI's language model, GPT-3, contains 175 billion learnable parameters, estimated to consume roughly 190 000 kWh to train [25], [26], [27]. Meanwhile, our brains operate within ∼12–20 W of power. This is in addition to churning through a multitude of sensory inputs, all the while ensuring that our involuntary biological processes do not shut down [28]. If our brains dissipated as much heat as state-of-the-art deep learning models, then natural selection would have wiped humanity out long before we could have invented machine learning. To be fair, none of the authors can emulate the style of Shakespeare or write up musical guitar tabs with the same artistic flair as ChatGPT.

A. Neuromorphic Computing: A Quick Snapshot

Neuromorphic ("brain-like") engineering strives to imitate the computational principles of the brain to drive down the energy cost of artificial intelligence systems. To replicate a biological system, we build on three parts.

1) **Neuromorphic sensors** that take inspiration from biological sensors, such as the retina or cochlear, and typically record *changes* in a signal instead of sampling it at regular intervals. Signals are only generated when a change occurs, and the signal is referred to as a "spike."
2) **Neuromorphic algorithms** that learn to make sense of spikes are known as SNNs. Instead of floating point values, SNNs work with single-bit, binary activations (spikes) that encode information over time, rather than in an intensity. As such, SNNs take advantage of low-precision parameters and high spatial and temporal sparsity.
3) These models are designed with power-efficient execution on specialized **neuromorphic hardware** in mind. Sparse activations reduce data movement both on and off a chip to accelerate neuromorphic workloads, which can lead to large power and latency gains compared to the same task on conventional hardware.

Armed with these three components, neuromorphic systems are equipped to bridge the efficiency gap between today's and future intelligent systems.

What lessons can be learned from the brain to build more efficient neural networks? Should we replicate the genetic makeup of a neuron right down to the molecular level [29], [30]? Do we look at the way memory and processing coalesce within neurons and synapses [31], [32]? Or should we aim to extract the learning algorithms that underpin the brain [33]? This article hones in on the intricacies of training brain-inspired neuromorphic algorithms, ultimately moving toward the goal of harnessing natural intelligence to further improve our use of artificial intelligence. SNNs can already be optimized using the tools available to the deep learning community. However, the brain-inspired nature of these emerging sensors, neuron models, and training methods is different enough to warrant a deep dive into biologically inspired neural networks.

B. Neuromorphic Systems in the Wild

The overarching aim is to combine artificial neural networks (ANNs), which have already proven their worth in a broad range of domains, with the potential efficiency of SNNs [34]. So far, SNNs have staked their claim to a range of applications where power efficiency is of utmost importance.

Fig. 1 offers a small window into the uses of SNNs, and their domain only continues to expand. Spiking algorithms have been used to implement low-power artificial intelligence algorithms across the medical, robotics, and mixed-reality domains, among many other fields. Given their power efficiency, initial commercial products often target edge computing applications, close to where the data are recorded.

In biosignal monitoring, nerve implants for the brain–machine or biosignal interfaces have to preprocess information locally at minimum power and lack the bandwidths to transmit data for cloud computation. Work done in that direction using SNNs includes on-chip spike sorting [35], [36], biosignal anomaly detection [37], [38], [39], [40], and brain–machine interfaces [41], [42], [43]. Beyond biomedical intervention, SNN models are also used in robotics in an effort to make them more human-like and to drive down the cost of operation [44], [45], [46]. Unmanned aerial vehicles must also operate in low-power environments to extract as much value from lightweight batteries and have benefited from using neuromorphic processors [47]. Audio signals can be processed with sub-milliwatt (mW) power consumption and low latency on neuromorphic hardware as SNNs provide an efficient computational mechanism for temporal signal processing [48].

A plethora of efficient computer vision applications using SNNs are reviewed in [49]. SNNs are equally suitable to track objects such as satellites in the sky for space situational awareness [50], [51] and have been researched to promote sustainable uses of artificial intelligence, such as in monitoring material strain in smart buildings [52] and wind power forecasting in remote areas that face power delivery challenges [53]. At the 2018–19 Telluride Neuromorphic and Cognition Workshops, a neuromorphic robot was even built to play foosball! [54].

Beyond neuromorphic applications, SNNs are also used to test theories about how natural intelligence may arise, from the higher level learning rules of the brain [55] and how memories are formed [56] down to the lower level dynamics at the neuronal and synaptic layers [57].

C. Overview of This Article

The brain's neural circuitry is a physical manifestation of its neural algorithm; understanding one will likely lead

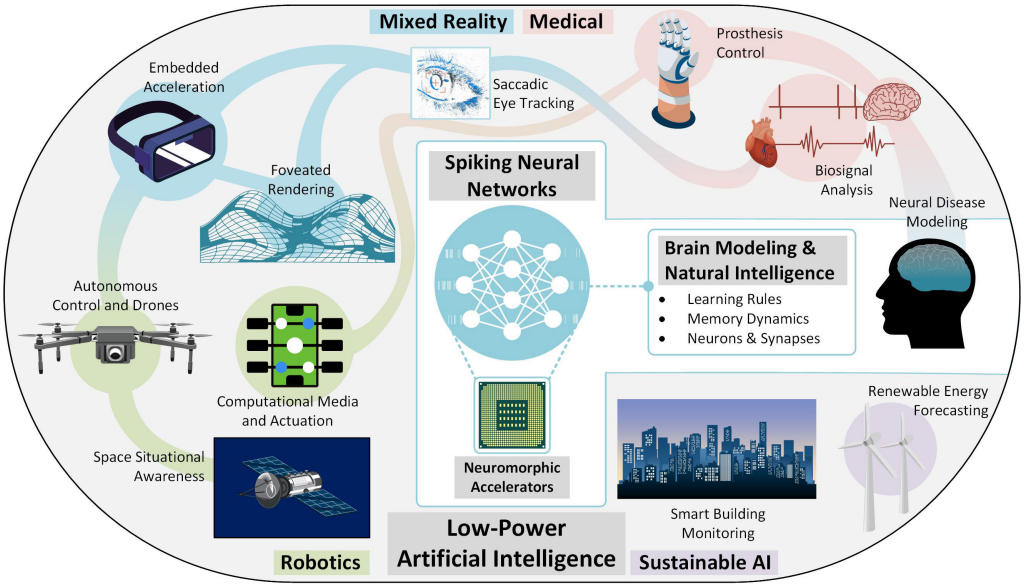


Fig. 1. ***SNNs have pervaded many streams of deep learning, which are in need of low-power, resource-constrained, and often portable operation. The utility of SNNs even extends to the modeling of neural dynamics across individual neurons and higher level neural systems.***

to an understanding of the other. This article will hone in on one particular aspect of neural models: those that are compatible with modern deep learning. Fig. 2 provides an illustrated overview of the structure of this article, and we will start from the ground up.

1) In Section II, we will rationalize the commonly accepted advantages of using spikes and derive a spiking neuron model from basic principles.
2) These spikes are assigned meaning in Section III by exploring various spike encoding strategies, how they impact the learning process, and how objective and regularization functions are used to sway the spiking patterns of an SNN.
3) In Section IV, the challenges of training SNNs using gradient-based optimization are explored, and several solutions are derived. These include defining derivatives at spike times and using approximations of the gradient.
4) In doing so, a subtle link between the backpropagation algorithm and the spike timing-dependent

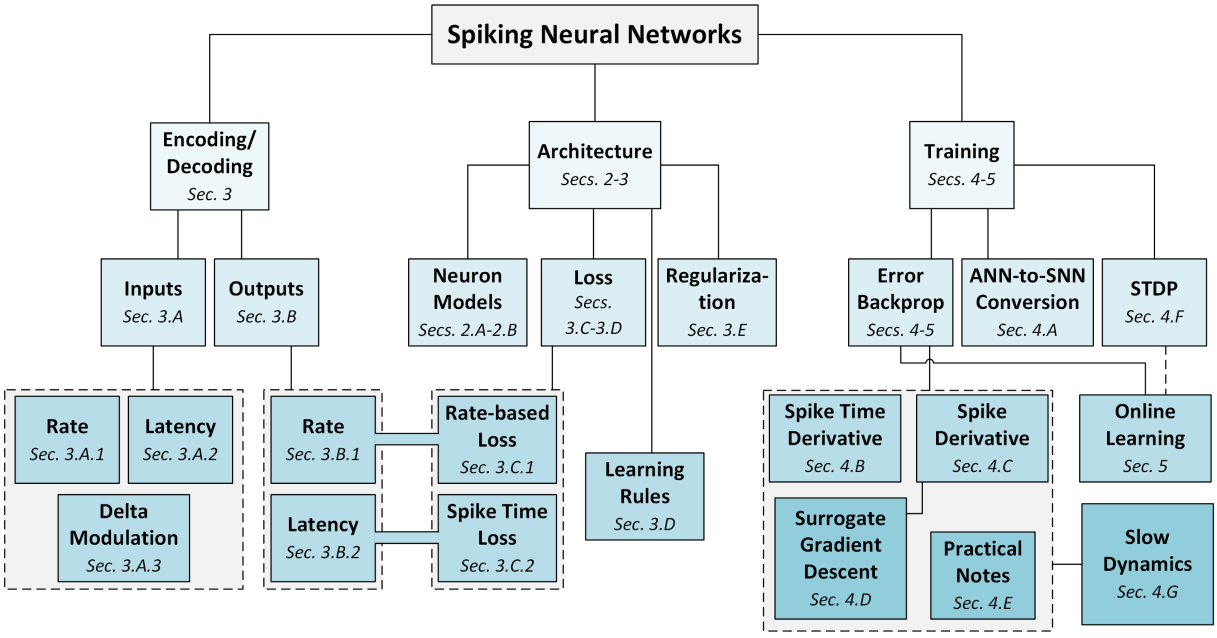


Fig. 2. ***Overview of this article's structure.***

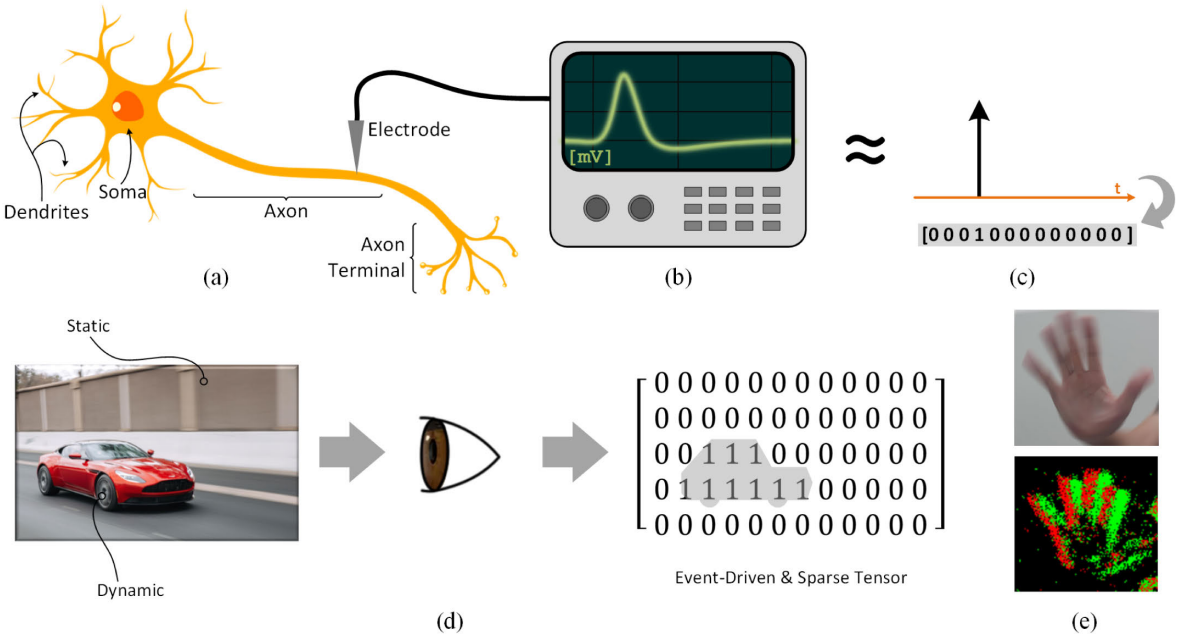


Fig. 3. *Neurons communicate via spikes. (a) Diagram of a neuron. (b) Measuring an action potential propagated along the axon of a neuron. Fluctuating subthreshold voltages are present in the soma but become severely attenuated over distances beyond 1 mm [58]. Only the action potential is detectable along the axon. (c) Neuron's spike is approximated with a binary representation. (d) Event-driven processing. Only dynamic segments of a scene are passed to the output ("1"), while static regions are suppressed ("0"). (e) Active pixel sensor and DVS.*

plasticity (STDP) learning rule emerges and is used in the subsequent section to derive online variants of backprop, which move toward biologically plausible learning mechanisms.

With that being said, it is time to dive into how we might combine the potential efficiency of SNNs with the high performance of ANNs.

II. FROM ARTIFICIAL TO SPIKING NEURAL NETWORKS

The neural code refers to how the brain represents information, and while many theories exist, the code is yet to be cracked. There are several persistent themes across these theories, which can be distilled down to *"the three S's"*: spikes, sparsity, and static suppression. These traits are a good starting point to show *why* the neural code might improve the efficiency of ANNs. Our first observation is given as follows.

1) *Spikes (biological neurons interact via spikes):* Neurons primarily process and communicate with action potentials or "spikes," which are electrical impulses of approximately 100 mV in amplitude. In most neurons, the occurrence of an action potential is far more important than the subtle variations of the action potential [58]. Many computational models of neurons simplify the representation of a spike to a discrete, single-bit, all-or-nothing event [see Fig. 3(a)–(c)]. Communicating high-precision activations between layers and routing them around and between chips are expensive undertakings. Multiplying high-precision activation with a high-precision weight requires conversion into integers and decomposition of multiplication into multiple additions, which introduces a carry propagation delay. On the other hand, a spike-based approach only requires a weight to be multiplied by a spike ("1"). This trades the cumbersome multiplication process with a simple memory read-out of the weight value.
   Despite the activation being constrained to a single bit, spiking networks are vastly different from binarized neural networks. What actually matters is the *timing* of the spike. Time is not a binarized quantity and can be implemented using clock signals that are already distributed across a digital circuit. After all, why not use what is already available?
2) *Sparsity:* Biological neurons spend most of their time at rest, silencing a majority of activations to *zero* at any given time.
   Sparse tensors are cheap to store. The space that a simple data structure requires to store a matrix grows with the number of entries to store. In contrast, a data structure to store a sparse matrix only consumes memory with the number of nonzero elements. Take the following list as an example:

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1].$$

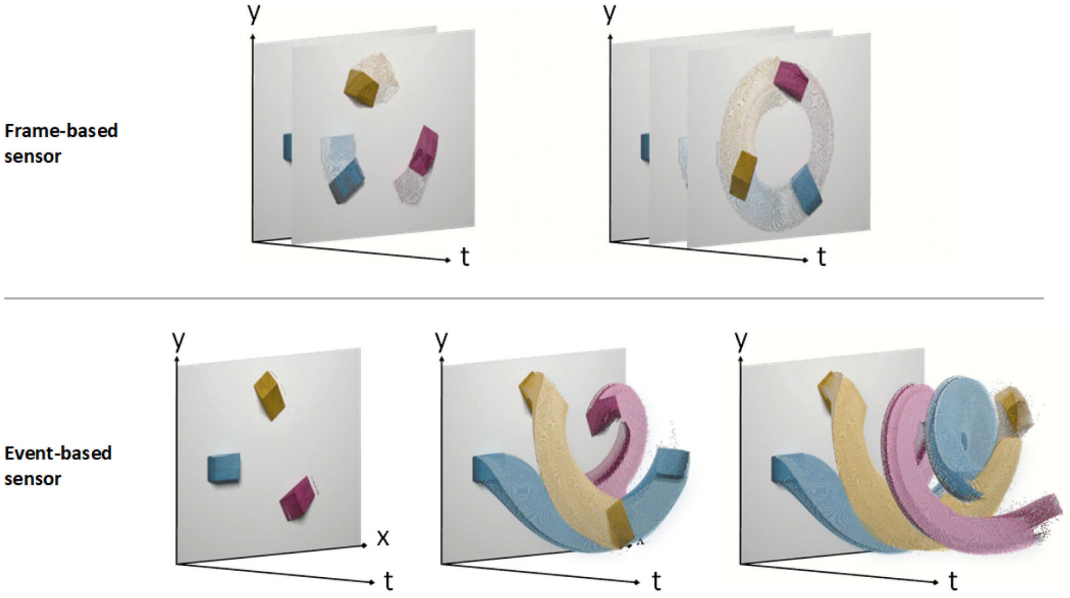


Fig. 4. *Functional difference between a conventional frame-based camera (top) and an event-based camera/silicon retina (bottom). The former records the scene as a sequence of images at a fixed frame rate. It operates independently of activity in the scene and can result in motion blur due to the global shutter. The silicon retina's output is directly driven by visual activity in the scene, as every pixel reacts to a change in illuminance.*

Since most of the entries are zero, writing out only the nonzero elements is a far more efficient representation, as would occur in run-length encoding (indexing from zero)

"1 at position 10; 1 at position 20."

For example, Fig. 3(c) shows how a single action potential can be represented by a sparsely populated vector. The sparser the list, the more space can be saved.

3) *Static suppression (a.k.a., event-driven processing):* The sensory system is more responsive to changes than to static input.

The sensory periphery features several mechanisms that promote neuron excitability when subject to dynamic, changing stimuli, while suppressing its response to static, unchanging information. In retinal ganglion cells and the primary visual cortex, the spatiotemporal receptive fields of neurons promote excitable responses to regions of spatial contrast (or edges) over regions of spatial invariance [59]. Analogous mechanisms in early auditory processing include spectrotemporal receptive fields that cause neurons to respond more favorably to changing frequencies in sound over static frequencies [60]. These processes occur on short timescales (milliseconds), while perceptual adaptation has also been observed on longer timescales (seconds) [61], [62], [63], causing neurons to become less responsive to prolonged exposure to fixed stimuli.

A real-world engineering example of event-driven processing is the dynamic vision sensor (DVS), or the "silicon retina," which is a camera that reports changes in brightness and stays silent otherwise [see Fig. 3(d) and (e)] [64], [65], [66], [67], [68]. This also means that each pixel activates independently of all other pixels, as opposed to waiting for a global shutter to produce a still frame. The reduction of active pixels leads to huge energy savings compared to conventional CMOS image sensors. This mix of low-power and asynchronous pixels allows for fast clock speeds, giving commercially available DVS cameras a microsecond temporal resolution without breaking a sweat [69]. The difference between a conventional frame-based camera and an event-based camera is illustrated in Fig. 4.

A. Spiking Neurons

ANNs and SNNs can model the same types of network topologies, but SNNs trade the artificial neuron model with a spiking neuron model instead (see Fig. 5). Much like the artificial neuron model [70], spiking neurons operate on a weighted sum of inputs. Rather than passing the result through a sigmoid or rectified linear unit (ReLU) nonlinearity, the weighted sum contributes to the membrane potential $U(t)$ of the neuron. If the neuron is sufficiently excited by this weighted sum, and the membrane potential reaches a threshold θ, then the neuron will emit a spike to its subsequent connections. However, most neuronal inputs are spikes of very short bursts of electrical activity. It is quite unlikely for all input spikes to arrive at the neuron body in unison [see Fig. 5(c)]. This indicates the presence

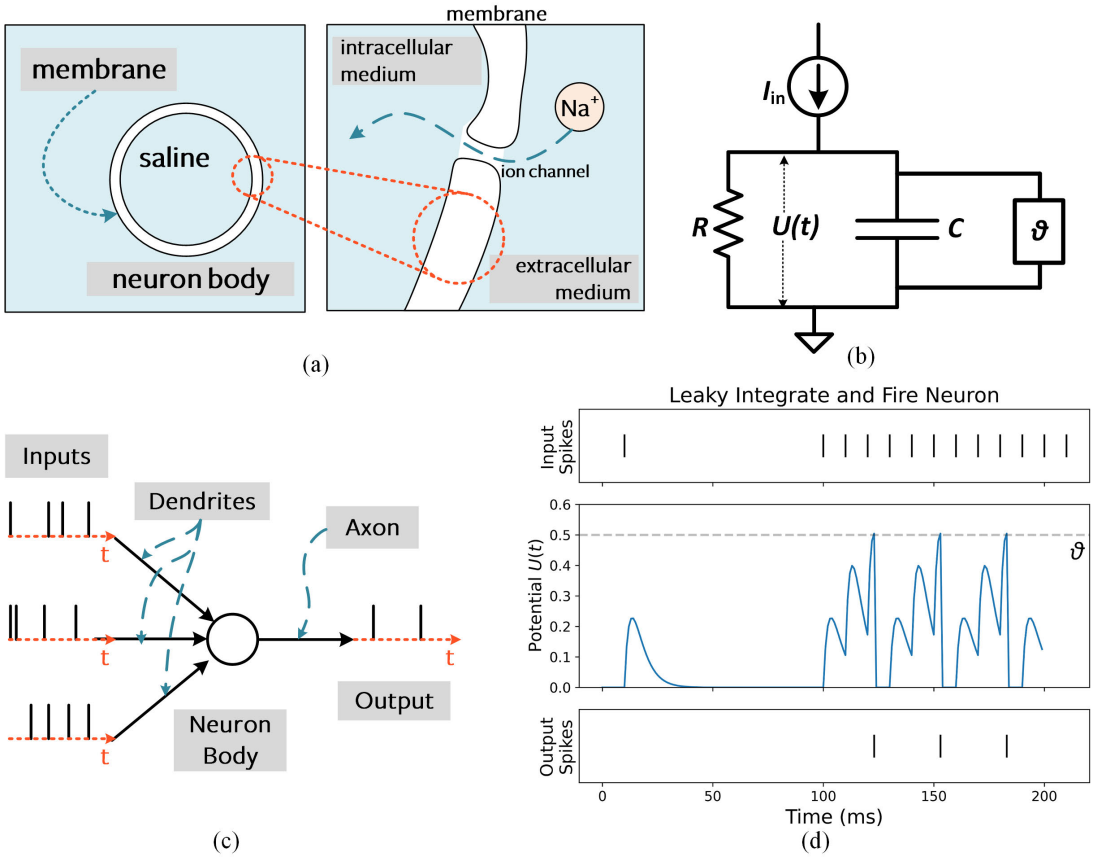


Fig. 5. ***Leaky IF neuron model. (a) Insulating bilipid membrane separates the intracellular and extracellular medium. Gated ion channels allow charge carriers, such as Na^+, to diffuse through the membrane. (b) Capacitive membrane and resistive ion channels form an RC circuit. When the membrane potential exceeds a threshold θ, a spike is generated. (c) Input spikes generated by I_{in} are passed to the neuron body via the dendritic tree. Sufficient excitation will cause spike emission at the output. (d) Simulation depicting the membrane potential U(t) reaching the threshold, arbitrarily set to $\theta = 0.5$ V, which generates output spikes.***

of temporal dynamics that "sustain" the membrane potential over time.

These dynamics were quantified back in 1907 [75]. Lapicque [75] stimulated the nerve fiber of a frog leg using a hacked-together current source and observed how long it took the frog leg to twitch based on the amplitude and duration of the driving current I_{in} [76]. He concluded that a spiking neuron coarsely resembles a low-pass filter circuit consisting of a resistor R and a capacitor C, later dubbed the leaky integrate-and-fire (IF) neuron [see Fig. 5(b)]. This holds up a century later: physiologically, the capacitance arises from the insulating lipid bilayer forming the membrane of a neuron. The resistance arises from gated ion channels that open and close, modulating charge carrier diffusion across the membrane [see Fig. 5(a) and (b)] [77]. The dynamics of the passive membrane modeled using an RC circuit can be represented as

$$\tau \frac{dU(t)}{dt} = -U(t) + I_{\text{in}}(t)R \tag{1}$$

where $\tau = RC$ is the time constant of the circuit. Typical values of τ fall on the order of 1–100 ms. The solution of (1) for a constant current input is

$$U(t) = I_{\text{in}}R + [U_0 - I_{\text{in}}R]e^{-\frac{t}{\tau}} \tag{2}$$

which shows how exponential relaxation of $U(t)$ to a steady-state value follows current injection, where U_0 is the initial membrane potential at $t = 0$. To make this time-varying solution compatible with a sequence-based neural network, the forward Euler method is used in the simplest case to find an approximate solution to (1)

$$U[t] = \beta U[t-1] + (1-\beta)I_{\text{in}}[t] \tag{3}$$

where time is explicitly discretized, $\beta = e^{-1/\tau}$ is the decay rate (or "inverse time constant") of $U[t]$, and the full derivation is provided in Appendix A1.

In deep learning, the weighting factor of an input is typically a learnable parameter. Relaxing the physically viable assumptions made thus far, the coefficient of input current in (3), $(1-\beta)$, is subsumed into a learnable weight W, and the simplification $I_{\text{in}}[t] = WX[t]$ is made

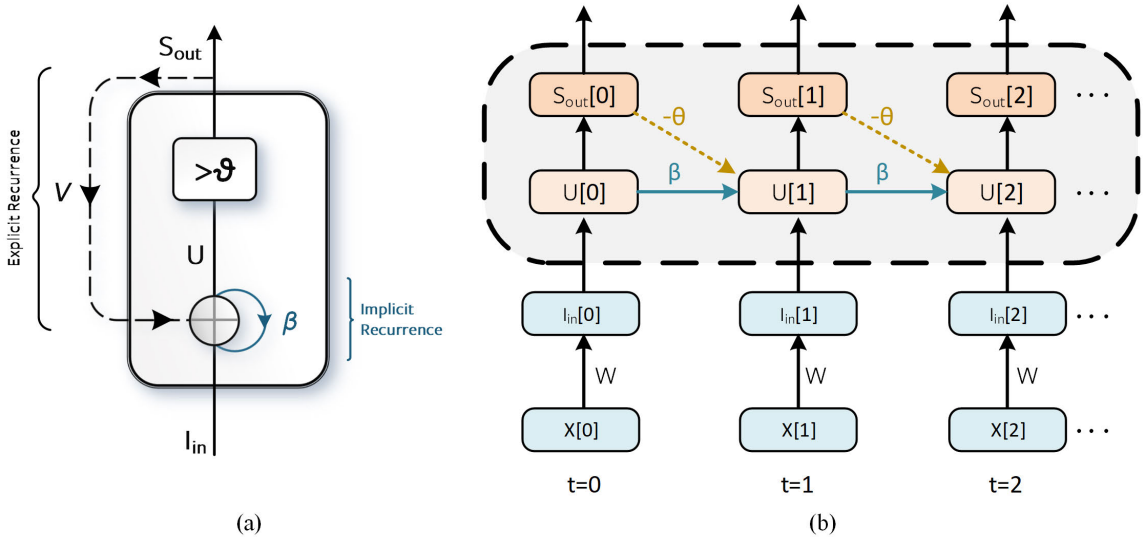


Fig. 6. *Computational steps in solving the leaky IF neuron model. (a) Recurrent representation of a spiking neuron. Hidden state decay is referred to as "implicit recurrence," and external feedback from the spike is "explicit recurrence," where V is the recurrent weight [omitted from (4)] [71]. (b) Unrolled computational graph of the neuron where time flows from left to right. $-\theta$ represents the reset term from (4), while β is the decay rate of U over time. Explicit recurrence is omitted for clarity. Note that kernel-based neuron models replace implicit recurrence with a time-varying filter [72], [73], [74].*

to decouple the effect of β on the input $X[t]$. Here, $X[t]$ is treated as a single input. A full-scale network would vectorize $X[t]$, and W would be a matrix but is treated here as a single input to a single neuron for simplicity. Finally, accounting for spiking and membrane potential reset gives

$$U[t] = \underbrace{\beta U[t-1]}_{\text{decay}} + \underbrace{WX[t]}_{\text{input}} - \underbrace{S_{\text{out}}[t-1]\theta}_{\text{reset}}. \tag{4}$$

$S_{\text{out}}[t] \in \{0, 1\}$ is the output spike generated by the neuron, where, if activated ($S_{\text{out}} = 1$), the reset term subtracts the threshold θ from the membrane potential. Otherwise, the reset term has no effect ($S_{\text{out}} = 0$). A complete derivation of (4) with all simplifying assumptions is provided in Appendix A1. A spike is generated if the membrane potential exceeds the threshold

$$S_{\text{out}}[t] = \begin{cases} 1, & \text{if } U[t] > \theta \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

In your exploration of leaky IF neurons, you may come across slight variants.

1) The spike threshold may be applied *before* updating the membrane potential. This induces a one-step delay between the input signal X when it can trigger a spike.
2) The above derivations use a "reset-by-subtraction" (or *soft reset*) mechanism. However, an alternative shown in Appendix A1 is a "reset-to-zero" mechanism (or *hard reset*).
3) The factor $(1 - \beta)$ from (3) may be included as a coefficient to the input term, WX. This will allow you to simulate a neuron model with realistic time constants but does not offer any advantages when ultimately applied to deep learning.

An extensive list of alternative neuron types is detailed in Section II-B, along with a brief overview of their use cases.

A graphical depiction of the LIF neuron is provided in Fig. 6. The recurrent neuron in (a) is "unrolled" across time steps in (b), where the reset mechanism is included via $S_{\text{out}}\theta$, and explicit recurrence $S_{\text{out}}V$ is omitted for brevity. The unrolled computational graph is a good way to think about leaky IF neurons. As you will soon see in Section IV, this unrolled representation allows us to borrow many of the tricks developed by deep learning researchers in training networks of spiking neurons.

B. Alternative Spiking Neuron Models

The leaky IF neuron is one of the many spiking neuron models. Some other models that you might encounter are listed as follows.

1) *IF:* The leakage mechanism is removed; $\beta = 1$ in (4).
2) *Current-based:* Often referred to as CuBa neuron models, these incorporate synaptic conductance variation into leaky IF neurons. If the default LIF neuron is a first-order low-pass filter, then CuBa neurons are a second-order low-pass filter. The input spike train undergoes two rounds of "smoothing," which means that the membrane potential has a finite rise time rather than experiencing discontinuous jumps in response to incoming spikes [78], [79], [80].

A depiction of such a neuron with a finite rise time of membrane potential is shown in Fig. 5(d).

3) *Recurrent neurons:* The output spikes of a neuron are routed back to the input, labeled in Fig. 6(a) with explicit recurrence. Rather than an alternative model, recurrence is a topology that can be applied to any other neuron and can be implemented in different ways: one-to-one recurrence, where each neuron routes its own spike to itself, or all-to-all recurrence, where the output spikes of a full layer are weighted and summed (e.g., via a dense or convolutional layer), before being fed back to the full layer [81].
4) *Kernel-based models:* Also known as the spike-response model, where a predefined kernel (such as the "alpha function": see Appendix C1) is convolved with input spikes [72], [73], [74]. Having the option to define the kernel to be any shape offers significant flexibility.
5) *Deep learning inspired spiking neurons:* Rather than drawing upon neuroscience, it is just as possible to start with primitives from deep learning and apply spiking thresholds. This helps with extending the short-term capacity of basic recurrent neurons. A couple of examples include spiking LSTMs [39] and Legendre memory units [82]. More recently, transformers have been used to further improve long-range memory dependencies in data. In a similar manner, SpikeGPT approximated self-attention into a recurrent model, providing the first demonstration of natural language generation with SNNs [83].
6) *Higher complexity neuroscience-inspired models:* A large variety of more detailed neuron models are out there. These account for biophysical realism and/or morphological details that are not represented in simple leaky integrators. The most renowned models include the Hodgkin–Huxley model [77] and the Izhikevich (or resonator) model [84], which can reproduce electrophysiological results with better accuracy.

The main takeaway is given as follows: use the neuron model that suits your task. Power-efficient deep learning will call for LIF models. Improving performance may call for using recurrent SNNs. Driving performance even further (often at the expense of efficiency) may demand methods derived from deep learning, such as spiking LSTMs and recurrent spiking transformers [83]. Or perhaps, deep learning is not your goal. If you are aiming to construct a brain model or are tasked with an exploration of linking low-level dynamics (ionic, conductance-driven, or otherwise) with higher order brain function, then perhaps, more detailed, biophysically accurate models will be your friend.

Having formulated a spiking neuron in a discrete-time, recursive form, we can now "borrow" the developments in training recurrent neural networks (RNNs) and sequence-based models. This recursion is illustrated using an "implicit" recurrent connection for the decay of the membrane potential and is distinguished from "explicit" recurrence where the output spikes S_{out} are fed back to the input as in recurrent SNNs (see Fig. 6).

While there are plenty more physiologically accurate neuron models [77], the leaky IF model is the most prevalent in gradient-based learning due to its computational efficiency and ease of training. Before moving onto training SNNs in Section IV, let us gain some insight into what spikes actually mean and how they might represent information in Section III.

III. NEURAL CODE

Light is what we see when the retina converts photons into spikes. Odors are what we smell when the nose processes volatilized molecules into spikes. Tactile perceptions are what we feel when our nerve endings turn pressure into spikes. The brain trades in the global currency of the *spike*. If all spikes are treated identically, then how do they carry meaning? With respect to spike encoding, there are two parts of a neural network that must be treated separately (see Fig. 7).

1) *Input encoding:* Conversion of input data into spikes which is then passed to a neural network.
2) *Output decoding:* Train the output of a network to spike in a way that is meaningful and informative.

A. Input Encoding

Input data to an SNN do not necessarily have to be encoded into spikes. It is acceptable to pass continuous values as input, much like how the perception of light starts with a continuous number of photons impinging upon our photoreceptor cells.

Static data, such as an image, can be treated as a direct current (dc) input with the same features passed to the input layer of the SNN at every time step. However, this does not exploit the way SNNs extract meaning from temporal data. In general, three encoding mechanisms have been popularized with respect to input data.

1) **Rate coding** converts input intensity into a **firing rate** or **spike count**.
2) **Latency (or temporal) coding** converts input intensity to a spike **time**.
3) **Delta modulation** converts a temporal **change** of input intensity into spikes and otherwise remains silent.

This is a nonexhaustive list, and these codes are not necessarily independent of each other.

1) Rate-Coded Inputs: How does the sensory periphery encode information about the world into spikes? When bright light is incident upon our photoreceptor cells, the retina triggers a spike train to the visual cortex. Hubel and Wiesel's Nobel prize-winning research on visual processing indicates that a brighter input or a favorable orientation of light corresponds to a higher firing rate [59]. As a

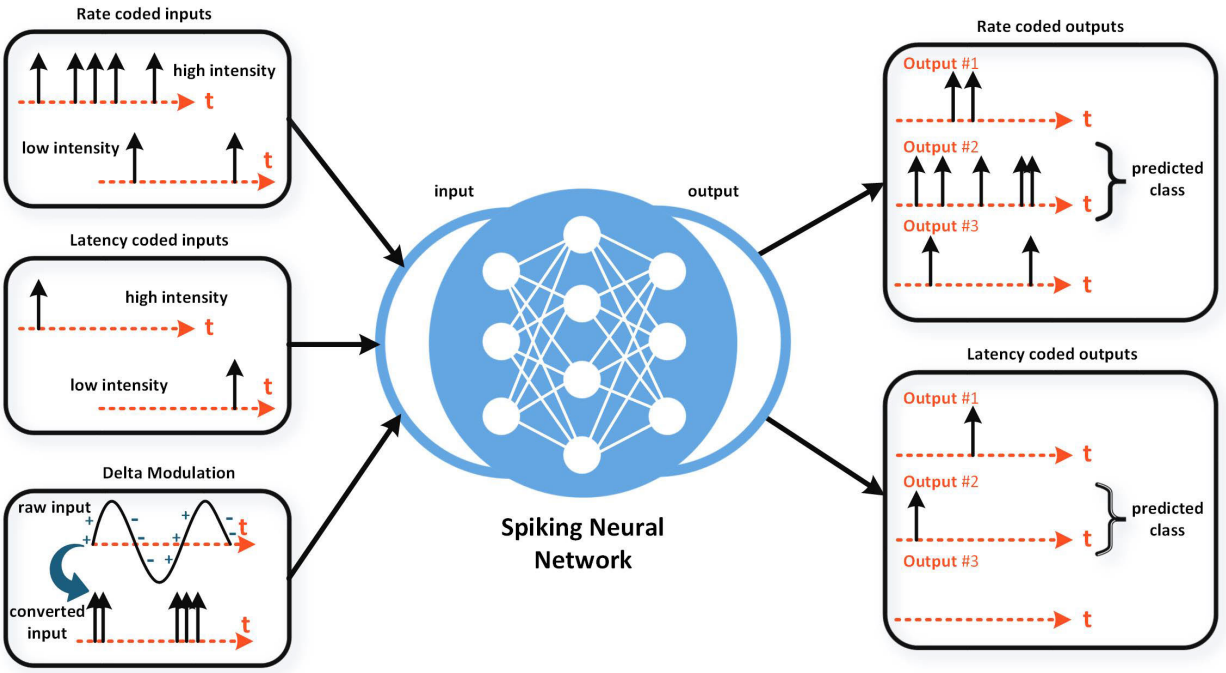


Fig. 7. ***Input data to an SNN may be converted into a firing rate, a firing time, or the data can be delta modulated. Alternatively, the input to the network can also be passed in without conversion, which experimentally represents a direct or variable current source applied to the input layer of neurons. The network itself may be trained to enable the correct class to have the highest firing rate or to fire first, among many other encoding strategies.***

rudimentary example, a bright pixel is encoded into a high-frequency firing rate, whereas a dark pixel would result in low-frequency firing. Measuring the firing rate of a neuron can become quite nuanced. The simplest approach is to apply an input stimulus to a neuron, count up the total number of action potentials it generates, and divide that by the duration of the trial. Although straightforward, the problem here is that the neuronal dynamics vary across time. There is no guarantee that the firing rate at the start of the trial is anything near the rate at the end.

An alternative method counts the spikes over a very short time interval Δt. For a small enough Δt, the spike count can be constrained to either 0 or 1, limiting the total number of possible outcomes to only two. By repeating this experiment multiple times, the average number of spikes (over trials) occurring within Δt can be found. This average must be equal to or less than 1, interpreted as the observed probability that a neuron will fire within the brief time interval. To convert it into a *time-dependent* firing rate, the trial-average is divided by the duration of the interval. This probabilistic interpretation of the rate code can be distributed across multiple neurons, where counting up the spikes from a collection of neurons advocates for a population code [85].

This representation is quite convenient for sequential neural networks. Each discrete-time step in an RNN can be thought of as lasting for a brief duration Δt in which a spike either occurs or does not occur. A formal example of how this takes place is provided in Appendix B1.

2) Latency-Coded Inputs: A latency, or temporal, code is concerned with the timing of a spike. The total number of spikes is no longer consequential. Rather, *when* the spike occurs is what matters. For example, a time-to-first-spike mechanism encodes a bright pixel as an early spike, whereas a dark input will spike last or simply never spike at all. Compared to the rate code, latency-encoding mechanisms assign much more meaning to each individual spike.

Neurons can respond to sensory stimuli over an enormous dynamic range. In the retina, neurons can detect individual photons to an influx of millions of photons [96], [97], [98], [99], [100]. To handle such widely varying stimuli, sensory transduction systems likely compress stimulus intensity with a logarithmic dependency. For this reason, a logarithmic relation between spike times and input feature intensity is ubiquitous in the literature (see Appendix B2) [101], [102].

Although sensory pathways appear to transmit rate-coded spike trains to our brains, it is likely that temporal codes dominate the actual processing that goes on within the brain. More on this in Section III-B4.

3) Delta Modulated Inputs: Delta modulation is based on the notion that neurons thrive on change, which underpins the operation of the silicon retina camera that only generates an input when there has been a sufficient change of input intensity over time. If there is no change in your field of view, then your photoreceptor cells are much less

Table 1 Examples of Neuromorphic Datasets Recorded With Event-Based Cameras and Cochlear Models

Vision datasets	
ASL-DVS [86]	100,800 samples of American sign language recorded with DAVIS.
DAVIS Dataset [87]	Includes spikes, frames and inertial measurement unit recordings of interior and outdoor scenes.
DVS Gestures [88]	11 different hand gestures recorded under 3 different lighting conditions.
DVS Benchmark [89]	DVS benchmark datasets for object tracking, action recognition, and object recognition.
MVSEC [90]	Spikes, frames and optical flow from stereo cameras for indoor and outdoor scenarios.
N-MNIST [91]	Spiking version of the classic MNIST dataset by converting digits from a screen using saccadic motion.
POKER DVS [92]	4 classes of playing cards flipped in rapid succession in front of a DVS.
DSEC [93]	A stereo event camera dataset for driving scenarios.
Audio datasets	
N-TIDIGITS [94]	Speech recordings from the TIDIGITS dataset converted to spikes with a silicon cochlear.
SHD [95]	Spiking version of the Heidelberg Digits dataset converted using a simulated cochlear model.
SSC [95]	Spiking version of the *Speech Commands* dataset converted using a simulated cochlear model.

prone to firing. Computationally, this would take a time-series input and feed a thresholded matrix difference to the network. While the precise implementation may vary, a common approach requires the difference to be both *positive* and *greater* than some predefined threshold for a spike to be generated. This encoding technique is also referred to as "threshold crossing." Alternatively, changes in intensity can be tracked over multiple time steps, and other approaches account for negative changes. For an illustration, see Fig. 4, where the "background" is not captured over time. Only the moving blocks are recorded, as it is those pixels that are changing.

The previous techniques tend to "convert" data into spikes. However, it is more efficient to natively capture data in "preencoded," spiking form. Each pixel in a DVS camera and channel in a silicon cochlear uses delta modulation to record changes in the visual or audio scene. Some examples of neuromorphic benchmark datasets are described in Table 1. A comprehensive series of neuromorphic-relevant datasets are accounted for in NeuroBench [103]. Many of these are readily available for use with open-source libraries, such as *Tonic* [104].

B. Output Decoding

Encoding input data into spikes can be thought of as how the sensory periphery transmits signals to the brain. On the other side of the same coin, decoding these spikes provides insight into how the brain handles these encoded signals. In the context of training an SNN, the encoding mechanism does not constrain the decoding mechanism. Shifting our attention from the input of an SNN, how might we interpret the firing behavior of output neurons?

1) **Rate coding** chooses the output neuron with the highest **firing rate**, or **spike count**, as the predicted class.
2) **Latency (or temporal) coding** chooses the output neuron that fires **first** as the predicted class.
3) **Population coding** relies on **multiple neurons** per class. This is typically used in conjunction with rate coding, rank order coding, or N-of-M coding [105], [106].

1) Rate-Coded Outputs: Consider a multiclass classification problem, where N_C is the number of classes. A non-SNN would select the neuron with the largest output activation as the predicted class. For a rate-coded spiking network, the neuron that fires with the highest frequency is used. As each neuron is simulated for the same number of time steps, simply choose the neuron with the highest spike count (see Appendix B3).

2) Latency-Coded Outputs: There are numerous ways a neuron might encode data in the timing of a spike. As in the case of latency-coded inputs, it could be that a neuron representing the correct class fires first. This addresses the energy burden that arises from the multiple spikes needed in rate codes. In hardware, the need for fewer spikes reduces the frequency of memory accesses, which is another computational burden in deep learning accelerators.

Biologically, does it make sense for neurons to operate on a time to first spike principle? How might we define "first" if our brains are not constantly resetting to some initial, default state? This is quite easy to conceptually address. The idea of a latency or temporal code is motivated by our response to a sudden input stimulus. For example, when viewing a static, unchanging visual scene, the retina undergoes rapid, yet subtle, saccadic motion. The scene projected onto the retina changes every few hundreds of milliseconds. It could very well be the case that the first spike must occur with respect to the reference signal generated by this saccade.

3) Population-Coded Outputs: We know that the reaction time of a human is roughly in the ballpark of 250 ms. If the average firing rate of a neuron in the human brain is on the order of 10 Hz, then we can only process about two to three spikes within our reaction time. However, this often cited 10-Hz assumption should be treated as an upper limit. When experimentalists are hunting for neurons using a single microelectrode, low-rate neurons might be bypassed as they do not generate enough spikes for data analysis or are simply missed altogether. As a result, high-rate neurons may have become significantly

overrepresented in the literature. This is supported by data collected from chronic implants in the macaque hippocampus, which routinely yields neurons with background firing rates below 0.1 Hz [114]. One would have to wait at least 10 s before observing a single spike!

This can be addressed by using a distributed representation of information across a population of neurons: if a single neuron is limited in its spike count within a brief time window, then just use more neurons [85]. The spikes from a subgroup of neurons can be pooled together to make more rapid decisions. Interestingly, population codes trade sequential processing for parallelism, which is more optimal when training SNNs on GPUs [81].

4) Rate Versus Latency Code: Whether neurons encode information as a rate, as latency, or as something wholly different, is a topic of much controversy. We do not seek to crack the neural code here but instead aim to provide intuition on when SNNs might benefit from one code over the other.

Advantages of Rate Codes:

1) *Error tolerance:* If a neuron fails to fire, there are ideally many more spikes to reduce the burden of this error.
2) *More spiking promotes more learning:* Additional spikes provide a stronger gradient signal for learning via error backpropagation. As will be described in Section IV, the absence of spiking can impede learning convergence (more commonly referred to as the "dead neuron problem").

Advantages of Latency Codes:

1) *Power consumption:* Generating and communicating fewer spikes mean less dynamic power dissipation in tailored hardware. It also reduces memory access frequency due to sparsity as a vector-matrix product for an all-zero input vector returns a zero output.
2) *Speed:* The reaction time of a human is roughly in the ballpark of 250 ms. If the average firing rate of a neuron in the human brain is on the order of 10 Hz (which is likely an overestimation [114]), then one can only process about two to three spikes in this reaction time window. In contrast, latency codes rely on a single spike to represent information. This issue with rate codes may be addressed by coupling it with a population code: if a single neuron is limited in its spike count within a brief time window, then just use more neurons [85]. This comes at the expense of further exacerbating the power consumption problem of rate codes.

The power consumption benefit of latency codes is also supported by observations in biology, where nature optimizes for efficiency. Olshausen and Field's [114] work in "What is the other 85% of V1 doing?" methodically demonstrates that rate-coding can only explain, at most, the activity of 15% of neurons in the primary visual cortex (V1). If our neurons indiscriminately defaulted to a rate code, this would consume an order of magnitude more energy than a temporal code. The mean firing rate of our cortical neurons must necessarily be rather low, which is supported by temporal codes.

Lesser explored encoding mechanisms in gradient-based SNNs include using spikes to represent a prediction or reconstruction error [115]. The brain may be perceived as an anticipatory machine that takes action based on its predictions. When these predictions do not match reality, spikes are triggered to update the system.

Some assert that the true code must lie between rate and temporal codes [116], while others argue that the two may coexist and only differ based on the timescale of observation: rates are observed for long timescales and latency for short timescales [117]. Some reject rate codes entirely [118]. This is one of those instances where a deep learning practitioner might be less concerned with what the brain does and prefers to focus on what is most useful.

C. Objective Functions

While it is unlikely that our brains use something as explicit as a cross-entropy loss function, it is fair to say that humans and animals have baseline objectives [121]. Biological variables, such as dopamine release, have been meaningfully related to objective functions from reinforcement learning [122]. Predictive coding models often aim to minimize the information entropy of sensory encodings such that the brain can actively predict incoming signals and inhibit what it already expects [123]. The multifaceted nature of the brain's function likely calls for the existence of multiple objectives [124]. How the brain can be optimized using these objectives remains a mystery though we might gain insight from multiobjective optimization [125].

A variety of loss functions can be used to encourage the output layer of a network to fire as a rate or temporal code. The optimal choice is largely unsettled and tends to be a function of the network hyperparameters and the complexity of the task at hand. All objective functions described in the following have successfully trained networks to competitive results on a variety of datasets though they come with their own tradeoffs.

1) Spike Rate Objective Functions: A summary of approaches commonly adopted in supervised learning classification tasks with SNNs to promote the correct neuron class to fire with the highest frequency is provided in Table 2. In general, either the cross-entropy loss or mse is applied to the spike count or the membrane potential of the output layer of neurons.

With a sufficient number of time steps, passing the spike count the objective function is more widely adopted as it operates directly on spikes. Membrane potential acts as a proxy for increasing the spike count and is also not considered an observable variable, which may partially offset the computational benefits of using spikes.

Cross-entropy approaches aim to suppress the spikes from incorrect classes, which may drive weights in a network to zero. This could cause neurons to go quiet in the

Table 2 Rate-Coded Objectives

	Cross-Entropy Loss	Mean Square Error
Spike Count	**Cross-Entropy Spike Rate:** The total number of spikes for each neuron in the output layer are accumulated over time into a spike count $\vec{c} \in \mathbb{N}^{N_C}$ (Equation (27) in Appendix B3), for N_C classes. A multi-class categorical probability distribution is obtained by treating the spike counts as logits in the softmax function. Cross entropy minimization is used to increase the spike count of the correct class, while suppressing the count of the incorrect classes [88], [107] (Appendix B4).	**Mean Square Spike Rate:** The spike counts of both correct and incorrect classes are specified as targets. The mean square errors between the actual and target spike counts for all output classes are summed together. In practice, the target is typically represented as a proportion of the total number of time steps: e.g., the correct class should fire at 80% of all time steps, while incorrect classes should fire 20% of the time [73], [108]–[110] (Appendix B5).
Membrane Potential	**Maximum Membrane:** The logits are obtained by taking the maximum value of the membrane potential over time, which are then applied to a softmax cross entropy function. By encouraging the membrane potential of the correct class to increase, it is expected to encourage more regular spiking [111]–[113]. A variant is to simply sum the membrane potential across all time steps to obtain the logits [113] (Appendix B6).	**Mean Square Membrane:** Each output neuron has a target membrane potential specified for each time step, and the losses are summed across both time and outputs. To implement a rate code, a superthreshold target should be assigned to the correct class across time steps (Appendix B7).

absence of additional regularization. By using the mean square spike rate, which specifies a target number of spikes for each class, output neurons can be placed on the cusp of firing. Therefore, the network may adapt to changing inputs with a faster response time than neurons that have their firing completely suppressed.

In networks that simulate a constrained number of time steps, a small change in weights is unlikely to cause a change in the spike count of the output. It might be preferable to apply the loss function directly to a more "continuous" signal, such as the membrane potential instead. This comes at the expense of operating on a full precision hidden state, rather than on spikes. Alternatively, using population coding can distribute the cost burden over multiple neurons to increase the probability that a weight update will alter the spiking behavior of the output layer. It also increases the number of pathways through which error backpropagation may take place and improve the chance that a weight update will generate a change in the global loss.

2) Spike Time Objectives: Loss functions that implement spike timing objectives are less commonly used than rate-coded objectives. Two possible reasons may explain why: 1) error rates are typically perceived to be the most important metric in deep learning literature, and rate codes are more tolerant to noise and 2) temporal codes are marginally more difficult to implement. A summary of approaches is provided in Table 3.

The use cases of these objectives are analogous to the spike rate objectives. A subtle challenge with using spike times is that the default implementation assumes each neuron spikes at least once, which is not necessarily the case. This can be handled by forcing a spike at the final time step in the event when a neuron does not fire [120].

Several state-of-the-art models wholly abandon spiking neurons at the output and train their models using a "read-out layer." This often consists of an IF layer with infinitely high thresholds (i.e., they will never fire) or with typical artificial neurons that use standard activation functions (ReLU, sigmoid, softmax, and so on). While this often improves accuracy, this may not qualify as a fully spiking network. Does this actually matter? If one can still achieve power efficiency, then engineers will be happy and that is often all that matters.

D. Learning Rules

1) Spatial and Temporal Credit Assignment: Once a loss has been determined, it must somehow be used to update the network parameters with the hope that the network will iteratively improve at the trained task. Each weight takes some blame for its contribution to the total loss, and this is known as "credit assignment." This can be split into the *spatial* and *temporal* credit assignment problems. Spatial credit assignment aims to find the spatial location of the weight contributing to the error, while the temporal credit assignment problem aims to find the time at which the weight contributes to the error. Backpropagation has proven to be an extremely robust way to address credit assignment, but the brain is far more constrained in developing solutions to these challenges.

Backpropagation solves spatial credit assignment by applying a distinct backward pass after a forward pass during the learning process [126]. The backward pass mirrors the forward pass, such that the computational pathway of the forward pass must be recalled. In contrast, action potential propagation along an axon is considered to be unidirectional, which may reject the plausibility of backprop taking place in the brain. Spatial credit assignment is not only concerned with calculating the weight's contribution to an error but also assigning the error back to the weight. Even if the brain could somehow calculate the gradient (or an approximation), a major challenge

Table 3 Latency-Coded Objectives

	Cross-Entropy Loss	Mean Square Error
Spike Time	**Cross-Entropy Spike Time:** The timing of the first spike of each neuron in the output layer is taken $\vec{f} \in \mathbb{R}^{N_C}$. As cross entropy minimization involves maximising the likelihood of the correct class, a monotonically decreasing function must be applied to $\vec{f}$ such that early spike times are converted to large numerical values, while late spikes become comparatively smaller. These 'inverted' values are then used as logits in the softmax function [74] (Appendix B8).	**Mean Square Spike Time:** The spike time of all neurons are specified as targets. The mean square errors between the actual and target spike times of all output classes are summed together. This can be generalized to multiple spikes as well [73], [119] (Appendix B9). **Mean Square Relative Spike Time:** A similar approach to above, but rather than specifying the precise time of each spike, only the relative time of the correct class must be specified. If the correct neuron fires a specified number of time steps earlier than incorrect neurons, the loss is fixed at zero [120] (Appendix B10).
Membrane Potential	Unreported in the literature.	**Mean Square Membrane:** Analogous to the rate-coded case, each output neuron has a target membrane potential specified for each time step, and the losses are summed across both time and outputs. To implement a temporal code, the correct class should specify a target membrane greater than the threshold of the neuron at an early time (Appendix B7).

would be projecting that gradient back to the synapse and knowing which gradient belongs to which synapse.

This constraint of neurons acting as directed edges is increasingly being relaxed, which could be a mechanism by which errors are assigned to synapses [127]. Numerous bidirectional, nonlinear phenomena occur within individual neurons, which may contribute toward helping errors find their way to the right synapse. For example, feedback connections are observed in most places where there are feedforward connections [128].

2) Biologically Motivated Learning Rules: With a plethora of neuronal dynamics that might embed variants of backpropagation, what options are there for modifying backprop to relax some of the challenges associated with biologically plausible spatial credit assignment? In general, the more broadly adopted approaches rely on either trading parts of the gradient calculation for stochasticity or otherwise swapping a global error signal for localized errors (see Fig. 8). Conjuring alternative methods to credit assignment that a real-time machine such as the brain can implement is not only useful for developing insight into biological learning [129] but also reduces the cost of data communication in hardware [130]. For example, using local errors can reduce the length a signal must travel across a chip. Stochastic approaches can trade computation with naturally arising circuit noise [131], [132], [133]. A brief summary of several common approaches to mitigating the spatial credit assignment problem is provided in the following [134].

1) *Perturbation learning:* A random perturbation of network weights is used to measure the change in error. If the error is reduced, the change is accepted. Otherwise, it is rejected [135], [136], [137]. The difficulty of learning scales with the number of weights, where the effect of a single weight change is dominated by the noise from all other weight changes. In practice, it may take a huge number of trials to average this noise away [55].
2) *Random feedback:* Backpropagation requires sequentially transporting the error signal through multiple layers, scaled by the forward weights of each layer. Random feedback replaces the forward weight matrices with random matrices, reducing the dependence of each weight update on distributed components of the network. While this does not fully solve the spatial credit assignment problem, it quells the *weight transport problem* [138], which is specifically concerned with a weight update in one layer depending upon the weights of far-away layers. Forward- and backward-propagating data are scaled by symmetric weight matrices, a mechanism that is absent in the brain. Random feedback has shown similar performance to backpropagation on simple networks and tasks, which gives hope that a precise gradient may not be necessary for good performance [138]. Random feedback has struggled with more complex tasks though variants have been proposed that reduce the gap [139], [140], [141], [142]. Nonetheless, the mere fact that such a core piece of the backpropagation algorithm can be replaced with random noise and yet somehow still work is a marvel. It is indicative that we still have much left to understand about gradient backpropagation.
3) *Local losses:* It could be that the six layers of the cortex are each supplied with their own cost function, rather

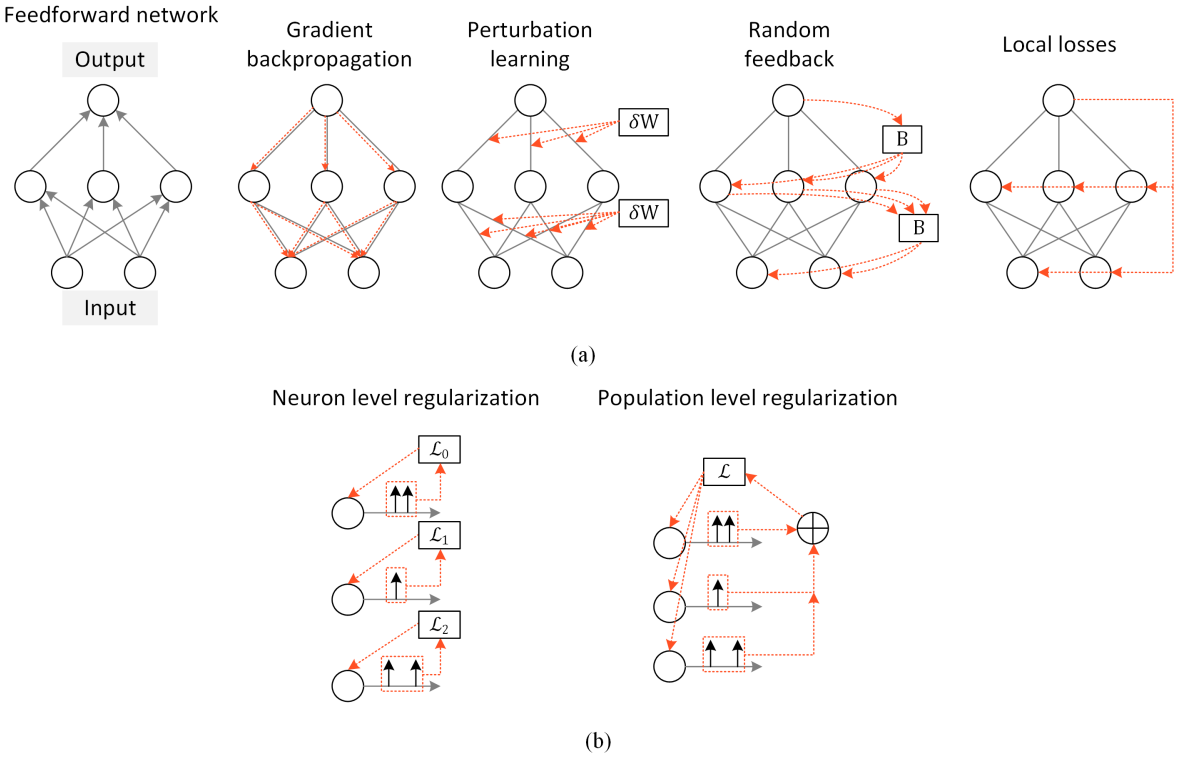


Fig. 8. ***Variety of learning rules can be used to train a network. (a) Objective functions. Gradient backpropagation: an unbiased gradient estimator of the loss is derived with respect to each weight. Perturbation learning: weights are randomly perturbed by δW, with the change accepted if the output error is reduced. Random feedback: all backward references to weights W are replaced with random feedback B. Local losses: each layer is provided with an objective function avoiding error backpropagation through multiple layers. (b) Activity regularization. Neuron level regularization: it aims to set a baseline spike count per neuron. Population level regularization: it aims to set an upper limit on the total number of spikes emitted from all neurons.***

than a global signal that governs a unified goal for the brain [124]. Early visual regions may try to minimize the prediction error in constituent visual features, such as orientations, while higher areas use cost functions that target abstractions and concepts. For example, a baby learns how to interpret receptive fields before consolidating them into facial recognition. In deep learning, greedy layerwise training assigns a cost function to each layer independently [143]. Each layer is sequentially assigned a cost function, so as to ensure that a shallow network is only ever trained. Target propagation is similarly motivated by assigning a reconstruction criterion to each layer [115]. Such approaches exploit the fact that training a shallow network is easier than training a deep one and aim to address spatial credit assignment by ensuring that the error signal does not need to propagate too far [127], [144].

4) *Forward–forward error propagation:* The backward pass of a model is replaced with a second forward pass where the input signal is altered based on error or some related metric. Initially proposed by Dellaferrera and Kreiman [145], Hinton's [146] forward–forward learning algorithm generated more traction soon after. These have not been ported to SNNs at the time of writing though someone is bound to step up to the mantle soon.

These approaches to learning are illustrated in Fig. 8(a). While they are described in the context of supervised learning, many theories of learning place emphasizes on self-organization and unsupervised approaches. Hebbian plasticity is a prominent example [147]. However, an intersection may exist in self-supervised learning, where the target of the network is a direct function of the data itself. Some types of neurons may be representative of facts, features, or concepts, only firing when exposed to the right type of stimuli. Other neurons may fire with the purpose of reducing a reconstruction error [148], [149]. By accounting for spatial and temporal correlations that naturally exist around us, such neurons may fire with the intent to predict what happens next. A more rigorous treatment of biological plausibility in objective functions can be found in [124].

E. Activity Regularization

A huge motivator behind using SNNs comes from the power efficiency when processed on appropriately tailored

hardware. This benefit is not only from single-bit interlayer communication via spikes but also the sparse occurrence of spikes. Some of the loss functions above, in particular those that promote rate codes, will indiscriminately increase the membrane potential and/or firing frequency without an upper bound, if left unchecked. Regularization of the loss can be used to penalize excessive spiking (or alternatively, penalize insufficient spiking, which is great for discouraging dead neurons). Conventionally, regularization is used to constrain the solution space of loss minimization, thus leading to a reduction in variance at the cost of increasing bias. Care must be taken, as too much activity regularization can lead to excessively high bias. Activity regularization can be applied to alter the behavior of individual neurons or populations of neurons, as depicted in Fig. 8(b).

1) *Population level regularization:* This is useful when the metric to optimize is a function of aggregate behavior. For example, the metric may be power efficiency, which is strongly linked to the total number of spikes from an entire network. L1-regularization can be applied to the total number of spikes emitted at the output layer to penalize excessive firing, which encourages sparse activity at the output [150]. Alternatively, for more fine-grain control over the network, an upper activity threshold can be applied. If the total number of spikes for *all* neurons in a layer exceeds the threshold, only then does the regularization penalty kick in [113] and [110] (see Appendix B11).
2) *Neuron level regularization:* If neurons completely cease to fire, then learning may become significantly more difficult. Regularization may also be applied at the individual neuron level by adding a penalty for each neuron. A lower activity threshold specifies the lower permissible limit of firing for *each* neuron before the regularization penalty is applied (see Appendix B12).

Recent experiments have shown that rate-coded networks (at the output) are robust to sparsity-promoting regularization terms [110], [111], [113]. However, networks that rely on time-to-first-spike schemes have had less success, which is unsurprising given that temporal outputs are already sparse.

Encouraging each neuron to have a baseline spike count helps with the backpropagation of errors through pathways that would otherwise be inactive. Together, the upper and lower limit regularization terms can be used to find the sweet spot of firing activity at each layer. As explained in detail in [151], the variance of activations should be as close as possible to "1" to avoid vanishing and exploding gradients. While modern deep learning practices rely on appropriate parameter initialization to achieve this, these approaches were not designed for nondifferentiable activation functions, such as spikes. By monitoring and appropriately compensating for neuron activity, this may turn out to be a key ingredient to successfully training deep SNNs.

IV. TRAINING SPIKING NEURAL NETWORKS

The rich temporal dynamics of SNNs give rise to a variety of ways in which a neuron's firing pattern can be interpreted. Naturally, this means that there are several methods for training SNNs. They can generally be classified into the following methods.

1) *Shadow training:* A nonspiking ANN is trained and converted into an SNN by interpreting the activations as a firing rate or spike time.
2) *Backpropagation using spikes:* The SNN is natively trained using error backpropagation, typically through time as is done with sequential models.
3) *Local learning rules:* Weight updates are a function of signals that are spatially and temporally local to the weight, rather than from a global signal as in error backpropagation.

Each approach has a time and place where it outshines the others. We will focus on approaches that apply backprop directly to an SNN, but useful insights can be attained by exploring shadow training and various local learning rules.

The goal of the backpropagation algorithm is loss minimization. To achieve this, the gradient of the loss is computed with respect to each learnable parameter by applying the chain rule from the final layer back to each weight [152], [153], [154]. The gradient is then used to update the weights such that the error is ideally always decreased. If this gradient is "0," there is no weight update. This has been one of the main road blocks to training SNNs using error backpropagation due to the nondifferentiability of spikes. This is also known as the dreaded "dead neuron" problem. There is a subtle, but important, difference between "vanishing gradients" and "dead neurons," which will be explained in Section IV-C.

To gain deeper insight behind the nondifferentiability of spikes, recall the discretized solution of the membrane potential of the leaky IF neuron from (4): $U[t] = \beta U[t-1] + WX[t]$, where the first term represents the decay of the membrane potential U, and the second term is the weighted input WX. The reset term and subscripts have been omitted for simplicity. Now, imagine that a weight update ΔW is applied to the weight W [see (4)]. This update causes the membrane potential to change by ΔU, but this change in potential fails to precipitate a further change to the spiking presence of the neuron [see (5)]. That is to say, $dS/dU = 0$ for all U, other than the threshold θ, where $dS/dU \rightarrow \infty$. This drives the term we are actually interested in, $d\mathcal{L}/dW$, or the gradient of the loss in weight space, to either "0" or "∞." In either case, there is no adequate learning

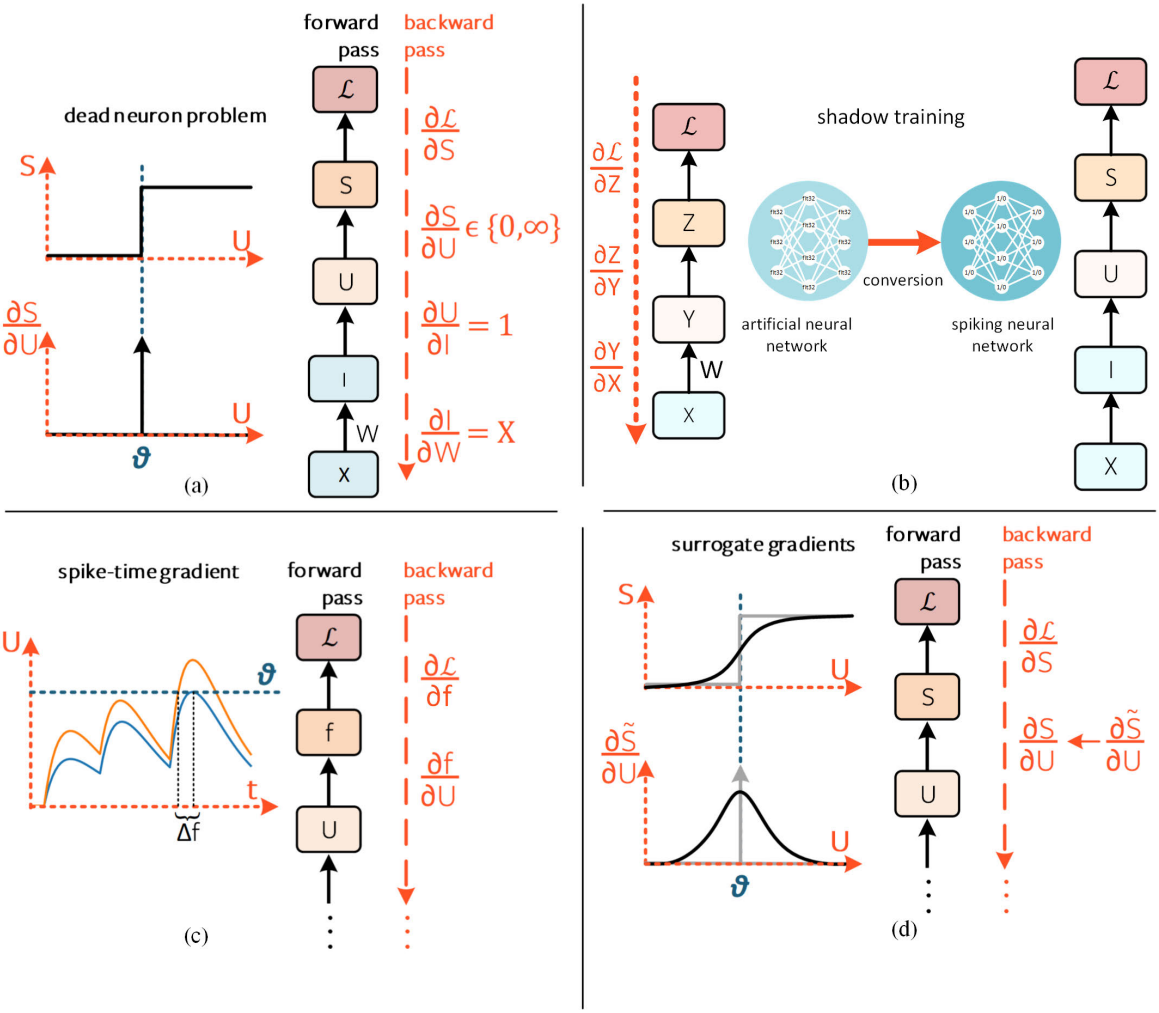


Fig. 9. ***Addressing the dead neuron problem. Only one time step is shown, where temporal connections and subscripts from Fig. 6 have been omitted for simplicity. (a) Dead neuron problem: the analytical solution of $\partial S/\partial U \in \{0, \infty\}$ results in a gradient that does not enable learning. (b) Shadow training: a nonspiking network is first trained and subsequently converted into an SNN. (c) Spike-time gradient: the gradient of spike time f is taken instead of the gradient of the spike generation mechanism, which is a continuous function as long as a spike necessarily occurs [119]. (d) Surrogate gradients: the spike generation function is approximated to a continuous function during the backward pass [113]. The left arrow (←) indicates function substitution. This is the most broadly adopted solution to the dead neuron problem.***

signal when backpropagating through a spiking neuron [see Fig. 9(a)].

A. Shadow Training

The dead neuron problem can be completely circumvented by instead training on a shadow ANN and converting it into an SNN [see Fig. 9(b)]. The high-precision activation function of each neuron is converted into either a spike rate [155], [156], [157], [158], [159] or a latency code [160]. One of the most compelling reasons to use shadow training is that advances in conventional deep learning can be directly applied to SNNs. For this reason, ANN-to-SNN conversion currently takes the crown for static image classification tasks on complex datasets, such as CIFAR-10 and ImageNet. Where inference efficiency is more important than training efficiency, and if input data are not time-varying, then shadow training could be the optimal way to go.

In addition to the inefficient training process, there are several drawbacks. First, the types of tasks that are most commonly benchmarked do not make use of the temporal dynamics of SNNs, and the conversion of sequential neural networks to SNNs is an underexplored area [157]. Second, converting high-precision activations into spikes typically requires a long number of simulation time steps, which may offset the power/latency benefits initially sought from SNNs. However, what really motivates doing away with ANNs is that the conversion process is necessarily an approximation. Therefore, a shadow-trained SNN is very unlikely to reach the performance of the original network.

The issue of long time sequences can be partially addressed by using a hybrid approach: start with a shadow-trained SNN and then perform backpropagation on the converted SNN [161]. Although this appears to degrade accuracy (reported on CIFAR-10 and ImageNet), it is possible to reduce the required number of steps by an order of magnitude. A more rigorous treatment of shadow training techniques and challenges can be found in [162].

B. Backpropagation Using Spike Times

An alternative method to side step the dead neuron problem is to instead take the derivative at spike times. In fact, this was the first proposed method for training multilayer SNNs using backpropagation [119]. The original approach in *SpikeProp* observes that, while spikes may be discontinuous, time is continuous. Therefore, taking the derivative of spike *timing* with respect to the weights achieves functional results. A thorough description is provided in Appendix C1.

Intuitively, SpikeProp calculates the gradient of the error with respect to the spike time. A change to the weight by ΔW causes a change of the membrane potential by ΔU, which ultimately results in a change of spike timing by Δf, where f is the firing time of the neuron. In essence, the nondifferentiable term $\partial S/\partial U$ has been traded with $\partial f/\partial U$. This also means that each neuron *must* emit a spike for a gradient to be calculable. This approach is illustrated in Fig. 9(c). Extensions of SpikeProp have made it compatible with multiple spikes [163], which are highly performant on data-driven tasks some of which have surpassed human-level performance on MNIST and N-MNIST [74], [164], [165], [166].

Several drawbacks arise. Once neurons become inactive, their weights become frozen. In most instances, no closed-form solutions exist to solve for the gradient if there is no spiking [167]. SpikeProp tackles this by modifying parameter initialization (i.e., increasing weights until a spike is triggered). However, since the inception of SpikeProp in 2002, the deep learning community's understanding of weight initialization has gradually matured. We now know initialization aims to set a constant activation variance between layers, the absence of which can lead to vanishing and exploding gradients through space and time [151], [168]. Modifying weights to promote spiking may detract from this. Instead, a more effective way to overcome the lack of firing is to lower the firing thresholds of the neurons. One may consider applying activity regularization to encourage firing in hidden layers though this has degraded classification accuracy when taking the derivative at spike times. This result is unsurprising, as regularization can only be applied at the spike time rather than when the neuron is quiet.

Another challenge is that it enforces stringent priors upon the network (e.g., each neuron must fire only once), which are incompatible with dynamically changing input data. This may be addressed by using periodic temporal codes that refresh at given intervals, in a similar manner to how visual saccades may set a reference time. However, it is the only approach that enables the calculation of an unbiased gradient without any approximations in multilayer SNNs. Whether this precision is necessary is a matter of further exploration of a broader range of tasks.

Another challenge is that it enforces stringent priors upon the network (e.g., each neuron must fire only once), which are incompatible with dynamically changing input data. This may be addressed by using periodic temporal codes that refresh at given intervals, in a similar manner to how visual saccades may set a reference time. However, it is the only approach that enables the calculation of an unbiased gradient without any approximations in multilayer SNNs. Whether this precision is necessary is a matter of further exploration of a broader range of tasks [165].

C. Backpropagation Using Spikes

Instead of computing the gradient with respect to spike times, the most commonly adopted approach over the past several years is to apply the generalized backpropagation algorithm to the unrolled computational graph [see Fig. 6(b)] [73], [107], [156], [169], [170], i.e., backpropagation through time (BPTT). Working backward from the final output of the network, the gradient flows from the loss to all descendants. In this way, computing the gradient through an SNN is mostly the same as that of an RNN by iterative application of the chain rule. Fig. 10(a) depicts the various pathways of the gradient $\partial\mathcal{L}/\partial W$ from the parent ($\mathcal{L}$) to its leaf nodes (W). In contrast, backprop using spike times only follows the gradient pathway whenever a neuron fires, whereas this approach takes every pathway regardless of the neuron firing. The final loss is the sum of instantaneous losses $\sum_t \mathcal{L}[t]$ though the loss calculation can take a variety of other forms, as described in Section III-C.

Finding the derivative of the total loss with respect to the parameters allows the use of gradient descent to train the network, so the goal is to find $\partial\mathcal{L}/\partial W$. The parameter W is applied at every time step, and the application of the weight at a particular step is denoted $W[s]$. Assume that an instantaneous loss $\mathcal{L}[t]$ can be calculated at each time step (taking caution that some objective functions, such as the mean square spike rate loss (see Section III-C1), must wait until the end of the sequence to accumulate all spikes and generate a loss). As the forward pass requires moving data through a directed acyclic graph, each application of the weight will only affect present and future losses. The influence of $W[s]$ on $\mathcal{L}[t]$ at $s = t$ is labeled as the *immediate influence* in Fig. 10(a). For $s < t$, we refer to the impact of $W[s]$ on $L[t]$ as the *prior influence*. The influence of all parameter applications on present and future losses is summed together to define the global gradient

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}[t]}{\partial W} = \sum_t \sum_{s \leq t} \frac{\partial \mathcal{L}[t]}{\partial W[s]} \frac{\partial W[s]}{\partial W}. \tag{6}$$

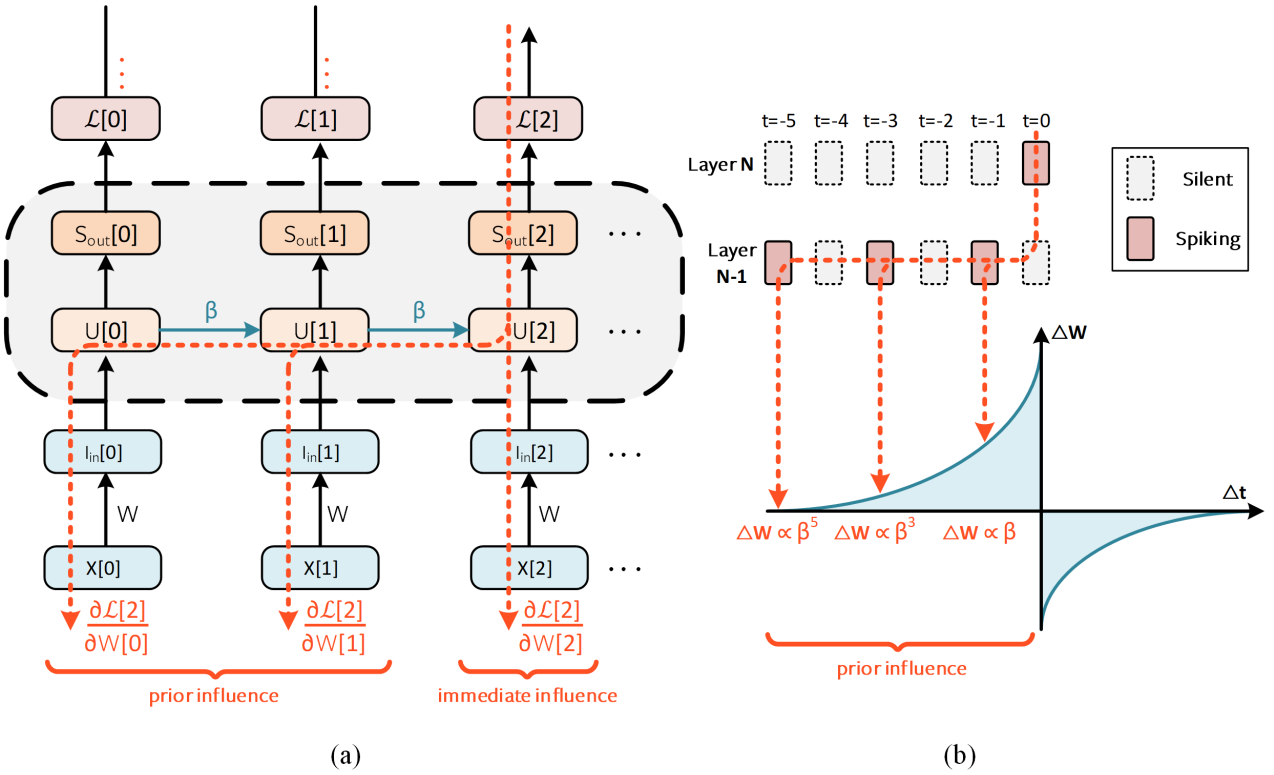


Fig. 10. *BPTT. (a) Present time application of W is referred to as the immediate influence, with the historical application of W described as the prior influence. Reset dynamics and explicit recurrence have been omitted for brevity. The error pathways through ℒ[0] and ℒ[1] are also hidden but follow the same idea as that of ℒ[2]. (b) Hybrid approach defaults to a nonzero gradient only at spike times. For present time t = 0, the derivative of each application of W[s] with respect to the loss decays exponentially moving back in time. The magnitude of the weight update ΔW for prior influences of W[s] follows a relationship qualitatively resembling that of STDP learning curves, where the strength of the synaptic update is dependent on the order and firing time of a pair of connected neurons [33].*

A recurrent system will constrain the weight to be shared across all steps: $W[0] = W[1] = \cdots = W$. Therefore, a change in $W[s]$ will have an equivalent effect on all other values of W, which suggests that $\partial W[s]/\partial W = 1$, and (6) simplifies to

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_t \sum_{s \leq t} \frac{\partial \mathcal{L}[t]}{\partial W[s]}. \tag{7}$$

Thankfully, gradients rarely need to be calculated by hand as most deep learning packages come with an automatic differentiation engine. Isolating the immediate influence at a single time step as in Fig. 9(c) makes it clear that we run into the spike nondifferentiability problem in the term $\partial S/\partial U \in \{0, \infty\}$. The act of thresholding the membrane potential is functionally equivalent to applying a shifted Heaviside operator, which is nondifferentiable.

The solution is actually quite simple. During the forward pass, as per usual, apply the Heaviside operator to $U[t]$ in order to determine whether the neuron spikes. However, during the backward pass, substitute the Heaviside operator with a continuous function, $\tilde{S}$ (e.g., sigmoid). The derivative of the continuous function is used as a substitute $\partial S/\partial U \leftarrow \partial \tilde{S}/\partial U$ and is known as the surrogate gradient approach [see Fig. 9(d)].

D. Surrogate Gradients

A major advantage of surrogate gradients is that they help with overcoming the dead neuron problem. To make the dead neuron problem more concrete, consider a neuron with a threshold of θ, and one of the following cases occurs.

1) The membrane potential is below the threshold: $U < \theta$.
2) The membrane potential is above the threshold: $U > \theta$.
3) The membrane potential is exactly at the threshold: $U = \theta$.

In Case 1, no spike is elicited, and the derivative would be $\partial S/\partial U_{U<\theta} = 0$. In Case 2, a spike would fire, but the derivative remains $\partial S/\partial U_{U>\theta} = 0$. Applying either of these to the chain of equations in Fig. 9(a) will null $\partial \mathcal{L}/\partial W = 0$. In the improbable event of Case 3, $\partial S/\partial U_{U=\theta} = \infty$, which swamps out any meaningful gradient when applied to the chain rule. However, approximating the gradient, $\partial \tilde{S}/\partial U$, solves this.

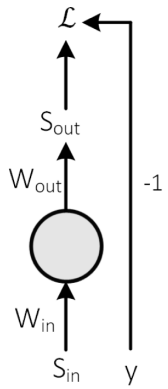


Fig. 11. ***Sequence of steps during the forward pass.***

One example is to replace the nondifferentiable term with the threshold-shifted sigmoid function but only during the backward pass. This is illustrated in Fig. 9(d). More formally

$$\sigma(\cdot) = \frac{1}{1 + e^{\theta - U}} \tag{8}$$

and therefore

$$\frac{\partial S}{\partial U} \leftarrow \frac{\partial \tilde{S}}{\partial U} = \sigma'(\cdot) = \frac{e^{\theta - U}}{(e^{\theta - U} + 1)^2}. \tag{9}$$

This means that learning only takes place if there is spiking activity. Consider a synaptic weight attached to the input of a spiking neuron, W_{in}, and another weight at the output of the same neuron, W_{out}. Say the following sequence of events occurs (see Fig. 11).

1) An input spike, S_{in}, is scaled by W_{in}.
2) The weighted spike is added as an input current injection to the spiking neuron [see (4)].
3) This may cause the neuron to trigger a spike, S_{out}.
4) The output spike is weighted by the output weight W_{out}.
5) This weighted output spike varies some arbitrary loss function, $\mathcal{L}$.

Let the loss function be the Manhattan distance between a target value y and the weighted spike

$$\mathcal{L} = |W_{\text{out}} S_{\text{out}} - y|$$

where updating W_{out} requires

$$\frac{\partial \mathcal{L}}{\partial W_{\text{out}}} = S_{\text{out}}.$$

More generally, a spike must be triggered for a weight to be updated. The surrogate gradient does not change this.

Now, consider the case for updating W_{in}, where the following derivative must be calculated:

$$\frac{\partial \mathcal{L}}{\partial W_{\text{in}}} = \underbrace{\frac{\partial \mathcal{L}}{\partial S_{\text{out}}}}_{A} \underbrace{\frac{\partial S_{\text{out}}}{\partial U}}_{B} \underbrace{\frac{\partial U}{\partial W_{\text{in}}}}_{C}.$$

1) **Term A** is simply W_{out} based on the above equation for $\mathcal{L}$.
2) **Term B** would almost always be 0, unless substituted for a surrogate gradient.
3) **Term C** is S_{in} (see (4) where $X = S_{\text{in}}$).

To summarize, the surrogate gradient enables errors to propagate to earlier layers, regardless of spiking. However, spiking is still needed to trigger a weight update.

As a practical note, various works empirically explore different surrogate gradients. These include triangular functions, fast sigmoid and sigmoid functions, straight-through estimators, and various other weird shapes. Is there a best surrogate gradient? In our experience, we have found the following function to be the best starting point:

$$\frac{\partial \tilde{S}}{\partial U} = \frac{1}{\pi} \frac{1}{1 + (\pi U)^2}.$$

You might see this referred to as the "arctan" surrogate gradient, first proposed in [171]. This is because the integral of this function is

$$\tilde{S} = \frac{1}{\pi} \arctan(\pi U).$$

As of 2023, this is the default surrogate gradient in snnTorch, and it is not wholly clear why it works so well.

To reiterate, surrogate gradients will not enable learning in the absence of spiking. This provokes an important distinction between the dead neuron problem and the vanishing gradient problem. A dead neuron is one that does not fire and, therefore, does not contribute to the loss. This means that the weights attached to that neuron have no "credit" in the credit assignment problem. The relevant gradient terms during the training process will remain at zero. Therefore, the neuron cannot learn to fire later on and so is stuck *forever*, not contributing to learning.

On the other hand, vanishing gradients can arise in ANNs and SNNs. For deep networks, the gradients of the loss function can become vanishingly small as they are successively scaled by values less than "1" when using several common activation functions (e.g., a sigmoid unit). In much the same way, RNNs are highly susceptible to vanishing gradients because they introduce an additional layer to the unrolled computational graph at each time step. Each layer adds another multiplicative factor in calculating the gradient, which makes it susceptible to vanishing if the factor is less than "1" or exploding if greater than "1." The ReLU activation became broadly adopted to reduce the

impact of vanishing gradients but remains underutilized in surrogate gradient implementations [151].

Surrogate gradients have been used in most state-of-the-art experiments that natively train an SNN [73], [107], [156], [169], [170]. A variety of surrogate gradient functions have been used to varying degrees of success, and the choice of function can be treated as a hyperparameter. While several studies have explored the impact of various surrogates on the learning process [113], [172], our understanding tends to be limited to what is known about biased gradient estimators. There is a lot left unanswered here. For example, if we can get away with approximating gradients, then, perhaps, surrogate gradients can be used in tandem with random feedback alignment. This involves replacing weights with random matrices during the backward pass. Rather than pure randomness, perhaps, local approximations can be made that follow the same spirit of a surrogate gradient.

In summary, taking the gradient only at spike times provides an unbiased estimator of the gradient at the expense of losing the ability to train dead neurons. Surrogate gradient descent flips this around, enabling dead neurons to backpropagate error signals by introducing a biased estimator of the gradient. There is a tug-of-war between bringing dead neurons back to life and introducing bias. Given how prevalent surrogate gradients have become, we will linger a little longer on the topic in describing their relation to model quantization. Understanding how approximations in gradient descent impact learning will very likely lead to a deeper understanding of why surrogate gradients are so effective, how they might be improved, and how backpropagation can be simplified by making approximations that reduce the cost of training without harming an objective.

E. Bag of Tricks in BPTT With SNNs

Many advances in deep learning stem from a series of incremental techniques that bolster the learning capacity of models. These techniques are applied in conjunction to boost model performance. For example, He et al.'s [173] work in "*Bag of tricks for image classification with convolutional neural networks*" not only captures the honest state of deep learning in the title alone but also performs an ablation study of "hacks" that can be combined to improve optimization during training. Some of these techniques can be ported straight from deep learning to SNNs, while others are SNN-specific. A nonexhaustive list of these techniques is provided in this section. These techniques are quite empirical, and each bullet would have its own "Practical Note" text box, but then this article would just turn into a bunch of boxes.

1) **The reset mechanism** in (4) is a function of the spike and is also nondifferentiable. It is important to ensure that the surrogate gradient is not cloned into the reset function as it has been empirically shown to degrade network performance [113]. Quite simply, we ignore it during the backward pass. snnTorch does this automatically by detaching the reset term in (4) from the computational graph by calling the "*.detach()*" function.
2) **Residual connections** work remarkably well for nonspiking nets and spiking models alike. Direct paths between layers are created by allowing the output of an earlier layer to be added to the output of a later layer, effectively skipping one or more layers in between. They are used to address the vanishing gradient problem and improve the flow of information during both forward propagation and backward propagation, which enabled the neural network community to construct far deeper architectures, starting with the ResNet family of models and now commonly used in transformers [174]. Unsurprisingly, they work extremely well for SNNs too [171].
3) **Learnable decay:** Rather than treating the decay rates of neurons as hyperparameters, it is also common practice to make them learnable parameters. This makes SNNs resemble conventional RNNs much more closely. Doing so has been shown to improve testing performance on datasets with time-varying features [57].
4) **Graded spikes:** Passive dendritic properties can attenuate action potentials, as can the cable-like properties of the axon. This feature can be coarsely accounted for as graded spikes. Each neuron has an additional learnable parameter that determines how to scale an output spike. Neuronal activations are no longer constrained to $\{1, 0\}$. Can this still be thought of as an SNN? From an engineering standpoint, if a spike must be broadcast to a variety of downstream neurons with an 8- or 16-bit destination address, then adding another several bits to the payload can be worth it. The second-generation Loihi chip from Intel Labs incorporates graded spikes in such a way that sparsity is preserved. Furthermore, the vector of learned values scales linearly with the number of neurons in a network, rather than quadratically with weights. It, therefore, contributes a minor cost in comparison to other components of an SNN [175].
5) **Learnable thresholds** have *not* been shown to help the training process. This is likely due to the discrete nature of thresholds, giving rise to nondifferentiable operators in a computational graph. On the other hand, normalizing the values that are passed into a threshold significantly helps. Adopting batch normalization in convolutional networks helps boost performance, and learnable normalization approaches may act as an effective surrogate for learnable thresholds [176], [177], [178].
6) **Pooling** is effective for downsampling large spatial dimensions in convolutional networks and achieving translational invariance. If max pooling is applied to a sparse, spiking tensor, then tie-breaking between 1's and 0's does not make much sense. One might expect

that we can borrow ideas from training binarized neural networks, where pooling is applied to the activations *before* they are thresholded to binarized quantities. This corresponds to applying pooling to the membrane potential in a manner that resembles a form of "local lateral inhibition." However, this does not necessarily lead to optimal performance in SNNs. Interestingly, Fang et al. applied pooling to the spikes instead. Where multiple spikes occurred in a pooling window, a tie-break would occur randomly among them [171]. While no reason was given for doing this, it, nonetheless, achieved state-of-the-art (at the time) performance on a series of computer vision problems. Our best guess is that this randomness acted as a type of regularization. Whether max pooling or average pooling is used can be treated as a hyperparameter. As an alternative, SynSense's neuromorphic hardware adopts sum pooling, where spatial dimensions are reduced by rerouting the spikes in a receptive field to a common postsynaptic neuron.

7) *Optimizer:* Most SNNs default to the Adam optimizer as they have classically been shown to be robust when used with sequential models [179]. As SNNs become deeper, stochastic gradient descent with momentum seems to increase in prevalence over the Adam optimizer. The reader is referred to Godbole et al.'s [180] deep learning tuning playbook for a systematic approach to hyperparameter optimization that applies generally.

F. Intersection Between Backprop and Local Learning

An interesting result arises when comparing backpropagation pathways that traverse varying durations of time. The derivative of the hidden state over time is $\partial U[t]/\partial U[t-1] = \beta$ as per (4). A gradient that backpropagates through n time steps is scaled by β^n. For a leaky neuron, we get $\beta < 1$, which causes the magnitude of a weight update to exponentially diminish with time between a pair of spikes. This proportionality is illustrated in Fig. 10(b). This result shows how the strength of a synaptic update is exponentially proportional to the spike time difference between presynaptic and postsynaptic neurons. In other words, weight updates from BPTT closely resemble weight updates from STDP learning curves (see Appendix C2) [33].

Is this link just a coincidence? BPTT was derived from function optimization. STDP is a model of biological observation. Despite being developed via completely independent means, they converge upon an identical result. This could have immediate practical implications, where hardware accelerators that train models can excise a chunk of BPTT and replace it with the significantly cheaper and local STDP rule. Adopting such an approach might be thought of as an online variant of BPTT or as a gradient-modulated form of STDP.

G. Long-Term Temporal Dependencies

Neural and synaptic time constants span timescales typically on the order of one to hundreds of milliseconds. With such time scales, it is difficult to solve problems that require long-range associations that are larger than the slowest neuron or synaptic time constant. Such problems are common in natural language processing and reinforcement learning, and are key to understanding behavior and decision-making in humans. This challenge is a huge burden on the learning process, where vanishing gradients drastically slow the convergence of the neural network. LSTMs [181] and, later, GRUs [182] introduced slow dynamics designed to overcome memory and vanishing gradient problems in RNNs. Thus, a natural solution for networks of spiking neurons is to complement the fast timescales of neural dynamics with a variety of slower dynamics. Mixing discrete and continuous dynamics may enable SNNs to learn features that occur on a vast range of timescales. Examples of slower dynamics include the following.

1) *Adaptive thresholds:* After a neuron fires, it enters a refractory period during which it is more difficult to elicit further spikes from the neuron. This can be modeled by increasing the firing threshold of the neuron θ every time the neuron emits a spike. After a sufficient time in which the neuron has spiked, the threshold relaxes back to a steady-state value. Homeostatic thresholds are known to promote neuronal stability in correlated learning rules, such as STDP, which favors long-term potentiation at high frequencies regardless of spike timing [183], [184]. More recently, it has been found to benefit gradient-based learning in SNNs as well [169] (see Appendix C3).
2) *Recurrent attention:* Hugely popularized from natural language generation, self-attention finds correlations between tokens of vast sequence lengths by feeding a model with all sequential inputs at once. This representation of data is not quite how the brain processes data. Several approaches have approximated self-attention into a sequence of recurrent operations, where SpikeGPT is the first application in the spiking domain and successfully achieved language generation [83]. In addition to more complex state-based computation, SpikeGPT additionally employs dynamical weights that vary over time.
3) *Axonal delays:* The wide variety of axon lengths means that there is a wide range of spike propagation delays. Some neurons have axons as short as 1 mm, whereas those in the sciatic nerve can extend up to a meter in length. The axonal delay can be a learned parameter spanning multiple time steps [73], [185], [186]. A lesser explored approach accounts for the varying delays in not only axons but also across the dendritic tree of a neuron. Coupling axonal and

dendritic delays together allows for a fixed delay per synapse.

4) *Membrane dynamics:* We already know how the membrane potential can trigger spiking, but how does spiking impact the membrane? Rapid changes in voltage cause an electric field build-up that leads to temperature changes in cells. Joule heating scales quadratically with voltage changes, which affects the geometric structure of neurons and cascades into a change in membrane capacitance (and, thus, time constants). Decay rate modulation as a function of spike emission can act as a second-order mechanism to generate neuron-specific refractory dynamics [187].
5) *Multistable neural activity:* Strong recurrent connections in biological neural networks can support multistable dynamics [188], which facilitates stable information storage over time. Such dynamics, often called attractor neural networks [189], are believed to underpin working memory in the brain [190], [191] and are often attributed to the prefrontal cortex. The training of such networks using gradient descent is challenging and has not been attempted using SNNs as of yet [192].

Several rudimentary slow timescale dynamics have been tested in gradient-based approaches to training SNNs with a good deal of success [73], [169], but there are several neuronal dynamics that are yet to be explored. LSTMs showed us the importance of temporal regulation of information and effectively cured the short-term memory problem that plagued RNNs. Translating more nuanced neuronal features into gradient-based learning frameworks can undoubtedly strengthen the ability of SNNs to represent dynamical data in an efficient manner.

V. ONLINE LEARNING

A. Temporal Locality

As incredible as our brains are, sadly, they are not time machines. It is highly unlikely that our neurons are breaching the space-time continuum to explicitly reference historical states to run the BPTT algorithm. As with all computers, brains operate on a physical substrate, which dictates the operations that it can handle and where memory is located. While conventional computers operate on an abstraction layer, memory is delocalized and communicated on demand, thus paying a considerable price in latency and energy. Brains are believed to operate on local information, which means that the best performing approaches in temporal deep learning, namely, BPTT, are biologically implausible. This is because BPTT requires the storage of the past inputs and states in memory. As a result, the required memory scales with time, a property that limits BPTT to small temporal dependencies. To solve this problem, BPTT assumes a finite sequence length before making an update while truncating the gradients in time. This, however, severely restricts the temporal dependencies that can be learned.

The constraint imposed on brain-inspired learning algorithms is that the calculation of a gradient should, much like the forward pass, be temporally local, i.e., they only depend on values available at either present time t or $t-1$. To address this, we turn to online algorithms that adhere to *temporal locality*. Real-time recurrent learning (RTRL) proposed back in 1989 is one prominent example.

B. Real-Time Recurrent Learning

RTRL estimates the same gradients as BPTT but relies on a set of different computations that make it temporally, but not spatially, local [193]. Since RTRL's memory requirement does not grow with time, why is it not used in favor of BPTT? BPTT's memory usage scales with the product of time and the number of neurons; it is $\mathcal{O}(nT)$. For RTRL, an additional set of computations must be introduced to enable the network to keep track of a gradient that evolves with time. These additional computations result in a $\mathcal{O}(n^3)$ memory requirement, which often exceeds the demands of BPTT. However, the push for continuously learning systems that can run indefinitely long has cast a spotlight back on RTRL (and variants [194], [195], [196], [197], [198]), with a focus on improving computational and memory efficiency.

Let us derive what new information needs to be propagated forward to enable real-time gradient calculation for an SNN. As in (7), let t denote real time in the calculation of $\partial\mathcal{L}/\partial W$, and let the instantaneous loss $\mathcal{L}[t]$ be a measure of how well the instantaneously predicted output $\hat{Y}[t]$ matches the target output $Y[t]$. Depending on the type of loss function in use, $\hat{Y}[t]$ might simply be the spike output of the final layer $S_{\text{out}}[t]$ or the membrane potential $U[t]$. In either case, $\partial\mathcal{L}[t]/U[t]$ does not depend on any values that are not present at t, so it is natural to calculate this term in an online manner. The key problem is deriving $\partial U[t]/\partial W$ such that it only relies on values presently available at $t-1$ and t.

First, we define the influence of parameter W on the membrane potential $U[t]$ as $m[t]$, which serves to track the derivative of the present-time membrane potential with respect to the weight. We then unpack it by one time step

$$m[t] = \frac{\partial U[t]}{\partial W} = \sum_{s\leq t} \frac{\partial U[t]}{\partial W[s]} = \underbrace{\sum_{s\leq t-1} \frac{\partial U[t]}{\partial W[s]}}_{\text{prior}} + \underbrace{\frac{\partial U[t]}{\partial W[t]}}_{\text{immediate}}. \tag{10}$$

The immediate and prior influence components are graphically illustrated in Fig. 10(a). The immediate influence is also natural to calculate online and evaluates the unweighted input to the neuron $X[t]$. The prior influence

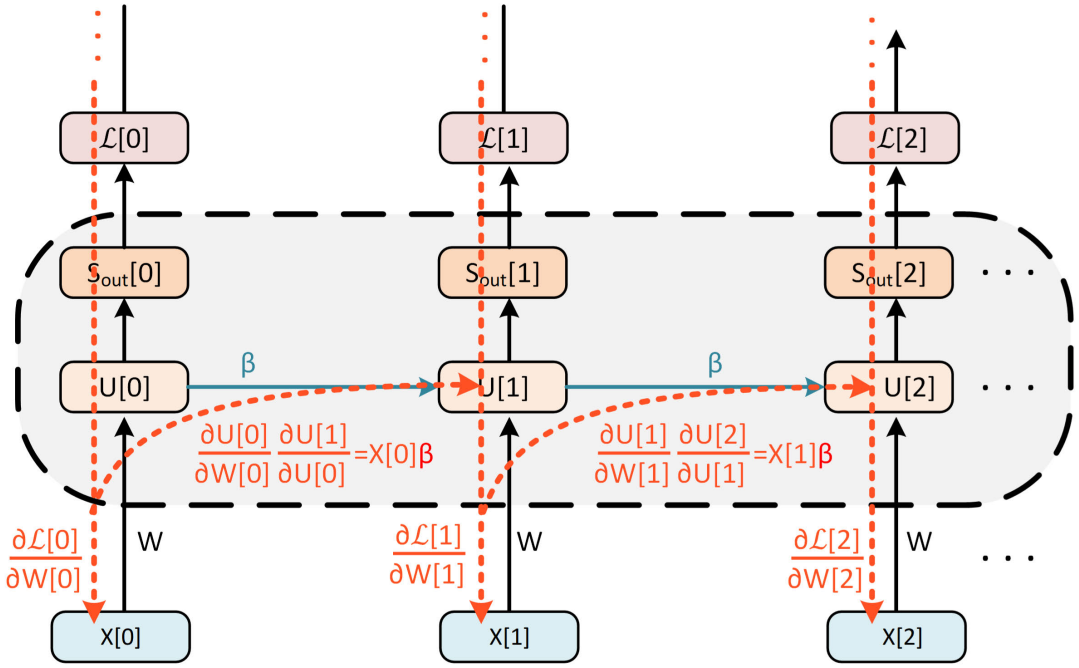


Fig. 12. *RTRL gradient pathways. The node for synaptic current, I, has been removed as it does not alter the result here.*

relies on historical components of the network

$$\sum_{s\leq t-1}\frac{\partial U[t]}{\partial W[s]}=\sum_{s\leq t-1}\underbrace{\frac{\partial U[t]}{\partial U[t-1]}}_{\text{temporal}}\frac{\partial U[t-1]}{\partial W[s]}. \tag{11}$$

Based on (4), in the absence of explicitly recurrent connections, the temporal term evaluates to β. From (10), the second term is the influence of parameters on $U[t-1]$, which is by definition $m[t-1]$. Substituting these back into (10) gives

$$m[t]=\beta m[t-1]+x[t]. \tag{12}$$

This recursive formula is updated by passing the unweighted input directly to $m[t]$ and recursively decaying the influence term by the membrane potential decay rate β. The gradient that is ultimately used with the optimizer can be derived with the chain rule

$$\frac{\partial\mathcal{L}[t]}{\partial W}=\frac{\partial\mathcal{L}[t]}{\partial U[t]}\frac{\partial U[t]}{\partial W}\equiv\bar{c}[t]m[t] \tag{13}$$

where $\bar{c}[t]=\partial\mathcal{L}[t]/\partial U[t]$ is the immediate credit assignment value obtained by backpropagating the instantaneous loss to the hidden state of the neuron, for example, by using a surrogate gradient approach. The calculation of $m[t]$ only ever depends on present time inputs and the influence at $t-1$, thus enabling the loss to be calculated in an online manner. The input spike now plays a role in not only modulating the membrane potential of the neuron but also the influence $m[t]$. The general flow of gradients is depicted in Fig. 12.

An intuitive, though incomplete, way to think about RTRL is given as follows. By reference to Fig. 12, at each time step, a backward pass that does not account for the history of weight updates is applied: $\partial\mathcal{L}[0]/\partial W[0]$ (the immediate influence). Rather than directing gradients backward through time, the partial derivative $\partial U[0]/\partial W[0]$ is "pushed" forward in time. In doing so, it is scaled by the temporal term, $X[0]\beta$. This term modulates the immediate influence at the next time step. This can be thought of as a gradient term that "snowballs" forward in time as a result of modulating and accumulating with the immediate influence term but also loses a bit of "momentum" every time the temporal term β decays it.

In the example above, the RTRL approach to training SNNs was only derived for a single neuron and a single parameter. A full-scale neural network replaces the influence value with an influence matrix $\boldsymbol{M}[t]\in\mathbb{R}^{n\times P}$, where n is the number of neurons and P is the number of parameters (approximately $\mathcal{O}(n^2)$ memory). Therefore, the memory requirements of the influence matrix scales with $\mathcal{O}(n^3)$.

Recent focus on online learning aims to reduce the memory and computational demands of RTRL. This is generally achieved by decomposing the influence matrix into simpler parts, approximating the calculation of $\boldsymbol{M}[t]$ by either completely removing terms or trading them for stochastic noise instead [194], [195], [196], [197]. Marschall et al. provide a systematic treatment of approximations to

RTRL in RNNs in [198], and variations of online learning have been applied specifically to SNNs in [109], [110], and [199].

C. RTRL Variants in SNNs

Since 2020, a flurry of forward-mode learning algorithms has been tailored to SNNs [200]. All such works either modify, rederive, or approximate RTRL.

1) *e-Prop [109]:* RTRL is combined with surrogate gradient descent. Recurrent spiking neurons are used where output spikes are linearly transformed and then fed back to the input of the same neurons. The computational graph is detached at the explicit recurrent operation but retained for implicit recurrence (i.e., where membrane potential evolves over time). Projecting output spikes into a higher dimensional recurrent space acts like a reservoir though it leads to biased gradient estimators that underperform compared to BPTT.
2) *decolle [110]:* "Deep continuous online learning" also combines RTRL with surrogate gradient descent. This time, greedy local losses are applied at **every** layer [143]. As such, errors only need to be propagated back to a single layer at a time. This means that errors do not need to traverse through a huge network, which reduces the burden of the spatial credit assignment problem. This brings about two challenges: not many problems can be cast into a form with definable local losses and greedy local learning prioritizes immediate gains without considering an overall objective.
3) *OSTL [201]:* "Online spatiotemporal learning" rederives RTRL. The spatial components of backpropagation and temporal components are factored into two separate terms, e.g., one that tracks the "immediate" influence and the other one that tracks the "prior influence" from (10).
4) *ETLP [202]:* "Event-based three-factor local plasticity" combines e-prop with direct random target projection (DRTP) [142]. In other words, the weights in the final layer are updated based on an approximation of RTRL. Earlier layers are updated based on partial derivatives that do not rely on a global loss and are spatially "local" to the layer. Instead, the target output is used to modulate these gradients. This addresses spatial credit assignment by using signals from a target, rather than backpropagating gradients in the immediate influence term of (10). The cost is that it both inherits drawbacks from e-prop and DRTP. DRTP prioritizes immediate gains without considering an overall objective, similar to greedy local learning.
5) *OSTTP [203]:* "Online spatiotemporal learning with target projection" combines OSTL (functionally equivalent to RTRL) with DRTP. It inherits the drawbacks of DRTP while addressing the spatial credit assignment problem.
6) *FPTT [204]:* "Forward propagation through time" considers RTRL for sequence-to-sequence models with time varying losses. A regularization term is applied to the loss at each step to ensure stability during the training process. Yin et al. [205] subsequently applied FPTT to SNNs with more complex neuron models with richer dynamics.

This is a nonexhaustive list of RTRL alternatives and can appear quite daunting at first. However, all approaches effectively stem from RTRL. The dominant trends include the following:

1) approximating RTRL to test how much of an approximation the training procedure can tolerate without completely failing [109];
2) replacing the immediate influence with global modulation of a loss or target to address spatial credit assignment [110], [202], [203];
3) modifying the objective to promote stable training dynamics [204];
4) identifying similarities to biology by factorizing RTRL into eligibility traces and/or three-factor learning rules [109], [202], [205].

Several RTRL variants claim to outperform BPTT in terms of loss minimization though we take caution with such claims as the two effectively become identical to BPTT for the case where weight updates are deferred to the end of a sequence. We also note caution with claims that suggest improvements over RTRL, as RTRL can be thought of as the most general case of forward-model learning applied to any generic architecture. Most reductions in computational complexity arise because they are narrowly considered for specific architectures or otherwise introduce approximations into their models. In contrast, Tallec and Ollivier [194] developed an "unbiased online recurrent optimization" scheme where stochastic noise is used and ultimately canceled out, leading to quadratic (rather than cubic) computational complexity with network size.

D. Spatial Locality

While temporal locality relies on a learning rule that depends only on the present state of the network, spatial locality requires each update to be derived from a node immediately adjacent to the parameter. The biologically motivated learning rules described in Section III-D address the spatial credit assignment problem by either replacing the global error signal with local errors or replacing analytical/numerical derivatives with random noise [138].

The more "natural" approach to online learning is perceived to be via unsupervised learning with synaptic plasticity rules, such as STDP [33], [206] and variants of STDP (see Appendix C2) [207], [208], [209], [210]. These approaches are directly inspired by experimental relationships between spike times and changes to synaptic conductance. Input data are fed to a network, and weights are updated based on the order and firing times of each

pair of connected neurons [see Fig. 10(b)]. The interpretation is that, if a neuron causes another neuron to fire, then their synaptic strength should be increased. If a pair of neurons appears uncorrelated, their synaptic strength should be decreased. It follows the Hebbian mantra of "*neurons that fire together wire together*" [147].

There is a common misconception that backprop and STDP-like learning rules are at odds with one other, competing to be the long-term solution for training connectionist networks. On the one hand, it is thought that STDP deserves more attention as it scales with less complexity than backprop. STDP adheres to temporal and spatial locality, as each synaptic update only relies on information from immediately adjacent nodes. However, this relationship necessarily arises as STDP was reported using data from "immediately adjacent" neurons. On the other hand, STDP fails to compete with backprop on remotely challenging datasets. However, backprop was designed with function optimization in mind, while STDP emerged as a physiological observation. The mere fact that STDP is capable at all of obtaining competitive results on tasks originally intended for supervised learning (such as classifying the MNIST dataset), no matter how simple, is quite a wonder. Rather than focusing on what divides backprop and STDP, the pursuit of more effective learning rules will more likely benefit by understanding how the two intersect.

We demonstrated in Section IV-F how surrogate gradient descent via BPTT subsumes the effect of STDP. Spike time differences result in exponentially decaying weight update magnitudes such that half of the learning window of STDP is already accounted for within the BPTT algorithm [see Fig. 10(b)]. Bengio et al. [211] previously made the case that STDP resembles stochastic gradient descent, provided that STDP is supplemented with gradient feedback [212]. This specifically relates to the case where a neuron's firing rate is interpreted as its activation. Here, we have demonstrated that no modification needs to be made to the BPTT algorithm for it to account for STDP-like effects and is not limited to any specific neural code, such as the firing rate. The common theme is that STDP may benefit from integrating error-triggered plasticity to provide meaningful feedback to training a network [213].

VI. OUTLOOK

Designing a neural network was once thought to be strictly an engineering problem, whereas mapping the brain was a scientific curiosity [214]. With the intersection between deep learning and neuroscience broadening, and brains being able to solve complex problems much more efficiently, this view is poised to change. From the scientist's view, deep learning and brain activity have shown many correlates, which leads us to believe that there is much untapped insight that ANNs can offer in the ambitious quest of understanding biological learning. For example, the activity across layers of a neural network has repeatedly shown similarities to experimental activity in the brain. This includes links between convolutional neural networks and measured activity from the visual cortex [215], [216], [217] and auditory processing regions [218]. Activity levels across populations of neurons have been quantified in many studies, but SNNs might inform us of the specific nature of such activity.

From the engineer's perspective, neuron models derived from experimental results have allowed us to design extremely energy-efficient networks when running on hardware tailored to SNNs [219], [220], [221], [222], [223], [224], [225]. Improvements in energy consumption of up to two to three orders of magnitude have been reported when compared to conventional ANN acceleration on embedded hardware, which provides empirical validation of the benefits available from the three S's: spikes, sparsity, and static data suppression (or event-driven processing) [20], [226], [227], [228], [229], [230]. These energy and latency benefits are derived from simply applying neuron models to connectionist networks, but there is so much more left to explore.

It is safe to say that the energy benefits afforded by spikes are uncontroversial. However, a more challenging question to address is: are spikes actually good for computation? It could be those years of evolution-determined spikes solved the long-range signal transmission problem in living organisms, and everything else had to adapt to fit this constraint. If this were true, then spike-based computation would be Pareto optimal with a proclivity toward energy efficiency and latency. However, until we amass more evidence of a spike's purpose, we have some intuition as to where spikes shine in computation.

1) *Hybrid dynamical systems:* SNNs can model a broad class of dynamical systems by coupling discrete and continuous time dynamics into one system. Discontinuities are present in many physical systems, and spiking neuron models are a natural fit to model such dynamics.
2) *Discrete function approximators:* Neural networks are universal function approximators, where discrete functions are considered to be modeled sufficiently well by continuous approximations. Spikes are capable of precisely defining discrete functions without approximation.
3) *Multiplexing:* Spikes can encode different information in spike rate, times, or burst counts. Repurposing the same spikes offers a sensible way to condense the amount of computation required by a system.
4) *Message packets:* By compressing the representation of information, spikes can be thought of as packets of messages that are unlikely to collide as they travel across a network. In contrast, a digital system requires a synchronous clock to signal that a communication channel is available for a message to pass through (even when modeling asynchronous systems).
5) *Coincidence detection:* Neural information can be encoded based on spatially disparate but temporally

proximate input spikes on a target neuron. It may be the case that isolated input spikes are insufficient to elicit a spike from the output neuron. However, if two incident spikes occur on a timescale faster than the target neuron membrane potential decay rate, this could push the potential beyond the threshold and trigger an output spike. In such a case, associative learning is taking place across neurons that are not directly connected. Although coincidence detection can be programmed in a continuous-time system without spikes, a theoretical analysis has shown that the processing rate of a coincidence detector neuron is faster than the rate at which information is passed to a neuron [231], [232].

6) *Noise robustness:* While analog signals are highly susceptible to noise, digital signals are far more robust in long-range communication. Neurons seem to have figured this out by performing analog computation via integration at the soma, and digital communication along the axon. It is possible that any noise incident during analog computation at the soma is subsumed into the subthreshold dynamics of the neuron and, therefore, eliminated. In terms of neural coding, a similar analogy can be made to spike rates and spike times. Pathways that are susceptible to adversarial attacks or timing perturbations could learn to be represented as a rate, which, otherwise, mitigates timing disturbances in temporal codes.
7) *Modality normalization:* A unified representation of sensory input (e.g., vision and auditory) as spikes is nature's way of normalizing data. While this benefit is not exclusive to spikes (i.e., continuous data streams in nonspiking networks may also be normalized), early empirical evidence has shown instances where multimodal SNNs outperform convolutional neural networks on equivalent tasks [20], [228].
8) *Mixed-mode differentiation:* While most modern deep learning frameworks rely on reverse-mode autodifferentiation [233], it is in stark contrast to how the spatial credit assignment problem is treated in biological organisms. If we are to draw parallels between backpropagation and the brain, it is far more likely that approximations of forward-mode autodifferentiation are being used instead. Equation (12) describes how to propagate gradient-related terms forward in time to implement online learning, where such terms could be approximated by eligibility traces that keep track of presynaptic neuron activity in the form of calcium ions and fades over time [109], [234]. SNNs offer a natural way to use mixed-mode differentiation by projecting temporal terms in the gradient calculation from (11) into the future via forward-mode differentiation while taking advantage of the computational complexity of reverse-mode autodifferentiation for spatial terms [71], [110].

A better understanding of the problems spikes are best suited for, beyond addressing just energy efficiency, will be important in directing SNNs to meaningful tasks. The above list is a nonexhaustive start to intuit where that might be. Thus far, we have primarily viewed the benefits of SNNs by examining individual spikes. For example, the advantages derived from sparsity and single-bit communication arise at the level of an individual spiking neuron: how a spike promotes sparsity, how it contributes to a neural encoding strategy, and how it can be used in conjunction with modern deep learning, backprop, and gradient descent. Despite the advances yielded by this spike-centric view, it is important not to develop tunnel vision. New advances are likely to come from a deeper understanding of spikes acting collectively, much like the progression from atoms to waves in physics.

Designing learning rules that operate with brain-like performance is far less trivial than substituting a set of artificial neurons with spiking neurons. It would be incredibly elegant if a unified principle governed how the brain learns. However, the diversity of neurons, functions, and brain regions implies that a heterogeneous system rich in objectives and synaptic update rules is more likely and might require us to use all of the weapons in our arsenal of machine learning tools. It is likely that a better understanding of biological learning will be amassed by observing the behavior of a collection of spikes distributed across brain regions. Ongoing advances in procuring large-scale electrophysiological recordings at the neuron level can give us a window into observing how populations of spikes are orchestrated to handle credit assignment so efficiently and, at the very least, give us a more refined toolkit to developing theories that may advance deep learning [235], [236]. After all, it was not a single atom that led to the silicon revolution but, rather, a mass of particles and their collective fields. A stronger understanding of the computational benefits of spikes may require us to think at a larger scale in terms of the "fields" of spikes.

As the known benefits of SNNs manifest in the physical quantities of energy and latency, it will take more than just a machine learning mind to navigate the tangled highways of 100 trillion synapses. It will take a concerted effort between machine learning engineers, neuroscientists, and circuit designers to put spikes in the front seat of deep learning.

ADDITIONAL MATERIALS

A series of interactive tutorials complementary to this article are available in the documentation for our Python package designed for gradient-based learning using SNNs, *snnTorch* [237], at the following link: https://snntorch.readthedocs.io/en/latest/tutorials/index.html.

We invite additional contributions and tutorial content from the neuromorphic community.

APPENDIX

A. From Artificial to Spiking Neural Networks

1) Forward Euler Method to Solving Spiking Neuron Models: The time derivative $dU(t)/dt$ is substituted into (1) without taking the limit $\Delta t \rightarrow 0$

$$\tau \frac{U(t+\Delta t) - U(t)}{\Delta t} = -U(t) + I_{\text{in}}(t)R. \tag{14}$$

For small enough values of Δt, this provides a sufficient approximation of continuous-time integration. Isolating the membrane potential at the next time step on the left-hand side of the equation gives

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t) + \frac{\Delta t}{\tau} I_{\text{in}}(t)R. \tag{15}$$

To single out the leaky membrane potential dynamics, assume that there is no input current $I_{\text{in}}(t) = 0\ A$

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t). \tag{16}$$

Let the ratio of subsequent values of U, i.e., $U(t+\Delta t)/U(t)$, be the decay rate of the membrane potential, also known as the inverse time constant. From (15), this implies that $\beta = (1 - \Delta t/\tau)$.

Assume that t is discretized into sequential time steps such that $\Delta t = 1$. To further reduce the number of hyperparameters from (15), assume that $R = 1\ \Omega$. This leads to the result in (3), where the following representation is shifted by one time step:

$$\beta = (1 - \frac{1}{\tau}) \implies U[t+1] = \beta U[t] + (1-\beta) I_{\text{in}}[t+1]. \tag{17}$$

The input current is weighted by $(1-\beta)$ and time-shifted by one step such that it can instantaneously contribute to membrane potential. While this is not a physiologically precise assumption, it casts the neuron model into a form that better resembles an RNN. β can be solved using the continuous-time solution from (2). In the absence of a current injection

$$U(t) = U_0 e^{-t/\tau} \tag{18}$$

where U_0 is the initial membrane potential at $t = 0$. Assuming that (18) is computed at discrete steps of t, $(t+\Delta t)$, $(t+2\Delta t)$, ..., then the ratio of membrane potential across two subsequent steps can be calculated using

$$\beta = \frac{U_0 e^{-(t+\Delta t)/\tau}}{U_0 e^{-t/\tau}} = \frac{U_0 e^{-(t+2\Delta t)/\tau}}{U_0 e^{-(t+\Delta t)/\tau}} = \dots$$
$$\implies \beta = e^{-\Delta t/\tau}. \tag{19}$$

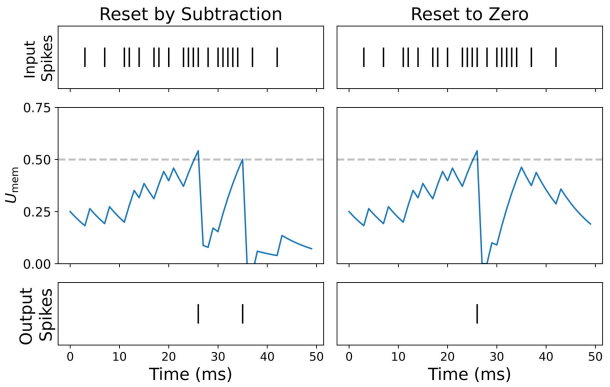


Fig. 13. ***Reset by subtraction versus reset-to-zero. Threshold set to $\theta = 0.5$.***

It is preferable to calculate β using (19) rather than $\beta = (1 - \Delta t/\tau)$, as the latter is only precise for $\Delta t << \tau$. This result for β can then be used in (17).

A second nonphysiological assumption is made, where the effect of $(1-\beta)$ is absorbed by a learnable weight W

$$WX[t] = I_{\text{in}}[t]. \tag{20}$$

This can be interpreted the following way. $X[t]$ is an input voltage, spike, or unweighted current, and is scaled by the synaptic conductance W to generate a current injection to the neuron. This leads to the following result:

$$U[t+1] = \beta U[t] + WX[t+1] \tag{21}$$

where the effects of W and β are decoupled, thus favoring simplicity over biological precision.

To arrive at (4), a reset function is appended, which activates every time an output spike is triggered. The reset mechanism can be implemented by either subtracting the threshold at the onset of a spike as in (4) or by forcing the membrane potential to zero (Fig. 13)

$$U[t+1] = \underbrace{\beta U[t]}_{\text{decay}} + \underbrace{WX[t]}_{\text{input}} - \underbrace{S_{\text{out}}(\beta U[t] + WX[t])}_{\text{reset-to-zero}}. \tag{22}$$

In general, reset-by-subtraction is thought to be better for performance as it retains residual superthreshold information, while reset-to-zero is more efficient as $U[t]$ will always be forced to zero when a spike is triggered. This has been formally demonstrated in ANN-SNN conversion approaches (see Section IV-A) though it has not yet been characterized for natively trained SNNs. The two approaches will converge for a small enough time window where $U[t]$ is assumed to increase in a finite period of time.

B. Spike Encoding

The following spike encoding mechanisms and loss functions are described with respect to a single sample of data.

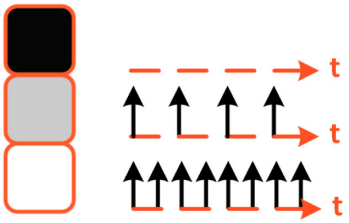


Fig. 14. *Rate-coded input pixel. An input pixel of greater intensity corresponds to a higher firing rate.*

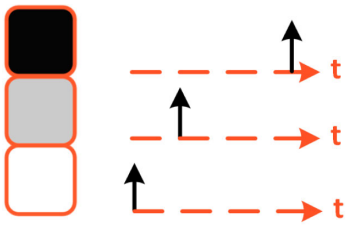


Fig. 15. *Latency-coded input pixel. An input pixel of greater intensity corresponds to an earlier spike time.*

They can be generalized to multiple samples as is common practice in deep learning to process data in batches.

1) Rate-Coded Input Conversion: An example of the conversion of an input sample to a rate-coded spike train follows (Fig. 14). Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ be a sample from the MNIST dataset, where $m = n = 28$. We wish to convert $\boldsymbol{X}$ to a rate-coded 3-D tensor $\boldsymbol{R} \in \mathbb{R}^{m \times n \times t}$, where t is the number of time steps. Each feature of the original sample X_{ij} is encoded separately, where the normalized pixel intensity (between 0 and 1) is the probability a spike occurs at any given time step. This can be treated as a Bernoulli trial, a special case of the binomial distribution $R_{ijk} \sim B(n, p)$, where the number of trials is $n = 1$, and the probability of success (spiking) is $p = X_{ij}$. Explicitly, the probability a spike occurs is

$$P(R_{ijk} = 1) = X_{ij} = 1 - P(R_{ijk} = 0). \tag{23}$$

Sampling from the Bernoulli distribution for every feature at each time step will populate the 3-D tensor $\boldsymbol{R}$ with 1's and 0's. For an MNIST image, a pure white pixel $X_{ij} = 1$ corresponds to a 100% probability of spiking. A pure black pixel $X_{ij} = 0$ will never generate a spike. A gray pixel of value $X_{ij} = 0.5$ will have an equal probability of sampling either "1" or "0." As the number of time steps $t \rightarrow \infty$, the proportion of spikes is expected to approach 0.5.

2) Latency-Coded Input Conversion: The logarithmic dependence between input feature intensity and spiking timing can be derived using an RC circuit model (Fig. 15). Starting with the general solution of the membrane potential with respect to the input current in (2) and nulling out the initial conditions $U_0 = 0$, we obtain

$$U(t) = I_{\text{in}} R(1 - e^{-t/\tau}). \tag{24}$$

For a constant current injection, $U(t)$ will exponentially relax toward a steady-state value of $I_{\text{in}}R$. Say a spike is emitted when $U(t)$ reaches a threshold θ. We solve for the time $U(t) = \theta$

$$t = \tau \left[\ln\left(\frac{I_{\text{in}} R}{I_{\text{in}} R - \theta} \right) \right]. \tag{25}$$

The larger the input current, the faster $U(t)$ charges up to θ, and the faster a spike occurs. The steady-state potential, $I_{\text{in}}R$, is set to the input feature x

$$t(x) = \begin{cases} \tau \left[\ln\left(\frac{x}{x - \theta} \right) \right], & x > \theta \\ \infty, & \text{otherwise.} \end{cases} \tag{26}$$

3) Rate-Coded Outputs: A vectorized implementation of determining the predicted class from rate-coded output spike trains is described (Fig. 16). Let $\vec{S}[t] \in \mathbb{R}^{N_C}$ be a time-varying vector that represents the spikes emitted from each output neuron across time, where N_C is the number of output classes. Let $\vec{c} \in \mathbb{R}^{N_C}$ be the spike count from each output neuron, which can be obtained by summing $\vec{S}[t]$ over T time steps

$$\vec{c} = \sum_{j=0}^{T} \vec{S}[t]. \tag{27}$$

The index of $\vec{c}$ with the maximum count corresponds to the predicted class

$$\hat{y} = \arg\max_i c_i. \tag{28}$$

4) Cross-Entropy Spike Rate: The spike count of the output layer $\vec{c} \in \mathbb{R}^{N_C}$ is obtained as in (27) (Fig. 17). c_i

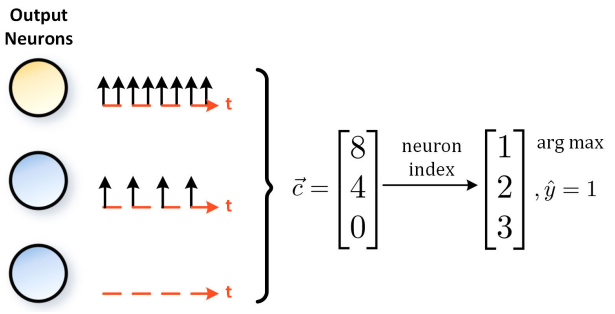


Fig. 16. *Rate-coded outputs. $\vec{c} \in \mathbb{R}^{N_C}$ is the spike count from each output neuron, where the example above shows the first neuron firing a total of eight times. $\hat{y}$ represents the index of the predicted output neuron, where it indicates that the first neuron is the correct class.*

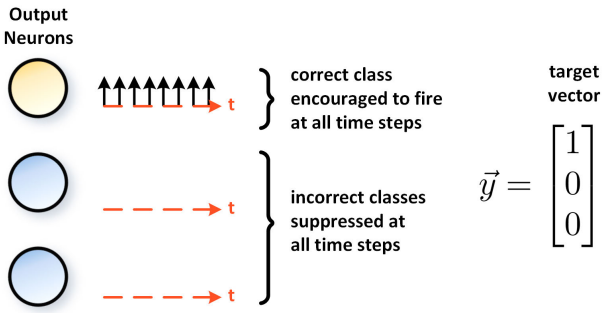


Fig. 17. ***Cross-entropy spike rate. The target vector $\vec{y}$ specifies the correct class as a one-hot encoded vector.***

is the ith element of $\vec{c}$, treated as the logits in the softmax function

$$p_i = \frac{e^{c_i}}{\sum_{i=1}^{N_C} e^{c_i}}. \tag{29}$$

The cross-entropy between p_i and the target $y_i \in \{0, 1\}^{N_C}$, which is a one-hot target vector, is obtained using

$$\mathcal{L}_{CE} = \sum_{i=0}^{N_C} y_i \log(p_i). \tag{30}$$

5) Mean Square Spike Rate: As in (27), the spike count of the output layer $\vec{c} \in \mathbb{R}^{N_C}$ is obtained (Fig. 18). c_i is the ith element of $\vec{c}$, and let $y_i \in \mathbb{R}$ be the target spike count over a period of time T for the ith output neuron. The target for the correct class should be greater than that of incorrect classes

$$\mathcal{L}_{\text{mse}} = \sum_{i}^{N_C} (y_i - c_i)^2. \tag{31}$$

6) Maximum Membrane: The logits $\vec{m} \in \mathbb{R}^{N_C}$ are obtained by taking the maximum value of the membrane potential of the output layer $\vec{U}[t] \in \mathbb{R}^{N_C}$ over time (Fig. 19)

$$\vec{m} = \max_t \vec{U}[t]. \tag{32}$$

The elements of $\vec{m}$ replace c_i in the softmax function from (29), with the cross-entropy of the result measured with respect to the target label.

Alternatively, the membrane potential is summed over time to obtain the logits

$$\vec{m} = \sum_{t}^{T} \vec{U}[t]. \tag{33}$$

7) Mean Square Membrane: Let $y_i[t]$ be a time-varying value that specifies the target membrane potential of the ith neuron at each time step (Fig. 20). The total mse is calculated by summing the loss for all T time steps and for all N_C output layer neurons

$$\mathcal{L}_{\text{mse}} = \sum_{i}^{N_C} \sum_{t}^{T} (y_i[t] - U[t])^2. \tag{34}$$

Alternatively, the time-varying target $y_i[t]$ can be replaced with a time-static target to drive the membrane potential of all neurons to a constant value. This can be an efficient implementation for a rate code, where the correct class target exceeds the threshold and all other targets are subthreshold values.

8) Cross-Entropy Latency Code: Let $\vec{f} \in \mathbb{R}^{N_C}$ be a vector containing the first spike time of each neuron in the output layer (Fig. 21). Cross-entropy minimization aims to maximize the logit of the correct class and reduce the logits of the incorrect classes. However, we wish for the correct class to spike first, which corresponds to a smaller value. Therefore, a monotonically decreasing function must be applied to $\vec{f}$. A limitless number of options are available. The work in [74] simply negates the spike times

$$\vec{f} := -\vec{f}. \tag{35}$$

Taking the inverse of each element f_i of $\vec{f}$ is also a valid option

$$f_i := \frac{1}{f_i}. \tag{36}$$

The new values of f_i then replace c_i in the softmax function from (29). Equation (36) must be treated with care, as it precludes spikes from occurring at $t = 0$; otherwise, $f_i \to \infty$.

9) Mean Square Spike Time: The spike time(s) of all neurons are specified as targets (Fig. 22). In the case where only the first spike matters, $\vec{f} \in \mathbb{R}^{N_C}$ contains the first spike time of each neuron in the output layer, and $y_i \in \mathbb{R}$ is the target spike time for the ith output neuron. The mses between the actual and target spike times of all output classes are summed together

$$\mathcal{L}_{\text{mse}} = \sum_{i}^{N_C} (y_i - f_i)^2. \tag{37}$$

This can be generalized to account for multiple spikes [73]. In this case, $\vec{f}_i$ becomes a list of emitted spike times, and $\vec{y_i}$ becomes a vector desired spike times for the ith neuron, respectively. The kth spike is sequentially taken from $\vec{f}_i$ and $\vec{y_i}$, and the mse between the two is calculated. This process is repeated n times, where n is the number of spike times that have been specified and the errors are summed

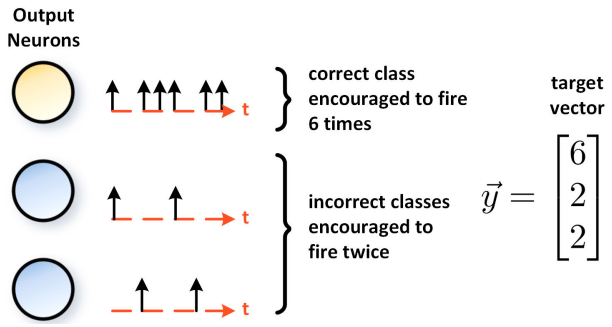


Fig. 18. *Mean square spike rate. The target vector $\bar{y}$ specifies the total desired number of spikes for each class.*

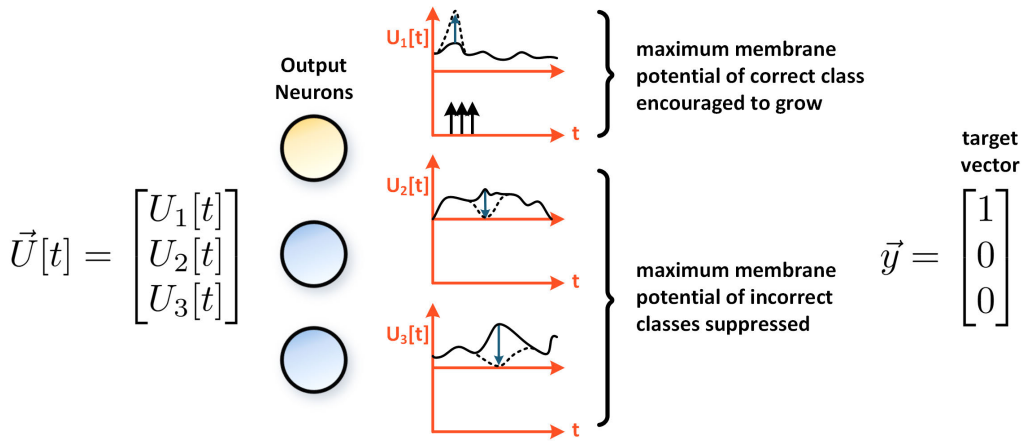


Fig. 19. *Maximum membrane. The peak membrane potential for each neuron is used in the cross-entropy loss function. This encourages the peak of the correct class to grow, while that of the incorrect class is suppressed. The effect of this is to promote more firing from the correct class and less from the incorrect class.*

together across spikes and classes

$$\mathcal{L}_{\text{mse}} = \sum_{k}^{n} \sum_{i}^{N_C} (y_{i,k} - f_{i,k})^2. \tag{38}$$

10) Mean Square Relative Spike Time: The difference between the spike time of correct and incorrect neurons is specified as a target (Fig. 23). As in Appendix B9, y_i is the desired spike time for the ith neuron, and f_i is the

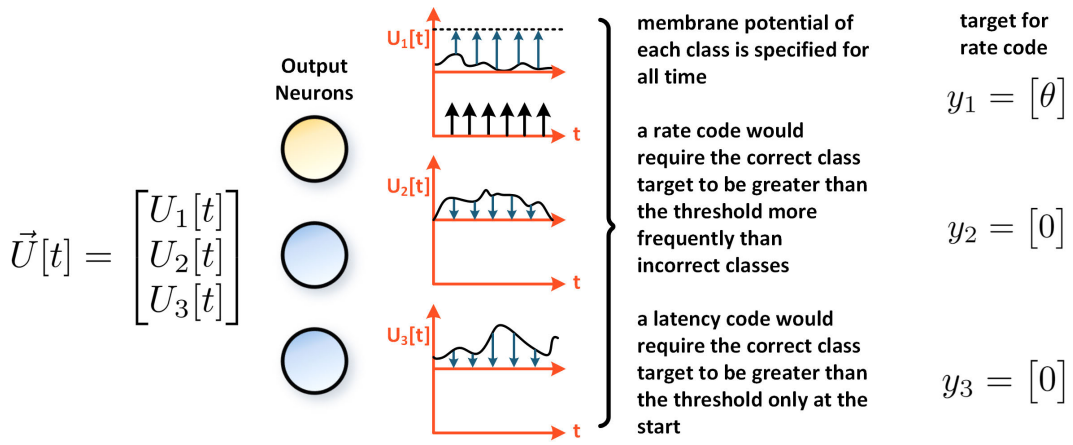


Fig. 20. *Mean square membrane. The membrane potential at each time step is applied to the mse loss function. This allows a defined membrane target. The example above sets the target at all time steps at the firing threshold for the correct class and to zero for incorrect classes.*

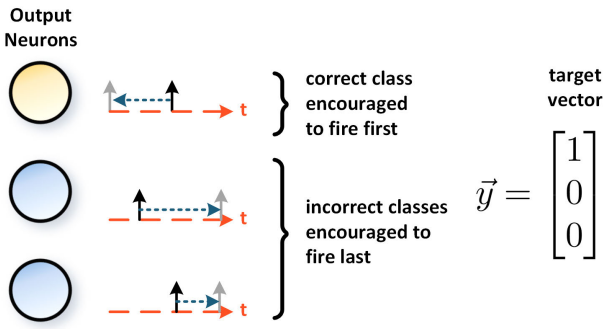


Fig. 21. ***Cross-entropy latency code. Applying the inverse (or negated) spike time to the cross-entropy loss pushes the correct class to fire first and the incorrect classes to fire later.***

actual emitted spike time. The key difference is that y_i can change throughout the training process.

Let the minimum possible spike time be $f_0 \in \mathbb{R}$. This sets the target firing time of the correct class. The target firing time of incorrect neuron classes y_i is set to

$$y_i = \begin{cases} f_0 + \gamma, & \text{if } f_i < f_0 + \gamma \\ f_i, & \text{if } f_i \geq f_0 + \gamma \end{cases} \tag{39}$$

where γ is a predefined latency, treated as a hyperparameter. In the first case, if an incorrect neuron fires at some time before the latency period γ, then a penalty will be applied. In the second case, where the incorrect neuron fires at γ steps after the correct neuron, the target is simply set to the actual spike time. These zero each other out during the loss calculation. This target y_i is then applied to the mse loss [see (38)].

11) Population Level Regularization: L1-regularization can be applied to the total number of spikes emitted at the output layer to penalize excessive firing [150], thus encouraging sparse activity at the output

$$\mathcal{L}_{L1} = \lambda_1 \sum_{t}^{T} \sum_{i}^{N_C} S_i[t] \tag{40}$$

where λ_1 is a hyperparameter controlling the influence of the regularization term and $S_i[t]$ is the spike of the ith class at time t.

Alternatively, an upper activity threshold θ_U can be applied where, if the total number of spikes for *all* neurons in layer l exceeds this threshold, only then does the regularization penalty apply

$$\mathcal{L}_U = \lambda_U \left(\left[\sum_{i}^{N} c_i^{(l)} - \theta_U \right]_+ \right)^L \tag{41}$$

where c_i is the total spike count over time for the ith neuron in layer l and N is the total number of neurons in layer l. λ_U is a hyperparameter influencing the strength of the upper activity regularization, and $[\cdot]_+$ is a linear rectification: if the total number of spikes from the layer is less than θ_U, the rectifier clips the negative result to zero such that a penalty is not added. L is typically chosen to be either 1 or 2 [113]. It is possible to swap out the spike count for a time-averaged membrane potential as well if using hidden-state variables is permissible [110].

12) Neuron Level Regularization: A lower activity threshold θ_L that specifies the lower permissible limit of firing for *each* neuron before the regularization penalty is applied

$$\mathcal{L}_L = \frac{\lambda_L}{N} \sum_{i}^{N} \left(\left[\theta_L - c_i^{(l)} \right]_+ \right)^2. \tag{42}$$

The rectification $[\cdot]_+$ now falls within the summation and is applied to the firing activity of each individual neuron, rather than a population of neurons, where λ_L is a hyperparameter that influences the strength of lower activity regularization [113]. As with population-level regularization, the spike count can also be substituted for a time-averaged membrane potential [110].

C. Training Spiking Neural Networks

1) Backpropagation Using Spike Times: A visual depiction of the following derivation is provided in Fig. 24. In the original description of SpikeProp from [119], a spike response model is used

$$\begin{aligned} U_j(t) &= \sum_{i,k} W_{i,j} I_i^{(k)}(t) \\ I_i^{(k)}(t) &= \epsilon(t - f_i^{(k)}) \end{aligned} \tag{43}$$

where $W_{i,j}$ is the weight between the ith presynaptic and jth postsynaptic neurons, $f_i^{(k)}$ is the firing time of the kth spike from the ith presynaptic neuron, and $U_j(t)$ is the membrane potential of the jth neuron. For simplicity, the "alpha function" defined in the following is frequently used for the kernel:

$$\epsilon(t) = \frac{t}{\tau} e^{1 - \frac{t}{\tau}} \Theta(t) \tag{44}$$

where τ and Θ are the time constant of the kernel and the Heaviside step function, respectively.

Consider an SNN where each target specifies the timing of the output spike emitted from the jth output neuron (y_j). This is used in the mean square spike time loss [see (37)], where f_j is the actual spike time. Rather than backpropagating in time through the entire history of the simulation, only the gradient pathway through the spike time of each neuron is taken. The gradient of the loss in weight space is then

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = \frac{\partial \mathcal{L}}{\partial f_j} \frac{\partial f_j}{\partial U_j} \frac{\partial U_j}{\partial W_{i,j}} \Big|_{t=f_j}. \tag{45}$$

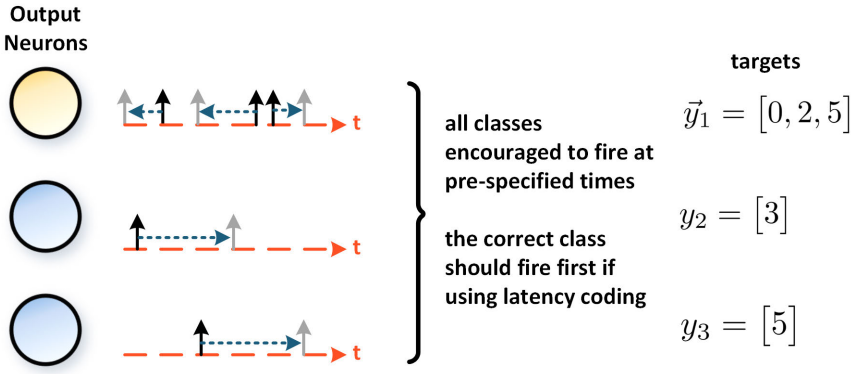


Fig. 22. ***Mean square spike time. The timing of all spikes is iterated over and sequentially applied to the mse loss function. This enables the timing for multiple spikes to be precisely defined.***

The first term on the right side evaluates to

$$\frac{\partial \mathcal{L}}{\partial f_j} = 2(y_j - f_j). \tag{46}$$

The third term can be derived from (43)

$$\frac{\partial U_j}{\partial W_{i,j}}\Big|_{t=f_j} = \sum_k I_i^{(k)}(f_j) = \sum_k \epsilon(f_j - f_i^{(k)}). \tag{47}$$

The second term in (45) can be calculated by calculating $-\partial U_j/\partial f_j\Big|_{t=f_j}$ instead and then taking the inverse. In [119], the evolution of $U_j(t)$ can be analytically solved using (43) and (44)

$$\begin{aligned}\frac{\partial f_j}{\partial U_j} &\leftarrow \left(\frac{\partial U_j}{\partial t}\Big|_{t=f_j}\right)^{-1} \\ &= \left(\sum_{i,k} W_{i,j}\frac{\partial I_i^{(k)}}{\partial t}\Big|_{t=f_j}\right)^{-1} \\ &= \left(\sum_{i,k} W_{i,j}\frac{f_j - f_i^{(k)} - \tau}{\tau^2}\left(e^{\frac{f_j - f_i^{(k)}}{\tau}} - 1\right)\right)^{-1}.\end{aligned} \tag{48}$$

Note that the input current is triggered at the onset of the presynaptic spike $t = f_i$ but is evaluated at the time of the postsynaptic spike $t = f_j$. The results can be combined to give

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = \frac{2(y_j - f_j)\sum_k I_i^{(k)}(f_j)}{\sum_{i,k} W_{i,j}(\partial I_j^{(k)}/\partial t)\Big|_{t=f_j}}. \tag{49}$$

This approach can be generalized to handle deeper layers, and the original formulation also includes delayed response kernels that are not included above for simplicity.

2) Backpropagation Using Spikes: STDP: The connection between a pair of neurons can be altered by the spikes emitted by both neurons. Several experiments have shown that the relative timing of spikes between presynaptic and postsynaptic neurons can be used to define a learning rule for updating the synaptic weight [33]. Let t_{pre} and t_{post} represent the timing of the presynaptic and postsynaptic spikes, respectively. The difference in spike time is

$$\Delta t = t_{\text{pre}} - t_{\text{post}}. \tag{50}$$

When the presynaptic neuron emits a spike before the postsynaptic neuron such that the presynaptic spike may have caused the postsynaptic spike, then the synaptic strength is expected to increase ("potentiation"). When reversed, i.e., the postsynaptic neuron spikes before the presynaptic neuron, the synaptic strength decreases ("depression"). This rule is known as STDP and has been shown to exist in various brain regions, including the visual cortex, somatosensory cortex, and hippocampus. Fitting curves to experimental measurements take the following form [33]:

$$\Delta W = \begin{cases} A_+ e^{\Delta t/\tau_+}, & \text{if } t_{\text{post}} > t_{\text{pre}} \\ A_- e^{-\Delta t/\tau_-}, & \text{if } t_{\text{post}} < t_{\text{pre}} \end{cases} \tag{51}$$

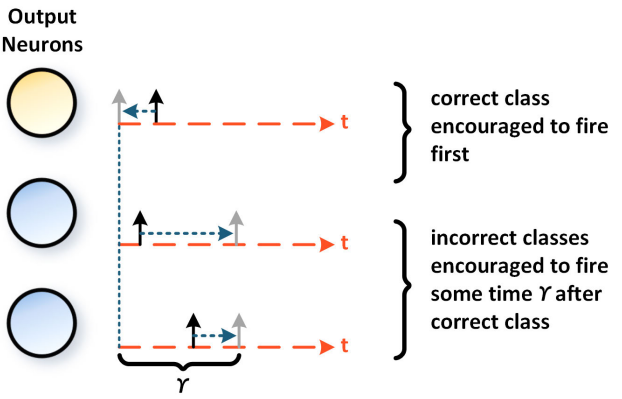


Fig. 23. ***Mean square relative spike time. The relative timing between all spikes is applied to the mse loss function, enabling a defined time window γ to occur between the correct class firing and incorrect classes firing.***

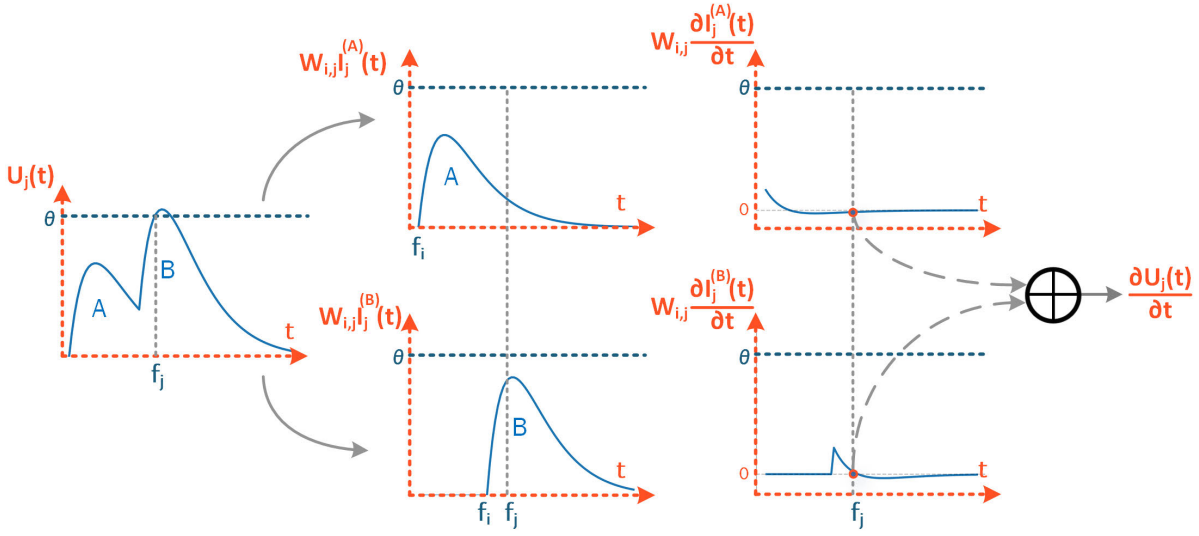


Fig. 24. *Calculation of derivative of membrane potential with respect to spike time. The superscripts (A) and (B) denote the separate contributions from each application of the kernel.*

where ΔW is the change in synaptic weight, A_+ and A_- represent the maximum amount of synaptic modulation that takes place as the difference between spike times approaches zero, and τ_+ and τ_- are the time constants that determine the strength of the update over a given interspike interval. This mechanism is illustrated in Fig. 25.

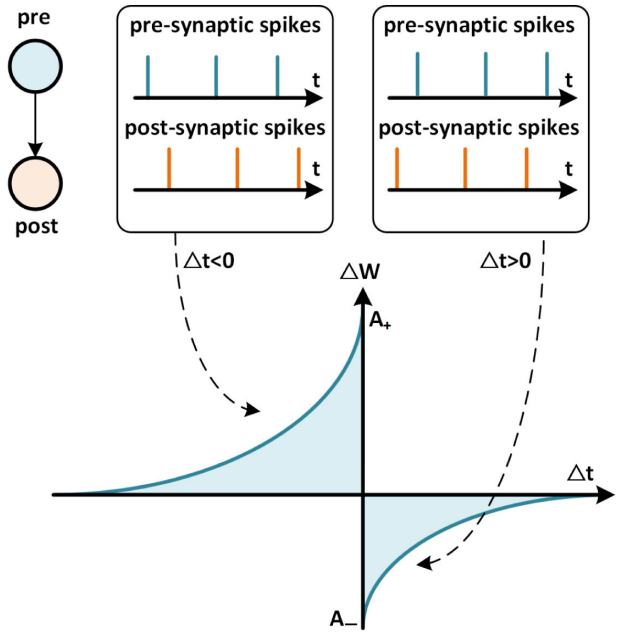


Fig. 25. *STDP learning window. If the presynaptic neuron spikes before the postsynaptic neuron, $\Delta t < 0 \implies \Delta W > 0$, and the synaptic strength between the two neurons is increased. If the presynaptic neuron spikes after the postsynaptic neuron, $\Delta t > 0 \implies \Delta W < 0$, and the synaptic strength is decreased.*

For a strong, excitatory synaptic connection, a presynaptic spike will trigger a large postsynaptic potential [refer to U in (4)]. As membrane potential approaches the threshold of neuronal firing, such an excitatory case suggests that a postsynaptic spike will likely follow a presynaptic spike. This will lead to a positive change in the synaptic weight, thus increasing the chance that a postsynaptic spike will follow a presynaptic spike in the future. This is a form of causal spiking, and STDP reinforces causal spiking by continuing to increase the strength of the synaptic connection.

Input sensory data are typically correlated in both space and time, so a network's response to a correlated spike train will increase the weights much faster than uncorrelated spike trains. This is a direct result of causal spiking. Intuitively, a group of correlated spikes from multiple presynaptic neurons will arrive at a postsynaptic neuron within a close time interval, causing stronger depolarization of the neuron membrane potential, and a higher probability of a postsynaptic spike being triggered.

However, without an upper bound, this will lead to unstable and indefinitely large growth of the synaptic weight. In practice, an upper limit should be applied to constrain potentiation. Alternatively, homeostatic mechanisms can also be used to offset this unbounded growth, such as an adaptive threshold that increases each time a spike is triggered from the neuron (see Appendix C3).

3) Long-Term Temporal Dependencies: One of the simplest implementations of an adaptive threshold is to choose a steady-state threshold θ_0 and a decay rate α

$$\theta[t] = \theta_0 + b[t] \tag{52}$$

$$b[t+1] = \alpha b[t] + (1-\alpha)S_{\text{out}}[t]. \tag{53}$$

Each time a spike is triggered from the neuron, $S_{\text{out}}[t] = 1$, the threshold jumps by $(1 - \alpha)$. This is added to the threshold through an intermediary state variable, $b[t]$. This jump decays at a rate of α at each subsequent step, causing the threshold to tend back to θ_0 in the absence of further spikes. The above form is loosely based on [169] though the decay rate α and the threshold jump factor $(1 - \alpha)$ can be decoupled from each other. α can be treated as either a hyperparameter or a learnable parameter.

Acknowledgment

The authors would like to thank Sumit Bam Shrestha, Garrick Orchard, and Albert Albesa-González for their insightful discussions over the course of putting together this article and iDataMap Corporation for their support. ■

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[3] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 91–99.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[6] J. K. Eshraghian, "Human ownership of artificial creativity," *Nature Mach. Intell.*, vol. 2, no. 3, pp. 157–160, Mar. 2020.

[7] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1764–1772.

[8] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 4960–4964.

[9] Y. Zhang, W. Chan, and N. Jaitly, "Very deep convolutional networks for end-to-end speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 4845–4849.

[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[11] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 160–167.

[12] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, *arXiv:1508.04025*.

[13] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[14] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc.*, Sep. 2010, pp. 1045–1048.

[15] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[16] O. Vinyals et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019.

[17] S. M. McKinney et al., "International evaluation of an AI system for breast cancer screening," *Nature*, vol. 577, no. 7788, pp. 89–94, 2020.

[18] A. Y. Hannun et al., "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature Med.*, vol. 25, no. 1, pp. 65–69, Jan. 2019.

[19] Y. Yang, N. D. Truong, J. K. Eshraghian, C. Maher, A. Nikpour, and O. Kavehei, "A multimodal AI system for out-of-distribution generalization of seizure identification," *IEEE J. Biomed. Health Inform.*, vol. 26, no. 7, pp. 3529–3538, Mar. 2022.

[20] M. R. Azghadi et al., "Hardware implementation of deep network accelerators towards healthcare and biomedical applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 6, pp. 1138–1159, Dec. 2020.

[21] Y. Yang, N. D. Truong, J. K. Eshraghian, A. Nikpour, and O. Kavehei, "Weak self-supervised learning for seizure forecasting: A feasibility study," *Roy. Soc. Open Sci.*, vol. 9, no. 8, p. 220374, Aug. 2022.

[22] *Kaggle*. Accessed: Aug. 29, 2023. [Online]. Available: https://www.kaggle.com

[23] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," 2020, *arXiv:2007.05558*.

[24] D. Amodei and D. Hernandez. (2019). *AI and Compute*. [Online]. Available: https://openai.com/blog/ai-and-compute/

[25] T. B. Brown et al., "Language models are few-shot learners," 2020, *arXiv:2005.14165*.

[26] P. Dhar, "The carbon impact of artificial intelligence," *Nature Mach. Intell.*, vol. 2, no. 8, pp. 423–425, Aug. 2020.

[27] L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," 2020, *arXiv:2007.03051*.

[28] W. B. Levy and V. G. Calvert, "Computation in the human cerebral cortex uses less than 0.2 Watts yet this great expense is optimal when considering communication costs," *bioRxiv*, Apr. 2020.

[29] G. Păun, G. Rozenberg, and A. Salomaa, *DNA Computing: New Computing Paradigms*. Berlin, Germany: Springer, 2005.

[30] L. Qian, E. Winfree, and J. Bruck, "Neural network computation with DNA strand displacement cascades," *Nature*, vol. 475, no. 7356, pp. 368–372, Jul. 2011.

[31] P. Chi et al., "Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, 2016.

[32] M. R. Azghadi et al., "Complementary metal-oxide semiconductor and memristive hardware for neuromorphic computing," *Adv. Intell. Syst.*, vol. 2, no. 5, 2020, Art. no. 1900189.

[33] G.-Q. Bi and M.-M. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *J. Neurosci.*, vol. 18, no. 24, pp. 10464–10472, Dec. 1998.

[34] T. Hamilton, "The best of both worlds," *Nature Mach. Intell.*, vol. 3, no. 3, pp. 194–195, 2021.

[35] Y. Liu, J. L. Pereira, and T. G. Constandinou, "Clockless continuous-time neural spike sorting: Method, implementation and evaluation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 538–541.

[36] G. Haessig, D. G. Lesta, G. Lenz, R. Benosman, and P. Dudek, "A mixed-signal spatio-temporal signal classifier for on-sensor spike sorting," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[37] F. C. Bauer, D. R. Muir, and G. Indiveri, "Real-time ultra-low power ECG anomaly detection using an event-driven neuromorphic processor," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 6, pp. 1575–1582, Dec. 2019.

[38] Z. Yan, J. Zhou, and W.-F. Wong, "Energy efficient ECG classification with spiking neural network," *Biomed. Signal Process. Control*, vol. 63, Jan. 2021, Art. no. 102170.

[39] Y. Yang, J. K. Eshraghian, N. D. Truong, A. Nikpour, and O. Kavehei, "Neuromorphic deep spiking neural networks for seizure detection," *Neuromorphic Comput. Eng.*, vol. 3, no. 1, Mar. 2023, Art. no. 014010.

[40] Y. He et al., "An implantable neuromorphic sensing system featuring near-sensor computation and send-on-delta transmission for wireless neural sensing of peripheral nerves," *IEEE J. Solid-State Circuits*, vol. 57, no. 10, pp. 3058–3070, Oct. 2022.

[41] F. Corradi and G. Indiveri, "A neuromorphic event-based neural recording system for smart brain-machine-interfaces," *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 5, pp. 699–709, Oct. 2015.

[42] F. Boi et al., "A bidirectional brain-machine interface featuring a neuromorphic hardware decoder," *Frontiers Neurosci.*, vol. 10, p. 563, Dec. 2016.

[43] L. F. H. Contreras et al., "Neuromorphic neuromodulation: Towards the next generation of on-device AI-revolution in electroceuticals," 2023, *arXiv:2307.12471*.

[44] Y. Sandamirskaya, M. Kaboli, J. Conradt, and T. Celikel, "Neuromorphic computing hardware and neural architectures for robotics," *Sci. Robot.*, vol. 7, no. 67, Jun. 2022, Art. no. eabl8419.

[45] C. Bartolozzi, G. Indiveri, and E. Donati, "Embodied neuromorphic intelligence," *Nature Commun.*, vol. 13, no. 1, pp. 1–14, Feb. 2022.

[46] S. Hussaini, M. Milford, and T. Fischer, "Spiking neural networks for visual place recognition via weighted neuronal assignments," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 4094–4101, Apr. 2022.

[47] J. Dupeyroux, J. J. Hagenaars, F. Paredes-Vallés, and G. C. H. E. de Croon, "Neuromorphic control for optic-flow-based landing of MAVs using the loihi processor," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 96–102.

[48] J. Timcheck et al., "The Intel neuromorphic DNS challenge," *Neuromorphic Comput. Eng.*, vol. 3, no. 3, Sep. 2023, Art. no. 034005.

[49] G. Gallego et al., "Event-based vision: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 1, pp. 154–180, Jan. 2022.

[50] G. Cohen et al., "Event-based sensing for space situational awareness," *J. Astron. Sci.*, vol. 66, no. 2, pp. 125–141, Jun. 2019.

[51] S. Afshar, A. P. Nicholson, A. van Schaik, and G. Cohen, "Event-based object detection and tracking for space situational awareness," *IEEE Sensors J.*, vol. 20, no. 24, pp. 15117–15132, Dec. 2020.

[52] A. Henkes, J. K. Eshraghian, and H. Wessels, "Spiking neural networks for nonlinear regression," 2022, *arXiv:2210.03515*.

[53] D. Wei, J. Wang, X. Niu, and Z. Li, "Wind speed forecasting system based on gated recurrent units and convolutional spiking neural networks," *Appl. Energy*, vol. 292, Jun. 2021, Art. no. 116842.

[54] G. Cohen, "Gooaall!!!: Why we built a neuromorphic robot to play foosball," *IEEE Spectr.*, vol. 59, no. 3, pp. 44–50, Mar. 2022.

[55] H. S. Seung, "Learning in spiking neural networks by reinforcement of stochastic synaptic transmission," *Neuron*, vol. 40, no. 6, pp. 1063–1073, Dec. 2003.

[56] J. Luboeinski and C. Tetzlaff, "Memory consolidation and improvement by synaptic tagging and capture in recurrent neural networks," *Commun. Biol.*, vol. 4, no. 1, pp. 1–17, Mar. 2021.

[57] N. Perez-Nieves, V. C. H. Leung, P. L. Dragotti, and D. F. M. Goodman, "Neural heterogeneity promotes robust learning," *Nature Commun.*, vol. 12, no. 1, pp. 1–9, Oct. 2021.

[58] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems* (Computational Neuroscience Series). Cambridge, MA, USA: MIT Press, 2001.

[59] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, Jan. 1962.

[60] A. M. H. J. Aertsen and P. I. M. Johannesma, "The spectro-temporal receptive field," *Biol. Cybern.*, vol. 42, no. 2, pp. 133–143, 1981.

[61] A. Benucci, A. B. Saleem, and M. Carandini, "Adaptation maintains population homeostasis in primary visual cortex," *Nature Neurosci.*, vol. 16, no. 6, pp. 724–729, Jun. 2013.

[62] B. Wark, B. N. Lundstrom, and A. Fairhall, "Sensory adaptation," *Current Opinion Neurobiol.*, vol. 17, no. 4, pp. 423–429, 2007.

[63] J. K. Eshraghian et al., "Formulation and implementation of nonlinear integral equations to model neural dynamics within the vertebrate retina," *Int. J. Neural Syst.*, vol. 28, no. 7, Sep. 2018, Art. no. 1850004.

[64] P.-F. Ruedi et al., "A 128 × 128 pixel 120-db dynamic-range vision-sensor chip for image contrast and orientation extraction," *IEEE J. Solid-State Circuits*, vol. 38, no. 12, pp. 2325–2333, Dec. 2003.

[65] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 × 128 120 db 30 mw asynchronous vision sensor that responds to relative intensity change," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2006, pp. 2060–2069.

[66] J. K. Eshraghian et al., "Neuromorphic vision hybrid RRAM-CMOS architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, pp. 2816–2829, Dec. 2018.

[67] D. E. Robey, W. Thio, H. H. C. Iu, and J. K. Eshraghian, "Naturalizing neuromorphic vision event streams using generative adversarial networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[68] G. Gallego et al., "Event-based vision: A survey," 2019, *arXiv:1904.08405*.

[69] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240 × 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor," *IEEE J. Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014.

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

[71] F. Zenke and E. O. Neftci, "Brain-inspired learning on neuromorphic substrates," *Proc. IEEE*, vol. 109, no. 5, pp. 935–950, May 2021.

[72] W. Gerstner, "A framework for spiking neuron models: The spike response model," in *Handbook of Biological Physics*, vol. 4. Amsterdam, The Netherlands: Elsevier, 2001, pp. 469–516.

[73] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 1419–1428.

[74] M. Zhang et al., "Spike-timing-dependent back propagation in deep spiking neural networks," 2020, *arXiv:2003.11837*.

[75] L. Lapicque, "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization," *J. Physiol. Pathol. Gen.*, vol. 9, pp. 620–635, Jan. 1907.

[76] N. Brunel and M. C. W. van Rossum, "Lapicque's 1907 paper: From frogs to integrate-and-fire," *Biol. Cybern.*, vol. 97, nos. 5–6, pp. 337–339, Dec. 2007.

[77] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, Aug. 1952.

[78] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input," *Biol. Cybern.*, vol. 95, no. 1, pp. 1–19, Jul. 2006.

[79] T. P. Vogels and L. F. Abbott, "Signal propagation and logic gating in networks of integrate-and-fire neurons," *J. Neurosci.*, vol. 25, no. 46, pp. 10786–10795, Nov. 2005.

[80] 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.

[81] P.-S. V. Sun et al., "Intelligence processing units accelerate neuromorphic learning," 2022, *arXiv:2211.10725*.

[82] A. Voelker, I. Kajić, and C. Eliasmith, "Legendre memory units: Continuous-time representation in recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–10.

[83] R.-J. Zhu, Q. Zhao, G. Li, and J. K. Eshraghian, "SpikeGPT: Generative pre-trained language model with spiking neural networks," 2023, *arXiv:2302.13939*.

[84] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[85] G. B. M. Mello, S. Soares, and J. J. Paton, "A scalable population code for time in the striatum," *Current Biol.*, vol. 25, no. 9, pp. 1113–1122, May 2015.

[86] Y. Bi, A. Chadha, A. Abbas, E. Bourtsoulatze, and Y. Andreopoulos, "Graph-based object classification for neuromorphic vision sensing," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 491–501.

[87] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM," *Int. J. Robot. Res.*, vol. 36, no. 2, pp. 142–149, Feb. 2017.

[88] A. Amir et al., "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7243–7252.

[89] Y. Hu, H. Liu, M. Pfeiffer, and T. Delbruck, "DVS benchmark datasets for object tracking, action recognition, and object recognition," *Frontiers Neurosci.*, vol. 10, p. 405, Aug. 2016.

[90] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis, "The multivehicle stereo event camera dataset: An event camera dataset for 3D perception," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2032–2039, Jul. 2018.

[91] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers Neurosci.*, vol. 9, p. 437, Nov. 2015.

[92] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their history, how they were made, and other details," *Frontiers Neurosci.*, vol. 9, p. 481, Dec. 2015.

[93] M. Gehrig, W. Aarents, D. Gehrig, and D. Scaramuzza, "DSEC: A stereo event camera dataset for driving scenarios," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4947–4954, Jul. 2021.

[94] J. Anumula, D. Neil, T. Delbruck, and S.-C. Liu, "Feature representations for neuromorphic audio spike streams," *Frontiers Neurosci.*, vol. 12, p. 23, Feb. 2018.

[95] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 2744–2757, Jul. 2022.

[96] S. Hecht, S. Shlaer, and M. H. Pirenne, "Energy, quanta, and vision," *J. Gen. Physiol.*, vol. 25, no. 6, pp. 819–840, Jul. 1942.

[97] H. A. van der Velden, "The number of quanta necessary for the perception of light of the human eye," *Ophthalmologica*, vol. 111, no. 6, pp. 321–331, 1946.

[98] F. Rieke and D. A. Baylor, "Single-photon detection by rod cells of the retina," *Rev. Mod. Phys.*, vol. 70, no. 3, pp. 1027–1036, Jul. 1998.

[99] J. K. Eshraghian et al., "Nonlinear retinal response modeling for future neuromorphic instrumentation," *IEEE Instrum. Meas. Mag.*, vol. 23, no. 1, pp. 21–29, Feb. 2020.

[100] S. Baek, J. K. Eshraghian, W. Thio, Y. Sandamirskaya, H. H. C. Iu, and W. D. Lu, "A real-time retinomorphic simulator using a conductance-based discrete neuronal network," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 79–83.

[101] S. Dehaene, "The neural basis of the Weber–Fechner law: A logarithmic mental number line," *Trends Cogn. Sci.*, vol. 7, no. 4, pp. 145–147, Apr. 2003.

[102] C. Arrow, H. Wu, S. Baek, H. H. C. Iu, K. Nazarpour, and J. K. Eshraghian, "Prosthesis control using spike rate coding in the retina photoreceptor cells," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[103] J. Yik et al., "NeuroBench: Advancing neuromorphic computing through collaborative, fair and representative benchmarking," 2023, *arXiv:2304.04640*.

[104] G. Lenz, K. Chaney, S. B. Shrestha, O. Oubari, S. Picaud, and G. Zarrella. (Jul. 2021). *Tonic: Event-Based Datasets and Transformations*. Accessed: Aug. 29, 2023. [Online]. Available: https://tonic.readthedocs.io, doi: 10.5281/zenodo.5079802.

[105] S. Thorpe and J. Gautrais, "Rank order coding," in *Computational Neuroscience*. Boston, MA, USA: Springer, 1998, pp. 113–118.

[106] S. B. Furber, W. J. Bainbridge, J. M. Cumpstey, and S. Temple, "Sparse distributed memory using *N*-of-*M* codes," *Neural Netw.*, vol. 17, no. 10, pp. 1437–1451, Dec. 2004.

[107] S. K. Esser et al., "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 41, pp. 11441–11446, Oct. 2016.

[108] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers Neurosci.*, vol. 12, p. 331, May 2018.

[109] G. Bellec et al., "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Commun.*, vol. 11, no. 1, pp. 1–15, Jul. 2020.

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

[111] N. Perez-Nieves and D. F. M. Goodman, "Sparse spiking gradient descent," 2021, *arXiv:2105.08810*.

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

[113] F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks," *Neural Comput.*, vol. 33, no. 4, pp. 899–925, Mar. 2021.

[114] B. A. Olshausen and D. J. Field, *What is the Other 85 Percent of V1 Doing?* vol. 23, L. van Hemmen and T. Sejnowski, Eds. Oxford, U.K.: Oxford Univ. Press, 2006, pp. 182–211.

[115] Y. Bengio, "How auto-encoders could provide credit assignment in deep networks via target propagation," 2014, *arXiv:1407.7906*.

[116] S. Shinomoto and S. Koyama, "A solution to the controversy between rate and temporal coding," *Statist. Med.*, vol. 26, no. 21, pp. 4032–4038, 2007.

[117] M. R. Mehta, A. K. Lee, and M. A. Wilson, "Role of

experience and oscillations in transforming a rate code into a temporal code," *Nature*, vol. 417, no. 6890, pp. 741–746, Jun. 2002.

[118] R. Brette, "Philosophy of the spike: Rate-based vs. spike-based theories of the brain," *Frontiers Syst. Neurosci.*, vol. 9, p. 151, Nov. 2015.

[119] S. M. Bohte, J. N. Kok, and H. L. Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, nos. 1–4, pp. 17–37, Oct. 2002.

[120] S. R. Kheradpisheh and T. Masquelier, "Temporal backpropagation for spiking neural networks with one spike per neuron," *Int. J. Neural Syst.*, vol. 30, no. 6, Jun. 2020, Art. no. 2050027.

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

[122] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, no. 5306, pp. 1593–1599, Mar. 1997.

[123] Y. Huang and R. P. N. Rao, "Predictive coding," *Wiley Interdiscipl. Rev., Cogn. Sci.*, vol. 2, no. 5, pp. 580–593, Sep./Oct. 2011.

[124] A. H. Marblestone, G. Wayne, and K. P. Kording, "Toward an integration of deep learning and neuroscience," *Frontiers Comput. Neurosci.*, vol. 10, p. 94, Sep. 2016.

[125] K. Deb, "Multi-objective optimization," in *Search Methodologies*. Boston, MA, USA: Springer, 2014, pp. 403–449.

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

[127] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers Neurosci.*, vol. 11, p. 324, Jun. 2017.

[128] E. M. Callaway, "Feedforward, feedback and inhibitory connections in primate visual cortex," *Neural Netw.*, vol. 17, nos. 5–6, pp. 625–632, Jun. 2004.

[129] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Rev. Neurosci.*, vol. 21, no. 6, pp. 335–346, 2020.

[130] M. Laskin et al., "Parallel training of deep networks with local updates," 2020, *arXiv:2012.03837*.

[131] C. Lammie, J. K. Eshraghian, W. D. Lu, and M. R. Azghadi, "Memristive stochastic computing for deep learning parameter optimization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1650–1654, May 2021.

[132] S. Gaba, P. Knag, Z. Zhang, and W. Lu, "Memristive devices for stochastic computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 2592–2595.

[133] F. Cai et al., "Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks," *Nature Electron.*, vol. 3, no. 7, pp. 409–418, Jul. 2020.

[134] S. Schmidgall et al., "Brain-inspired learning in artificial neural networks: A review," 2023, *arXiv:2305.11252*.

[135] R. J. Williams, "Toward a theory of reinforcement-learning connectionist systems," College Comput. Sci., Northeastern Univ., Boston, MA, USA, Tech. Rep. NU-CCS-88-3, 1988.

[136] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.

[137] J. Werfel, X. Xie, and H. S. Seung, "Learning curves for stochastic gradient descent in linear feedforward networks," *Neural Comput.*, vol. 17, no. 12, pp. 2699–2718, Dec. 2005.

[138] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random feedback weights support learning in deep neural networks," 2014, *arXiv:1411.0247*.

[139] T. H. Moskovitz, A. Litwin-Kumar, and L. F. Abbott, "Feedback alignment in deep convolutional networks," 2018, *arXiv:1812.06488*.

[140] S. Bartunov, A. Santoro, B. A. Richards, L. Marris, G. E. Hinton, and T. P. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 9390–9400.

[141] W. Xiao, H. Chen, Q. Liao, and T. Poggio, "Biologically-plausible learning algorithms can scale to large datasets," 2018, *arXiv:1811.03567*.

[142] C. Frenkel, M. Lefebvre, and D. Bol, "Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks," *Frontiers Neurosci.*, vol. 15, Feb. 2021, Art. no. 629892.

[143] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 153–160.

[144] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers Neurosci.*, vol. 12, p. 608, Aug. 2018.

[145] G. Dellaferrera and G. Kreiman, "Error-driven input modulation: Solving the credit assignment problem without a backward pass," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 4937–4955.

[146] G. Hinton, "The forward-forward algorithm: Some preliminary investigations," 2022, *arXiv:2212.13345*.

[147] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York, NY, USA: Science Editions, 1949.

[148] L. Kushnir and S. Denève, "Learning temporal structure of the input with a network of integrate-and-fire neurons," 2019, *arXiv:1912.10262*.

[149] S. Denève, A. Alemi, and R. Bourdoukan, "The brain as an efficient and robust adaptive learner," *Neuron*, vol. 94, no. 5, pp. 969–977, Jun. 2017.

[150] F. Zenke. (2019). *SpyTorch*. [Online]. Available: https://github.com/fzenke/spytorch

[151] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[152] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[153] S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors," M.S. thesis, Dept. Comput. Sci., Univ. Helsinki, Helsinki, Finland, 1970, pp. 6–7.

[154] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System Modeling and Optimization*. Berlin, Germany: Springer, 1982, pp. 762–770.

[155] J. A. Pérez-Carrasco et al., "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing-application to feedforward ConvNets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, Nov. 2013.

[156] E. Hunsberger and C. Eliasmith, "Spiking deep networks with LIF neurons," 2015, *arXiv:1510.08829*.

[157] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Oct. 2016, pp. 1–8.

[158] Y. Hu, H. Tang, and G. Pan, "Spiking deep residual network," 2018, *arXiv:1805.01352*.

[159] 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.

[160] C. Stöckl and W. Maass, "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes," *Nature Mach. Intell.*, vol. 3, no. 3, pp. 230–238, Mar. 2021.

[161] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–14.

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

[163] O. Booij and H. T. Nguyen, "A gradient descent rule for spiking neurons emitting multiple spikes," *Inf. Process. Lett.*, vol. 95, no. 6, pp. 552–558, Sep. 2005.

[164] Y. Xu, X. Zeng, L. Han, and J. Yang, "A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks," *Neural Netw.*, vol. 43, pp. 99–113, Jul. 2013.

[165] S. M. Jin, D. Kim, D. H. Yoo, J. Eshraghian, and D. S. Jeong, "BPLC + NOSO: Backpropagation of errors based on latency code with neurons that only spike once at most," *Complex Intell. Syst.*, pp. 1–18, Feb. 2023, doi: 10.1007/s40747-023-00983-y.

[166] T. C. Wunderlich and C. Pehle, "Event-based backpropagation can compute exact gradients for spiking neural networks," *Sci. Rep.*, vol. 11, no. 1, pp. 1–17, Jun. 2021.

[167] I. M. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala, "Temporal coding in spiking neural networks with alpha synaptic function," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 8529–8533.

[168] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[169] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," 2018, *arXiv:1803.09574*.

[170] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," 2017, *arXiv:1706.04698*.

[171] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian, "Deep residual learning in spiking neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 21056–21069.

[172] 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.

[173] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 558–567.

[174] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[175] F. Ottati et al., "To spike or not to spike: A digital hardware perspective on deep learning acceleration," 2023, *arXiv:2306.15749*.

[176] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[177] Y. Kim and P. Panda, "Revisiting batch normalization for training low-latency deep spiking neural networks from scratch," *Frontiers Neurosci.*, vol. 15, p. 1638, Dec. 2021.

[178] C. Duan, J. Ding, S. Chen, Z. Yu, and T. Huang, "Temporal effective batch normalization in spiking neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 34377–34390.

[179] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[180] V. Godbole, G. E. Dahl, J. Gilmer, C. J. Shallue, and Z. Nado. (2023). *Deep Learning Tuning Playbook, Version 1.0*. [Online]. Available: http://github.com/google-research/tuning_playbook

[181] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8,

pp. 1735–1780, Nov. 1997.
[182] K. Cho et al., "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.
[183] A. J. Watt, "Homeostatic plasticity and STDP: Keeping a neuron's cool in a fluctuating world," *Frontiers Synaptic Neurosci.*, vol. 2, p. 5, Jun. 2010.
[184] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, "Rate, timing, and cooperativity jointly determine cortical synaptic plasticity," *Neuron*, vol. 32, no. 6, pp. 1149–1164, Dec. 2001.
[185] B. Schrauwen and J. Van Campenhout, "Extending SpikeProp," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 1, Jul. 2004, pp. 471–475.
[186] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "DL-ReSuMe: A delay learning-based remote supervised method for spiking neurons," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3137–3149, Dec. 2015.
[187] R. Zhu, J. K. Eshraghian, and Z. Kuncic, "Memristive reservoirs learn to learn," 2023, *arXiv:2306.12676*.
[188] A. Renart, N. Brunel, and X.-J. Wang, "Mean-field theory of irregularly spiking neuronal populations and working memory in recurrent cortical networks," in *Computational Neuroscience: A Comprehensive Approach*. 2004, pp. 431–490.
[189] D. J. Amit, *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1992.
[190] A. Renart, P. Song, and X.-J. Wang, "Robust spatial working memory through homeostatic synaptic scaling in heterogeneous cortical networks," *Neuron*, vol. 38, no. 3, pp. 473–485, May 2003.
[191] M. Rigotti, "Internal representation of task rules by recurrent dynamics: The importance of the diversity of neural responses," *Frontiers Comput. Neurosci.*, vol. 4, p. 24, Oct. 2010.
[192] J. Miller and M. Hardt, "Stable recurrent models," 2019, *arXiv:1805.10369*.
[193] 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.
[194] C. Tallec and Y. Ollivier, "Unbiased online recurrent optimization," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–14.
[195] A. Mujika, F. Meier, and A. Steger, "Approximating real-time recurrent learning with random Kronecker factors," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6594–6603.
[196] C. Roth, I. Kanitscheider, and I. Fiete, "Kernel RNN learning (keRNl)," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–11.
[197] J. M. Murray, "Local online learning in recurrent networks with random feedback," *eLife*, vol. 8, May 2019, Art. no. e43299.
[198] 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. 1, pp. 5320–5353, 2020.
[199] T. Bohnstingl, S. Woźniak, W. Maass, A. Pantazi, and E. Eleftheriou, "Online spatio-temporal learning in deep neural networks," 2020, *arXiv:2007.12723*.
[200] A. Mehonic and J. Eshraghian, "Brains and bytes: Trends in neuromorphic technology," *APL Mach. Learn.*, vol. 1, no. 2, Jun. 2023, Art. no. 020401.
[201] T. Bohnstingl, S. Woźniak, A. Pantazi, and E. Eleftheriou, "Online spatio-temporal learning in deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 16, 2022, doi: 10.1109/TNNLS.2022.3153985.
[202] F. M. Quintana, F. Perez-Peña, P. L. Galindo, E. O. Neftci, E. Chicca, and L. Khacef, "ETLP: Event-based three-factor local plasticity for online learning with neuromorphic hardware," 2023, *arXiv:2301.08281*.
[203] T. Ortner, L. Pes, J. Gentinetta, C. Frenkel, and A. Pantazi, "Online spatio-temporal learning with target projection," 2023, *arXiv:2304.05124*.
[204] A. Kag and V. Saligrama, "Training recurrent neural networks via forward propagation through time," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 5189–5200.
[205] B. Yin, F. Corradi, and S. M. Bohte, "Accurate online training of dynamical spiking neural networks through forward propagation through time," *Nature Mach. Intell.*, vol. 5, pp. 518–527, May 2023.
[206] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers Comput. Neurosci.*, vol. 9, p. 99, Aug. 2015.
[207] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," *Neural Comput.*, vol. 19, no. 11, pp. 2881–2912, 2007.
[208] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward categorization on AER motion events using cortex-like features in a spiking neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 9, pp. 1963–1978, Sep. 2015.
[209] M. Beyeler, N. D. Dutt, and J. L. Krichmar, "Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule," *Neural Netw.*, vol. 48, pp. 109–124, Dec. 2013.
[210] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Trans. Nanotechnol.*, vol. 12, no. 3, pp. 288–295, May 2013.
[211] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, "Towards biologically plausible deep learning," 2015, *arXiv:1502.04156*.
[212] G. Hinton et al., "Can the brain do back-propagation?" in *Proc. Stanford Univ. Colloq. Comput. Syst.* Stanford, CA, USA: Stanford Univ., Center for Professional Development, Apr. 2016.
[213] M. Payvand, M. E. Fouda, F. Kurdahi, A. M. Eltawil, and E. O. Neftci, "On-chip error-triggered learning of multi-layer memristive spiking neural networks," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 4, pp. 522–535, Dec. 2020.
[214] F. Crick, "The recent excitement about neural networks," *Nature*, vol. 337, no. 6203, pp. 129–132, Jan. 1989.
[215] R. M. Cichy, A. Khosla, D. Pantazis, A. Torralba, and A. Oliva, "Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence," *Sci. Rep.*, vol. 6, no. 1, pp. 1–13, Jun. 2016.
[216] R. Rajalingham, E. B. Issa, P. Bashivan, K. Kar, K. Schmidt, and J. J. DiCarlo, "Large-scale, high-resolution comparison of the core visual object recognition behavior of humans, monkeys, and state-of-the-art deep artificial neural networks," *J. Neurosci.*, vol. 38, no. 33, pp. 7255–7269, Aug. 2018.
[217] M. Schrimpf et al., "Brain-score: Which artificial neural network for object recognition is most brain-like?" *bioRxiv*, Sep. 2018.
[218] A. J. E. Kell, D. L. K. Yamins, E. N. Shook, S. V. Norman-Haignere, and J. H. McDermott, "A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy," *Neuron*, vol. 98, no. 3, pp. 630–644.e16, May 2018.
[219] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
[220] 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.
[221] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.
[222] A. Neckar et al., "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proc. IEEE*, vol. 107, no. 1, pp. 144–164, Jan. 2019.
[223] J. Pei et al., "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, Aug. 2019.
[224] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, Nov. 2019.
[225] V. Kornijcuk and D. S. Jeong, "Recent progress in real-time adaptable digital neuromorphic hardware," *Adv. Intell. Syst.*, vol. 1, no. 6, Oct. 2019, Art. no. 1900030.
[226] C. Frenkel, D. Bol, and G. Indiveri, "Bottom-up and top-down approaches for the design of neuromorphic processing systems: Tradeoffs and synergies between natural and artificial intelligence," *Proc. IEEE*, vol. 111, no. 6, pp. 623–652, Jun. 2023.
[227] F. Modaresi, M. Guthaus, and J. K. Eshraghian, "OpenSpike: An OpenRAM SNN accelerator," 2023, *arXiv:2302.01015*.
[228] E. Ceolini et al., "Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing," *Frontiers Neurosci.*, vol. 14, p. 637, Aug. 2020.
[229] M. Davies et al., "Advancing neuromorphic computing with Loihi: A survey of results and outlook," *Proc. IEEE*, vol. 109, no. 5, pp. 911–934, May 2021.
[230] M. Sharifshazileh, K. Burelo, J. Sarnthein, and G. Indiveri, "An electronic neuromorphic system for real-time detection of high frequency oscillations (HFO) in intracranial EEG," *Nature Commun.*, vol. 12, no. 1, pp. 1–14, May 2021.
[231] R. Krips and M. Furst, "Stochastic properties of coincidence-detector neural cells," *Neural Comput.*, vol. 21, no. 9, pp. 2524–2553, Sep. 2009.
[232] R. Brette, "Computing with neural synchrony," *PLoS Comput. Biol.*, vol. 8, no. 6, Jun. 2012, Art. no. e1002561.
[233] A. Paszke et al., "Automatic differentiation in PyTorch," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst., Workshop Autodiff Submission*, 2017, pp. 1–4.
[234] M. Sanhueza and J. Lisman, "The CaMKII/NMDAR complex as a molecular memory," *Mol. Brain*, vol. 6, no. 1, pp. 1–8, 2013.
[235] J. J. Jun et al., "Fully integrated silicon probes for high-density recording of neural activity," *Nature*, vol. 551, no. 7679, pp. 232–236, Nov. 2017.
[236] N. A. Steinmetz et al., "Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings," *Science*, vol. 372, no. 6539, Apr. 2021, Art. no. eabf4588.
[237] J. K. Eshraghian et al. (2021). *snnTorch*. [Online]. Available: https://github.com/jeshraghian/snntorch

ABOUT THE AUTHORS

Jason K. Eshraghian (Member, IEEE) received the B.E. degree in electrical and electronics engineering and the Bachelor of Laws and Ph.D. degrees from The University of Western Australia, Crawley, WA, Australia, in 2016 and 2019, respectively.

From 2019 to 2022, he was a Postdoctoral Researcher with the University of Michigan, Ann Arbor, MI, USA. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of California at Santa Cruz, Santa Cruz, CA, USA. His research interests include neuromorphic computing, resistive random access memory (RRAM) circuits, and spiking neural networks.

Dr. Eshraghian was awarded the 2023 IEEE Transactions on Circuits and Systems Darlington Best Paper Award, the 2019 IEEE Very Large Scale Integration Systems Best Paper Award, the Best Paper Award at the 2019 IEEE Artificial Intelligence Circuits and Systems Conference, and the Best Live Demonstration Award at the 2020 IEEE International Conference on Electronics, Circuits and Systems. He was a recipient of the Fulbright Fellowship (Australian-American Fulbright Commission), the Forrest Research Fellowship (Forrest Research Foundation), and the Endeavour Fellowship (Australian Government). He also serves as the Secretary of the IEEE Neural Systems and Applications Committee.

Max Ward received the B.S. and Ph.D. degrees from The University of Western Australia, Crawley, WA, Australia, in 2019.

He spent several years in the industry at Google, Sydney, NSW, Australia. He is currently a Lecturer with the Department of Computer Science and Software Engineering, University of Western Australia. He has recently returned to academia. He did postdoctoral research at The University of Adelaide, Adelaide, SA, Australia, and the Department of Molecular and Cellular Biology, Harvard University, Cambridge, MA, USA. His research interests include ribonucleic acid (RNA) bioinformatics, graph theory and algorithms, and discrete mathematics.

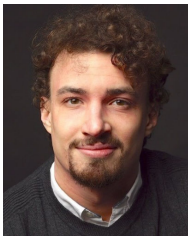
Dr. Ward was selected as a finalist for the ACS 1962 Medal for the Most Outstanding Doctoral Research in Western Australia.

Emre O. Neftci (Member, IEEE) received the M.Sc. degree in physics from the École polytechnique fédérale de Lausanne, Lausanne, Switzerland, and the Ph.D. degree from the Institute of Neuroinformatics, University of Zürich, Zürich, Switzerland, and ETH Zürich, Zürich, in 2010.

He is currently a Full Professor with RWTH Aachen University, Aachen, Germany, an Institute Leader with Forschungszentrum Jülich, Jülich, Germany, and an Associate Professor with the University of California at Irvine, Irvine, CA, USA. His current research explores the bridges between neuroscience and machine learning, with a focus on the theoretical and computational modeling of learning algorithms that are best suited to neuromorphic hardware and non-von Neumann computing architectures.

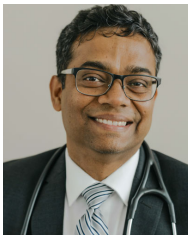
Xinxin Wang (Student Member, IEEE) received the B.S. degree from the Department of Microelectronics, Peking University, Beijing, China, in 2018. She is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA.

Her current research interest is hardware neural network accelerator design based on resistive random access memory (RRAM) crossbar arrays.



Gregor Lenz received the B.Sc. and M.Sc. degrees in biomedical engineering from the University of Applied Sciences Technikum Wien, Vienna, Austria, in 2012 and 2014, respectively, and the Ph.D. degree in neuromorphic engineering from the Institut de la Vision, Paris, France, in 2021.

After a one-year stint at Imperial College London, London, U.K., as a Research Assistant at the Biomedical Image Analysis Group, he moved to the Institut de la Vision for his Ph.D. degree in neuromorphic engineering. He is currently a Neuromorphic Machine Learning Engineer with SynSense, Zürich, Switzerland. He has published articles on event-based sensing, spiking neural networks, and the application thereof on neuromorphic hardware.



Girish Dwivedi received the Ph.D. degree in noninvasive cardiac imaging from The University of Manchester, Manchester, U.K., in 2010.

He is currently the Wesfarmers Chair in cardiology with the Harry Perkins Institute of Medical Research, The University of Western Australia, Crawley, WA, Australia, and a Consultant Cardiologist with the Fiona Stanley Hospital, Murdoch, WA, Australia. Prior to taking up this position in 2017, he was appointed as a Clinician Scientist and a Consultant Cardiologist at the University of Ottawa Heart Institute (UOHI), Ottawa, ON, Canada.

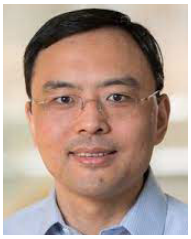
Dr. Dwivedi received accreditation in cardiac computerized tomography (CT) from the Society of Cardiovascular Computed Tomography; echocardiography from the American Society of Echocardiography Comprehensive Examination Certification and the European Society of Cardiology; the American Board Certifications in Cardiac Magnetic Resonance from the Society for Cardiovascular Magnetic Resonance, USA; and the nuclear cardiology including positron emission tomography (PET) from the Certification Board of Nuclear Cardiology, USA.

Mohammed Bennamoun (Senior Member, IEEE) is currently a Professor with the Department of Computer Science and Software Engineering, The University of Western Australia (UWA), Crawley, WA, Australia. He is also a researcher in computer vision, machine/deep learning, robotics, and signal/speech processing. He has published four books (available on Amazon), one edited book, one Encyclopedia article, 14 book chapters, more than 170 journal articles, more than 280 conference publications, and 16 invited & keynote publications.

Dr. Bennamoun won the Best Supervisor of the Year Award at the Queensland University of Technology (QUT) in 1998. He received the Award for Research Supervision at UWA in 2008 and 2016, and the Vice-Chancellor Award for Mentorship in 2016.

Doo Seok Jeong (Member, IEEE) received the B.E. and M.E. degrees in materials science from Seoul National University, Seoul, South Korea, in 2002 and 2005, respectively, and the Ph.D. degree in materials science from RWTH Aachen University, Aachen, Germany, in 2008.

He was with the Korea Institute of Science and Technology, Seoul, from 2008 to 2018. He is currently an Associate Professor with Hanyang University, Seoul. His research interests include spiking neural networks for sequence learning and future prediction. Learning algorithms, spiking neural network design, and digital neuromorphic processor design are his current research focus.



Wei D. Lu (Fellow, IEEE) received the B.S. degree in physics from Tsinghua University, Beijing, China, in 1996, and the Ph.D. degree in physics from Rice University, Houston, TX, USA, in 2003.

He was a Postdoctoral Research Fellow with Harvard University, Cambridge, MA, USA, from 2003 to 2005. He joined the faculty of the University of Michigan, Ann Arbor, MI, USA, in 2005. He is currently a Professor with the Electrical Engineering and Computer Science Department, University of Michigan. His research interests include resistive-random access memory (RRAM)/memristor devices, neuromorphic systems, aggressively scaled transistor devices, and low-dimensional systems.