TRAINING SPIKING NEURAL NETWORKS USING LESSONS FROM DEEP LEARNING

Jason K. Eshraghian
University of Michigan
University of Western Australia
jasonesh@umich.edu

Max Ward
University of Adelaide
max.ward-graham@adelaide.edu.au

Emre Neftci
Forschungszentrum Jülich
RWTH Aachen
e.neftci@fz-juelich.de

Xinxin Wang
University of Michigan
xinxinw@umich.edu

Gregor Lenz
SynSense
gregor.lenz@synsense.ai

Girish Dwivedi
University of Western Australia
girish.dwivedi@uwa.edu.au

Mohammed Bennamoun
University of Western Australia
mohammed.bennamoun@uwa.edu.au

Doo Seok Jeong
Hanyang University
dooseokj@hanyang.ac.kr

Wei D. Lu
University of Michigan
wluee@umich.edu

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 paper serves as a tutorial and perspective showing how to apply the lessons learnt from several decades of research in deep learning, gradient descent, backpropagation and neuroscience to biologically plausible spiking neural neural networks. 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 spiking neural networks; the subtle link between temporal backpropagation and spike timing dependent plasticity, and how deep learning might move towards biologically plausible online learning. Some ideas are well accepted and commonly used amongst the neuromorphic engineering community, while others are presented or justified for the first time here. A series of companion interactive tutorials complementary to this paper using our Python package, *snnTorch*, are also made available.[1]

1 Introduction

Deep learning has solved numerous problems in computer vision [1–6], speech recognition [7–9], and natural language processing [10–14]. Neural networks have been instrumental in outperforming world champions in a diverse range of games, from Go to Starcraft [15, 16]. They are now surpassing the diagnostic capability of clinical specialists in numerous medical tasks [17–20]. But for all the state-of-the-art models designed every day, a Kaggle [21] 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 $10\times$ per year from 2012 to 2019 [22, 23]. 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 [24–26]. Meanwhile, our brains operate within $\sim$12-20 W of power. This is in addition to churning through a multitude of sensory input, all the while ensuring our involuntary biological processes do not shut down [27]. If our brains dissipated as much heat as state-of-the-art deep learning models, then natural selection would have wiped

[1] Link to interactive tutorials: https://snntorch.readthedocs.io/en/latest/tutorials/index.html.

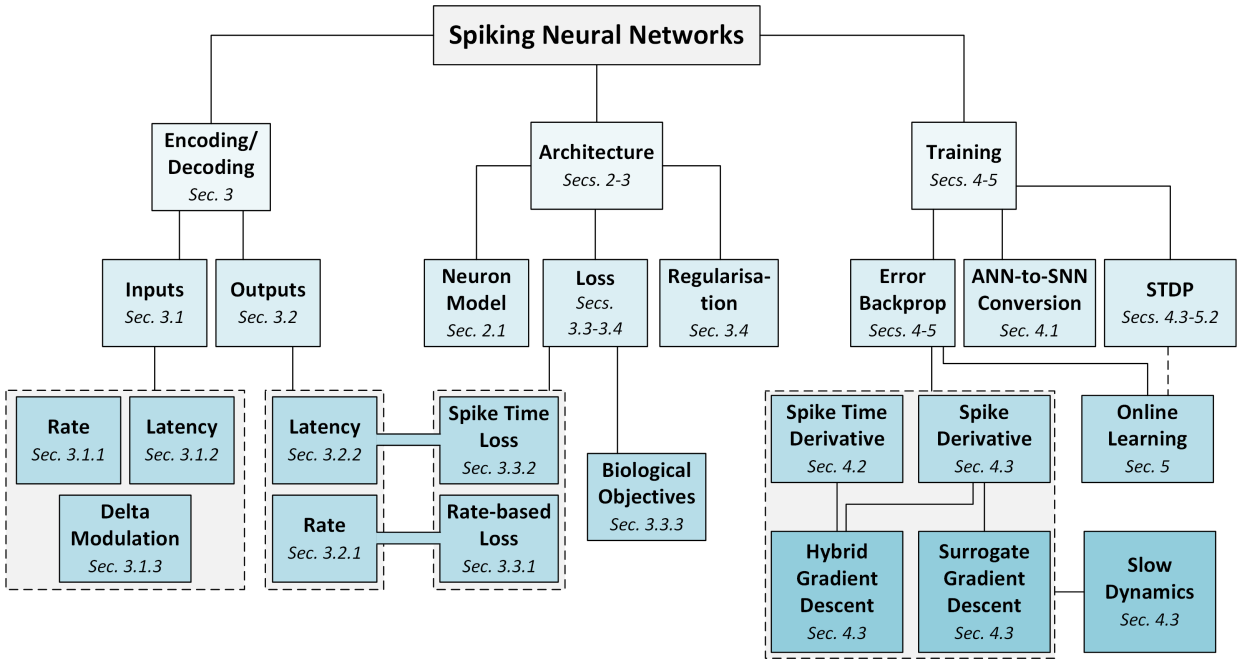

Figure 1: An overview of the paper structure.

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 of GPT-3.

So what lessons can be learnt from the brain to build more efficient neural networks? Should we replicate the genetic makeup of a neuron right down to the molecular level [28, 29]? Do we look at the way memory and processing coalesce within neurons and synapses [30, 31]? Or do we devote ourselves to extracting the learning algorithms that underpin the brain [32]?

The brain's neural circuitry is a physical manifestation of its neural algorithm; to understand one will likely lead to an understanding of the other. This paper will hone in one particular aspect of neural models: those that are compatible with modern deep learning. Figure 1 provides an illustrated overview of the structure of this paper, and we will start from the ground up. First, we will rationalise the commonly accepted advantages of using spikes, and derive a spiking neuron model from basic principles. These spikes will be assigned meaning in Section 3 by exploring various spike encoding strategies, how they impact the learning process, and how objective and regularisation functions can be used to sway the spiking patterns of an SNN. In Section 4, the challenges of training SNNs using gradient-based optimisation will be explored, and several solutions will be derived. These include defining derivatives at spike times, using approximations of the gradient, and a hybrid method that sits somewhere between the two. In doing so, a subtle link between the backpropagation algorithm and the spike timing dependent plasticity (STDP) learning rule will emerge, and used in the subsequent section to derive online variants of backprop that move towards biologically plausible learning mechanisms (Figure 1). The 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 [33].

2 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:

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 [34]. Many computational models of neurons simplify the representation of

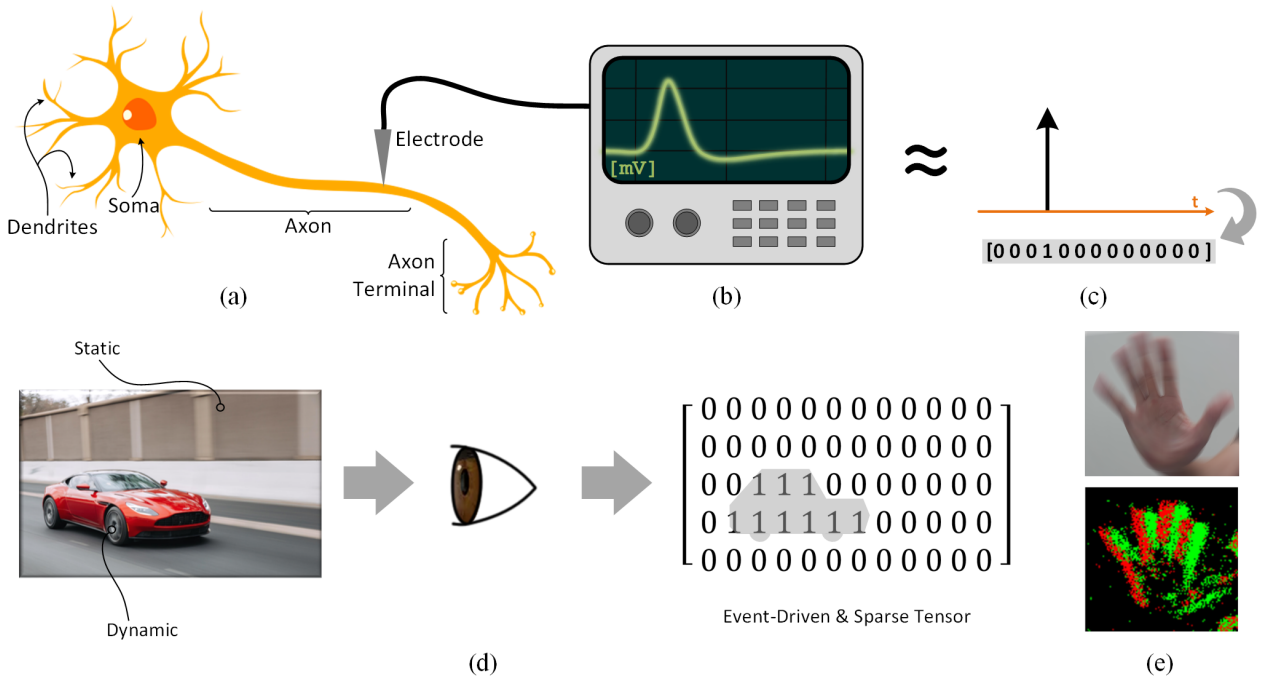

Figure 2: Neurons communicate via spikes. (a) Diagram of a neuron. (b) Measuring an action potential propagated along the axon of a neuron. Fluctuating subthreshold voltages are present in the soma, but become severely attenuated over distances beyond 1 mm [34]. Only the action potential is detectable along the axon. (c) The 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 Dynamic Vision Sensor.

a spike to a discrete, single-bit, all-or-nothing event (Figure 2(a-c)). Communicating high-precision activations between layers, routing them around and between chips is an expensive undertaking. Multiplying a high-precision activation with a high-precision weight requires conversion into integers, 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 to binarised neural networks. What actually matters is the *timing* of the spike. Time is not a binarised 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 non-zero elements. Take the following list as an example:

$$[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 7,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 5]$$

Since most of the entries are zero, we could save time by writing out only the non-zero elements as would occur in run-length encoding (indexing from zero):

"7 at position 10; 5 at position 20"

For example, Figure 2(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

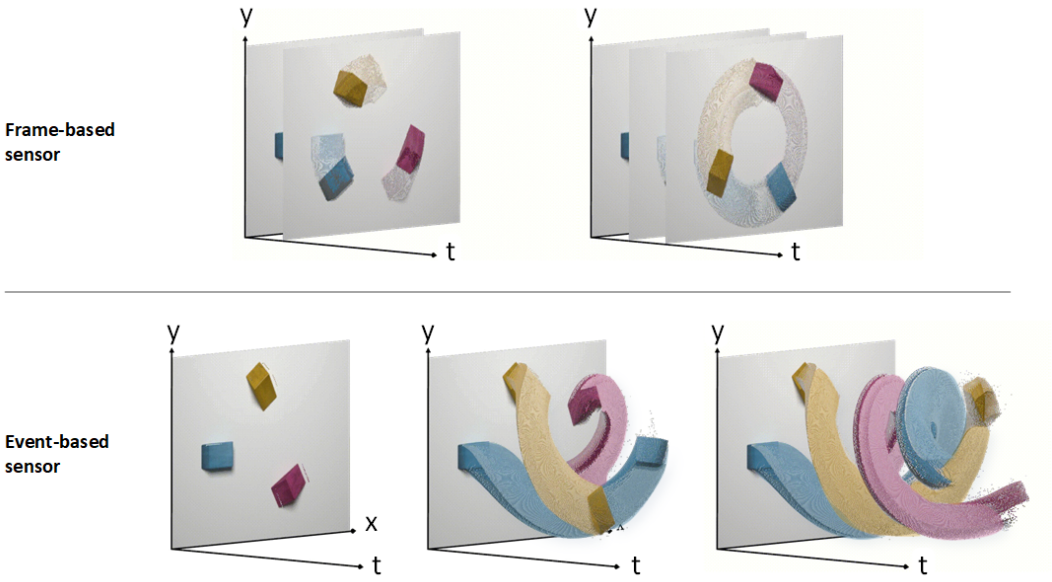

Figure 3: Functional difference between a conventional frame-based camera (above) and an event-based camera/silicon retina (below). The former records the scene as a sequence of images at a fixed frame rate. It operates independently of activity in the scene and can result in motion blur due to the global shutter. The silicon retina's output is directly driven by visual activity in the scene, as every pixel reacts to a change in illuminance.

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 [35]. Analogous mechanisms in early auditory processing include spectro-temporal receptive fields, which cause neurons to respond more favourably to changing frequencies in sound over static frequencies [36]. These processes occur on short timescales (milliseconds), while perceptual adaptation has also been observed on longer timescales (seconds) [37–39], 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 (Figure 2(d-e)) [40–44]. 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 when 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 [45]. The difference between a conventional frame-based camera and an event-based camera is illustrated in Figure 3.

2.1 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 (Figure 4). Much like the artificial neuron model [46], spiking neurons operate on a weighted sum of inputs. Rather than passing the result through a sigmoid or 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. But 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 (Figure 4(c)). This indicates the presence of temporal dynamics that 'sustain' the membrane potential over time.

These dynamics were quantified back in 1907 [47]. Louis Lapicque 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} [48]. 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 neuron (Figure 4(b)). This holds up a century

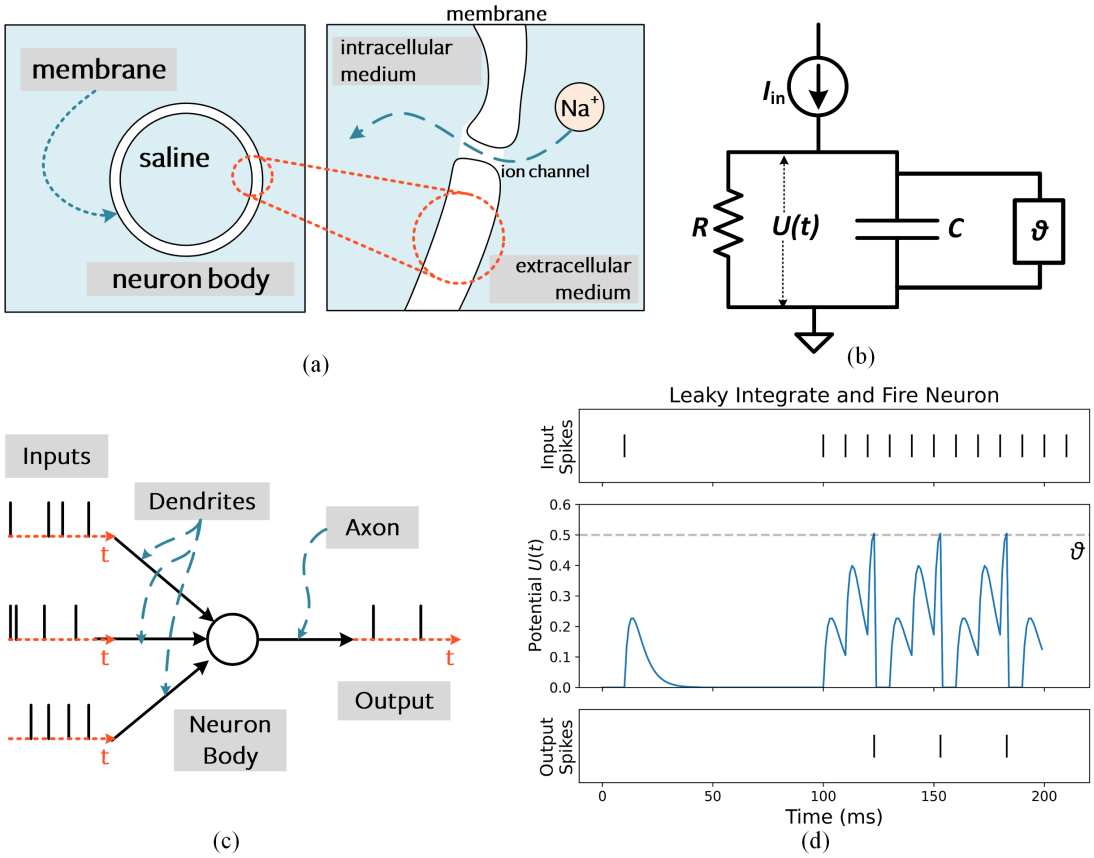

Figure 4: Leaky Integrate-and-Fire Neuron Model. (a) An insulating bilipid membrane separates the intracellular and extracellular medium. Gated ion channels allow charge carriers, such as Na^+, to diffuse through the membrane. (b) The capacitive membrane and resistive ion channels form an RC circuit. 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) A simulation depicting the membrane potential $U(t)$ reaching the threshold, arbitrarily set to $\theta = 0.5V$, which generates output spikes.

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 (Figure 4(a–b)) [49]. The dynamics of the passive membrane modelled 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 milliseconds. The general solution of (1) is:

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

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 to find an approximate solution to Equation (1):

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

where time is explicitly discretised, $\beta = e^{-1/\tau}$ is the decay rate (or 'inverse time constant') of $U[t]$, and the full derivation is provided in Appendix A.1.

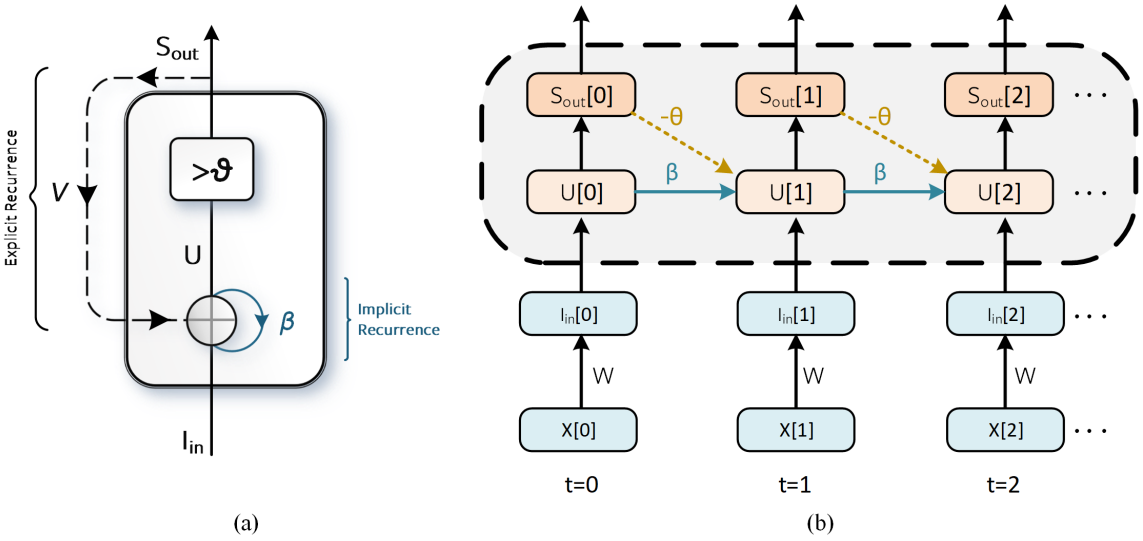

Figure 5: Computational steps in solving the leaky integrate-and-fire neuron model. (a) A recurrent representation of a spiking neuron. Hidden state decay is referred to as 'implicit recurrence', and external feedback from the spike is 'explicit recurrence', where V is the recurrent weight (omitted from Equation (4)) [56]. (b) An unrolled computational graph of the neuron where time flows from left to right. $-\theta$ represents the reset term from Equation (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 [53–55].

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 Equation (3), $(1-\beta)$, is subsumed into a learnable weight W, and the simplification $I_{\text{in}}[t] = WX[t]$ is made 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 vectorise $X[t]$ 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 Equation (4) with all simplifying assumptions is provided in Appendix A.1. A spike is generated if the membrane potential exceeds the threshold:

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

Variants of the leaky integrate-and-fire model build upon this approach. Some of the more commonly used adaptations incorporate synaptic conductance variation, and assume the input current acts as an additional low-pass filtering step to the input spike train [50–52]. Other approaches convolve pre-defined kernels (such as the 'alpha function': see Appendix C.1) with input spikes [53–55]. Kernel-based neurons have shown competitive performance in supervised learning tasks, and offer flexibility in terms of how the membrane potential evolves over time. But they also demand more memory during both the forward and backward passes, as solving each time step depends on the time stamp of all spikes, whereas the approach in Equation (4) only requires information from the previous time step. A depiction of a leaky integrate-and-fire neuron with a finite rise time of membrane potential is shown in Figure 4(d).

This formulation of a spiking neuron in a discrete, recursive form is perfectly poised to take advantage of the developments in training recurrent neural networks (RNNs) and sequence-based models. This is illustrated using an 'implicit' recurrent connection for the decay of the membrane potential, and is distinguished from 'explicit' recurrence where the output spike S_{out} would be fed back to the input (Figure 5). While there are plenty more physiologically

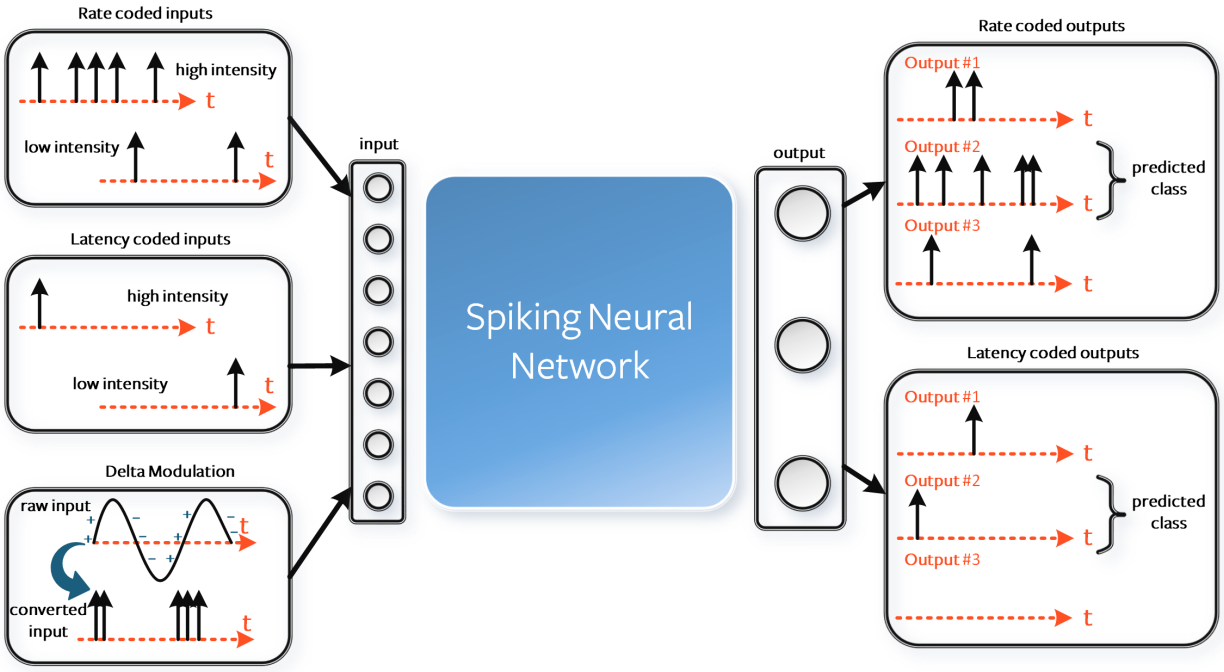

Figure 6: Input data to an SNN may be converted into a firing rate, firing time, or the data can be delta modulated. Alternatively, the input to the network can also be passed in without conversion which experimentally represents a direct or variable current source applied to the input layer of neurons. The network itself may be trained to enable the correct class to have the highest firing rate or to fire first, amongst many other encoding strategies.

accurate neuron models [49], the leaky integrate and fire 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 4, let us gain some insight to what spikes actually mean, and how they might represent information.

3 The Neural Code

Light is what we see when the retina converts photons into spikes. Odors are what we smell when the nose processes volatilised 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 (Figure 6):

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

3.1 Input encoding

Input data to a SNN does not necessarily have to be encoded into spikes. Static data 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. But this does not exploit the way SNNs extract meaning from temporal data. In general, three encoding mechanisms have been popularised 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 non-exhaustive list, and these codes are not necessarily independent of each other.

3.1.1 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 favourable orientation of light corresponds to a higher firing rate [35]. As a 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 dynamics of a neuron varies across time. There is no guarantee the firing rate at the start of the trial is anything near the rate at the end of the trial.

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 [57].

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 B.1.

3.1.2 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. When 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 [58–62]. 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 (Appendix B.2) [63, 64].

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 3.2.3.

3.1.3 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 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 pre-defined 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. 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.

3.2 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 on 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** applies the above coding schemes (typically a rate code) with **multiple neurons** per class

Table 1: Examples of neuromorphic datasets recorded with event-based cameras and cochlear models.

Vision datasets	
ASL-DVS [65]	100,800 samples of American sign language recorded with DAVIS.
DAVIS Dataset [66]	Includes spikes, frames and inertial measurement unit recordings of interior and outdoor scenes.
DVS Gestures [67]	11 different hand gestures recorded under 3 different lighting conditions.
DVS Benchmark [68]	DVS benchmark datasets for object tracking, action recognition, and object recognition.
MVSEC [69]	Spikes, frames and optical flow from stereo cameras for indoor and outdoor scenarios.
N-MNIST [70]	Spiking version of the classic MNIST dataset by converting digits from a screen using saccadic motion.
POKER DVS [71]	4 classes of playing cards flipped in rapid succession in front of a DVS.
DSEC [72]	A stereo event camera dataset for driving scenarios.
Audio datasets	
N-TIDIGITS [73]	Speech recordings from the TIDIGITS dataset converted to spikes with a silicon cochlear.
SHD [74]	Spiking version of the Heidelberg Digits dataset converted using a simulated cochlear model.
SSC [74]	Spiking version of the *Speech Commands* dataset converted using a simulated cochlear model.

3.2.1 Rate Coded Outputs

Consider a multi-class classification problem, where N_C is the number of classes. A non-spiking neural network 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 (Appendix B.3).

3.2.2 Latency Coded Outputs

There are numerous ways a neuron might encode data in the timing of a spike. As in the case with 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 less 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.

3.2.3 Rate vs. 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

- **Error tolerance:** if a neuron fails to fire, there are ideally many more spikes to reduce the burden of this error.
- **More spiking promotes more learning:** additional spikes provide a stronger gradient signal for learning via error backpropagation. As will be described in Section 4, the absence of spiking can impede learning convergence (more commonly referred to as the 'dead neuron problem').

Advantages of Latency Codes

- **Power consumption:** generating and communicating less spikes means 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.
- **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 [75]), then one can only process about 2-3 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 [57]. 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 optimises for efficiency. Olshausen and Field's 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) [75]. 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 [76]. 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 the true code must lie between rate and temporal codes [77], while others argue that the two may co-exist and only differ based on the timescale of observation: rates are observed for long timescales, latency for short timescales [78]. Some reject rate codes entirely [79]. 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.

3.3 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 [80]. Biological variables, such as dopamine release, have been meaningfully related to objective functions from reinforcement learning [81]. Predictive coding models often aim to minimise the information entropy of sensory encodings, such that the brain can actively predict incoming signals and inhibit what it already expects [82]. The multi-faceted nature of the brain's function likely calls for the existence of multiple objectives [83]. How the brain can be optimised using these objectives remains a mystery, though we might be able to gain insight from multi-objective optimisation [84].

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 complexity of the task at hand. All objective functions described below have successfully trained networks to competitive results on a variety of datasets, though come with their own trade-offs.

3.3.1 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 mean square error is applied to the spike count or the membrane potential of the output layer of neurons.

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 (26) in Appendix B.3), 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 minimisation is used to increase the spike count of the correct class, while suppressing the count of the incorrect classes [67, 85] (Appendix B.4).	**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 [54,86–88] (Appendix B.5).
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 [89–91]. A variant is to simply sum the membrane potential across all time steps to obtain the logits [91] (Appendix B.6).	**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 B.7).

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 absence of additional regularisation. 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 is expected to 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. The absence of change drives the derivative of the loss with respect to the learnable parameters to zero, such that no learning takes place. When relying on a small range of time steps, as may necessarily be the case when using memory-constrained resources, it might be preferable to apply the loss function directly to the membrane potential instead. 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.

3.3.2 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.

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 minimisation 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 [55] (Appendix B.8).	**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 generalised to multiple spikes as well [54, 92] (Appendix B.9). **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 [93] (Appendix B.10).
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 B.7).

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 a neuron does not fire [93].

In the absence of a sufficient number of time steps, the mean square membrane loss may become useful as it has a well-defined gradient with respect to the network weights. This comes at the expense of operating on a high precision hidden state, rather than on spikes. Alternatively, a population of output neurons for each class increases the number of pathways through which backpropagation may take place, and improve the chance that a weight update will generate a change in the global loss.

3.3.3 Biologically Motivated Learning Rules

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 [94]. 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 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 [95]. Numerous bi-directional, non-linear phenomena occur within individual neurons which may contribute towards helping errors find their way to the right synapse. For example, feedback connections are observed in most places where there are feedforward connections [96].

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 localised errors (Figure 7). Conjuring alternative methods to credit assignment that a real-time machine such as the brain can implement is not only useful for developing insight to biological learning [97], but also reduces the cost of data communication in hardware [98]. 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 [99–101]. A brief summary of several common approaches to ameliorating the spatial credit assignment problem are provided below:

- **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 [102–104]. 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 [105].

- **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* [106], which is specifically concerned with a weight update in one layer depending upon the weights of far-away layers. Forward and backwards-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 [106]. Random feedback has struggled with more complex tasks, though variants have been proposed that reduce the gap [107–110]. 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.

- **Local Losses:** It could be that the six layers of the cortex are each supplied with their own cost function, rather than a global signal that governs a unified goal for the brain [83]. Early visual regions may try to minimise 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 layer-wise training assigns a cost function to each layer independently [111]. Each layer is sequentially assigned a cost function so as to ensure a shallow network is only ever trained. Target propagation is similarly motivated, by assigning a reconstruction criterion to each layer [76]. 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 the error signal does not need to propagate too far [95, 112].

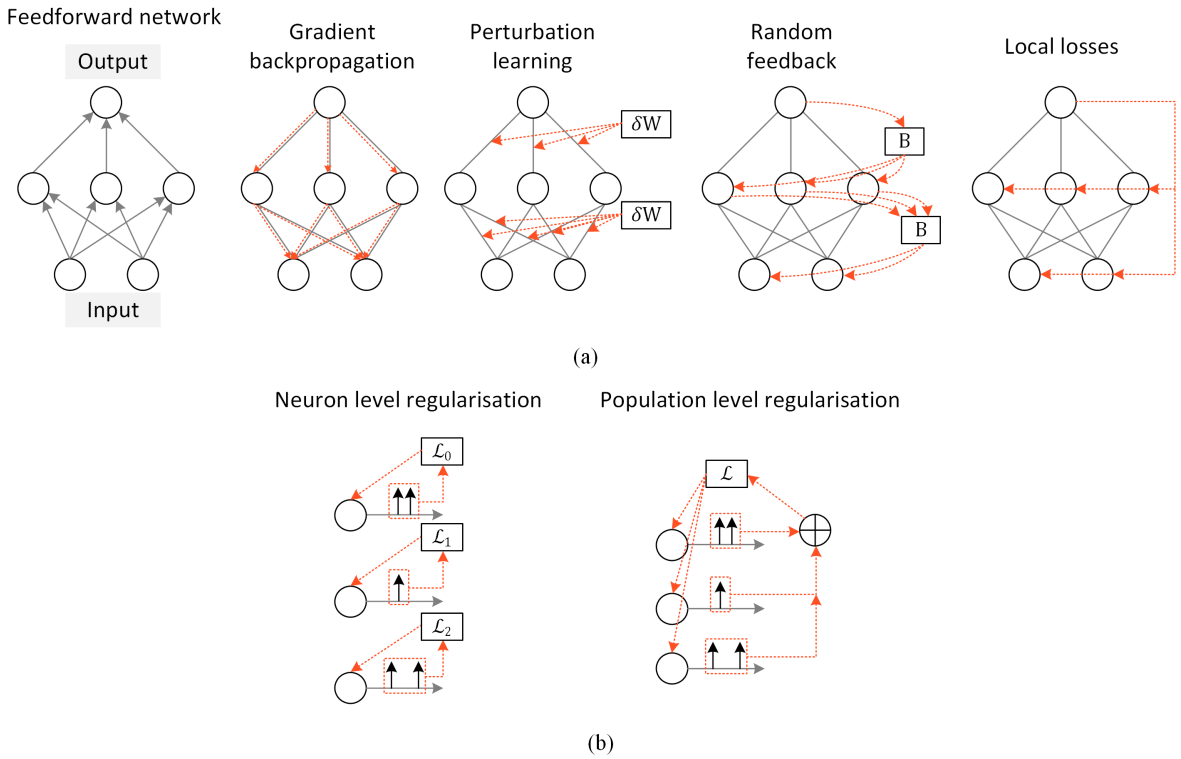

Figure 7: A variety of learning rules can be used to train a network. (a) Objective Functions. **Gradient backpropagation:** an unbiased gradient estimator of the loss is derived with respect to each weight. **Perturbation learning:** weights are randomly perturbed by δW, with the change accepted if 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 Regularisation. **Neuron level regularisation:** aims to set a baseline spike count per neuron. **Population level regularisation:** aims to set an upper limit on the total number of spikes emitted from all neurons.

These approaches to learning are illustrated in Figure 7(a). While they are described in the context of supervised learning, many theories of learning place emphasis on self-organisation and unsupervised approaches. Hebbian plasticity is a prominent example [113]. But 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 [114, 115]. 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 [83].

3.4 Activity Regularisation

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 inter-layer 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. Regularisation of the loss can be used to penalise excessive spiking (or alternatively, penalise insufficient spiking which is great for discouraging dead neurons). Conventionally, regularisation is used to constrain the solution space of loss minimisation, thus leading to a reduction in variance at the cost of increasing bias. Care must be taken, as too much activity regularisation can lead to excessively high bias. Activity regularisation can be applied to alter the behavior of individual neurons or populations of neurons, as depicted in Figure 7(b).

- **Population level regularisation:** this is useful when the metric to optimise 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-regularisation can be applied to the total number of spikes emitted at the output layer to penalise excessive firing, which encourages sparse activity at the output [116]. 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 exceed the threshold, only then does the regularisation penalty kick in [88, 91] (Appendix B.11).

- **Neuron level regularisation:** If neurons completely cease to fire, then learning may become significantly more difficult. Regularisation 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 regularisation penalty is applied (Appendix B.12).

Population level regularisation can also be used to reduce susceptibility to exploding gradients. This may be an issue where the decay rate of membrane potential β (Equation (4)) is a learnable parameter, and the output neurons are rate-coded. In such a case, if the firing rate lacks an upper bound then β will tend towards large values to promote more spikes. If $\beta > 1$, the gradient of the loss with respect to β will scale exponentially over time and tend to infinity. A penalty applied to a population of terms can regulate this undesirable behaviour. Recent experiments have shown that rate-coded networks (at the output) are robust to sparsity-promoting regularisation terms [88, 89, 91]. However, networks that rely on time-to-first-spike schemes have had less success, which is unsurprising given that temporal outputs are already sparse.

By encouraging each neuron to have a baseline spike count, this helps with the backpropagation of errors through pathways that would otherwise be inactive. Together, the upper and lower-limit regularisation terms can be used to find the sweet spot of firing activity at each layer. As explained in detail in [117], 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 non-differentiable 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.

4 Training Spiking Neural Networks

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

- **Shadow training:** A non-spiking ANN is trained and converted into an SNN by interpreting the activations as a firing rate or spike time
- **Backpropagation using spikes:** The SNN is natively trained using error backpropagation, typically through time as is done with sequential models
- **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 minimisation. 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 [118–120]. 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 non-differentiability 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 4.3.

To gain deeper insight behind the non-differentiability of spikes, recall the discretised solution of the membrane potential of the leaky integrate and fire neuron from Equation (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 a weight update ΔW is applied to the weight W (Equation (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 (Equation (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 signal when backpropagating through a spiking neuron (Figure 8(a)).

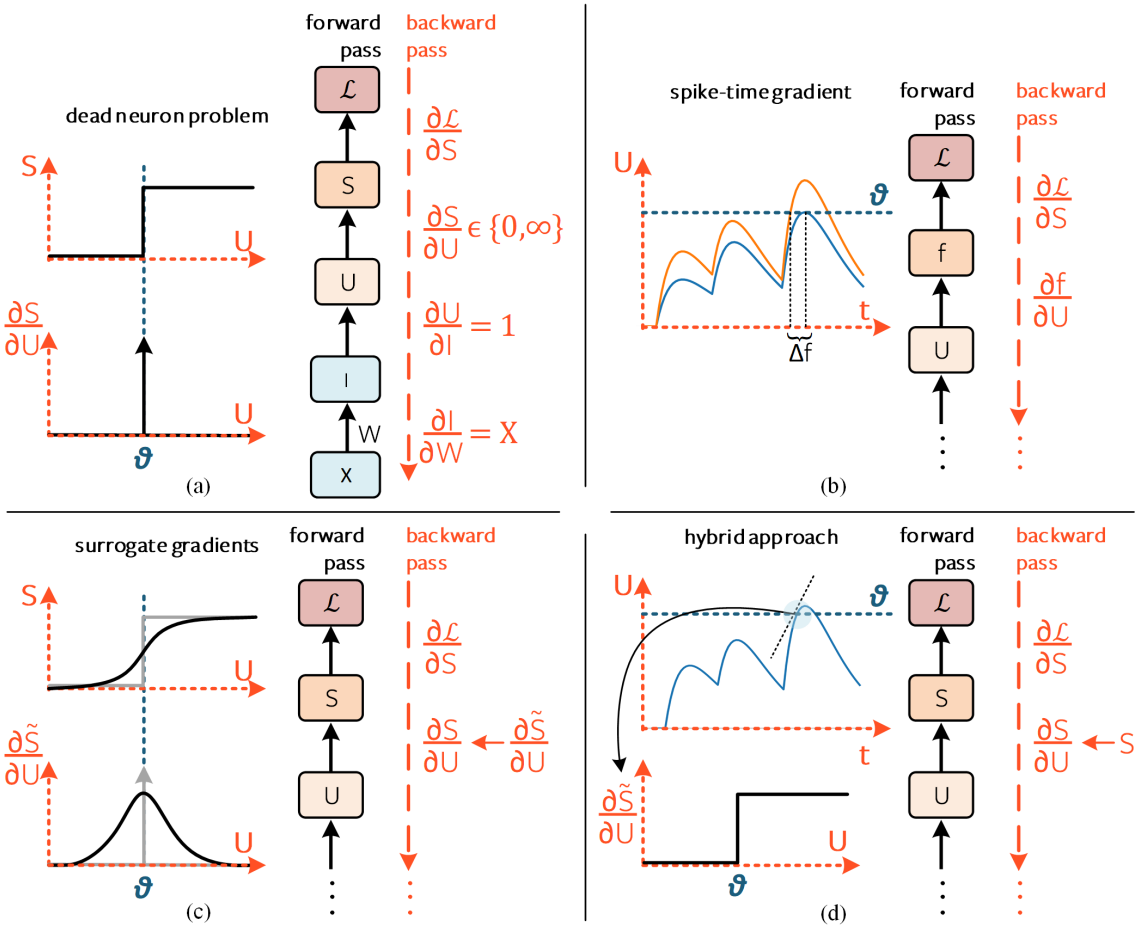

Figure 8: Addressing the dead neuron problem. Only one time step is shown, where temporal connections and subscripts from Figure 5 have been omitted for simplicity. (a) The dead neuron problem: the analytical solution of $\partial S/\partial U \in \{0, \infty\}$ results in a gradient that does not enable learning. (b) 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 [92]. (c) Surrogate gradients: the spike generation function is approximated to a continuous function during the backward pass [91]. The left arrow ($\leftarrow$) indicates function substitution. This is the most broadly adopted solution to the dead neuron problem. (d) Hybrid approach: as in (b), the gradient is '0' in the absence of a spike. At the onset of a spike, assume the membrane evolution is linearised and therefore, the gradient approximation is simply the spike itself, or a scaled version of the spike [121].

.

4.1 Shadow Training

The dead neuron problem can be completely circumvented by instead training on a shadow ANN and converting it into an SNN. The high precision activation function of each neuron is converted into either a spike rate [122–126] or a latency code [127]. 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 is not time-varying, then shadow training could be the optimal way to go.

In addition to the inefficient training process, there are several drawbacks. Firstly, 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 under-explored area [124]. Secondly, 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. But 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 [128]. 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 [129].

4.2 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 to training multi-layer SNNs using backpropagation [92]. 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 C.1.

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 non-differentiable 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 Figure 8(b). Extensions of SpikeProp have made it compatible with multiple spikes [130], which are highly performant on data-driven tasks some of which have surpassed human level performance on MNIST and N-MNIST [55, 131, 132].

Several drawbacks arise. Once neurons become inactive, their weights become frozen. In most instances, no closed-form solutions exist to solving for the gradient if there is no spiking [133]. SpikeProp tackles this by modifying parameter initialization (i.e., increasing weights until a spike is triggered). But 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 [117, 134]. 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) that 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. But it is the only approach that enables the calculation of an unbiased gradient without any approximations in multi-layer SNNs. Whether this precision is necessary is a matter of further exploration on a broader range of tasks.

4.3 Backpropagation Using Spikes

Instead of computing the gradient with respect to spike times, the more commonly adopted approach over the past several years is to apply the generalised backpropagation algorithm to the unrolled computational graph (Figure 5(b)) [54, 85, 123, 135, 136], i.e., backpropagation through time (BPTT). Working backwards 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. Figure 9(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 3.3.

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 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 (Section 3.3.1), 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 labelled the *immediate influence* in Figure 9(a). For $s < t$, we refer to the impact of $W[s]$ on $L[t]$ as

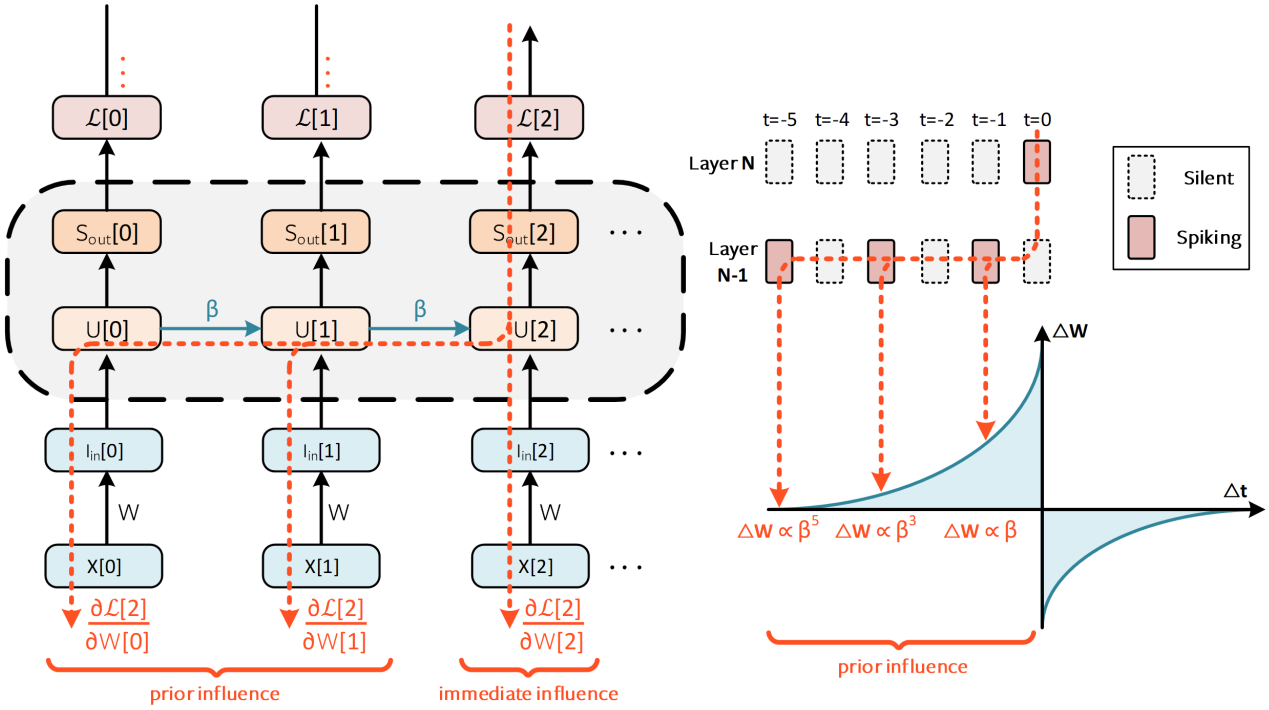

Figure 9: Backpropagation through time. (a) The present time application of W is referred to as the immediate influence, with historical application of W described as the prior influence. Reset dynamics and explicit recurrence have been omitted for brevity. The error pathways through $\mathcal{L}[0]$ and $\mathcal{L}[1]$ are also hidden but follow the same idea as that of $\mathcal{L}[2]$. (b) The hybrid approach defaults to a non-zero 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 [32].

the *prior influence*. The influence of all parameter applications on present and future losses are summed together to define the global gradient:

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

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 Equation (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 Figure 8(c) makes it clear that we run into the spike non-differentiability problem in the term $\partial S/\partial U$. 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. But 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 (Figure 8(c)).

Surrogate Gradients

A major advantage of surrogate gradients is they effectively overcome the dead neuron problem. Consider a case where the threshold is $\theta = 1$, and the membrane potential is $U = 0$. Conventionally, no spike would be elicited as $U < \theta \implies S = 0$, and $\partial S/\partial U = 0$. But a surrogate gradient, $\partial \tilde{S}/\partial U$, solves this. Let $\sigma(\cdot)$ be the sigmoid function centered about the threshold: $1/(1 + e^{-U+\theta})$, where $\sigma'(U = 0) \approx 0.20 \implies \partial \tilde{S}/\partial U = 0.20$, which enables a biased estimate of the gradient to flow backward across multiple layers. Now, even where a neuron doesn't fire, a non-zero approximation of the gradient can still enable learning.

For clarity, we provide a 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 the weights attached to that neuron have no 'credit' in the credit assignment problem. Therefore, the neuron cannot learn to fire later on and so is stuck *forever*, not contributing to learning. Dead neurons are caused by non-differentiable functions that force the gradient to zero, whereas vanishing gradients can arise in ANNs as well as 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 underutilised in surrogate gradient implementations [117].

A further note of caution: the reset mechanism in Equation (4) is a function of the spike, and is also non-differentiable. It is important to ensure the surrogate gradient is not cloned into the reset function as it has been empirically shown to degrade network performance [91]. Either the original analytical derivative can be used such that it (almost always) provides no error feedback to the gradient update, or more cautiously, the reset pathway can be detached from the computation graph.

Surrogate gradients have been used in most state-of-the-art experiments that natively train an SNN [54, 85, 123, 135, 136]. 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. Most common surrogate functions in use tend to take the form of the following functions centered about the threshold: the sigmoid function, the fast sigmoid function: $(U - \theta)/(1 + |U - \theta|)$, and a triangular function: $\max(1 - |U - \theta|, 0)$. While several studies have explored the impact of various surrogates on the learning process [91, 137], our understanding tends to be limited to what is known about biased gradient estimators. This draws a parallel to random feedback alignment, where weights are replaced with random matrices during the backward pass. Understanding how approximations in gradient descent impacts learning will very likely lead to a deeper understanding of why surrogate gradients are so effective, which surrogate might be most effective for a given task, and how they might be improved.

Many studies normalise the surrogate gradient to a peak value of '1', rather than the infinite peak of the Dirac-Delta function [85, 91, 116, 135, 137]. In the absence of formal justification, we have some loose intuition as to why this is done. Shrestha and Orchard [54] interpret a normalised surrogate gradient as the probability that a change in U will lead to a change in S. An alternative justification is that normalisation reduces the impact of vanishing and exploding gradients, although using sigmoid-like surrogates still opens up vulnerability to vanishing gradients, much like a standard ANN. We expect this to be heavily influenced by how modern deep learning frameworks (PyTorch, Tensorflow, etc.) implement parameter initialisation, which have been optimised for a given set of activation functions (e.g., Sigmoid and ReLU) [117, 134]. That is to say, our approach to designing SNNs may currently be moulded to fit with the current best practices in deep learning.

Hybrid Approaches

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. Alternatively, these two approaches can be merged. Let a neuron be separated into two operating modes: one where it does not spike, and one where it does spike. For simplicity, the threshold is normalised to $\theta = 1$. Consider the case where the neuron does not spike. From Equation (5), $U < 1 \implies S = 0 \times U \implies \partial \tilde{S}/\partial U = 0$.

Now consider when the neuron spikes, and the temporal precision of the network is high enough such that $U \approx \theta = 1$, and $S = U \times 1 = 1 \implies \partial\tilde{S}/\partial U = 1$. To summarise this result:

$$\frac{\partial \tilde{S}}{\partial U} \leftarrow S = \begin{cases} 1, & \text{if } U > \theta \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

That is to say, the estimate of the derivative used in the backward pass is substituted with the spike calculated in the forward pass (Figure 8(d)). It is assumed that S and U are sampled at a fixed point in time and so time-dependence is omitted here.

A computational benefit arises as the membrane potential U no longer needs to be used or stored to calculate the surrogate gradient. A memory benefit is obtained by 1) constraining the spike derivative term to a single-bit value, and 2) sparse firing patterns means a sparse gradient vector. Technically, this approach is still a surrogate of sorts as $\partial\tilde{S}/\partial U$ is necessarily an approximation for $U \geq \theta$.

Separating the derivative into two regimes of spiking and non-spiking means the gradient term of quiet neurons will always evaluate to '0'. Therefore, the only backpropagation pathways that give non-zero gradients will occur at spike times. This might seem somewhat redundant; why bother unrolling the computational graph when the derivative will only ever evaluate to '0' in the absence of spiking? Why not just take the derivative at spike times instead, as in the approach in Section 4.2? The answer is that there is no fixed requirement to actually treat the derivative of a subthreshold neuron as '0'. Instead, the lack of gradient can be used as a flag that enables random feedback to replace $\partial\tilde{S}/\partial U$ when $U < \theta$. When trained across many samples, noise centered about a mean of zero will integrate out and provide an unbiased estimator of the gradient for quiet neurons, while overcoming the dead neuron problem. Alternatively, the subthreshold flag can also scale the gradient by a small constant, which would then be the equivalent of applying the gradient of a threshold-shifted leaky ReLU activation during the backward pass. Furthermore, using a threshold-shifted ReLU gradient (see Figure 8(d)) also offers the benefit of avoiding vanishing gradients that sigmoidal approximations are prone to [117].

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 Equation (4). A gradient that backpropagates through n time steps is scaled by β^n. For a leaky neuron, $\beta < 1$, which causes the magnitude of a weight update to exponentially diminish with time between a pair of spikes. This proportionality is illustrated in Figure 9(b). This result shows how the strength of a synaptic update is exponentially proportional to the spike time difference between a pre- and post-synaptic neuron. In other words, weight updates from BPTT in this hybrid approach closely resembles weight updates from spike-timing dependent plasticity (STDP) learning curves (Appendix C.2) [32].

4.4 Long-Term Temporal Dependencies

Neural and synaptic time constants span timescales typically on the order of 1-100s 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 a huge burden on the learning process, where vanishing gradients drastically slow the convergence of the neural network. LSTMs [138] and, later, GRUs [139] 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:

- **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 favours long term potentiation at high frequencies regardless of spike timing [140, 141]. More recently, it has been found to benefit gradient-based learning in SNNs as well [135] (Appendix C.3).
- **Axonal delays:** The wide variety of axon lengths means 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 [54, 142, 143]. 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.

- **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.
- **Multistable Neural Activity:** Strong recurrent connections in biological neural networks can support multi-stable dynamics [144], which facilitates stable information storage over time. Such dynamics, often called attractor neural networks [145], are believed to underpin working memory in the brain [146, 147], and is 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 [148].

Several rudimentary slow timescale dynamics have been tested in gradient-based approaches to training SNNs with a good deal of success [54, 135], 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.

5 Online Learning

5.1 Temporal Locality

As with all computers, brains operate on a physical substrate which dictates the operations it can handle and where memory is located. While conventional computers operate on an abstraction layer, memory is delocalised and communicated on demand, thus paying a considerable price in latency and energy. Brains are believed to operate on local information, which means 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 which 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.* that 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. RTRL estimates the same gradients as BPTT, but relies on a set of different computations that make it temporally, but not spatially, local [149]. Since RTRL's memory requirement does not grow with time, then why is it not used in favour 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. But the push for continuously-learning systems that can run indefinitely long has cast a spotlight back on RTRL (and variants [150–154]), 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 Equation (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:

$$m[t] = \frac{\partial U[t]}{\partial W} = \sum_{s \leq t} \frac{\partial U[t]}{\partial W[s]} = \underbrace{\sum_{s \leq t-1} \frac{\partial U[t]}{\partial W[s]}}_{\text{prior}} + \underbrace{\frac{\partial U[t]}{\partial W[t]}}_{\text{immediate}} \tag{9}$$

The immediate and prior influence components are graphically illustrated in Figure 9(a). The immediate influence is also natural to calculate online, and evaluates to the unweighted input to the neuron $X[t]$. The prior influence relies on historical components of the network:

$$\sum_{s\leq t-1}\frac{\partial U[t]}{\partial W[s]}=\sum_{s\leq t-1}\underbrace{\frac{\partial U[t]}{\partial U[t-1]}}_{\text{temporal}}\frac{\partial U[t-1]}{\partial W[s]} \tag{10}$$

Based on Equation (4), in the absence of explicitly recurrent connections, the temporal term evaluates to β. From Equation (9), the second term is the influence of parameters on $U[t-1]$, which is by definition $m[t-1]$. Substituting these back into Equation (9) gives:

$$m[t]=\beta m[t-1]+x[t] \tag{11}$$

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:

$$\frac{\partial\mathcal{L}[t]}{\partial W}=\frac{\partial\mathcal{L}[t]}{\partial U[t]}\frac{\partial U[t]}{\partial W}\equiv\bar{c}[t]m[t] \tag{12}$$

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 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 in 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 [150–153]. Marschall *et al.* provides a systematic treatment of approximations to RTRL in RNNs in [154], and variations of online learning have been applied specifically to SNNs in [87, 88, 155].

Several practical considerations should be accounted for when implementing online learning algorithms. For an approach that closely resembles BPTT, the gradient accumulated at the end of the sequence can be used to update the network, which is referred to as a 'deferred' update. Alternatively, it is possible to update the network more regularly as a gradient is consistently available. While this latter option is a more accurate reflection of biological learning (i.e., training and inference are not decoupled processes), there are two issues that must be treated with care. Firstly, adaptive optimizers such as Adam naturally reduce the learning rate as parameters approach optimal values [156]. When applying frequent updates on a given batch of data, future batches will have less influence on weight updates. The result is a learning procedure that assigns a higher weighting to early data than to later data. If the sampled data does not satisfy the i.i.d assumption, which is the case when a system experiences data in an online fashion, learning may not perform well. Secondly, the reverse problem is catastrophic forgetting where new information causes the network to forget what it has previously learnt [157]. This is especially problematic in real-time systems because a "real-world batch size is equal to 1". Several approaches to overcome catastrophic forgetting in continual learning have been proposed, including using higher dimensional synapses [158], ensembles of networks [159], pseudo-replay [160], and penalizing weights that change excessively fast [161].

5.2 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 3.3.3 address the spatial credit assignment problem by either replacing the global error signal with local errors, or replacing analytical/numerical derivatives with random noise [106].

The more 'natural' approach to online learning is perceived to be via unsupervised learning with synaptic plasticity rules, such as STDP [32, 162] and variants of STDP (Appendix C.2) [163–166]. These approaches are directly inspired by experimental relationships between spike times and changes to synaptic conductance. Input data is fed to a network, and weights are updated based on the order and firing times of each pair of connected neurons (Figure 9(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 appear uncorrelated, their synaptic strength should be decreased. It follows the Hebbian mantra of *'neurons that fire together wire together'* [113].

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. But 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 4.3 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 (Figure 9(b)). Bengio *et al.* previously made the case that STDP resembles stochastic gradient descent, provided that STDP is supplemented with gradient feedback [167, 168]. 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 [169].

6 Outlook

Designing a neural network was once thought to be strictly an engineering problem whereas mapping the brain was a scientific curiosity [170]. 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 lead 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 have repeatedly shown similarities to experimental activity in the brain. This includes links between convolutional neural networks and measured activity from the visual cortex [171–174], and auditory processing regions [175]. 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 [176–182]. Improvements in energy consumption of up to 2–3 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, 183–185]. 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 the energy benefits afforded by spikes are uncontroversial. But a more challenging question to address is: are spikes actually good for computation? It could be that 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 towards energy efficiency and latency. But until we amass more evidence of a spike's purpose, we have some intuition as to where spikes shine in computation:

- **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.
- **Discrete Function Approximators:** Neural networks are universal function approximators, where discrete functions are considered to be modelled sufficiently well by continuous approximations. Spikes are capable of precisely defining discrete functions without approximation.

- **Multiplexing:** Spikes can encode different information in spike rate, times, or burst counts. Re-purposing the same spikes offers a sensible way to condense the amount of computation required by a system.
- **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 modelling asynchronous systems).
- **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. But 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 [186, 187].
- **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.
- **Modality normalisation:** A unified representation of sensory input (e.g., vision, auditory) as spikes is nature's way of normalising data. While this benefit is not exclusive to spikes (i.e., continuous data streams in non-spiking networks may also be normalised), early empirical evidence has shown instances where multi-modal SNNs outperform convolutional neural networks on equivalent tasks [20, 183, 188].
- **Mixed-mode differentiation:** While most modern deep learning frameworks rely on reverse-mode autodifferentiation [189], 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 autodifferentation are being used instead. Equation (11) in Section 5 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 pre-synaptic neuron activity in the form of calcium ions, and fades over time [87, 190]. SNNs offer a natural way to use mixed-mode differentiation by projecting temporal terms in the gradient calculation from Equation (10) into the future via forward-mode differentation, while taking advantage of the computational complexity of reverse-mode autodifferentation for spatial terms [56, 88].

A better understanding of the types of 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 non-exhaustive 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 conjuction 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. But the diversity of neurons, functions, and brain regions imply 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 [191, 192]. 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.

Acknowledgements

We would like to thank Sumit Bam Shrestha and Garrick Orchard for their insightful discussions over the course of putting together this paper, and iDataMap Corporation for their support. Jason K. Eshraghian is supported by the Forrest Research Foundation.

Additional Materials

A series of interactive tutorials complementary to this paper are available in the documentation for our Python package designed for gradient-based learning using spiking neural networks, *snnTorch* [193], at the following link: https://snntorch.readthedocs.io/en/latest/tutorials/index.html.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[3] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1440–1448, 2015.

[4] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28:91–99, 2015.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[6] Jason K Eshraghian. Human ownership of artificial creativity. *Nature Machine Intelligence*, 2(3):157–160, 2020.

[7] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772. PMLR, 2014.

[8] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964. IEEE, 2016.

[9] Yu Zhang, William Chan, and Navdeep Jaitly. Very deep convolutional networks for end-to-end speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4845–4849. IEEE, 2017.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[11] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167, 2008.

[12] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[14] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *11th Annual Conference of the International Speech Communication Association*, 2010.

[15] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[16] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[17] Scott Mayer McKinney, Marcin Sieniek, Varun Godbole, Jonathan Godwin, Natasha Antropova, Hutan Ashrafian, Trevor Back, Mary Chesus, Greg S Corrado, Ara Darzi, et al. International evaluation of an AI system for breast cancer screening. *Nature*, 577(7788):89–94, 2020.

[18] Awni Y Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H Tison, Codie Bourn, Mintu P Turakhia, and Andrew Y Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature medicine*, 25(1):65–69, 2019.

[19] Yikai Yang, Nhan Duy Truong, Jason K Eshraghian, Christina Maher, Armin Nikpour, and Omid Kavehei. A multimodal AI system for out-of-distribution generalization of seizure detection. *bioRxiv*, 2021.

[20] Mostafa Rahimi Azghadi, Corey Lammie, Jason K Eshraghian, Melika Payvand, Elisa Donati, Bernabe Linares-Barranco, and Giacomo Indiveri. Hardware implementation of deep network accelerators towards healthcare and biomedical applications. *IEEE Transactions on Biomedical Circuits and Systems*, 14(6):1138–1159, 2020.

[21] Kaggle. https://www.kaggle.com.

[22] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

[23] Dario Amodei and Danny Hernandez. AI and compute. Online: https://openai.com/blog/ai-and-compute/. 2019.

[24] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[25] Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2:423–5, 2020.

[26] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

[27] William B Levy and Victoria 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*, 2020.

[28] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *DNA computing: new computing paradigms*. Springer Science & Business Media, 2005.

[29] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368–372, 2011.

[30] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. *ACM SIGARCH Computer Architecture News*, 44(3):27–39, 2016.

[31] Mostafa Rahimi Azghadi, Ying-Chen Chen, Jason K Eshraghian, Jia Chen, Chih-Yang Lin, Amirali Amirsoleimani, Adnan Mehonic, Anthony J Kenyon, Burt Fowler, Jack C Lee, et al. Complementary metal-oxide semiconductor and memristive hardware for neuromorphic computing. *Advanced Intelligent Systems*, 2(5):1900189, 2020.

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

[33] Tara Hamilton. The best of both worlds. *Nature Machine Intelligence*, 3(3):194–195, 2021.

[34] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. Computational Neuroscience Series, 2001.

[35] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[36] Ad M.H.J. Aertsen and P.I.M. Johannesma. The spectro-temporal receptive field. *Biological Cybernetics*, 42(2):133–143, 1981.

[37] Andrea Benucci, Aman B Saleem, and Matteo Carandini. Adaptation maintains population homeostasis in primary visual cortex. *Nature Neuroscience*, 16(6):724–729, 2013.

[38] Barry Wark, Brian Nils Lundstrom, and Adrienne Fairhall. Sensory adaptation. *Current opinion in neurobiology*, 17(4):423–429, 2007.

[39] Jason K Eshraghian, Seungbum Baek, Jun-Ho Kim, Nicolangelo Iannella, Kyoungrok Cho, Yong Sook Goo, Herbert HC Iu, Sung-Mo Kang, and Kamran Eshraghian. Formulation and implementation of nonlinear integral equations to model neural dynamics within the vertebrate retina. *International Journal of Neural Systems*, 28(07):1850004, 2018.

[40] P-F Ruedi, Pascal Heim, François Kaess, Eric Grenet, Friedrich Heitger, P-Y Burgi, Stève Gyger, and Pascal Nussbaum. A 128× 128 pixel 120-db dynamic-range vision-sensor chip for image contrast and orientation extraction. *IEEE Journal of Solid-State Circuits*, 38(12):2325–2333, 2003.

[41] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128 x 128 120db 30mw asynchronous vision sensor that responds to relative intensity change. In *2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers*, pages 2060–2069. IEEE, 2006.

[42] Jason Kamran Eshraghian, Kyoungrok Cho, Ciyan Zheng, Minho Nam, Herbert Ho-Ching Iu, Wen Lei, and Kamran Eshraghian. Neuromorphic vision hybrid RRAM-CMOS architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12):2816–2829, 2018.

[43] Dennis E Robey, Wesley Thio, Herbert HC Iu, and Jason K Eshraghian. Naturalizing neuromorphic vision event streams using generative adversarial networks. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[44] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *arXiv preprint arXiv:1904.08405*, 2019.

[45] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[46] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[47] Louis Lapique. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology and Pathology*, 9:620–635, 1907.

[48] Nicolas Brunel and Mark CW Van Rossum. Lapicque's 1907 paper: From frogs to integrate-and-fire. *Biological Cybernetics*, 97(5):337–339, 2007.

[49] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[50] Anthony N Burkitt. A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. *Biological Cybernetics*, 95(1):1–19, 2006.

[51] Tim P Vogels and Larry F Abbott. Signal propagation and logic gating in networks of integrate-and-fire neurons. *Journal of Neuroscience*, 25(46):10786–10795, 2005.

[52] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[53] Wulfram Gerstner. A framework for spiking neuron models: The spike response model. In *Handbook of Biological Physics*, volume 4, pages 469–516. Elsevier, 2001.

[54] Sumit Bam Shrestha and Garrick Orchard. SLAYER: Spike layer error reassignment in time. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1419–1428, 2018.

[55] Malu Zhang, Jiadong Wang, Zhixuan Zhang, Ammar Belatreche, Jibin Wu, Yansong Chua, Hong Qu, and Haizhou Li. Spike-timing-dependent back propagation in deep spiking neural networks. *arXiv preprint arXiv:2003.11837*, 2020.

[56] Friedemann Zenke and Emre O Neftci. Brain-inspired learning on neuromorphic substrates. *Proceedings of the IEEE*, 109(5):935–950, 2021.

[57] Gustavo BM Mello, Sofia Soares, and Joseph J Paton. A scalable population code for time in the striatum. *Current Biology*, 25(9):1113–1122, 2015.

[58] Selig Hecht, Simon Shlaer, and Maurice Henri Pirenne. Energy, quanta, and vision. *Journal of General Physiology*, 25(6):819–840, 1942.

[59] H.A. Van Der Velden. The number of quanta necessary for the perception of light of the human eye. *Ophthalmologica*, 111(6):321–331, 1946.

[60] Foster Rieke and Denis A Baylor. Single-photon detection by rod cells of the retina. *Reviews of Modern Physics*, 70(3):1027, 1998.

[61] Jason K Eshraghian, Seungbum Baek, Timothée Levi, Takashi Kohno, Said Al-Sarawi, Philip HW Leong, Kyoungrok Cho, Derek Abbott, and Omid Kavehei. Nonlinear retinal response modeling for future neuromorphic instrumentation. *IEEE Instrumentation & Measurement Magazine*, 23(1):21–29, 2020.

[62] Seungbum Baek, Jason K Eshraghian, Wesley Thio, Yulia Sandamirskaya, Herbert HC Iu, and Wei D Lu. A real-time retinomorphic simulator using a conductance-based discrete neuronal network. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 79–83. IEEE, 2020.

[63] Stanislas Dehaene. The neural basis of the Weber–Fechner law: A logarithmic mental number line. *Trends in Cognitive Sciences*, 7(4):145–147, 2003.

[64] Coen Arrow, Hancong Wu, Seungbum Baek, Herbert HC Iu, Kia Nazarpour, and Jason K Eshraghian. Prosthesis control using spike rate coding in the retina photoreceptor cells. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[65] Y Bi, A Chadha, A Abbas, , E Bourtsoulatze, and Y Andreopoulos. Graph-based object classification for neuromorphic vision sensing. In *2019 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2019.

[66] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36(2):142–149, 2017.

[67] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017.

[68] Yuhuang Hu, Hongjie Liu, Michael Pfeiffer, and Tobi Delbruck. DVS benchmark datasets for object tracking, action recognition, and object recognition. *Frontiers in Neuroscience*, 10:405, 2016.

[69] Alex Zihao Zhu, Dinesh Thakur, Tolga Özaslan, Bernd Pfrommer, Vijay Kumar, and Kostas Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, 2018.

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

[71] Teresa Serrano-Gotarredona and Bernabe Linares-Barranco. Poker-DVS and MNIST-DVS. Their history, how they were made, and other details. *Frontiers in Neuroscience*, 9:481, 2015.

[72] Mathias Gehrig, Willem Aarents, Daniel Gehrig, and Davide Scaramuzza. Dsec: A stereo event camera dataset for driving scenarios. *IEEE Robotics and Automation Letters*, 6(3):4947–4954, 2021.

[73] Jithendar Anumula, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. Feature representations for neuromorphic audio spike streams. *Frontiers in Neuroscience*, 12:23, 2018.

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

[75] Bruno A Olshausen and David J Field. What is the other 85 percent of V1 doing? *L. van Hemmen, & T. Sejnowski (Eds.)*, 23:182–211, 2006.

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

[77] Shigeru Shinomoto and Shinsuke Koyama. A solution to the controversy between rate and temporal coding. *Statistics in Medicine*, 26(21):4032–4038, 2007.

[78] MR Mehta, AK Lee, and MA Wilson. Role of experience and oscillations in transforming a rate code into a temporal code. *Nature*, 417(6890):741–746, 2002.

[79] Romain Brette. Philosophy of the spike: Rate-based vs. spike-based theories of the brain. *Frontiers in Systems Neuroscience*, 9:151, 2015.

[80] Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, et al. A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770, 2019.

[81] Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[82] Yanping Huang and Rajesh PN Rao. Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):580–593, 2011.

[83] Adam H Marblestone, Greg Wayne, and Konrad P Kording. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10:94, 2016.

[84] Kalyanmoy Deb. Multi-objective optimization. In *Search Methodologies*, pages 403–449. Springer, 2014.

[85] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113(41):11441–11446, 2016.

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

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

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

[89] Nicolas Perez-Nieves and Dan FM Goodman. Sparse spiking gradient descent. *arXiv preprint arXiv:2105.08810*, 2021.

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

[91] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Computation*, 33(4):899–925, 2021.

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

[93] Saeed Reza Kheradpisheh and Timothée Masquelier. Temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, 30(06):2050027, 2020.

[94] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *Elife*, 6:e22901, 2017.

[95] Emre O Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, 11:324, 2017.

[96] Edward M Callaway. Feedforward, feedback and inhibitory connections in primate visual cortex. *Neural Networks*, 17(5-6):625–632, 2004.

[97] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[98] Michael Laskin, Luke Metz, Seth Nabarrao, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel training of deep networks with local updates. *arXiv preprint arXiv:2012.03837*, 2020.

[99] Corey Lammie, Jason K Eshraghian, Wei D Lu, and Mostafa Rahimi Azghadi. Memristive stochastic computing for deep learning parameter optimization. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(5):1650–1654, 2021.

[100] Siddharth Gaba, Phil Knag, Zhengya Zhang, and Wei Lu. Memristive devices for stochastic computing. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2592–2595. IEEE, 2014.

[101] Fuxi Cai, Suhas Kumar, Thomas Van Vaerenbergh, Xia Sheng, Rui Liu, Can Li, Zhan Liu, Martin Foltin, Shimeng Yu, Qiangfei Xia, et al. Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks. *Nature Electronics*, 3(7):409–418, 2020.

[102] Ronald J Williams. Toward a theory of reinforcement-learning connectionist systems. *Technical Report NU-CCS-88-3, Northeastern University*, 1988.

[103] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[104] Justin Werfel, Xiaohui Xie, and H Sebastian Seung. Learning curves for stochastic gradient descent in linear feedforward networks. *Neural Computation*, 17(12):2699–2718, 2005.

[105] Hyunjune Sebastian Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40(6):1063–1073, 2003.

[106] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[107] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[108] Sergey Bartunov, Adam Santoro, Blake A Richards, Luke Marris, Geoffrey E Hinton, and Timothy P Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9390–9400, 2018.

[109] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. *arXiv preprint arXiv:1811.03567*, 2018.

[110] Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in Neuroscience*, 15, 2021.

[111] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160, 2007.

[112] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *Frontiers in Neuroscience*, 12:608, 2018.

[113] Donald Olding Hebb. *The organisation of behaviour: A neuropsychological theory*. Science Editions New York, 1949.

[114] Lyudmila Kushnir and Sophie Denève. Learning temporal structure of the input with a network of integrate-and-fire neurons. *arXiv preprint arXiv:1912.10262*, 2019.

[115] Sophie Denève, Alireza Alemi, and Ralph Bourdoukan. The brain as an efficient and robust adaptive learner. *Neuron*, 94(5):969–977, 2017.

[116] Friedemann Zenke. Spytorch. Online: https://github.com/fzenke/spytorch. 2019.

[117] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[118] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[119] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), University of Helsinki*, pages 6–7, 1970.

[120] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System Modeling and Optimization*, pages 762–770. Springer, 1982.

[121] Amirhossein Tavanaei and Anthony Maida. BP-STDP: approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, 330:39–47, 2019.

[122] José Antonio Pérez-Carrasco, Bo Zhao, Carmen Serrano, Begona Acha, Teresa Serrano-Gotarredona, Shouchun Chen, and Bernabe Linares-Barranco. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2706–2719, 2013.

[123] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with LIF neurons. *arXiv preprint arXiv:1510.08829*, 2015.

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

[125] Yangfan Hu, Huajin Tang, and Gang Pan. Spiking deep residual network. *arXiv preprint arXiv:1805.01352*, 2018.

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

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

[128] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *International Conference on Learning Representations*, 2019.

[129] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12:774, 2018.

[130] Olaf Booij and Hieu tat Nguyen. A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, 95(6):552–558, 2005.

[131] Yan Xu, Xiaoqin Zeng, Lixin Han, and Jing Yang. A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Networks*, 43:99–113, 2013.

[132] Timo C Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):1–17, 2021.

[133] Iulia M Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8529–8533. IEEE, 2020.

[134] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[135] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint arXiv:1803.09574*, 2018.

[136] Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. *arXiv preprint arXiv:1706.04698*, 2017.

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

[138] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[139] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[140] Alanna J Watt and Niraj S Desai. Homeostatic plasticity and STDP: Keeping a neuron's cool in a fluctuating world. *Frontiers in Synaptic Neuroscience*, 2:5, 2010.

[141] Per Jesper Sjöström, Gina G Turrigiano, and Sacha B Nelson. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron*, 32(6):1149–1164, 2001.

[142] Benjamin Schrauwen and Jan Van Campenhout. Extending Spikeprop. In *2004 IEEE International Joint Conference on Neural Networks*, volume 1, pages 471–475. IEEE, 2004.

[143] Aboozar Taherkhani, Ammar Belatreche, Yuhua Li, and Liam P Maguire. DL-ReSuMe: a delay learning-based remote supervised method for spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, 26(12):3137–3149, 2015.

[144] Alfonso Renart, Nicolas Brunel, and Xiao-Jing Wang. Mean-field theory of irregularly spiking neuronal populations and working memory in recurrent cortical networks. *Computational Neuroscience: A Comprehensive Approach*, pages 431–490, 2004.

[145] Daniel J Amit. *Modeling brain function: The world of attractor neural networks*. Cambridge University Press, 1992.

[146] Alfonso Renart, Pengcheng Song, and Xiao-Jing Wang. Robust spatial working memory through homeostatic synaptic scaling in heterogeneous cortical networks. *Neuron*, 38(3):473–485, 2003.

[147] Mattia Rigotti, Daniel D Ben Dayan Rubin, Xiao-Jing Wang, and Stefano Fusi. Internal representation of task rules by recurrent dynamics: The importance of the diversity of neural responses. *Frontiers in Computational Neuroscience*, 4:24, 2010.

[148] John Miller and Moritz Hardt. Stable recurrent models. In *International Conference on Learning Representations*, 2019.

[149] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[150] Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. In *International Conference on Learning Representations*, 2018.

[151] Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random Kronecker factors. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6594–6603, 2018.

[152] Christopher Roth, Ingmar Kanitscheider, and Ila Fiete. Kernel RNN learning (keRNl). In *International Conference on Learning Representations*, 2018.

[153] James M Murray. Local online learning in recurrent networks with random feedback. *ELife*, 8:e43299, 2019.

[154] Owen Marschall, Kyunghyun Cho, and Cristina Savin. A unified framework of online learning algorithms for training recurrent neural networks. *Journal of Machine Learning Research*, 2020.

[155] Thomas Bohnstingl, Stanisław Woźniak, Wolfgang Maass, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. *arXiv preprint arXiv:2007.12723*, 2020.

[156] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[157] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989.

[158] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[159] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[160] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*, 2017.

[161] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

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

[163] Joseph M Brader, Walter Senn, and Stefano Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Computation*, 19(11):2881–2912, 2007.

[164] Bo Zhao, Ruoxi Ding, Shoushun Chen, Bernabe Linares-Barranco, and Huajin Tang. Feedforward categorization on AER motion events using cortex-like features in a spiking neural network. *IEEE Transactions on Neural Networks and Learning Systems*, 26(9):1963–1978, 2014.

[165] Michael Beyeler, Nikil D Dutt, and Jeffrey L Krichmar. Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule. *Neural Networks*, 48:109–124, 2013.

[166] Damien Querlioz, Olivier Bichler, Philippe Dollfus, and Christian Gamrat. Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Transactions on Nanotechnology*, 12(3):288–295, 2013.

[167] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

[168] Geoffrey Hinton et al. Can the brain do back-propagation? In *Invited talk at Stanford University Colloquium on Computer Systems*, 2016.

[169] Melika Payvand, Mohammed E Fouda, Fadi Kurdahi, Ahmed M Eltawil, and Emre O Neftci. On-chip error-triggered learning of multi-layer memristive spiking neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):522–535, 2020.

[170] Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

[171] Radoslaw Martin Cichy, Aditya Khosla, Dimitrios Pantazis, Antonio Torralba, and Aude Oliva. Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence. *Scientific Reports*, 6(1):1–13, 2016.

[172] Rishi Rajalingham, Elias B Issa, Pouya Bashivan, Kohitij Kar, Kailyn Schmidt, and James 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. *Journal of Neuroscience*, 38(33):7255–7269, 2018.

[173] Martin Schrimpf, Jonas Kubilius, Ha Hong, Najib J Majaj, Rishi Rajalingham, Elias B Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Franziska Geiger, et al. Brain-score: Which artificial neural network for object recognition is most brain-like? *bioRxiv*, 2020.

[174] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun. *A guide to convolutional neural networks for computer vision*, volume 8. Morgan & Claypool Publishers, 2018.

[175] Alexander JE Kell, Daniel LK Yamins, Erica N Shook, Sam V Norman-Haignere, and Josh H McDermott. A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron*, 98(3):630–644, 2018.

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

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

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

[179] Alexander Neckar, Sam Fok, Ben V Benjamin, Terrence C Stewart, Nick N Oza, Aaron R Voelker, Chris Eliasmith, Rajit Manohar, and Kwabena Boahen. Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164, 2018.

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

[181] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

[182] Vladimir Kornijcuk and Doo Seok Jeong. Recent progress in real-time adaptable digital neuromorphic hardware. *Advanced Intelligent Systems*, 1(6):1900030, 2019.

[183] Enea Ceolini, Charlotte Frenkel, Sumit Bam Shrestha, Gemma Taverni, Lyes Khacef, Melika Payvand, and Elisa Donati. Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing. *Frontiers in Neuroscience*, 14, 2020.

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

[185] Mohammadali Sharifshazileh, Karla Burelo, Johannes Sarnthein, and Giacomo Indiveri. An electronic neuromorphic system for real-time detection of high frequency oscillations (HFO) in intracranial eeg. *Nature Communications*, 12(1):1–14, 2021.

[186] Ram Krips and Miriam Furst. Stochastic properties of coincidence-detector neural cells. *Neural Computation*, 21(9):2524–2553, 2009.

[187] Romain Brette. Computing with neural synchrony. *PLoS Computational Biology*, 8(6):e1002561, 2012.

[188] Rudi Primorac, Roberto Togneri, Mohammed Bennamoun, and Ferdous Sohel. Generalized joint sparse representation for multimodal biometric fusion of heterogeneous features. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2018.

[189] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *Proceedings of the 31st International Conference on Neural Information Processing Systems: Workshop Autodiff Submission*, 2017.

[190] Magdalena Sanhueza and John Lisman. The CaMKII/NMDAR complex as a molecular memory. *Molecular brain*, 6(1):1–8, 2013.

[191] James J Jun, Nicholas A Steinmetz, Joshua H Siegle, Daniel J Denman, Marius Bauza, Brian Barbarits, Albert K Lee, Costas A Anastassiou, Alexandru Andrei, Çağatay Aydın, et al. Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679):232–236, 2017.

[192] Nicholas A Steinmetz, Cagatay Aydin, Anna Lebedeva, Michael Okun, Marius Pachitariu, Marius Bauza, Maxime Beau, Jai Bhagat, Claudia Böhm, Martijn Broux, et al. Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science*, 372(6539), 2021.

[193] Jason K. Eshraghian. snnTorch. Online: https://github.com/jeshraghian/snntorch. 2021.

A Appendix A: From Artificial to Spiking Neural Networks

A.1 Forward Euler Method to Solving Spiking Neuron Models

The time derivative $dU(t)/dt$ is substituted into Equation (1) without taking the limit $\Delta t \rightarrow 0$:

$$\tau \frac{U(t+\Delta t) - U(t)}{\Delta t} = -U(t) + I_{\text{in}}(t)R \tag{13}$$

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 side of the equation gives:

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t) + \frac{\Delta t}{\tau} I_{\text{in}}(t)R \tag{14}$$

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

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t) \tag{15}$$

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 Equation (14), this implies that $\beta = (1 - \Delta t/\tau)$.

Assume t is discretised into sequential time-steps, such that $\Delta t = 1$. To further reduce the number of hyperparameters from Equation (14), assume $R = 1\Omega$. This leads to the result in Equation (3), where the following representation is shifted by one time step:

$$\beta = (1 - \frac{1}{\tau}) \implies U[t+1] = \beta U[t] + (1-\beta) I_{\text{in}}[t+1] \tag{16}$$

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 Equation (2). In absence of current injection:

$$U(t) = U_0 e^{-t/\tau} \tag{17}$$

where U_0 is the initial membrane potential at $t = 0$. Assuming Equation (17) 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:

$$\begin{aligned} \beta =& \frac{U_0 e^{-(t+\Delta t)/\tau}}{U_0 e^{-t/\tau}} = \frac{U_0 e^{-(t+2\Delta t)/\tau}}{U_0 e^{-(t+\Delta t)/\tau}} = ... \\ \implies& \beta = e^{-\Delta t/\tau} \end{aligned} \tag{18}$$

It is preferable to calculate β using Equation (18) 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 Equation (16).

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

$$WX[t] = I_{\text{in}}[t] \tag{19}$$

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:

$$U[t+1] = \beta U[t] + WX[t+1] \tag{20}$$

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

To arrive at Equation (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 Equation (4), or by forcing the membrane potential to zero:

$$U[t+1] = \underbrace{\beta U[t]}_{\text{decay}} + \underbrace{WX[t]}_{\text{input}} - \underbrace{S_{\text{out}}(\beta U[t] + WX[t])}_{\text{reset-to-zero}} \tag{21}$$

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 (Section 4.1), though has not yet been characterised 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:

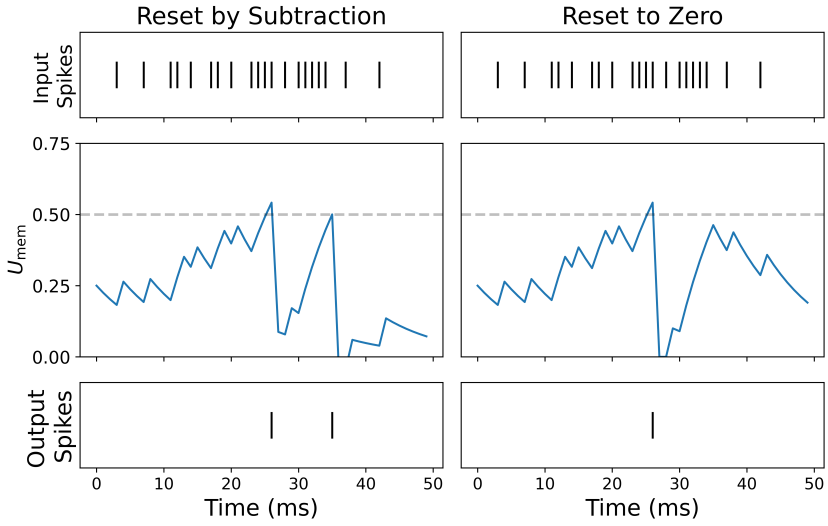

Figure S.1: Reset by subtraction vs reset-to-zero. Threshold set to $\theta = 0.5$.

B Appendix B: Spike Encoding

The following spike encoding mechanisms and loss functions are described with respect to a single sample of data. They can be generalised to multiple samples as is common practice in deep learning to process data in batches.

B.1 Rate Coded Input Conversion

An example of conversion of an input sample to a rate coded spike train follows. 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 normalised 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:

$$\mathrm{P}(R_{ijk} = 1) = X_{ij} = 1 - \mathrm{P}(R_{ijk} = 0) \tag{22}$$

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 a '1' or a '0'. As the number of time steps $t \to \infty$, the proportion of spikes is expected to approach 0.5.

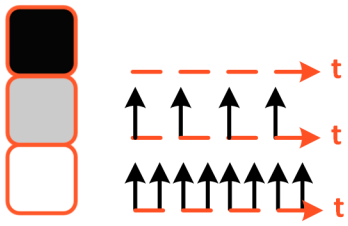

Figure S.2: Rate coded input pixel. An input pixel of greater intensity corresponds to a higher firing rate.

B.2 Latency Coded Input Conversion

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

$$U(t) = I_{\text{in}} R(1 - e^{-t/\tau}) \tag{23}$$

For a constant current injection, $U(t)$ will exponentially relax towards 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 $U(t) = \theta$:

$$t = \tau \left[\ln\left(\frac{I_{\text{in}} R}{I_{\text{in}} R - \theta} \right) \right] \tag{24}$$

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:

$$t(x) = \begin{cases} \tau \left[\ln\left(\frac{x}{x - \theta} \right) \right], & x > \theta \\ \infty, & \text{otherwise} \end{cases} \tag{25}$$

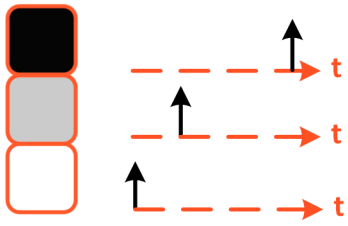

Figure S.3: Latency coded input pixel. An input pixel of greater intensity corresponds to an earlier spike time.

B.3 Rate Coded Outputs

A vectorised implementation of determining the predicted class from rate-coded output spike trains is described. 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:

$$\vec{c} = \sum_{j=0}^{T} \vec{S}[t] \tag{26}$$

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

$$\hat{y} = \arg\max_i c_i \tag{27}$$

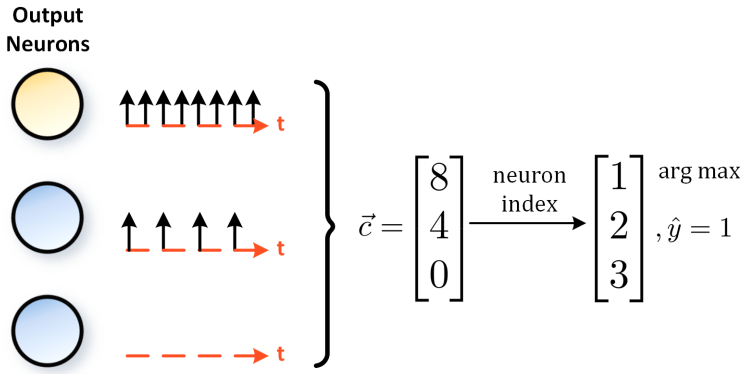

Figure S.4: Rate coded outputs. $\vec{c} \in \mathbb{R}^{N_C}$ is the spike count from each output neuron, where the example above shows the first neuron firing a total of 8 times. $\hat{y}$ represents the index of the predicted output neuron, where it indicates the first neuron is the correct class.

B.4 Cross Entropy Spike Rate

The spike count of the output layer $\vec{c} \in \mathbb{R}^{N_C}$ is obtained as in Equation (26). c_i is the i^{th} element of $\vec{c}$, treated as the logits in the softmax function:

$$p_i = \frac{e^{c_i}}{\sum_{i=1}^{N_C} e^{c_i}} \tag{28}$$

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:

$$\mathcal{L}_{CE} = \sum_{i=0}^{N_C} y_i \log(p_i) \tag{29}$$

B.5 Mean Square Spike Rate

As in Equation (26), the spike count of the output layer $\vec{c} \in \mathbb{R}^{N_C}$ is obtained. c_i is the i^{th} element of $\vec{c}$, and let $y_i \in \mathbb{R}$ be the target spike count over a period of time T for the i^{th} output neuron. The target for the correct class should be greater than that of incorrect classes:

$$\mathcal{L}_{MSE} = \sum_{i}^{N_C} (y_i - c_i)^2 \tag{30}$$

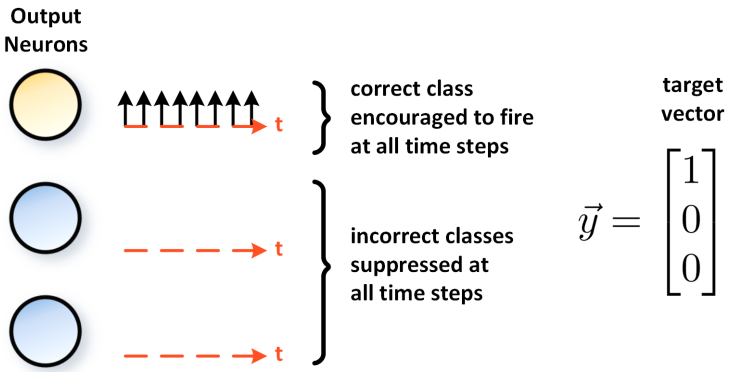

Figure S.5: Cross Entropy Spike Rate. The target vector $\vec{y}$ specifies the correct class as a one-hot encoded vector.

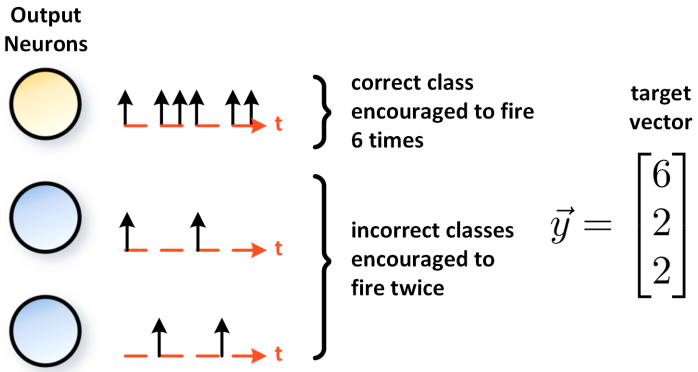

Figure S.6: Mean Square Spike Rate. The target vector $\vec{y}$ specifies the total desired number of spikes for each class.

B.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:

$$\vec{m} = \max_t \vec{U}[t] \tag{31}$$

The elements of $\vec{m}$ replace c_i in the softmax function from Equation (28), 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:

$$\vec{m} = \sum_t^T \vec{U}[t] \tag{32}$$

B.7 Mean Square Membrane

Let $y_i[t]$ be a time-varying value that specifies the target membrane potential of the i^{th} neuron at each time step. The total mean square error is calculated by summing the loss for all T time steps and for all N_C output layer neurons:

$$\mathcal{L}_{MSE} = \sum_i^{N_C} \sum_t^T (y_i[t] - U[t])^2 \tag{33}$$

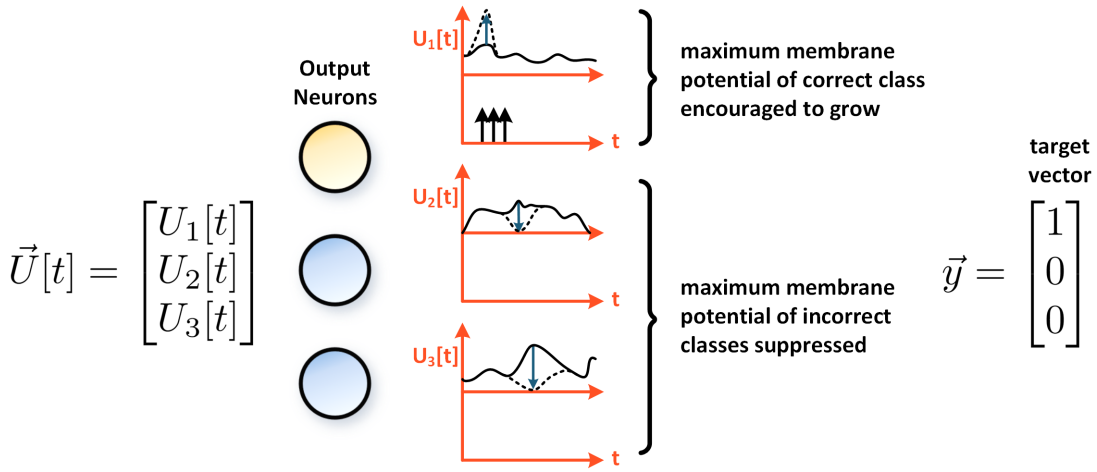

Figure S.7: Maximum Membrane. The peak membrane potential for each neuron is used in the cross entropy loss function. This encourages the peak of the correct class to grow, while that of the incorrect class is suppressed. The effect of this is to promote more firing from the correct class and less from the incorrect class.

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.

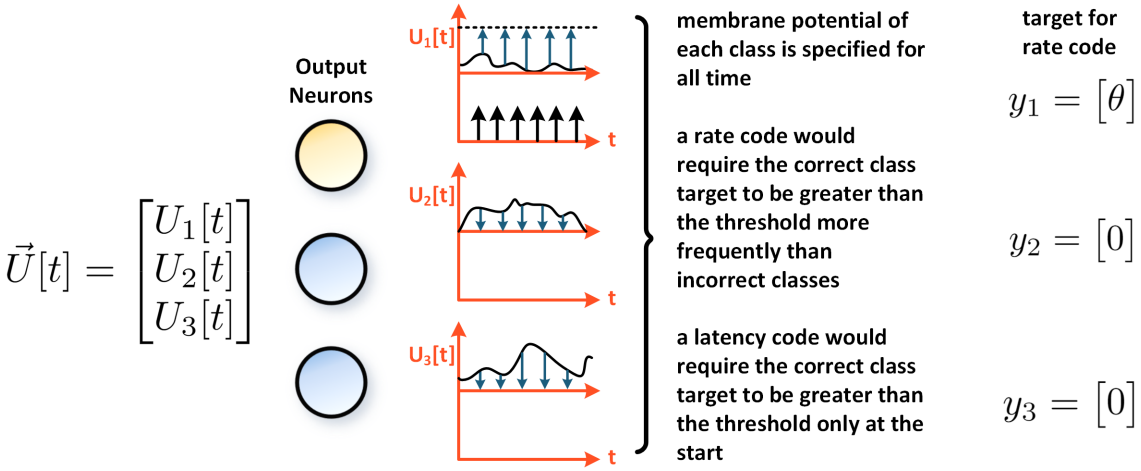

Figure S.8: Mean Square Membrane. The membrane potential at each time step is applied to the mean square error loss function. This allows a defined membrane target. The example above sets the target at all time steps at the firing threshold for the correct class, and to zero for incorrect classes.

B.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. Cross entropy minimisation aims to maximise 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 [55] simply negates the spike times:

$$\vec{f} := -\vec{f} \tag{34}$$

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

$$f_i := \frac{1}{f_i} \tag{35}$$

The new values of f_i then replace c_i in the softmax function from Equation (28). Equation (35) must be treated with care, as it precludes spikes from occurring at $t = 0$, otherwise $f_i \to \infty$.

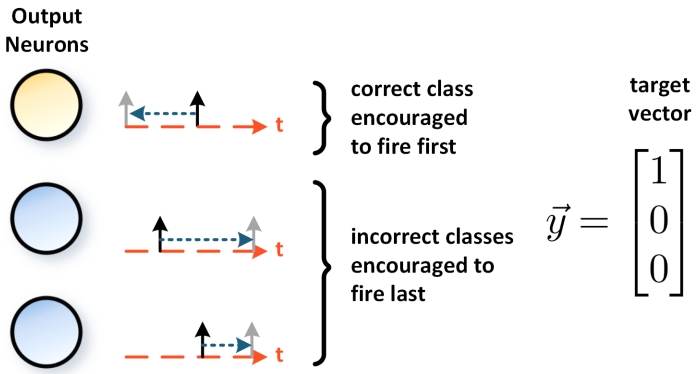

Figure S.9: Cross Entropy Latency Code. Applying the inverse (or negated) spike time to the cross entropy loss pushes the correct class to fire first, and incorrect classes to fire later.

B.9 Mean Square Spike Time

The spike time(s) of all neurons are specified as targets. 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, $y_i \in \mathbb{R}$ is the target spike time for the i^{th} output neuron. The mean square errors between the actual and target spike times of all output classes are summed together:

$$\mathcal{L}_{MSE} = \sum_{i}^{N_C} (y_i - f_i)^2 \tag{36}$$

This can be generalised to account for multiple spikes [54]. 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 i^{th} neuron, respectively. The k^{th} spike is sequentially taken from $\vec{f_i}$ and $\vec{y_i}$, and the mean square error 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 together across spikes and classes:

$$\mathcal{L}_{MSE} = \sum_{k}^{n} \sum_{i}^{N_C} (y_{i,k} - f_{i,k})^2 \tag{37}$$

B.10 Mean Square Relative Spike Time

The difference between the spike time of correct and incorrect neurons is specified as a target. As in Appendix B.9, y_i is the desired spike time for the i^{th} neuron and f_i is the 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:

$$y_i = \begin{cases} f_0 + \gamma, & \text{if } f_i < f_0 + \gamma \\ f_i, & \text{if } f_i \geq f_0 + \gamma \end{cases} \tag{38}$$

where γ is a pre-defined 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

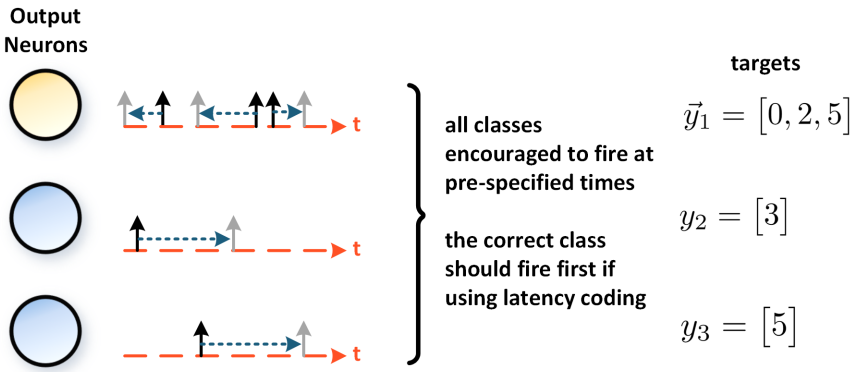

Figure S.10: Mean Square Spike Time. The timing of all spikes are iterated over, and sequentially applied to the mean square error loss function. This enables the timing for multiple spikes to be precisely defined.

after the correct neuron, then 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 mean square error loss (Equation (37)).

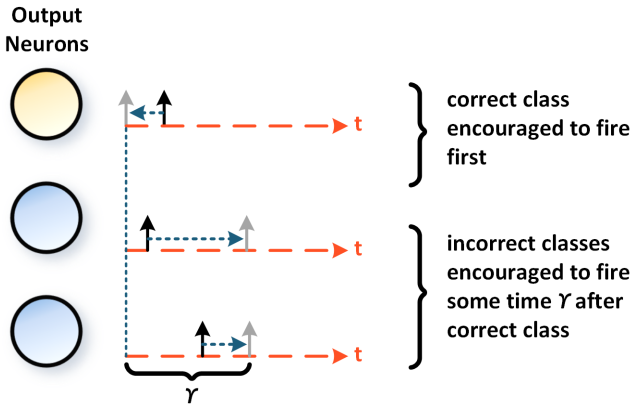

Figure S.11: Mean Square Relative Spike Time. The relative timing between all spikes are applied to the mean square error loss function, enabling a defined time window γ to occur between the correct class firing and incorrect classes firing.

B.11 Population Level Regularisation

L1-regularisation can be applied to the total number of spikes emitted at the output layer to penalise excessive firing [116], thus encouraging sparse activity at the output:

$$\mathcal{L}_{L1} = \lambda_1 \sum_{t}^{T} \sum_{i}^{N_C} S_i[t] \tag{39}$$

where λ_1 is a hyperparameter controlling the influence of the regularisation term, and $S_i[t]$ is the spike of the i^{th} 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 regularisation penalty apply:

$$\mathcal{L}_U = \lambda_U \Big(\Big[\sum_{i}^{N} c_i^{(l)} - \theta_U \Big]_+ \Big)^L \tag{40}$$

where c_i is the total spike count over time for the i^{th} 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 regularisation, 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 [91]. It is possible to swap out the spike count for a time-averaged membrane potential as well, if using hidden-state variables is permissible [88].

B.12 Neuron Level Regularisation

A lower-activity threshold θ_L that specifies the lower permissible limit of firing for *each* neuron before the regularisation penalty is applied:

$$\mathcal{L}_L = \frac{\lambda_L}{N} \sum_i^N \left(\left[\theta_L - c_i^{(l)} \right]_+ \right)^2 \tag{41}$$

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 regularisation [91]. As with population-level regularisation, the spike count can also be substituted for a time-averaged membrane potential [88].

C Appendix C: Training Spiking Neural Networks

C.1 Backpropagation Using Spike Times

In the original description of SpikeProp from [92], a spike response model is used:

$$\begin{aligned} U_j(t) &= \sum_{i,k} W_{i,j} I_i^{(k)}(t), \\ I_i^{(k)}(t) &= \epsilon(t - f_i^{(k)}), \end{aligned} \tag{42}$$

where $W_{i,j}$ is the weight between the i^{th} presynaptic and j^{th} postsynaptic neurons, $f_i^{(k)}$ is the firing time of the k^{th} spike from the i^{th} presynaptic neuron, and $U_j(t)$ is the membrane potential of the j^{th} neuron. For simplicity, the 'alpha function' defined below is frequently used for the kernel:

$$\epsilon(t) = \frac{t}{\tau} e^{1-\frac{t}{\tau}} \Theta(t), \tag{43}$$

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

Consider an SNN where each target specifies the timing of the output spike emitted from the j^{th} output neuron (y_j). This is used in the mean square spike time loss (Equation (36), Appendix B.9), 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:

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = \frac{\partial \mathcal{L}}{\partial f_j} \frac{\partial f_j}{\partial U_j} \frac{\partial U_j}{\partial W_{i,j}} \Big|_{t=f_j}. \tag{44}$$

The first term on the right side evaluates to:

$$\frac{\partial \mathcal{L}}{\partial f_j} = 2(y_j - f_j). \tag{45}$$

The third term can be derived from Equation (42):

$$\frac{\partial U_j}{\partial W_{i,j}} \Big|_{t=f_j} = \sum_k I_i^{(k)}(f_j) = \sum_k \epsilon(f_j - f_i^{(k)}). \tag{46}$$

The second term in Equation (44) can be calculated by calculating $-\partial U_j/\partial f_j\Big|_{t=f_j}$ instead, and then taking the inverse. In [92], the evolution of $U_j(t)$ can be analytically solved using Equations (42) and (43):

$$\frac{\partial f_j}{\partial U_j} \leftarrow -\left(\frac{\partial U_j}{\partial t}\Big|_{t=f_j}\right)^{-1} = -\left(\sum_{i,k} W_{i,j}\frac{\partial I_i^{(k)}}{\partial t}\Big|_{t=f_j}\right)^{-1} = \sum_{i,k}\frac{\tau^2}{f_j - f_i^{(k)} - \tau} e^{\frac{f_j - f_i^{(k)}}{\tau} - 1}\Theta\left(f_j - f_i^{(k)}\right). \quad (47)$$

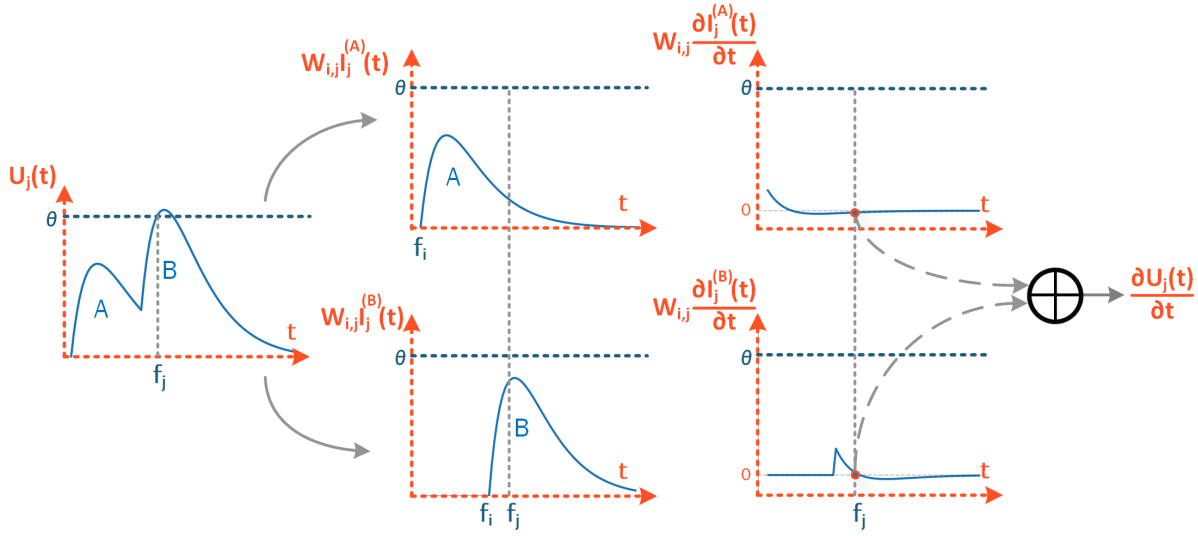

Figure S.12: Calculation of derivative of membrane potential with respect to spike time. The superscripts $^{(A)}$ and $^{(B)}$ denote the separate contributions from each application of the kernel.

Note, the input current is triggered at the onset of the pre-synaptic spike $t = f_i$, but is evaluated at the time of the post-synaptic spike $t = f_j$. The results can be combined to give:

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = -\frac{2(y_j - f_j)\sum_k I_i^{(k)}(f_j)}{\sum_{i,k} W_{i,j}(\partial I_j^{(k)}/\partial t)\Big|_{t=f_j}} \quad (48)$$

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.

C.2 Backpropagation Using Spikes

Spike Timing Dependent Plasticity

The connection between a pair of neurons can be altered by the spikes emitted by both neurons. Several experiments have shown the relative timing of spikes between pre- and post-synaptic neurons can be used to define a learning rule for updating the synaptic weight [32]. Let t_{pre} and t_{post} represent the timing of the pre- and post-synaptic spikes, respectively. The difference in spike time is:

$$\Delta t = t_{\text{pre}} - t_{\text{post}} \quad (49)$$

When the pre-synaptic neuron emits a spike before the post-synaptic neuron, such that the pre-synaptic spike may have caused the post-synaptic spike, then the synaptic strength is expected to increase ('potentiation'). When reversed, i.e., the post-synaptic neuron spikes before the pre-synaptic neuron, the synaptic strength decreases ('depression'). This rule is known as spike timing dependent plasticity (STDP), and has been shown to exist in various brain regions including

the visual cortex, somatosensory cortex and the hippocampus. Fitting curves to experimental measurements take the following form [32]:

$$\Delta W = \begin{cases} A_+ e^{\Delta t/\tau_+}, & \text{if } t_{\text{post}} > t_{\text{pre}} \\ A_- e^{-\Delta t/\tau_-}, & \text{if } t_{\text{post}} < t_{\text{pre}} \end{cases} \tag{50}$$

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, τ_+ and τ_- are the time constants that determine the strength of the update over a given interspike interval. This mechanism is illustrated in Figure S.13.

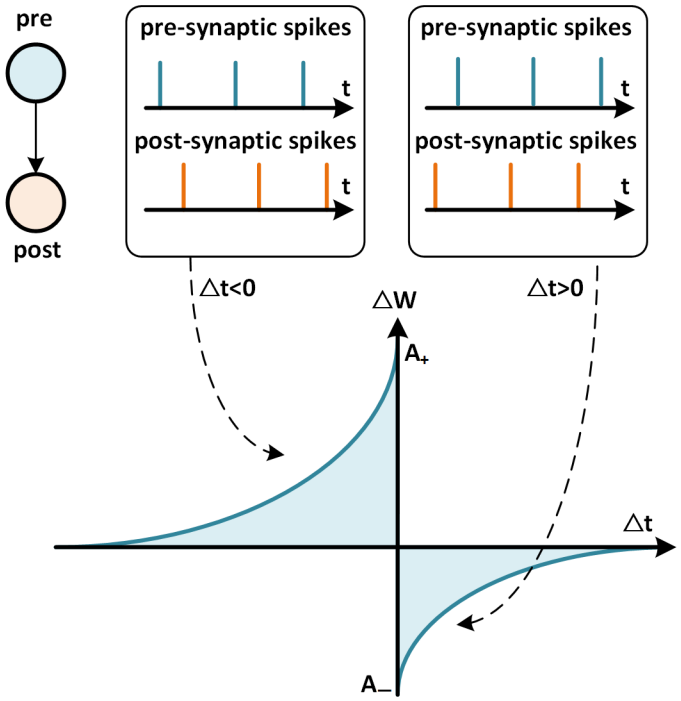

Figure S.13: STDP Learning Window. If the pre-synaptic neuron spikes before the post-synaptic neuron, $\Delta t < 0 \implies \Delta W > 0$ and the synaptic strength between the two neurons is increased. If the pre-synaptic neuron spikes after the post-synaptic neuron, $\Delta t > 0 \implies \Delta W < 0$ and the synaptic strength is decreased.

For a strong, excitatory synaptic connection, a pre-synaptic spike will trigger a large post-synaptic potential (refer to U in Equation (4)). As membrane potential approaches the threshold of neuronal firing, such an excitatory case suggests that a post-synaptic spike will likely follow a pre-synaptic spike. This will lead to a positive change of the synaptic weight, thus increasing the chance that a post-synaptic spike will follow a pre-synaptic spike in 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 is typically correlated in both space and time, so a network's response to a correlated spike train will be to 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 pre-synaptic neurons will arrive at a post-synaptic neuron within a close time interval, causing stronger depolarization of the neuron membrane potential, and a higher probability of a post-synaptic 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 (Appendix C.3).

C.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:

$$\theta[t] = \theta_0 + b[t] \tag{51}$$

$$b[t+1] = \alpha b[t] + (1-\alpha) S_{\text{out}}[t] \tag{52}$$

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 absence of further spikes. The above form is loosely based on [135], though the decay rate α and threshold jump factor $(1-\alpha)$ can be decoupled from each other. α can be treated as either a hyperparameter or a learnable parameter.