Hyeryung Jang, Osvaldo Simeone,
Brian Gardner, and André Grüning

An Introduction to Probabilistic Spiking Neural Networks

Probabilistic models, learning rules, and applications

©ISTOCKPHOTO.COM/JUST_SUPER

Digital Object Identifier 10.1109/MSP.2019.2935234
Date of current version: 29 October 2019

Spiking neural networks (SNNs) are distributed trainable systems whose computing elements, or neurons, are characterized by internal analog dynamics and by digital and sparse synaptic communications. The sparsity of the synaptic spiking inputs and the corresponding event-driven nature of neural processing can be leveraged by energy-efficient hardware implementations, which can offer significant energy reductions as compared to conventional artificial neural networks (ANNs). The design of training algorithms for SNNs, however, lags behind hardware implementations: most existing training algorithms for SNNs have been designed either for biological plausibility or through conversion from pretrained ANNs via rate encoding.

This article provides an introduction to SNNs by focusing on a probabilistic signal processing methodology that enables the direct derivation of learning rules that leverage the unique time-encoding capabilities of SNNs. We adopt discrete-time probabilistic models for networked spiking neurons and derive supervised and unsupervised learning rules from first principles via variational inference. Examples and open research problems are also provided.

Introduction

ANNs have become the de facto standard tool to carry out supervised, unsupervised, and reinforcement learning tasks. Their recent successes range from image classifiers that outperform human experts in medical diagnosis to machines that defeat professional players at complex games, such as Go. These breakthroughs have built upon various algorithmic advances but have also heavily relied on the unprecedented availability of computing power and memory in data centers and cloud computing platforms. The resulting considerable energy requirements run counter to the constraints imposed by implementations on low-power mobile or embedded devices for such applications as personal health monitoring or neural prosthetics [1].

ANNs versus SNNs

Various new hardware solutions have recently emerged that attempt to improve the energy efficiency of ANNs as inference

machines by trading complexity for accuracy in the implementation of matrix operations. A different line of research, which is the subject of this article, seeks an alternative framework that enables efficient online inference and learning by taking inspiration from the working of the human brain.

The human brain is capable of performing general and complex tasks via continuous adaptation at a minute fraction of the power required by state-of-the-art supercomputers and ANN-based models [2]. Neurons in the human brain are qualitatively different from those in an ANN: they are dynamic devices featuring recurrent behavior, rather than static nonlinearities, and they process and communicate using sparse spiking signals over time, rather than real numbers. Inspired by this observation, as illustrated in Figure 1, SNNs have been introduced in the theoretical neuroscience literature as networks of dynamic spiking neurons [3]. SNNs have the unique capability to process information encoded in the timing of events, or spikes. Spikes are also used for synaptic communications, with synapses delaying and filtering signals before they reach the postsynaptic neuron. Because of the presence of synaptic delays, neurons in an SNN can be naturally connected via arbitrary recurrent topologies, unlike standard multilayer ANNs or chain-like recurrent neural networks.

Proof-of-concept and commercial hardware implementations of SNNs have demonstrated orders-of-magnitude improvements in terms of energy efficiency over ANNs [4]. Given the extremely low idle energy consumption, the energy spent by SNNs for learning and inference is essentially proportional to the number of spikes processed and communicated by the neurons, with the energy per spike being as low as a few picojoules [5].

Deterministic versus probabilistic SNN models

The most common SNN model consists of a network of neurons with deterministic dynamics whereby a spike is emitted as soon as an internal state variable, known as the *membrane potential*, crosses a given threshold value. A typical example is the leaky integrate-and-fire model, in which the membrane potential increases with each spike recorded in the incoming synapses while decreasing in the absence of inputs. When information is encoded in the rate of spiking of the neurons, an SNN can approximate the behavior of a conventional ANN with the same topology. This has motivated a popular line of work that aims at converting a pretrained ANN into a potentially more efficient SNN implementation (see [6] and the "Models" section for further details).

To make full use of the temporal processing capabilities of SNNs, learning problems should be formulated as the minimization of a loss function that directly accounts for the timing of the spikes emitted by the neurons. As for ANNs, this minimization can, in principle, be done using stochastic gradient descent (SGD). Unlike ANNs, however, this conventional approach is made challenging by the nondifferentiability of the output of the SNN with respect to the synaptic weights due to the threshold crossing-triggered behavior of spiking neurons. The potentially complex recurrent topology of SNNs also makes it difficult to implement the standard backpropagation procedure used in multilayer ANNs to compute gradients. To obviate this problem, a number of existing learning rules approximate the derivative by smoothing out the membrane potential as a function of the weights [7]–[9].

In contrast to deterministic models for SNNs, a probabilistic model defines the outputs of all spiking neurons as jointly distributed binary random processes. The joint distribution is differentiable in the synaptic weights, and, as a result, so are principled learning criteria from statistics and information theory, such as likelihood function and mutual information. The maximization of such criteria can apply to arbitrary topologies and does not require the implementation of backpropagation mechanisms. Hence, a stochastic viewpoint has significant analytic advantages, which translate into the derivation of flexible learning rules from first principles. These rules recover as special cases many known algorithms proposed for SNNs in the theoretical neuroscience literature as biologically plausible algorithms [10].

Scope and overview

This article aims to provide a review on the topic of probabilistic SNNs with a specific focus on the most commonly used generalized linear models (GLMs). We cover models, learning rules, and applications, highlighting principles and tools. The main goal is to make key ideas in this emerging field accessible to researchers in signal processing, who may otherwise find it difficult to navigate the theoretical neuroscience literature on the subject, given its focus on biological plausibility rather than

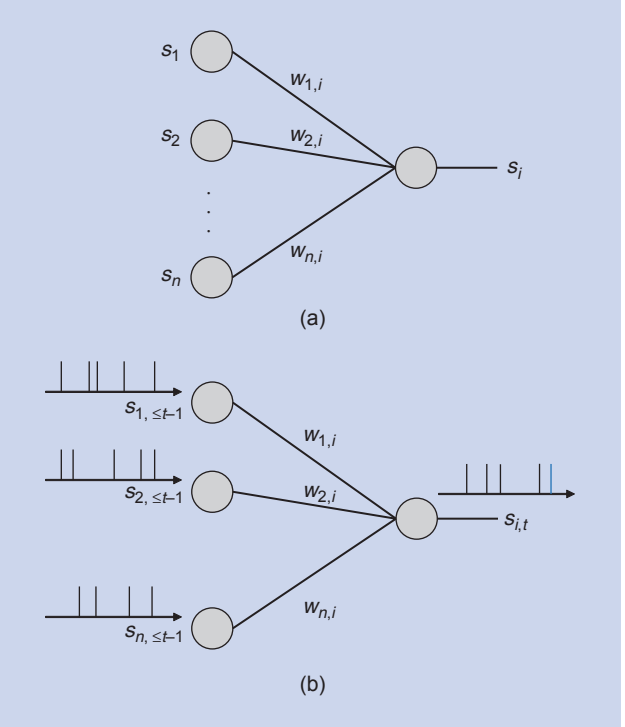

FIGURE 1. Illustrations of neural network models: (a) an ANN, where each neuron i processes real numbers $s_1, \ldots, s_n$ to output and communicates a real number s_i as a static nonlinearity, and (b) an SNN, where dynamic spiking neurons process and communicate sparse spiking signals over time t in a causal manner to output and communicate a binary spiking signal $s_{i,t}$.

theoretical and algorithmic principles [10]. At the end of the article, we also review alternative probabilistic formulations of SNNs, extensions, and open problems.

Learning tasks

An SNN is a network of spiking neurons. As seen in Figure 2, the input and output interfaces of an SNN typically transfer spiking signals. Input spiking signals can either be recorded directly from neuromorphic sensors, such as silicon cochleas and retinas [Figure 2(a)], or be converted from a natural signal to a set of spiking signals [Figure 2(b)]. Conversion can be done by following different rules, including rate encoding, whereby amplitudes are converted into the (instantaneous) spiking rate of a neuron; time encoding, in which amplitudes are translated into spike timings; and population coding, whereby amplitudes are encoded into the (instantaneous) firing rates [11] or relative firing times of a subset of neurons (see [10] for a review). In a similar manner, output spiking signals can either be fed directly to a neuromorphic actuator, such as neuromorphic controllers or prosthetic systems [Figure 2(a)], or be converted from spiking signals to natural signals [Figure 2(b)]. This can be done by following rate, time, or population decoding principles.

The SNN generally acts as a dynamic mapping between inputs and outputs that is defined by the model parameters, including, most notably, the interneuron synaptic weights. This mapping can be designed or trained to carry out inference or control tasks. When training is enabled, the model parameters are automatically adapted based on data fed to the network, with the goal of maximizing a given performance criterion. Training can be carried out in a supervised, unsupervised, or reinforcement learning manner, depending on the availability of data and feedback signals, as further discussed subsequently. For both inference/control and training, data can be presented to the SNN in a batch mode (also known as a *frame-based mode*) or in an *online mode* (see the "Training SNNs" section).

With supervised learning, the training data specify both the input and desired output. Input and output pairs are either in the form of a number of separate examples, in the case of batch learning, or presented over time in a streaming fashion for online learning. As an example, the training set may include a number of spike-encoded images and corresponding correct labels, or a single time sequence to be used to extrapolate predictions (see also the "Batch Learning Examples" and "Online Learning Examples" sections). Under unsupervised learning, the training data specify only the desired input or output to the SNN, which can again be presented in a batch or online fashion. Examples of applications include representation learning, which aims to translate the input into a more compact, interpretable, or useful representation, and generative modeling, which seeks to generate outputs with statistics akin to the training data (see, e.g., [12]). Finally, with reinforcement learning, the SNN is used to control an agent on the basis of input observations from the environment to accomplish a given goal. To this end, the SNN is provided with feedback on the selected outputs that guides the SNN in updating its parameters in a batch or online manner [13].

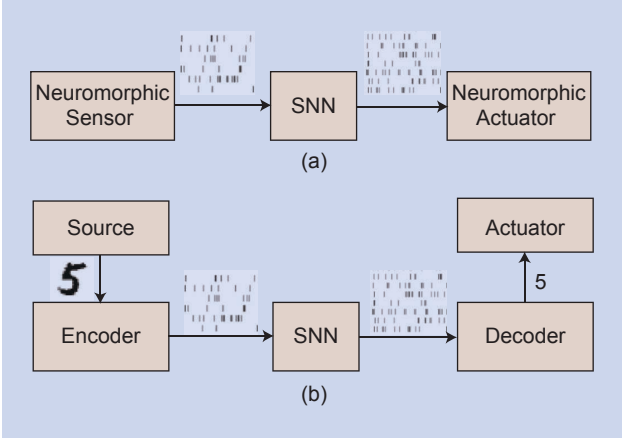

FIGURE 2. Depictions of the input/output interfaces of an SNN: (a) a direct interface with a neuromorphic sensor and actuator and (b) an indirect interface through encoding and decoding.

Models

Here, we describe the standard discrete-time GLM for SNNs, also known as the *spike response model with escape noise* (see, e.g., [14] and [15]). Discrete-time models reflect the operation of a number of neuromorphic chips, including Intel's Loihi [4], while continuous-time models are more commonly encountered in the computer neuroscience literature [10].

Graphical representation

As illustrated in Figure 3, an SNN consists of a network of N spiking neurons. At any time $t = 0, 1, 2, \ldots$, each neuron i outputs a binary signal $s_{i,t} \in \{0, 1\}$, with value $s_{i,t} = 1$ corresponding to a spike emitted at time t. We collect in vector $\boldsymbol{s}_t = (s_{i,t} : i \in \mathcal{V})$ the binary signals emitted by all neurons at time t, where $\mathcal{V}$ is the set of all neurons. Each neuron $i \in \mathcal{V}$ receives the signals emitted by a subset $\mathcal{P}_i$ of neurons through directed links, known as *synapses*. Neurons in set $\mathcal{P}_i$ are referred to as *presynaptic* for *postsynaptic* neuron i.

Membrane potential and filtered traces

The internal, analog state of each spiking neuron $i \in \mathcal{V}$ at time t is defined by its membrane potential $u_{i,t}$ (and possibly also by other secondary variables to be discussed) [15]. The value of the membrane potential indicates the probability of neuron i to spike. As illustrated in Figure 4, the membrane potential is the sum of the contributions from the incoming spikes of the presynaptic neurons and from the past spiking behavior of the neuron itself, where both contributions are filtered by the respective kernels a_t and b_t. To elaborate, we denote as $\boldsymbol{s}_{i,\leq t} = (s_{i,0}, \ldots, s_{i,t})$ the spike signal emitted by neuron i up to time t. Given past input spike signals from the presynaptic neurons $\mathcal{P}_i$, denoted as $\boldsymbol{s}_{\mathcal{P}_i,\leq t-1} = \{\boldsymbol{s}_{j,\leq t-1}\}_{j\in\mathcal{P}_i}$, and the local spiking history $\boldsymbol{s}_{i,\leq t-1}$, the membrane potential of postsynaptic neuron i at time t can be written as [15]

$$u_{i,t} = \sum_{j\in\mathcal{P}_i} w_{j,i}\vec{s}_{j,t-1} + w_i\overleftarrow{s}_{i,t-1} + \gamma_i, \tag{1}$$

where the quantities $w_{j,i}$ for $j \in \mathcal{P}_i$ are synaptic (feedforward) weights, w_i is a feedback weight, γ_i is a bias parameter, and the quantities

$$\vec{s}_{i,t} = a_t * s_{i,t} \quad \text{and} \quad \overleftarrow{s}_{i,t} = b_t * s_{i,t} \tag{2}$$

are known as *filtered feedforward* and *feedback traces* of neuron i, respectively, where $*$ denotes the convolution operator $f_t * g_t = \Sigma_{\delta \geq 0} f_\delta g_{t-\delta}$.

Kernels and model weights

In (1) and (2), the filter a_t defines the synaptic response to a spike from a presynaptic neuron at the postsynaptic neuron. This filter is known as the *feedforward*, or *synaptic*, *kernel*. The filtered contribution of a spike from the presynaptic neuron $j \in \mathcal{P}_i$ is multiplied by a learnable weight $w_{j,i}$ for the synapse from neuron j to neuron $i \in \mathcal{V}$. When the filter is of finite duration τ, computing the feedforward trace $\vec{s}_{i,t}$ requires keeping track of the window $\{s_{i,t}, s_{i,t-1}, \ldots, s_{i,t-(\tau-1)}\}$ of prior synaptic inputs as part of the neuron's state [16]. An example is given by the "alpha" function $a_t = (\exp(-t/\tau_1) - \exp(-t/\tau_2))$ for $t = 0, \ldots, \tau - 1$ and zero otherwise, with time constants τ_1 and τ_2 and duration τ, as illustrated in Figure 5(a). When the kernel is chosen as an infinitely long decaying exponential, i.e., as $a_t = \exp(-t/\tau_1)$, the feedforward trace $\vec{s}_{i,t}$ can be directly computed using an autoregressive update that requires the storage of only a single scalar variable in the neuron's state [16], i.e., $\vec{s}_{i,t} = \exp(-1/\tau_1)(\vec{s}_{i,t-1} + s_{i,t})$. In general, the time constants and kernel shapes determine the synaptic memory and synaptic delays.

The filter b_t describes the response of a neuron to a local output spike and is known as a *feedback kernel*. A negative feedback kernel, such as $b_t = -\exp(-t/\tau_m)$, with time constant τ_m [see Figure 5(b)], models the refractory period upon the emission of a spike, with the time constant of the feedback kernel determining the duration of the refractory period. As per (1), the filtered contribution of a local output spike is weighted by a learnable parameter w_i. Similar considerations as for the feedforward traces apply regarding the computation of the feedback trace.

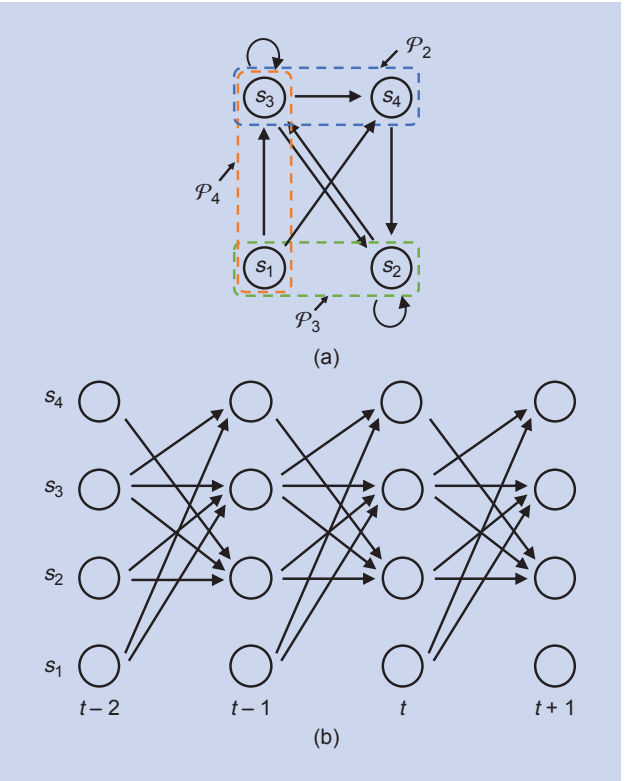

FIGURE 3. (a) An architecture of an SNN with $N = 4$ spiking neurons. The directed links between two neurons represent causal feedforward, or synaptic, dependencies, while the self-loop links represent feedback dependencies. The directed graph may have loops, including self-loops, indicating recurrent behavior. (b) A time-expanded view of the temporal dependencies implied by (a) with synaptic and feedback memories equal to one time step.

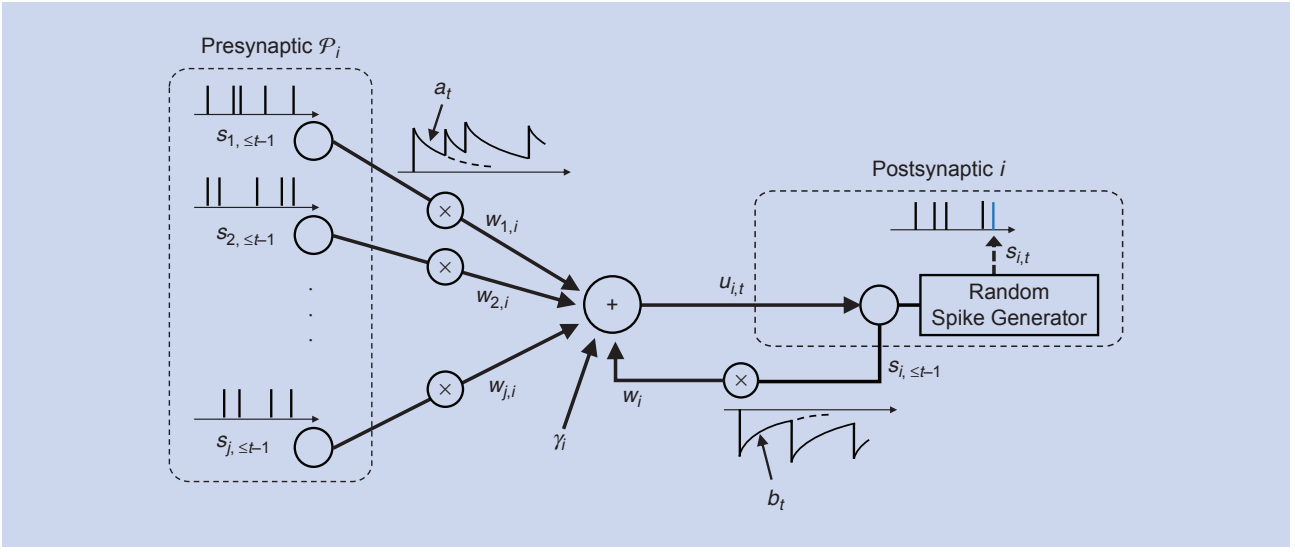

FIGURE 4. An illustration of the membrane potential model, with exponential feedforward and feedback kernels (see also Figure 5).

Generalizing the model described previously, a synapse can be associated with K_a learnable synaptic weights $\{w_{j,i,k}\}_{k=1}^{K_a}$. In this case, the contribution from presynaptic neuron j in (1) can be written as [14]

$$\left(\sum_{k=1}^{K_a} w_{j,i,k} a_{k,t}\right) * s_{j,t}, \tag{3}$$

where we have defined K_a fixed basis functions $\{a_{k,t}\}_{k=1}^{K_a}$, with learnable weights $\{w_{j,i,k}\}_{k=1}^{K_a}$. The feedback kernel can be similarly parameterized as the weighted sum of fixed K_b basis functions. Parameterization (3) makes it possible to adapt the shape of the filter applied by the synapse by learning the weights $\{w_{j,i,k}\}_{k=1}^{K_a}$. Typical examples of basis functions are the raised cosine functions shown in Figure 5(c). With this choice, the system can learn the sensitivity of each synapse to different synaptic delays, each corresponding to a different basis function, by adapting the weights $\{w_{j,i,k}\}_{k=1}^{K_a}$. In the rest of this article, with the exception of the "Batch Learning Examples" and "Online Learning Examples" sections, we focus on the simpler model of (1) and (2).

Practical implementations of the membrane potential model (1) can leverage the fact that linear filtering of binary spiking signals requires only carrying out sums while doing away with the need to compute expensive floating-point multiplications [5].

GLM

As discussed, a probabilistic model defines the joint probability distribution of the spike signals emitted by all neurons. In general, with the notation $\boldsymbol{s}_{\le t} = (\boldsymbol{s}_0, \ldots, \boldsymbol{s}_t)$ using the chain rule, the log probability of the spike signals $\boldsymbol{s}_{\le T} = (\boldsymbol{s}_0, \ldots, \boldsymbol{s}_T)$ emitted by all neurons in the SNN up to time T can be written as

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\boldsymbol{s}_{\le T}) &= \sum_{t=0}^{T} \log p_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{s}_{\le t-1}) \\ &= \sum_{t=0}^{T} \sum_{i \in \mathcal{V}} \log p_{\boldsymbol{\theta}_i}(s_{i,t} \mid \boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \le t-1}),\end{aligned} \tag{4}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_i\}_{i \in \mathcal{V}}$ is the learnable parameter vector, with $\boldsymbol{\theta}_i$ being the local parameters of neuron i. The decomposition (4) is in terms of the conditional probabilities $p_{\boldsymbol{\theta}_i}(s_{i,t} \mid \boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \le t-1})$, which represent the spiking probability of neuron i at time t, given its past spike timings and the past behaviors of its presynaptic neurons $\mathcal{P}_i$.

Under the GLM, the dependency of the spiking behavior of neuron $i \in \mathcal{V}$ on the history $\boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \le t-1}$ is mediated by the neuron's membrane potential $u_{i,t}$. Specifically, the instantaneous firing probability of neuron i at time t is equal to

$$p_{\boldsymbol{\theta}_i}(s_{i,t} = 1 \mid \boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \le t-1}) = p(s_{i,t} = 1 \mid u_{i,t}) = \sigma(u_{i,t}), \tag{5}$$

with $\sigma(\cdot)$ being the sigmoid function, i.e., $\sigma(x) = 1/(1+\exp(-x))$. According to (5), a larger potential $u_{i,t}$ increases the probability that neuron i spikes. The model (5) is parameterized by the local learnable vector $\boldsymbol{\theta}_i = \{\gamma_i, \{w_{j,i}\}_{j \in \mathcal{P}_i}, w_i\}$ of neuron i. SNNs modeled according to the described GLM framework can be thought of as a generalization of dynamic models of belief networks [17], and they can also be interpreted as a discrete-time version of Hawkes processes [18].

In a variant of this model, probability (5) can be written as $\sigma(u_{i,t}/\Delta u)$, where Δu is a bandwidth parameter that dictates the smoothness of the firing rate about the threshold. When taking the limit $\Delta u \to 0$, we obtain the deterministic integrate-and-fire model [19].

Relationship with ANNs

Under rate encoding, as long as the duration T is large enough, the deterministic integrate-and-fire model can mimic the operation of a conventional feedforward ANN with a nonnegative activation function. To this end, consider an ANN with an arbitrary topology defined by an acyclic directed graph. The corresponding SNN has the same topology, a feedforward kernel defined by a single basis function implementing a perfect integrator (i.e., a filter with a constant impulse response), the same synaptic weights of the ANN, and no feedback kernel. In this way, the value of the filtered feedforward trace for each synapse approximates the spiking rate of the presynaptic neuron as T increases. The challenge in enabling a conversion from ANN to SNN is to choose the thresholds γ_i and possibly a renormalization of the weights, so that the spiking rates of all neurons in the SNN approximate the outputs of the neurons in

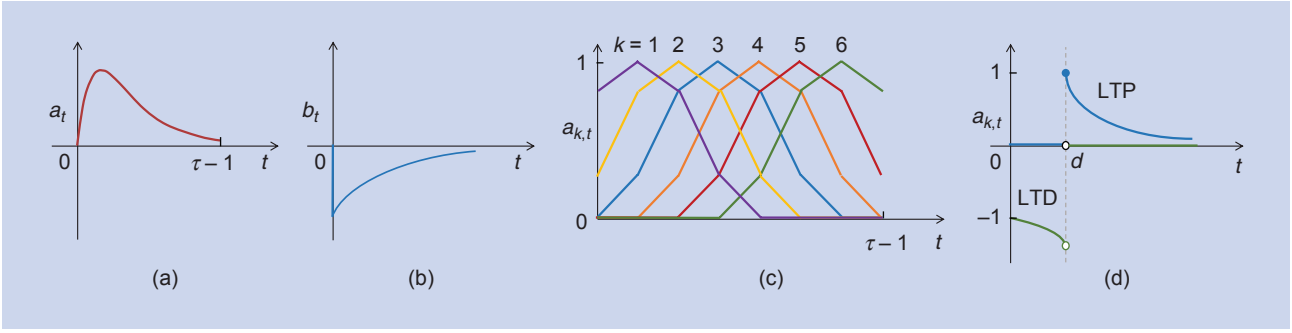

FIGURE 5. Examples of feedforward/feedback kernels: (a) an exponentially decaying feedforward kernel a_t, (b) an exponentially decaying feedback kernel b_t, (c) raised cosine basis functions $a_{k,t}$ in [14], and (d) spike-timing-dependent plasticity basis functions $a_{k,t}$ for long-term potentiation (LTP) and long-term depression (LTD), where the synaptic conduction delay equals d [16].

the ANN [6]. When including loops, deterministic SNNs can also implement recurrent NNs [20].

Gradient of the log-likelihood

The gradient of the log probability, or log-likelihood, $\mathcal{L}_{s\leq T}(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{s}_{\leq T})$ in (4), with respect to the learnable parameters $\boldsymbol{\theta}$, plays a key role in the problem of training a probabilistic SNN. Focusing on any neuron $i \in \mathcal{V}$, from (1) to (5), the gradient of the log-likelihood with respect to the local parameters $\boldsymbol{\theta}_i$ for neuron i is given as

$$\nabla_{\boldsymbol{\theta}_i} \mathcal{L}_{s\leq T}(\boldsymbol{\theta}) = \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}_i} \log p_{\boldsymbol{\theta}_i}(s_{i,t} \,|\, \boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \leq t-1}), \tag{6}$$

where the individual entries of the gradient of time t can be obtained as

$$\nabla_{\gamma_i} \log p_{\boldsymbol{\theta}_i}(s_{i,t} \,|\, \boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \leq t-1}) = s_{i,t} - \sigma(u_{i,t}), \tag{7a}$$

$$\nabla_{w_{j,i}} \log p_{\boldsymbol{\theta}_i}(s_{i,t} \,|\, \boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \leq t-1}) = \vec{s}_{j,t-1}(s_{i,t} - \sigma(u_{i,t})), \tag{7b}$$

and

$$\nabla_{w_i} \log p_{\boldsymbol{\theta}_i}(s_{i,t} \,|\, \boldsymbol{s}_{\mathcal{P}_i \cup \{i\}, \leq t-1}) = \overleftarrow{s}_{i,t-1}(s_{i,t} - \sigma(u_{i,t})). \tag{7c}$$

The gradients (7) depend on the difference between the desired spiking behavior and its average behavior under the model distribution (5). The implications of this result for learning will be discussed in the next sections.

Training SNNs

SNNs can be trained using supervised, unsupervised, and reinforcement learning. To this end, the network follows a learning rule, which defines how the model parameters $\boldsymbol{\theta}$ are updated on the basis of the available observations. As we will detail, learning rules can be applied in a batch mode at the end of a full period T of use of the SNN, based on multiple observations of duration T, or in an online fashion, i.e., after each time instant t, based on an arbitrarily long observation.

Locality

A learning rule is local if its operation can be decomposed into atomic steps that can be carried out in parallel at distributed processors based only on locally available information and limited communication on the connectivity graph (see Figure 3). Local information at a neuron includes the membrane potential, the feedforward filtered traces for the incoming synapses, the local feedback filtered trace, and the local model parameters. The processors will be considered here to be conventionally implemented at the level of individual neurons. Besides local signals, learning rules may also require global feedback signals, as discussed next.

Three-factor rule

While the details differ for each learning rule and task, a general form of the learning rule for the synaptic weights follows the three-factor rule [21], [22]. Accordingly, the synaptic weight $w_{j,i}$ from presynaptic neuron $j \in \mathcal{P}_i$ to a postsynaptic neuron $i \in \mathcal{V}$ is updated as

$$w_{j,i} \leftarrow w_{j,i} + \eta \times \ell \times \text{pre}_j \times \text{post}_i, \tag{8}$$

where η is a learning rate, ℓ is a scalar global learning signal that determines the sign and magnitude of the update, pre_j is a function of the activity of the presynaptic neuron $j \in \mathcal{P}_i$, and post_i depends on the activity of the postsynaptic neuron $i \in \mathcal{V}$. For most learning rules, pre- and postsynaptic terms are local to each neuron, while the learning signal ℓ, if present, plays the role of a global feedback signal. As a special case, the rule (8) can implement Hebb's hypothesis that "neurons that spike together wire together." This is indeed the case if the product of the pre_j and post_i terms is large when the two neurons spike at nearly the same time, resulting in a large change of the synaptic weight $w_{j,i}$ [10].

In the next two sections, we will see how learning rules of the form (8) can be derived in a principled manner as SGD updates obtained under the described probabilistic SNN models.

Training SNNs: Fully observed models

Fully observed versus partially observed models

Neurons in an SNN can be divided into the subsets of visible, or observed, neurons, which encode inputs and outputs, and hidden, or latent, neurons, whose role is to facilitate the desired behavior of the SNN. During training, the behavior of visible neurons is specified by the training data. For example, under supervised learning, input neurons are clamped to the input data, while the spiking signals of output neurons are determined by the desired output. Another related example is a reinforcement learning task in which the SNN models a policy, with input neurons encoding the state and output neurons encoding the action previously taken by the learner in response to the given input [23].

In the case of fully observed models, the SNN contains only visible neurons while, in the case of partially observed models, the SNN also includes hidden neurons. We first consider the simpler former case and then extend the discussion to partially observed models.

Maximum likelihood learning via SGD

The standard training criterion for probabilistic models for both supervised and unsupervised learning is maximum likelihood (ML). ML selects model parameters that maximize the probability of the observed data and, hence, of the desired input/output behavior under the model. To elaborate, we consider an example $\boldsymbol{x}_{\leq T}$ consisting of fully observed spike signals for all neurons in the SNN, including both input and output neurons. Using the notation in the "Models" section, we hence have $\boldsymbol{s}_{\leq T} = \boldsymbol{x}_{\leq T}$. During training, the spike signals for all neurons are thus clamped to the values assumed in the data point $x_{\leq T}$, and the log-likelihood is given as $\mathcal{L}_{x\leq T}(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T})$

in (4), with $\boldsymbol{s}_{\leq T} = \boldsymbol{x}_{\leq T}$. As we will see next, for batch learning, there are multiple such examples $\boldsymbol{x}_{\leq T}$ in the training set while, for online learning, we have a single arbitrary long example $\boldsymbol{x}_{\leq T}$ for large T.

Batch SGD

In the batch training mode, a training set $\mathcal{D} = \{\boldsymbol{x}_{\leq T}^{m}\}_{m=1}^{M}$ of M fully observed examples is available to enable the learning of the model parameters. The batch SGD-based rule proceeds iteratively by selecting an example $\boldsymbol{x}_{\leq T}$ from the training set $\mathcal{D}$ at each iteration (see, e.g., [24]). The model parameters $\boldsymbol{\theta}$ are then updated in the direction of the gradient (6) and (7), with $\boldsymbol{s}_{\leq T} = \boldsymbol{x}_{\leq T}$, as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}), \tag{9}$$

where the learning rate η is assumed to be fixed here for simplicity of notation. Note that the update (9) is applied at the end of the observation period T. The batch algorithm can be generalized by summing over a minibatch of examples at each iteration [24].

Online SGD

In the online training mode, an arbitrary long example $\boldsymbol{x}_{\leq T}$ is available, and the model parameters $\boldsymbol{\theta}$ are updated at each time t (or, more generally, periodically every few time instants). This can be done by introducing an eligibility trace $\boldsymbol{e}_{i,t}$ for each neuron i [19], [22]. As summarized in Algorithm 1, the eligibility trace $\boldsymbol{e}_{i,t}$ in (A1), with $\kappa < 1$, computes a weighted average of current and past gradient updates. In this update, the current gradient $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_t \,|\, \boldsymbol{x}_{\leq t-1})$ is weighted by a factor $(1-\kappa)$, and the gradient that is evaluated l steps in the past is multiplied by the exponentially decaying coefficient $(1-\kappa)\cdot\kappa^{l}$. The eligibility trace captures the impact of past updates on the current spiking behavior, and it can help stabilize the online training by reducing the variance of the updates (for sufficiently large κ) [13].

Algorithm 1. ML training via online SGD.

Input: Training example $\boldsymbol{x}_{\leq T}$ and learning rates η and κ
Output: Learned model parameters $\boldsymbol{\theta}$

initialize parameters $\boldsymbol{\theta}$
repeat
  for each $t = 0, 1, \ldots, T$
  for *each neuron* $i \in \mathcal{V}$ **do**
    compute the gradient $\nabla_{\theta_i} \log p_{\boldsymbol{\theta}_i}(x_{i,t} | \boldsymbol{x}_{\mathcal{P}_i \cup \{i\}, \leq t-1})$ with respect to the local parameters $\boldsymbol{\theta}_i$ from (7)
    compute the eligibility trace $\boldsymbol{e}_{i,t}$

$$\boldsymbol{e}_{i,t} = \kappa \boldsymbol{e}_{i,t-1} + (1-\kappa) \nabla_{\boldsymbol{\theta}_i} \log p_{\boldsymbol{\theta}_i}(x_{i,t} | \boldsymbol{x}_{\mathcal{P}_i \cup \{i\}, \leq t-1}) \tag{A1}$$

    update the local model parameters

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + \eta \boldsymbol{e}_{i,t} \tag{A2}$$

  end
until *stopping criterion is satisfied.*

Interpretation

The online gradient update for any synaptic weight $w_{j,i}$ can be interpreted in light of the general form of rule (8). In fact, the gradient (7b) has a two-factor form, whereby the global learning signal is absent; the presynaptic term is given by the filtered feedforward trace $\vec{x}_{j,t-1}$ of the presynaptic neuron $j \in \mathcal{P}_i$, and the postsynaptic term is given by the error term $x_{i,t} - \sigma(u_{i,t})$. This error measures the difference between the desired spiking behavior of the postsynaptic neuron i at any time t and its average behavior under the model distribution (5).

This update can be related to the standard spike-timing-dependent plasticity (STDP) rule [10], [16], [25]. In fact, STDP stipulates that the long-term potentiation (LTP) of a synapse occurs when the presynaptic neuron spikes right before a postsynaptic neuron, while long-term depression (LTD) of a synapse takes place when the presynaptic neuron spikes right after a postsynaptic neuron. With the basis functions depicted in Figure 5(d), if a presynaptic spike occurs more than d steps prior to the postsynaptic spike at time t, an increase in the synaptic weight, or LTP, occurs, while a decrease in the synaptic weight, or LTD, takes place otherwise [16]. The parameter d can hence be interpreted as synaptic delay.

As for the synaptic weights, all other gradients (7) also depend on an error signal measuring the gap between the desired and average model behavior. In (7a)–(7c), the desired behavior is given by samples $s_{i,t} = x_{i,t}$ in the training example. The contribution of this error signal can be interpreted as a form of (task-specific) homeostatic plasticity, in that it regulates the neuronal firing rates around desirable set-point values [10], [26].

Locality and implementation

Given the absence of a global learning signal, the online SGD rule in Algorithm 1 and the batch SGD rule can be implemented locally, so that each neuron i updates its own local parameters $\boldsymbol{\theta}_i$. Each neuron i uses information about the local spike signal $x_{i,t}$, the feedforward filtered traces $\vec{x}_{j,t-1}$ for all presynaptic neurons $j \in \mathcal{P}_i$, and the local feedback filtered trace $\overleftarrow{x}_{i,t-1}$ to compute the first terms in (7a)–(7c), while the second terms in (7a)–(7c) are obtained from (5) by using the neuron's membrane potential $u_{i,t}$.

Training SNNs: Partially observed models

Latent neurons

As mentioned previously, the set $\mathcal{V}$ of neurons can be partitioned into the disjoint subsets of observed (input and output) and hidden neurons. The N_{X} neurons in the subset $\mathcal{X}$ are observed, and the N_{H} neurons in the subset $\mathcal{H}$ are hidden, or latent, and we have $\mathcal{V} = \mathcal{X} \cup \mathcal{H}$. We write as $\boldsymbol{x}_t = (x_{i,t} : i \in \mathcal{X})$ and $\boldsymbol{h}_t = (h_{i,t} : i \in \mathcal{H})$, the binary signals emitted by the observed and hidden neurons at time t, respectively. Therefore, using the notation in the "Models" section, we have $s_{i,t} = x_{i,t}$ for any observed neuron $i \in \mathcal{X}$ and $s_{i,t} = h_{i,t}$ for any latent neuron $i \in \mathcal{H}$ as well as $\boldsymbol{s}_t = (\boldsymbol{x}_t, \boldsymbol{h}_t)$ for the overall set of spike signals at time t. During training, the spike signals $\boldsymbol{x}_{\leq T}$ of the

observed neurons are clamped to the examples in the training set while the probability distribution of the signals $\boldsymbol{h}_{\leq T}$ of the hidden neurons can be adapted to ensure the desired input/output behavior. Mathematically, the probabilistic model is defined as in (4) and (5), with $\boldsymbol{s}_{\leq T} = (\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T})$.

ML via SGD and variational learning

Here, we review a standard learning rule that tackles the ML problem by using SGD. Unlike in the fully observed case, as we will see, variational inference is needed to cope with the complexity of computing the gradient of the log-likelihood of the observed spike signals in the presence of hidden neurons [12].

Log-likelihood

The log-likelihood of an example of observed spike signals $\boldsymbol{x}_{\leq T}$ is obtained via marginalization by summing over all possible values of the latent spike signals $\boldsymbol{h}_{\leq T}$ as $\mathcal{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}) = \log \sum_{\boldsymbol{h}_{\leq T}} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T})$. Let us denote as $\langle \cdot \rangle_p$ the expectation over a distribution p, as in $\langle f(x) \rangle_{p(x)} = \sum_x f(x)p(x)$, for some function $f(x)$. The gradient of the log-likelihood with respect to the model parameters $\boldsymbol{\theta}$ can be expressed as (see, e.g., [12, Ch. 6])

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}) = \langle \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) \rangle_{p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})}, \tag{10}$$

where the expectation is with respect to the posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$ of the latent variables $\boldsymbol{h}_{\leq T}$, given the observation $\boldsymbol{x}_{\leq T}$. Note that the gradient $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x_{\leq T}, h_{\leq T}) = \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{h}_t | \boldsymbol{x}_{\leq t-1}, \boldsymbol{h}_{\leq t-1})$ is obtained from (7), with $\boldsymbol{s}_{\leq T} = (\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T})$. Computing the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$ amounts to the Bayesian inference of the hidden spike signals for the observed values $\boldsymbol{x}_{\leq T}$. Given that we have the equality $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) / p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T})$, this task requires the evaluation of the marginal distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}) = \Sigma_{\boldsymbol{h}_{\leq T}} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T})$. For problems of practical size, this computation is intractable, and, hence, so is evaluating the gradient (10).

Variational learning

Variational inference, or variational Bayes, approximates the true posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$ by means of any arbitrary variational posterior distribution $q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$ parameterized by a vector $\boldsymbol{\phi}$ of learnable parameters. For any variational distribution $q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$, using Jensen's inequality, the log-likelihood $\mathcal{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta})$ can be lower-bounded as (see, e.g., [12, Ch. 6 and Ch. 8])

$$\begin{aligned} \mathcal{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}) &= \log \sum_{\boldsymbol{h}_{\leq T}} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) \\ &\geq \sum_{\boldsymbol{h}_{\leq T}} q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T})}{q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})} \\ &= \langle \ell_{\boldsymbol{\theta}, \boldsymbol{\phi}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) \rangle_{q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})} \coloneqq L_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi}), \end{aligned} \tag{11}$$

where we have defined the learning signal as

$$\ell_{\boldsymbol{\theta}, \boldsymbol{\phi}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) \coloneqq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T}). \tag{12}$$

A baseline variational learning rule, also known as the *variational expectation maximization* algorithm, is based on the maximization of the evidence lower bound (ELBO) $L_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ in (11) with respect to both the model parameters $\boldsymbol{\theta}$ and the variational parameters $\boldsymbol{\phi}$. Accordingly, for a given observed example $\boldsymbol{x}_{\leq T} \in \mathcal{D}$, the learning rule is given by gradient ascent updates, where the gradients can be computed as

$$\nabla_{\boldsymbol{\theta}} L_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \langle \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) \rangle_{q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})}, \tag{13a}$$

and

$$\begin{aligned} &\nabla_{\boldsymbol{\phi}} L_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \\ &\quad \langle \ell_{\boldsymbol{\theta}, \boldsymbol{\phi}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) \cdot \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T}) \rangle_{q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})}, \end{aligned} \tag{13b}$$

respectively. The gradient (13a) is derived in a manner analogous to (10), and the gradient (13b) is obtained from the standard REINFORCE, or score function, gradient [12, Ch. 8], [27]. Importantly, the gradients (13) require expectations with respect to the known variational posterior $q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$ evaluated at the current value of variational parameters $\boldsymbol{\phi}$ rather than with respect to the hard-to-compute posterior $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$. An alternative to the computation of the gradient over the variational parameters $\boldsymbol{\phi}$ as in (13b) is given by the so-called reparameterization trick [28], as briefly discussed in the "Conclusions and Open Problems" section.

In practice, computing the averages in (13) is still intractable because of the large domain of the hidden variables $\boldsymbol{h}_{\leq T}$. Therefore, the expectations over the variational posterior are typically approximated by means of Monte Carlo empirical averages. This is possible as long as sampling from the variational posterior $q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$ is feasible. As an example, if a single spike signal $\boldsymbol{h}_{\leq T}$ is sampled from $q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T})$, we obtain the Monte Carlo approximations of (13) as

$$\nabla_{\boldsymbol{\theta}} \hat{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}), \tag{14a}$$

and

$$\nabla_{\boldsymbol{\phi}} \hat{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \ell_{\boldsymbol{\theta}, \boldsymbol{\phi}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) \cdot \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_{\leq T} | \boldsymbol{x}_{\leq T}). \tag{14b}$$

Batch doubly SGD

In a batch training formulation, at each iteration, an example $\boldsymbol{x}_{\leq T}$ is selected from the training set $\mathcal{D}$. At the end of the observation period T, both model and variational parameters can be updated in the direction of the gradients $\nabla_{\theta} \hat{L}_{\boldsymbol{x}_{\leq T}}(\theta, \phi)$ and $\nabla_{\phi} \hat{L}_{\boldsymbol{x}_{\leq T}}(\theta, \phi)$ in (14) as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \hat{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi}), \tag{15a}$$

and

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \eta_{\boldsymbol{\phi}} \nabla_{\boldsymbol{\phi}} \hat{L}_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \boldsymbol{\phi}), \tag{15b}$$

respectively, where the learning rates $\eta_{\boldsymbol{\theta}}$ and $\eta_{\boldsymbol{\phi}}$ are assumed to be fixed for simplicity. Rule (15) is known as *doubly SGD*

since sampling is carried out over both the observed examples $\boldsymbol{x}_{\leq T}$ in the training set and the hidden spike signals $\boldsymbol{h}_{\leq T}$.

The doubly stochastic gradient estimator (14b) typically exhibits a high variance. To reduce the variance, a common approach is to subtract a baseline control variate from the learning signal. This can be done by replacing the learning signal in (14b) with the centered learning signal $\ell_{\boldsymbol{\theta},\phi}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) - \bar{\ell}$, where the baseline $\bar{\ell}$ is calculated as a moving average of learning signals computed at previous iterations [22], [27], [29].

Online doubly SGD

The batch doubly SGD rule (15) applies with any choice of variational distribution $q_\phi(\boldsymbol{h}_{\leq T}|\boldsymbol{x}_{\leq T})$, as long as it is feasible to sample from it and to compute the gradient in (14b). However, the locality properties and complexity of the learning rule are strongly dependent on the choice of the variational distribution. We now discuss a specific choice considered in [16], [22], and [29]–[31] that yields an online rule, summarized in Algorithm 2.

The approach approximates the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T}|\boldsymbol{x}_{\leq T})$ with a feedforward distribution that ignores the stochastic dependence of the hidden spike signals $\boldsymbol{h}_t$ at time t on the future values of the observed spike signals $\boldsymbol{x}_{\leq T}$. The corresponding variational distribution can be written as

$$q_{\boldsymbol{\theta}^{\mathrm{H}}}(\boldsymbol{h}_{\leq T}|\boldsymbol{x}_{\leq T}) = \prod_{t=0}^{T} p_{\boldsymbol{\theta}^{\mathrm{H}}}(\boldsymbol{h}_t|\boldsymbol{x}_{\leq t-1}, \boldsymbol{h}_{\leq t-1}) = \prod_{t=0}^{T} \prod_{i\in\mathcal{H}} p(h_{i,t}|u_{i,t}), \tag{16}$$

Algorithm 2. ML training via online doubly SGD.

Input: Training data $\boldsymbol{x}_{\leq T}$ and learning rates η and κ
Output: Learned model parameters $\boldsymbol{\theta}$

initialize parameters $\boldsymbol{\theta}$
repeat
  feedforward sampling:
  for each hidden neuron $i \in \mathcal{H}$ **do**
    emit a spike $h_{i,t} = 1$ with probability $\sigma(u_{i,t})$
  end
  global feedback:
    a central processor collects the log probabilities $p(x_{i,t}|u_{i,t})$ in (5) from all observed neurons $i \in \mathcal{X}$, computes an eligibility trace from the learning signal (17) as

$$\ell_t = \kappa \ell_{t-1} + (1-\kappa) \sum_{i\in\mathcal{X}} \log p(x_{i,t}|u_{i,t}), \tag{A3}$$

    and feeds back the global learning signal ℓ_t to all latent neurons
  parameter update:
  for each neuron $i \in \mathcal{V}$ **do**
    evaluate the eligibility trace $\boldsymbol{e}_{i,t}$ as

$$\boldsymbol{e}_{i,t} = \kappa \boldsymbol{e}_{i,t-1} + (1-\kappa) \nabla_{\boldsymbol{\theta}_i} \log p_{\boldsymbol{\theta}_i}(s_{i,t}|\boldsymbol{x}_{\leq t-1}\boldsymbol{h}_{\leq t-1}), \tag{A4}$$

    with $s_{i,t} = x_{i,t}$ if $i \in \mathcal{X}$ and $s_{i,t} = h_{i,t}$ if $i \in \mathcal{H}$
    update the local model parameters as

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + \eta \cdot \begin{cases} \boldsymbol{e}_{i,t}, & \text{if } i \in \mathcal{X} \\ \ell_t \boldsymbol{e}_{i,t}, & \text{if } i \in \mathcal{H} \end{cases} \tag{A5}$$

  end
until *stopping criterion is satisfied.*

where we denote as $\boldsymbol{\theta}^{\mathrm{H}} = \{\boldsymbol{\theta}_i\}_{i\in\mathcal{H}}$ the collection of the model parameters for hidden neurons, and $p(h_{i,t} = 1|u_{i,t}) = \sigma(u_{i,t})$ by (5), with $s_{i,t} = h_{i,t}$. We note that (16) is an approximation of the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T}|\boldsymbol{x}_{\leq T}) = \prod_{t=0}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{h}_t|\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq t-1})$ since it neglects the correlation between variables $\boldsymbol{h}_t$ and the future observed samples $\boldsymbol{x}_{\geq t}$. In (16), we have emphasized that the variational parameters ϕ are tied to a subset of the model parameters, as per the equality $\phi = \boldsymbol{\theta}^{\mathrm{H}}$. As a result, this choice of variational distribution does not include additional learnable parameters apart from the model parameters $\boldsymbol{\theta}$. The learning signal (12) with the feedforward distribution (16) reads

$$\begin{aligned} \ell_{\boldsymbol{\theta}^{\mathrm{X}}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) &= \sum_{t=0}^{T} \log p_{\boldsymbol{\theta}^{\mathrm{X}}}(\boldsymbol{x}_t|\boldsymbol{x}_{\leq t-1}, \boldsymbol{h}_{\leq t-1}) \\ &= \sum_{t=0}^{T} \sum_{i\in\mathcal{X}} \log p(x_{i,t}|u_{i,t}), \end{aligned} \tag{17}$$

where $\boldsymbol{\theta}^{\mathrm{X}} = \{\boldsymbol{\theta}_i\}_{i\in\mathcal{X}}$ is the collection of the model parameters for observed neurons.

With the choice of (16) for the variational posterior, the batch doubly SGD update rule (15) can be turned into an online rule by generalizing Algorithm 1, as detailed in Algorithm 2. At each step of the online procedure, each hidden neuron $i \in \mathcal{H}$ emits a spike, i.e., $h_{i,t} = 1$, at any time t by following the current model distribution (16), i.e., with probability $\sigma(u_{i,t})$. Note that the membrane potential $u_{i,t}$ of any neuron i at time t is obtained from (1), with observed neurons clamped to the training example $\boldsymbol{x}_{\leq t-1}$ and hidden neurons clamped to the samples $\boldsymbol{h}_{\leq t-1}$. Then, a central processor collects the log probabilities $p(x_{i,t}|u_{i,t})$ under the current model from all observed neurons $i \in \mathcal{X}$ to compute the eligibility trace of the learning signal ℓ_t, as in (A3) and feeds back the global learning signal to all latent neurons.

Intuitively, this learning signal indicates to the hidden neurons how effective their current signaling is in ensuring the desired input/output behavior with high probability. Finally, each observed and hidden neuron i computes the eligibility trace $\boldsymbol{e}_{i,t}$ of the gradient, i.e., $\nabla_{\boldsymbol{\theta}_i} \log p_{\boldsymbol{\theta}_i}(x_{i,t}|\boldsymbol{x}_{\leq t-1}, \boldsymbol{h}_{\leq t-1})$ and $\nabla_{\boldsymbol{\theta}_i} \log p_{\boldsymbol{\theta}_i}(h_{i,t}|\boldsymbol{x}_{\leq t-1}, \boldsymbol{h}_{\leq t-1})$, respectively, as in (A4). The local parameters $\boldsymbol{\theta}_i$ of each observed neuron $i \in \mathcal{X}$ are updated in the direction of the eligibility trace $\boldsymbol{e}_{i,t}$, while each hidden neuron $i \in \mathcal{H}$ updates the parameter using $\boldsymbol{e}_{i,t}$ and the learning signal ℓ_t in (A3).

Sparsity and regularization

As discussed, the energy consumption of SNNs depends on the number of spikes emitted by the neurons. Since the ML criterion does not enforce any sparsity constraint, an SNN trained using the methods discussed so far may present dense spiking signals [18]. This is especially the case for the hidden neurons, whose behavior is not tied to the training data. To obviate this problem, it is possible to add a regularization term $-\alpha \cdot \mathrm{KL}(q_\phi(\boldsymbol{h}_{\leq T}|\boldsymbol{x}_{\leq T}) \| r(\boldsymbol{h}_{\leq T}))$ to the learning objective $L_{\boldsymbol{x}_{\leq T}}(\boldsymbol{\theta}, \phi)$ in (11), where $\mathrm{KL}(p\|q) = \sum_x p(x) \log(p(x)/q(x))$ is the Kullback–Leibler divergence between distributions p and q, $r(\boldsymbol{h}_{\leq T})$ represents a baseline distribution with the desired

level of sparsity, and $\alpha > 0$ is a parameter adjusting the amount of regularization. This regularizing term, which penalizes variational distributions far from the baseline distribution, can also act as a regularizer to minimize overfitting by enforcing a bounded rationality constraint [32]. The learning rule in Algorithm 2 can be modified accordingly.

Interpretation

The update (A5) for the synaptic weight $w_{j,i}$ of any observed neuron $i \in \mathcal{X}$ follows the local two-factor rule, as described in the "Interpretation" section. In contrast, for any hidden neuron $i \in \mathcal{H}$, the update applies a three-factor nonlocal learning rule (8). Accordingly, the postsynaptic error signal of hidden neuron i and the filtered feedforward trace of presynaptic neuron j are multiplied by the global learning signal (17). As anticipated, the global learning signal can be interpreted as an internal reward signal. To see this more generally, we can rewrite (17) as

$$\ell_{\boldsymbol{\theta}^{\mathrm{X}}}(\boldsymbol{x}_{\leq T}, \boldsymbol{h}_{\leq T}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}|\boldsymbol{h}_{\leq T}) - \log \frac{q_{\boldsymbol{\theta}^{\mathrm{H}}}(\boldsymbol{h}_{\leq T}|\boldsymbol{x}_{\leq T})}{p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T})}. \quad (18)$$

According to (18), the learning signal rewards hidden spike signals $\boldsymbol{h}_{\leq T}$, producing observations $\boldsymbol{x}_{\leq T}$ that yield a large likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\leq T}|\boldsymbol{h}_{\leq T})$ for the desired behavior. Furthermore, it penalizes values of hidden spike signals $\boldsymbol{h}_{\leq T}$ that have large variational probability $q_{\boldsymbol{\theta}^{\mathrm{H}}}(\boldsymbol{h}_{\leq T}|\boldsymbol{x}_{\leq T})$ while having a low prior probability $p_{\boldsymbol{\theta}}(\boldsymbol{h}_{\leq T})$ under the model.

As discussed in the "Learning Tasks" section, SNNs can be trained in a batch or online mode. In the next sections, we provide a representative, simple, and reproducible example for each case.

Batch learning examples

As an example of batch learning, we consider the standard handwritten digit classification task on the USPS data set [35]. We adopt an SNN with two layers, the first encoding the input and the second the output, with directed synaptic links existing from all neurons in the input layer to all neurons in the output layer. No hidden neurons exist, and, hence, training can be done as described in the section "Training SNNs: Fully Observed Models." Each 16×16 input image, representing either a one or a seven handwritten digit, is encoded in the spike domain by using rate encoding. Each gray pixel is converted into an input spiking signal by generating an independent identically distributed (i.i.d.) Bernoulli vector of T samples, with the spiking probability proportional to the pixel intensity and limited to between zero and 0.5. As a result, we have 256 input neurons, with one per pixel of the input image. The digit labels {1, 7} are also rate encoded using each one of the two output neurons. The neuron corresponding to the correct label index emits spikes with a frequency of one every three samples, while the other output neurons are silent. We refer the reader to [33] and the supplementary material [34] for further details on the numerical setup.

Figure 6 shows the classification accuracy in the test set versus the duration T of the operation of the SNN after the convergence of the training process. The classification accuracy of a conventional ANN with the same topology and a soft-max output layer is added for comparison. Note that, unlike the SNN, the ANN outputs real values, namely, the logits for each class processed by the soft-max layer. From the figure, the SNN is seen to provide a graceful tradeoff between accuracy and complexity of learning: as T increases, the number of spikes that are processed and the output by the SNN grow larger, entailing a larger inference complexity but also an improved accuracy that tends to that of the baseline ANN.

Online learning examples

We now consider an online prediction task in which the SNN sequentially observes a time sequence $\{a_l\}$ and the SNN is trained to predict, in an online manner, the next value of sequence a_l, given the observation of the previous values $a_{\leq l-1}$. The time sequence $\{a_l\}$ is encoded in the spike domain, producing a spike signal $\{\boldsymbol{x}_t\}$, consisting of N_{X} spiking signals $\boldsymbol{x}_t = (x_{1,t}, \ldots, x_{N_{\mathrm{X}},t})$ with $\Delta T \geq 1$ samples for each sample a_l. We refer to ΔT as a time expansion factor. Each of the spiking signals $x_{i,t}$ is associated with one of N_{X} visible neurons.

We adopt a fully connected SNN topology that also includes N_{H} hidden neurons. In this online prediction task, we trained the SNN using Algorithm 2, with the addition of a sparsity regularization term. This is obtained by assuming an i.i.d. reference Bernoulli distribution with a desired spiking rate $r \in [0, 1]$, i.e., $\log r(\boldsymbol{h}_{\leq T}) = \sum_{t=0}^{T} \sum_{i \in \mathcal{H}} h_{i,t} \log r + (1 - h_{i,t}) \log(1 - r)$ (see the supplementary material [34] for details). The source sequence is randomly generated as follows: at every $T_s = 25$ time steps, one of three possible sequences of duration T_s is selected, namely, an all-zero sequence with probability 0.7, a sequence of class 1 from the SwedishLeaf data set of the UCR archive [36], or a sequence of class 6 from the same archive, with equal probability [see Figure 7(a) for an illustration].

Encoding and decoding

Each value a_l of the time sequence is converted into ΔT samples $\boldsymbol{x}_{l\Delta T+1}, \boldsymbol{x}_{l\Delta T+2}, \ldots, \boldsymbol{x}_{(l+1)\Delta T}$ of the N_{X} spike signals $\{\boldsymbol{x}_t\}$

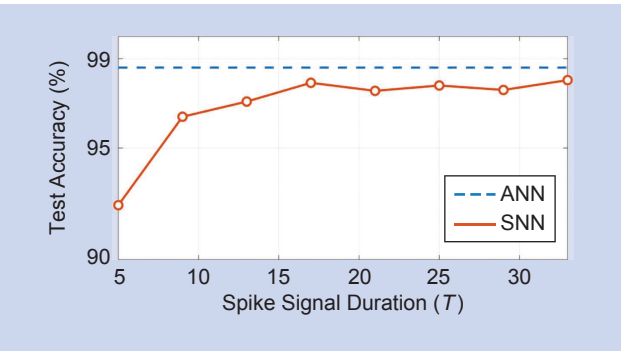

FIGURE 6. Performance of classification based on a two-layer SNN trained via batch ML learning in terms of accuracy versus the duration T of the operation of the SNN. The accuracy of an ANN with the same topology is also shown as a baseline (see [33] and [34] for details).

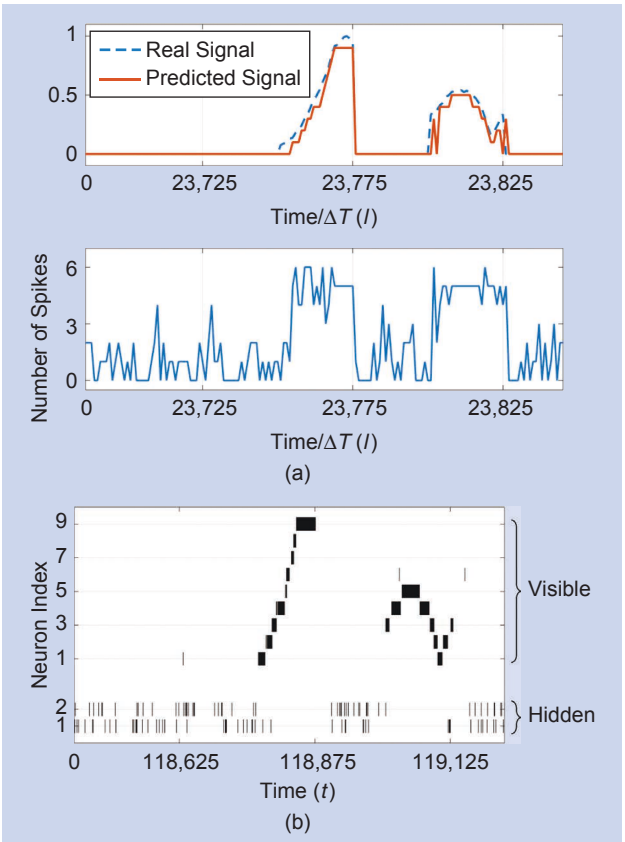

FIGURE 7. An online prediction task based on an SNN with $N_X = 9$ visible neurons and $N_H = 2$ hidden neurons trained via Algorithm 2. (a) A real analog time signal and a predicted decoded signal (top), and the total number of spikes emitted by the SNN (bottom). (b) A spike raster plot of visible neurons (top) and a spike raster plot of hidden neurons (bottom).

via rate or time coding, as illustrated in Figure 8. With rate coding, the value a_l is first discretized into $N_X + 1$ uniform quantization levels using rounding to the largest lower value. The lowest, silent level is converted to all-zero signals $\boldsymbol{x}_{l\Delta T+1}$, $\boldsymbol{x}_{l\Delta T+2}, \ldots, \boldsymbol{x}_{(l+1)\Delta T}$. Each of the other N_X levels is assigned to a visible neuron, so that the neuron associated with the quantization level corresponding to value a_l emits ΔT consecutive spikes while the other neurons are silent. Rate decoding predicts value a_{l+1} by generating the samples $\boldsymbol{x}_{(l+1)\Delta T+1}, \ldots, \boldsymbol{x}_{(l+2)\Delta T}$ from the trained model and then selecting the neuron with the largest number of spikes in this window.

For time coding, each of the N_X visible neurons is associated with a different shifted, truncated Gaussian receptive field [37]. Accordingly, as seen in Figure 8(b), for each value a_l, each visible neuron i emits a signal $x_{i,l\Delta T+1}, x_{i,l\Delta T+2}, \ldots, x_{i,(l+1)\Delta T}$ that contains no spike if the value a_l is outside the receptive field and, otherwise, contains one spike, with the timing determined by the value of the corresponding truncated Gaussian receptive field quantized to values $\{1, \ldots, \Delta T\}$ using rounding to the nearest value. Time decoding considers the first spike timing of the samples $x_{i,(l+1)\Delta T+1}, \ldots, x_{i,(l+2)\Delta T}$ for each visible neuron i and predicts a value a_{l+1} using a least-squares criterion on the values of the receptive fields (see [11] and [37]). We refer to the supplementary material [34] for further details on the numerical setup.

Rate coding

First, assuming rate encoding with $\Delta T = 5$, we train an SNN with $N_X = 9$ visible neurons and $N_H = 2$ hidden neurons using Algorithm 2. In the top portion of Figure 7(a), we see a segment

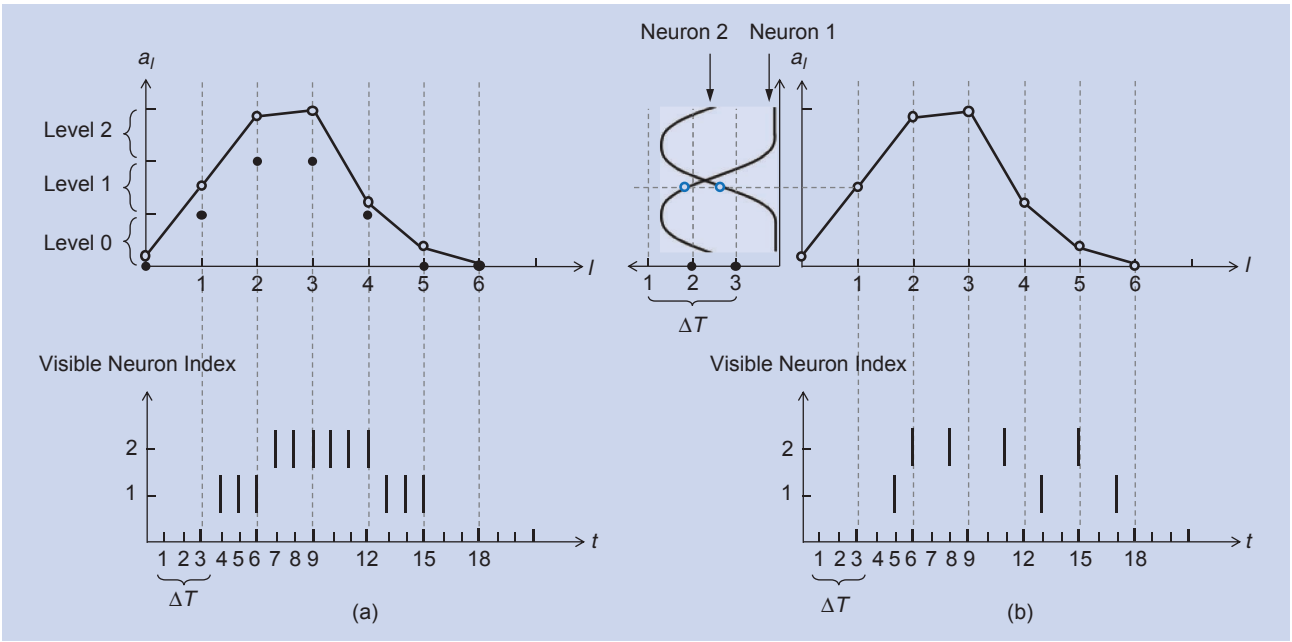

FIGURE 8. Examples of coding schemes with $N_X = 2$ visible neurons and time expansion factor $\Delta T = 3$. (a) With rate coding, each value a_l is discretized into $N_X + 1 = 3$ levels (top), and $\Delta T = 3$ consecutive spikes are assigned to input neuron i for level $i = 1, 2$, and no spikes are assigned otherwise (bottom). (b) With time coding, value a_l is encoded for each visible neuron into zero or one spike, whose timing is given by the value of the corresponding Gaussian receptive field [37].

of the signal and of the prediction for a time window after the observation of the 23,700 plus training samples of the sequence. The corresponding spikes emitted by the SNN [Figure 7(b)] are also shown (top), along with the total number of spikes per time instant [Figure 7(a, bottom)]. The SNN is seen to be able to provide an accurate prediction. Furthermore, the number of spikes, and, hence, the operating energy, depend on the level of activity of the input signal. This demonstrates the potential of SNNs for always-on event-driven applications. As a final note, in this particular example, the hidden neurons are observed to act as a detector of activity versus silence, which facilitates the correct behavior of the visible neurons.

The role of the number N_{H} of hidden neurons is further investigated in Figure 9, which shows the prediction error as a function of the number of observed training samples for different values of N_{H}. Increasing the number of hidden neurons is seen to improve the prediction accuracy as long as training is carried out for a sufficiently long time. The prediction error is measured in terms of average mean absolute error (MAE). For reference, we also compare the prediction performance with a persistent baseline (dashed line) that outputs the previous sample, upon quantization to N_{X} levels for fairness.

Rate versus time encoding

We now discuss the impact of the coding schemes on the online prediction task. We train an SNN with $N_{\mathrm{X}} = 2$ visible neurons and $N_{\mathrm{H}} = 5$ hidden neurons. Figure 10(a) shows the prediction error and Figure 10(b) the number of spikes in a window of 2,500 samples of the input sequence, after the observation of the 17,500 training samples, versus the time expansion factor ΔT. From the figure, rate encoding is seen to be preferable for smaller values of ΔT, while time encoding achieves better prediction error for larger ΔT with fewer spikes and, hence, energy consumption.

This result is a consequence of the different use that the two schemes make of the time expansion ΔT. With rate encoding, a larger ΔT entails a large number of spikes for the neuron encoding the correct quantization level, which provides increased robustness to noise. In contrast, with time encoding, the value ΔT controls the resolution of the mapping between input value a_l and the spiking times of the visible neurons. This demonstrates the efficiency benefits of SNNs that may arise from their unique time encoding capabilities.

Conclusions and open problems

As illustrated by the examples in the previous section, SNNs provide a promising alternative solution to conventional ANNs for the implementation of low-power learning and inference. When using rate encoding, they can approximate the performance of any ANN while also providing a graceful tradeoff between accuracy, on the one hand, and energy consumption and delay, on the other. Most importantly, they have the unique capacity to process time-encoded information, yielding sparse, event-driven, and low-complexity inference and learning solutions.

The recent advances in hardware design reviewed in [5] are motivating renewed efforts to tackle the current lack of well-established direct training algorithms that are able to harness the potential efficiency gains of SNNs. This article has argued that this gap is, at least in part, a consequence of the insistence on the use of deterministic models, which is in turn due to their dominance in the context of ANNs. As discussed, not only can probabilistic models allow the recovery of learning rules that are well known in theoretical neuroscience, but

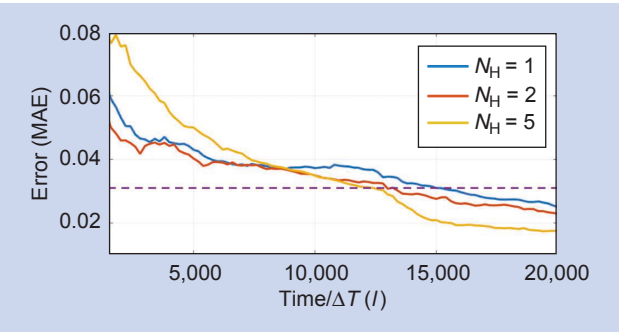

FIGURE 9. Prediction error versus training time for SNNs with $N_X = 9$ visible neurons and $N_H = 1, 2$, and 5 hidden neurons trained via ML learning using Algorithm 2. The dashed line indicates the performance of a baseline persistent predictor that outputs the previous sample (quantized to N_X levels, as described in the text).

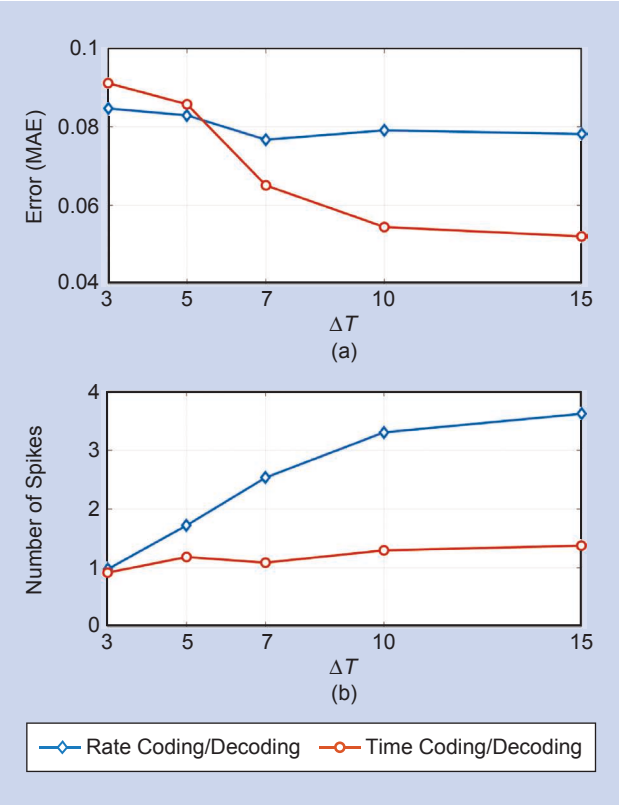

FIGURE 10. An online prediction task based on an SNN consisting of $N_X = 2$ visible neurons and $N_H = 5$ hidden neurons, with rate and time coding schemes: (a) prediction error and (b) number of spikes emitted by the SNN versus the time expansion factor ΔT.

they can also provide a principled framework for the derivation of more general training algorithms. Notably, these algorithms differ significantly from the standard backpropagation approach used for ANNs, owing to their locality coupled with global feedback signaling.

With the main aim of inspiring more research on the topic, this article has presented a review of models and training methods for probabilistic SNNs within a probabilistic signal processing framework. We focused on GLM spiking neuron models, given their flexibility and tractability, and on ML-based training methods. We conclude this article with some discussion on extensions in terms of models and algorithms as well as on open problems.

The SNN models and algorithms we have considered can be extended and modified along various directions. In terms of models, while randomness is defined here at the level of neurons' outputs, alternative models introduce randomness at the level of synapses or thresholds [38], [39]. Furthermore, while the models studied in this article encode information in the temporal behavior of the network within a given interval of time, information can also be retrieved from the asymptotic steady-state spiking rates, which define a joint probability distribution [4], [40], [41]. Specifically, when the GLM (4), (5) has symmetric synaptic weights, i.e., $w_{j,i} = w_{i,j}$, the memory of the synaptic filter is $\tau = 1$, and there is no feedback filter, the conditional probabilities (5) for all neurons define a Gibbs sampling procedure for a Boltzmann machine that can be used for this purpose. As another extension, more general connections among neurons can be defined, including instantaneous firing correlations, and more information, such as a sign, can be encoded in a spike [33]. Finally, while here we focus on signal processing aspects, at a semantic level, SNNs can process logical information by following different principles [11].

In terms of algorithms, the doubly stochastic SGD approach reviewed here for ML training can be extended and improved by leveraging an alternative estimator of the ELBO and its gradients with respect to the variational parameters that is known as the *reparameterization trick* [28]. Furthermore, similar techniques can be developed to tackle other training criteria, such as Bayesian optimal inference [31], reward maximization in reinforcement learning [23], and mutual information maximization for representation learning (see [12] for a discussion in the context of general probabilistic models).

Interesting open problems include the development of metalearning algorithms, whereby the goal is learning how to train or adapt a network to a new task (see, e.g., [41]); the design of distributed learning techniques; and the definition of clear use cases and applications with the quantification of advantages in terms of power efficiency [42]. Another important problem is the design of efficient input/output interfaces between information sources and the SNN, at one end, and between the SNN and actuators or end users, on the other. In the absence of such efficient mechanisms, SNNs risk replacing the so-called memory wall of standard computing architectures with an input/output wall.

Acknowledgments

This work was supported in part by the European Research Council under the European Union's Horizon 2020 research and innovation program under grant 725731 and by the U.S. National Science Foundation under grant ECCS 1710009. André Grüning (partly) and Brian Gardner (fully) are supported by the European Union's Horizon 2020 Framework Programme for Research and Innovation under the specific grant agreement 785907 (Human Brain Project SGA2).

Authors

Hyeryung Jang (hyeryung.jang@kcl.ac.uk) received her B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, in 2010, 2012, and 2017, respectively. She is currently a research associate in the Department of Informatics, King's College London, United Kingdom. Her recent research interests lie in the mathematical modeling, learning, and inference of probabilistic graphical models, with a specific focus on spiking neural networks and communication systems. Her past research works also include network economics, game theory, and distributed algorithms in communication networks.

Osvaldo Simeone (osvaldo.simeone@kcl.ac.uk) received his M.Sc. degree (with honors) and Ph.D. degree in information engineering from Politecnico di Milano, Italy, in 2001 and 2005, respectively. He is a professor of information engineering with the Centre for Telecommunications Research, Department of Informatics, King's College London, United Kingdom. He is a corecipient of the 2019 IEEE Communication Society Best Tutorial Paper Award, the 2018 IEEE Signal Processing Society Best Paper Award, the 2017 Best Paper by *Journal of Communications and Networks*, the 2015 IEEE Communication Society Best Tutorial Paper Award, and the IEEE International Workshop on Signal Processing Advances in Wireless Communications 2007 and IEEE Wireless Rural and Emergency Communications Conference 2007 Best Paper Awards. He currently serves on the editorial board of *IEEE Signal Processing Magazine* and is a Distinguished Lecturer of the IEEE Information Theory Society. He is a Fellow of the Institution of Engineering and Technology and of the IEEE.

Brian Gardner (b.gardner@surrey.ac.uk) received his M.Phys. degree from the University of Exeter, United Kingdom, in 2011 and his Ph.D. degree in computational neuroscience from the University of Surrey, Guildford, United Kingdom, in 2016. He is a research fellow in the Department of Computer Science, University of Surrey. Currently, his research focuses on the theoretical aspects of learning in spiking neural networks. He is also working as a part of the Human Brain Project and is involved with the implementation of spike-based learning algorithms in neuromorphic systems for embedded applications.

André Grüning (andre.gruening@hochschule-stralsund.de) received his undergraduate degree in theoretical physics from the University of Göttingen, Germany, and his Ph.D. degree in computer science from the University of Leipzig, Germany. He is a professor of mathematics and computational intelligence at the University of Applied Sciences, Stralsund, Germany. He is a visiting member of the European Institute for Theoretical Neuroscience, Paris, France. Previously, he was a senior lecturer (associate professor) in the Department of Computer Science, University of Surrey, Guildford, United Kingdom. He held research posts in computational neuroscience at the Scuola Internazionale Superiore di Studi Avanzati, Trieste, Italy, and in cognitive neuroscience at the University of Warwick, Coventry, United Kingdom. His research concentrates on computational and cognitive neuroscience, especially learning algorithms for spiking neural networks. He is a partner in the Human Brain Project, a European Union Horizon 2020 Flagship Project.

References

[1] M. Welling, "Intelligence per kilowatt-hour," YouTube, 2018. [Online]. Available: https://youtu.be/7QhkvG4MUbk

[2] H. Paugam-Moisy and S. Bohte, "Computing with spiking neuron networks," in *Handbook of Natural Computing*. Springer-Verlag, 2012, pp. 335–376.

[3] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997.

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

[5] B. Rajendran, A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou, Low-power neuromorphic hardware for signal processing applications. 2019. [Online]. Available: https://arxiv.org/abs/1901.03690

[6] B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *Proc. IEEE Int. Symp. Circuits and Systems*, Florence, Italy, 2018, pp. 1–5. doi: 10.1109/ISCAS.2018.8351295.

[7] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Front. Neurosci.*, vol. 10, Nov. 2016. doi: 10.3389/fnins.2016.00508.

[8] P. O'Connor and M. Welling, Deep spiking networks. 2016. [Online]. Available: https://arxiv.org/abs/1602.08323

[9] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Front. Neurosci.*, vol. 12, May 2018. doi: 10.3389/fnins.2018.00331.

[10] P. Dayan and L. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA: MIT Press, 2001.

[11] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT Press, 2004.

[12] O. Simeone, "A brief introduction to machine learning for engineers," *Found. Trends Signal Process.*, vol. 12, no. 3-4, pp. 200–431, 2018.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 2018.

[14] J. W. Pillow, J. Shlens, L. Paninski, A. Sher, A. M. Litke, E. J. Chichilnisky, and E. P. Simoncelli, "Spatio-temporal correlations and visual signalling in a complete neuronal population," *Nature*, vol. 454, no. 7207, Aug. 2008.

[15] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, United Kingdom: Cambridge Univ. Press, 2002.

[16] T. Osogami, Boltzmann machines for time-series. 2017. [Online]. Available: https://arxiv.org/abs/1708.06004

[17] R. M. Neal, "Connectionist learning of belief networks," *Artif. Intell.*, vol. 56, no. 1, pp. 71–113, 1992.

[18] F. Gerhard, M. Deger, and W. Truccolo, "On the stability and dynamics of stochastic spiking neuron models: Nonlinear Hawkes process and point process GLMs," *PLoS Comput. Biol.*, vol. 13, no. 2, 2017. doi: 10.1371/journal.pcbi.1005390.

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

[20] E. O. Neftci, H. Mostafa, and F. Zenke, Surrogate gradient learning in spiking neural networks. 2019. [Online]. Available: https://arxiv.org/abs/1901.09948

[21] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Front. Neural Circuits*, vol. 9, Jan. 2016. doi: 10.3389/fncir.2015.00085.

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

[23] B. Rosenfeld, O. Simeone, and B. Rajendran, "Learning first-to-spike policies for neuromorphic control using policy gradients," in *Proc. IEEE Int. Workshop Signal Processing Advances Wireless Communications (SPAWC)*, Cannes, France, 2019. doi: 10.1109/SPAWC.2019.8815546.

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

[25] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex," *J. Neurosci.*, vol. 2, no. 1, pp. 32–48, 1982.

[26] A. J. Watt and N. S. Desai, "Homeostatic plasticity and STDP: Keeping a neurons cool in a fluctuating world," *Front. Synaptic Neurosci.*, vol. 2, June 2010. doi: 10.3389/fnsyn.2010.00005.

[27] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," in *Proc. Int. Conf. Machine Learning (ICML)*, Beijing, 2014, pp. 1791–1799.

[28] D. P. Kingma and M. Welling, Auto-encoding variational Bayes. 2013. [Online]. Available: arXiv:1312.6114

[29] D. J. Rezende and W. Gerstner, "Stochastic variational learning in recurrent spiking networks," *Front. Comput. Neurosci.*, vol. 8, Apr. 2014. doi: 10.3389/fncom.2014.00038.

[30] G. E. Hinton and A. D. Brown, "Spiking Boltzmann machines," in *Proc. Advances Neural Information Processing Systems (NIPS)*, Denver, CO, 2000, pp. 122–128.

[31] D. Kappel, S. Habenschuss, R. Legenstein, and W. Maass, "Network plasticity as Bayesian inference," *PLoS Comput. Biol.*, vol. 11, no. 11, 2015. doi: 10.1371/journal.pcbi.1004485.

[32] F. Leibfried and D. A. Braun, "A reward-maximizing spiking neuron as a bounded rational decision maker," *Neural Comput.*, vol. 27, no. 8, pp. 1686–1720, 2015.

[33] H. Jang and O. Simeone, "Training dynamic exponential family models with causal and lateral dependencies for generalized neuromorphic computing," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, U.K., 2019, pp. 3382–3386.

[34] H. Jang, O. Simeone, B. Gardner, and A. Grüning, "An introduction to spiking neural networks: Probabilistic models, learning rules, and applications [supplementary material]," 2019. [Online]. Available: https://nms.kcl.ac.uk/osvaldo.simeone/spm-supp.pdf

[35] J. J. Hull, "A database for handwritten text recognition research," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 550–554, 1994.

[36] H. A. Dau et al., "The UCR time series classification archive," Oct. 2018. [Online]. Available: https://www.cs.ucr.edu/~eamonn/time_series_data_2018

[37] S. M. Bohte, H. La Poutré, and J. N. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 426–435, 2002.

[38] N. Kasabov, "To spike or not to spike: A probabilistic spiking neuron model," *Neural Netw.*, vol. 23, no. 1, pp. 16–19, 2010.

[39] H. Mostafa and G. Cauwenberghs, "A learning framework for winner-take-all networks with stochastic synapses," *Neural Comput.*, vol. 30, no. 6, pp. 1542–1572, 2018.

[40] W. Maass, "Noise as a resource for computation and learning in networks of spiking neurons," *Proc. IEEE*, vol. 102, no. 5, pp. 860–880, 2014.

[41] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Proc. Advances Neural Information Processing Systems (NIPS)*, Montreal, 2018, pp. 787–797.

[42] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, Benchmarking keyword spotting efficiency on neuromorphic hardware. 2018. [Online]. Available: https://arxiv.org/abs/1812.01739