frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 12 April 2021
doi: 10.3389/fncom.2021.617862

Supervised Learning With First-to-Spike Decoding in Multilayer Spiking Neural Networks

Brian Gardner[1]* *and André Grüning*[2]

[1] Department of Computer Science, University of Surrey, Guildford, United Kingdom, [2] Faculty of Electrical Engineering and Computer Science, University of Applied Sciences, Stralsund, Germany

Experimental studies support the notion of spike-based neuronal information processing in the brain, with neural circuits exhibiting a wide range of temporally-based coding strategies to rapidly and efficiently represent sensory stimuli. Accordingly, it would be desirable to apply spike-based computation to tackling real-world challenges, and in particular transferring such theory to neuromorphic systems for low-power embedded applications. Motivated by this, we propose a new supervised learning method that can train multilayer spiking neural networks to solve classification problems based on a rapid, first-to-spike decoding strategy. The proposed learning rule supports multiple spikes fired by stochastic hidden neurons, and yet is stable by relying on first-spike responses generated by a deterministic output layer. In addition to this, we also explore several distinct, spike-based encoding strategies in order to form compact representations of presented input data. We demonstrate the classification performance of the learning rule as applied to several benchmark datasets, including MNIST. The learning rule is capable of generalizing from the data, and is successful even when used with constrained network architectures containing few input and hidden layer neurons. Furthermore, we highlight a novel encoding strategy, termed "scanline encoding," that can transform image data into compact spatiotemporal patterns for subsequent network processing. Designing constrained, but optimized, network structures and performing input dimensionality reduction has strong implications for neuromorphic applications.

Keywords: spiking neural networks, multilayer SNN, supervised learning, backpropagation, temporal coding, classification, MNIST

OPEN ACCESS

Edited by:
Guenther Palm,
University of Ulm, Germany

Reviewed by:
Thomas Wennekers,
University of Plymouth,
United Kingdom
Steve B. Furber,
The University of Manchester,
United Kingdom

***Correspondence:**
Brian Gardner
b.gardner@surrey.ac.uk

Received: *15 October 2020*
Accepted: *08 March 2021*
Published: *12 April 2021*

Citation:
Gardner B and Grüning A (2021) Supervised Learning With First-to-Spike Decoding in Multilayer Spiking Neural Networks. Front. Comput. Neurosci. 15:617862. doi: 10.3389/fncom.2021.617862

1. INTRODUCTION

Neurons constitute complex biological circuits, and work to convey information via rapid, spike-based signaling. These neural circuits interconnect with one another, forming the basis of large scale networks in the brain, and are often organized as consecutive processing layers operating at increasing levels of abstraction. For example, within the visual system, information regarding object features can be temporally encoded as spikes in little over just 10 ms, and its identity determined through feedforward processing pathways within 200 ms of pattern onset (Hung et al., 2005; Kiani et al., 2005; Gollisch and Meister, 2008). There is also evidence indicating that first spike times relative to stimulus onset, rather than comparatively long neural firing rate estimates, are utilized in the human somatosensory system to enable rapid behavioral responses (VanRullen et al., 2005).

Interestingly, such temporal-based coding strategies are thought to describe a rank order code (Thorpe et al., 2001), whereby the order in which groups of encoding neurons generate first-spike responses corresponds to the importance of the information they signal. In terms of forming such representations, the adaptation of synaptic connections between neurons is hypothesized to underlie the learning process: principally based on correlated neuronal activity patterns and regulatory, homeostatic plasticity mechanisms (Morrison et al., 2008). In particular, a Hebbian-like learning scheme, termed spike-timing-dependent plasticity (STDP), is considered to play a prominent role (Gerstner and Kistler, 2002), whereby the strength of a synaptic connection is modified according to the relative timing difference between paired pre- and postsynaptic firing events (Bi and Poo, 1998). Drawing on these principles, theoretical work has sought to model goal-directed learning in the brain using spiking neural networks (SNNs): typically incorporating concepts from machine learning such as supervised and reinforcement learning for this purpose (Grüning and Bohte, 2014). Despite progress in this respect, a more comprehensive theoretical description of learning that also aims to more fully exploit the rapidity and precision of spike-based temporal coding is still largely lacking; consequently, finding real-world applications for spike-based learning and computing techniques, including their transfer to neuromorphic platforms, remains an open issue.

For the most part, theoretical studies into spike-based learning in neural networks have been devised based on the dynamics of the leaky integrate-and-fire (LIF) neuron model, typically when formalized as the simplified spike response model (SRM_0) (Gerstner and Kistler, 2002), owing to its convenient trade-off between analytical tractability and model realism in run simulations. Additionally, the application of gradient descent in order to minimize the value of some predefined cost function, as taken for supervised learning, is a useful starting point in order to obtain weight update rules for SNNs (Gütig, 2014). A common learning objective has been to train neurons to precisely fire at one or more prescribed target firing times; to this end, the cost function of an SNN is usually defined in terms of the separation between target and actual firing times with respect to one or more of its readout neurons. Hence, by applying gradient descent, weight update rules for the network can be derived, and accordingly implemented during training in order to support neuronal firing at these target timings (Bohte et al., 2002; Florian, 2012; Sporea and Grüning, 2013; Gardner et al., 2015; Zenke and Ganguli, 2018). Furthermore, some supervised approaches have incorporated a trained neuron's subthreshold voltage into the network cost function: for example to support a more efficient mode of operation in addition to learning target firing times (Albers et al., 2016). In a similar effort, the minimum distance between a neuron's voltage and its firing threshold, as measured over some predetermined observation period, has been selected as the point at which a neuron should be driven to fire, in order to provide a highly efficient spike-based classifier rule (Gütig and Sompolinsky, 2006; Urbanczik and Senn, 2009). Aside from setting up an initial cost function, some studies have instead taken a statistical approach to learning; for instance, based on a maximum-likelihood principle that works to maximize the likelihood of an SNN generating desired target firing times (Pfister et al., 2006; Gardner and Grüning, 2016), or similarly by minimizing an upper bound on the KL-divergence between the actual and target firing distribution for more complex, spike-based, recurrent neural network (RNN) architectures (Brea et al., 2013). A variational, online learning rule for training recurrent SNNs has also recently been proposed in Jang et al. (2020); a detailed review of probabilistic learning methods can be found in Jang et al. (2019). Otherwise, the procedure used to learn target firing times may be mapped from Perceptron-like learning, as has originally been used to train the readout weights of spike-based RNNs, also known as liquid state machines (Maass et al., 2002). Perceptron-like learning has also been used to learn targets with high precision in feedforward SNNs (Memmesheimer et al., 2014), or more heuristically by modifying weights according to a spike-based adaptation of the Widrow-Hoff rule (Ponulak and Kasiński, 2010; Mohemmed et al., 2012; Yu et al., 2013).

A large part of the studies described so far have been concerned with training SNNs to learn target output firing times, which in most cases tend to be arbitrarily selected; this usually follows from focusing on a proof-of-concept of a derived learning procedure, rather than measuring its technical performance on benchmark datasets. Moreover, biological plausibility is a common concern with spike-based learning approaches, which can place further constraints on a model and detract from its performance. Although there is likely to be strong potential in utilizing spike-based computation for data classification purposes, it remains unclear which temporal coding strategy is best suited for this purpose. For instance, learning multiple, temporally-precise sequences of spikes in order to categorize input patterns into different classes might inadvertently lead to model overfitting, and hinder generalization to previously unseen samples; this is more likely to be an issue with high precision rules, for example the E-learning variant of the Chronotron (Florian, 2012) or the HTP algorithm (Memmesheimer et al., 2014). Therefore, a preferable coding strategy for a spike-based classifier might instead rely on selecting output responses which place the least constraint on the trained parameters: for example as demonstrated by the single-layer Tempotron rule (Gütig and Sompolinsky, 2006).

Although the minimally-constrained Tempotron has proven capable of high performance with respect to certain problem domains, such as vocabulary recognition (Gütig and Sompolinsky, 2009), there may arise limitations in terms of its flexibility as applied to increasingly challenging datasets such as MNIST, for which networks containing hidden neurons are indicated for its solution; interestingly, however, recently submitted work has addressed this issue by implementing a Tempotron-inspired cost function combined with a multilayer learning procedure, and to good effect (Zenke and Vogels, 2020). Despite such progress, there still exist comparatively few learning rules for multilayer SNNs compared with single-layer ones, owing to the complexity in solving ill-defined gradients of hidden layer spike trains when applying the technique of backpropagation. A number of approaches have relied on approximating such gradients: for example by taking a linear approximation of a neuron's response close to its firing

threshold (Bohte et al., 2002), estimating a spike train by its underlying firing density (Sporea and Grüning, 2013), or using a surrogate gradient to substitute a neuron's spike-gradient with an analytically tractable one (Zenke and Ganguli, 2018; Neftci et al., 2019). Furthermore, some studies have taken a statistical approach which instead consider the likelihood of a neuron's firing response, as applied to feedforward (Gardner et al., 2015) and recurrent (Brea et al., 2013; Jimenez Rezende and Gerstner, 2014) network structures. In these cases, however, the networks have been constrained to learning predefined, target firing patterns, with less of a focus on utilizing efficient temporal encoding and decoding strategies for data classification purposes. Of the studies which have focused on applying multilayer or deep SNNs to more challenging datasets, some have demonstrated that training rate-based artificial neural networks (ANNs) and transferring the learned weights to similarly designed SNNs for test inference can provide performance competitive with state-of-the-art systems (O'Connor et al., 2013; Diehl et al., 2015). A main limitation of this approach, however, is that these equivalent ANNs must be trained offline before being mapped to an online system, making this technique somewhat restrictive in terms of its application to adaptive learning tasks. A further study exploring deep SNN architectures considered a scheme which involved low-pass filtering spike events in order to establish smooth gradients for backpropagation, although this came with the caveat of introducing auxiliary variables which needed to be computed separately (Lee et al., 2016). Intriguingly, one study took a semi-supervised approach to training deep SNNs containing convolutional and pooling layers, by applying STDP to modify weights at each layer based on first spike responses (Kheradpisheh et al., 2018): in this way, the hidden layers learned the features of objects in an unsupervised manner, appropriate for subsequent classification by a linear SVM classifier. Otherwise, some studies have arrived at alternative solutions by approximating simulated LIF neurons as rectified linear units (ReLUs) (Tavanaei and Maida, 2019; Kheradpisheh and Masquelier, 2020), or instead simulating non-leaky integrate-and-fire neurons for analytical tractability (Mostafa, 2017), thereby establishing closed-form expressions for the weight updates. These methods have resulted in competitive performance on the MNIST dataset, and the first-to-spike decoding methods implemented by Mostafa (2017) and Kheradpisheh and Masquelier (2020), which classify data samples according to which output neuron is the first to respond with a spike, have proven to be particularly rapid at forming predictions. Interestingly, the recent work of Bagheri et al. (2018), which examined training probabilistic, single-layer SNNs as applied to MNIST, has also indicated at the merits of utilizing a rapid, first-to-spike decoding scheme.

In this article, we introduce a new supervised learning algorithm to train multilayer SNNs for data classification purposes, based on a first-to-spike decoding strategy. This algorithm extends on our previous MultilayerSpiker rule described in Gardner et al. (2015), by redefining the network's objective function as a cost over first spike arrival times in the output layer, and instead implementing deterministic output neurons for more robust network responses. Our method also supports multiple spikes generated by hidden layer neurons: conferring an additional level of processing capability compared with other, single-spike based, learning methods. We test our new first-to-spike multilayer classifier rule on several benchmark classification tasks, including the ubiquitous MNIST dataset of handwritten digits, in order to provide an indication of its technical capability. Additionally, we explore several different spike-based encoding strategies to efficiently represent the input data, including one novel technique that can transform visual patterns into compact spatio-temporal patterns via "scanline encoding." We determine that such an encoding strategy holds strong potential when applied to constrained network architectures, as might exist with a neuromorphic hardware platform. In the next section we start our analysis by describing the specifics of our first-to-spike neural classifier model.

2. MATERIALS AND METHODS

2.1. Neuron Model

We consider the simplified SRM_0, as defined in Gerstner and Kistler (2002), to describe the dynamics of a postsynaptic neuron's membrane potential with time t:

$$u_i(t) := \sum_{j\in\Gamma_i} w_{ij}\left(\epsilon * S_j\right)(t) + \left(\kappa * S_i\right)(t)\,, \tag{1}$$

where the neuron is indexed i, and its membrane potential is measured relative to a resting potential arbitrarily set to 0 mV The first term on the RHS of the above equation describes a weighted sum over the neuron's received presynaptic spikes, where Γ_i denotes the set of direct neural predecessors of neuron i, or its presynaptic neurons, and the parameter w_{ij} refers to the synaptic weight projecting from presynaptic neuron j. The term $\left(\epsilon * S_j\right)(t) \equiv \int_0^t \epsilon(s)S_j(t-s)\mathrm{d}s$ refers to a convolution of the postsynaptic potential (PSP) kernel ϵ and the j-th presynaptic spike train S_j, where a spike train is formalized as a sum of Dirac-delta functions, $S_j(t) = \sum_f \delta_D(t - t_j^f)$, over a list of presynaptic firing times $\mathcal{F}_j = \{t_j^1, t_j^2, \dots\}$. The second term on the RHS of Equation (1) signifies the dependence of the postsynaptic neuron on its own firing history, where κ is the reset kernel and S_i is the neuron's spike train for postsynaptic firing times $\mathcal{F}_i = \{t_i^1, t_i^2, \dots\}$. A postsynaptic spike is considered to be fired at time t_i^f when u_i crosses the neuron's fixed firing threshold ϑ from below. The PSP and reset kernels are, respectively, given by:

$$\epsilon(s) = \epsilon_0\left[\exp\left(-\frac{s}{\tau_m}\right) - \exp\left(-\frac{s}{\tau_s}\right)\right]\Theta(s)\,, \tag{2}$$

$$\kappa(s) = \kappa_0 \exp\left(-\frac{s}{\tau_m}\right)\Theta(s)\,. \tag{3}$$

With respect to Equation (2), $\epsilon_0 = 4\,\mathrm{mV}$ is a scaling constant, $\tau_m = 10\,\mathrm{ms}$ the membrane time constant, $\tau_s = 5\,\mathrm{ms}$ a synaptic time constant and $\Theta(s)$ the Heaviside step function. With respect to Equation (3), the reset strength is given by $\kappa_0 = -(\vartheta - u_r)$, where $u_r = 0\,\mathrm{mV}$ is the value the neuron's membrane potential is reset to immediately after a postsynaptic spike is fired upon

crossing the threshold $\vartheta = 15\,\text{mV}$. From Equation (1) it follows that the neuron's resting potential is equal to its reset value, i.e., $u_{\text{rest}} = u_{\text{r}} = 0\,\text{mV}$.

2.2. Learning Rule

2.2.1. Notation

The technique of backpropagation is applied to a feedforward multilayer SNN containing hidden layers of neurons, where the objective of the network is to perform pattern recognition on multiple input classes by learning error-minimizing weights. Network layers are indexed by l, with $l \in \{1, 2, \ldots, L-1, L\}$, where $l = 1, L$ correspond to the input and last layers, respectively. The number of neurons in the l-th layer is denoted N_l. In our analysis, each input class corresponds to a distinct output neuron: hence, if the total number of classes is equal to c then the number of output neurons is given by $N_L = c$. Using this notation, the SRM_0 defined by Equation (4) is rewritten as

$$u_i^l(t) := \sum_{j \in \Gamma_i^l} w_{ij}^l \left(\epsilon * S_j^{l-1}\right)(t) + \left(\kappa * S_i^l\right)(t)\,, \tag{4}$$

for a postsynaptic neuron in the l-th layer receiving its input from previous layer neurons belonging to the set Γ_i^l. The spike train of a neuron i in layer l is now denoted by $S_i^l(t) = \sum_f \delta_D(t - t_i^f)$, and its associated list of firing times, $\mathcal{F}_i^l = \{t_i^1, t_i^2, \ldots\}$.

2.2.2. Cost Function

The objective is to train a multilayer SNN to efficiently classify input patterns based on a temporal decoding scheme. To this end, a first-to-spike code seems appropriate, since it encourages rapidity of neural processing (Thorpe et al., 2001) and avoids arbitrarily constraining the network to generate spikes with specific timings. There is also experimental evidence supporting the notion of a latency code in relation to visual and neural processing pathways (Hung et al., 2005; VanRullen et al., 2005; Gollisch and Meister, 2008). Hence, we focus on implementing a minimally-constrained, competitive learning scheme: such that the output neuron with the earliest and strongest activation, resulting in a first-spike response, decides the class of input pattern.

Taking the above points into consideration, a suitable choice for the i-th output layer neuron's activation is a softmax, given by

$$a_i^L = \frac{\exp\left(-\nu\tau_i\right)}{\sum_{i'} \exp\left(-\nu\tau_{i'}\right)}\,, \tag{5}$$

where ν is a scale parameter controlling the sharpness of the distribution, i' indexes each output neuron, $1 \leq i' \leq c$, and $\tau_{i'}$ is the first firing time of neuron i'. If a neuron i' fails to fire any spike, then it is assumed $\tau_{i'} \to \infty$. The set of activations can be interpreted as a conditional probability distribution over the predicted class labels. A natural choice of cost function using softmax activation is the cross-entropy, given by

$$C(\mathbf{y}, \mathbf{a}^L) = -\sum_i y_i \log a_i^L\,, \tag{6}$$

where $\mathbf{y} \in \mathbb{R}^c$ is a c-dimensional target activation vector of the network, associated with the presented input pattern, and $\mathbf{a}^L \in \mathbb{R}^{N_L}$ is the vector of output layer neuron activations. Since we are concerned with a classification problem a one-hot encoding scheme is used to describe a target vector, such that all components of $\mathbf{y}$ are set to zero except for the one corresponding to the pattern class. For example, if a dataset were comprised of three input pattern classes, then patterns belonging to the second class would be associated with $\mathbf{y} = (0, 1, 0)$. Hence, using this coding strategy, and using y to denote the index of the target class label, Equation (6) reduces to

$$C(\mathbf{y}, \mathbf{a}^L) = -\log a_y^L\,, \tag{7}$$

where a_y^L now denotes the activation of the single output neuron corresponding to the correct class. The above choices of cost and activation functions is inspired by the approach taken in Mostafa (2017), although here we instead consider LIF neurons and extend our analysis to include entire spike trains generated by input and hidden layer neurons.

2.2.3. Error Signal

The technique of backpropagation is applied in order to train weights within the multilayer network, by minimizing the cross-entropy loss defined by Equation (7). We begin by taking the gradient of Equation (7) with respect to the membrane potential of a neuron i in the final layer, a term which will be useful later:

$$\frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial u_i^L} = -\frac{\partial \log a_y^L}{\partial u_i^L}\,, \tag{8}$$

which can be rewritten, using the chain rule, as

$$\frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial u_i^L} = -\frac{1}{a_y^L}\frac{\partial a_y^L}{\partial u_i^L}\,. \tag{9}$$

Furthermore, the gradient of the neuron's activation can be expanded using the chain rule as follows:

$$\frac{\partial a_y^L}{\partial u_i^L} = \frac{\partial a_y^L}{\partial \tau_i}\frac{\tau_i}{\partial u_i^L}\,. \tag{10}$$

Using Equation (5), the first gradient on the RHS of the above can be solved to provide one of two cases:

$$\frac{\partial a_y^L}{\partial \tau_i} = \begin{cases} a_y^L(a_i^L - 1) & \text{if } i = y\,, \\ a_i^L a_y^L & \text{if } i \neq y\,. \end{cases} \tag{11}$$

The second gradient on the RHS of Equation (10) is ill-defined, but can be approximated by making certain assumptions regarding the neuron's dynamics close to its firing threshold. Specifically, for a deterministic LIF neuron it follows that the gradient of the neuron's membrane potential must be positive at its firing threshold when a spike is fired, such that $\partial u_i^L / \partial t(\tau_i) > 0$. Hence, following Bohte et al. (2002), we make a first order approximation of u_i^L for a small region about $t = \tau_i$, giving rise to

the relation $\delta\tau_i = -\delta u_i^L/\alpha$, where the local gradient is given by $\alpha = \partial u_i^L/\partial t(\tau_i)$. Taken together, the gradient of the neuron's first firing time is approximated by

$$\begin{aligned}\frac{\tau_i}{\partial u_i^L} &\approx \left.\frac{\partial \tau_i(u_i^L)}{\partial u_i^L(t)}\right|_{u_i^L=\theta} \\ &\approx -\left.\frac{1}{\partial u_i^L/\partial t}\right|_{t=\tau_i} = -\frac{1}{\alpha}\,, \end{aligned} \tag{12}$$

where for numerical stability reasons α is considered to be a positive, constant value. For the sake of brevity this constant is set to unity in the remainder of this analysis, and gives no qualitative change in the final result. Thus, Equations (9)–(12) are combined to give one of two possible output neuron error signals:

$$\frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial u_i^L} = \begin{cases} a_i^L - 1 & \text{if } i = y\,, \\ a_i^L & \text{if } i \neq y\,, \end{cases} \tag{13}$$

depending on whether the i-th neuron corresponds to the target label y. Using our earlier notation for the network's target activation vector $\mathbf{y} = (y_1, y_2, \ldots, y_c)$ as used in Equation (6), the above can be written more compactly as

$$\begin{aligned}\delta_i^L &:= \frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial u_i^L} \\ &:= a_i^L - y_i\,, \end{aligned} \tag{14}$$

where we define δ_i^L to be the error signal due to the i-th neuron in the final layer.

2.2.4. Output Weight Updates

We apply gradient descent to Equation (7) with respect to final layer weights, such that the weight between the i-th output neuron and j-th previous layer, hidden neuron is modified according to

$$\Delta w_{ij}^L = -\eta \frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial w_{ij}^L}\,, \tag{15}$$

where $\eta > 0$ is the learning rate. The second term on the RHS is expanded using the chain rule to give

$$\frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial w_{ij}^L} = \frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial u_i^L} \left.\frac{\partial u_i^L(t)}{\partial w_{ij}^L}\right|_{t=\tau_i}\,, \tag{16}$$

where the gradient of the output neuron's membrane potential is evaluated at the time of its first spike. The first gradient term on the RHS of this equation corresponds to the neuron's error signal, as provided by Equation (14), hence the above can be rewritten as

$$\frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial w_{ij}^L} = \delta_i^L \left.\frac{\partial u_i^L(t)}{\partial w_{ij}^L}\right|_{t=\tau_i}\,. \tag{17}$$

Using the definition of the neuron's membrane potential given by Equation (4), and neglecting the contribution due to refractory effects which is valid for sufficiently low output firing rates, the above becomes

$$\Delta w_{ij}^L = -\eta \delta_i^L \left(\epsilon * S_j^{L-1}\right)(\tau_i)\,, \tag{18}$$

where the constant α, as introduced by Equation (12), is folded into η for simplicity. By inspecting each of the terms of the above equation, we note that the synaptic factor, $\left(\epsilon * S_j^{L-1}\right)(\tau_i)$, acts as a correlation trace: capturing the causal contribution of the set of previous layer firing events from neuron j to the generation of the first spike fired by output neuron i. Additionally, the term δ_i^L signals the degree of network error contributed due to this output spike; hence, this rule works to minimize the network error by changing the weight in the opposite direction, and proportionate to these two terms. An example of the above weight update rule taking place in a simulated SNN is visualized in **Figure 1**.

Integrated formula. Integrating out the spike train in Equation (18) gives

$$\Delta w_{ij}^L = -\eta \delta_i^L \sum_{t_j^f \in \mathcal{F}_j^{L-1}} \epsilon(\tau_i - t_j^f)\,, \tag{19}$$

where $\mathcal{F}_j^{L-1}$ is used to denote the list of spike times contributed by the j-th neuron in the previous layer, $L-1$.

2.2.5. Hidden Weight Updates

With respect to hidden layer weight updates, gradient descent is taken on Equation (7) according to

$$\Delta w_{ij}^{L-1} = -\eta \frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial w_{ij}^{L-1}}\,, \tag{20}$$

where the weight update between the i-th hidden neuron in layer $L-1$ and the j-th presynaptic neuron in layer $L-2$ is derived. Hence, using the chain rule, the gradient on the RHS is expanded as follows:

$$\begin{aligned}\frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial w_{ij}^{L-1}} &= \sum_{k \in \Gamma^{i,L-1}} \frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial u_k^L} \left.\frac{\partial u_k^L(t)}{\partial w_{ij}^{L-1}}\right|_{t=\tau_k} \\ &= \sum_{k \in \Gamma^{i,L-1}} \delta_k^L \left.\frac{\partial u_k^L(t)}{\partial w_{ij}^{L-1}}\right|_{t=\tau_k}\,, \end{aligned} \tag{21}$$

where $\Gamma^{i,L-1}$ denotes the immediate set of neural successors of neuron i in layer $L-1$, or the set of output layer neurons, and having used the identity of the output error signal given by Equation (14). Using Equation (4), the gradient of the K-th membrane potential becomes

$$\begin{aligned}\left.\frac{\partial u_k^L(t)}{\partial w_{ij}^{L-1}}\right|_{t=\tau_k} &= w_{ki}^L \frac{\partial}{\partial w_{ij}^{L-1}} \left(\epsilon * S_i^{L-1}\right)(\tau_k) \\ &= w_{ki}^L \left(\epsilon * \frac{\partial S_i^{L-1}}{\partial w_{ij}^{L-1}}\right)(\tau_k)\,, \end{aligned} \tag{22}$$

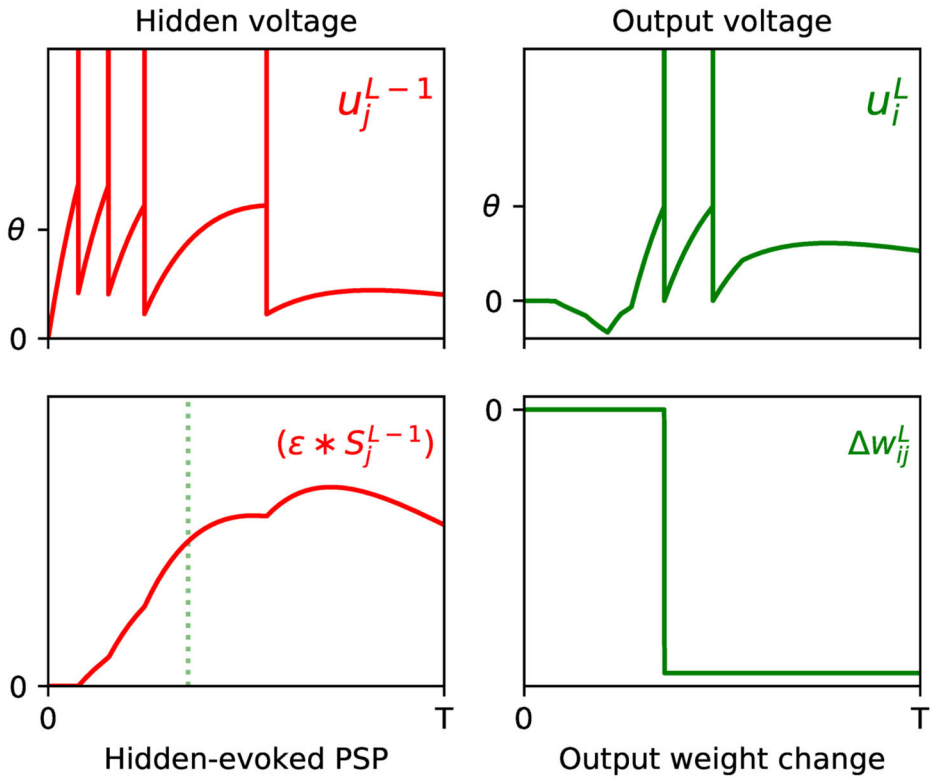

FIGURE 1 | Example of the weight update process with respect to the output layer, $l = L$, of a multilayer SNN, according to Equation (18). **(Top)** The left panel shows the membrane potential of an excited presynaptic neuron j in the second-to-last layer, $l = L - 1$, over a small observation time T (arbitrary units). The vertical lines indicate the neuron's firing times, and ϑ its firing threshold. The right panel shows the response of a postsynaptic neuron i in the network's output layer, which is stimulated in part by neuron j. **(Bottom)** The left panel shows the PSP evoked due to neuron j. The first output spike fired by neuron i is indicated by the green dotted line. Hence, as shown in the right panel, the magnitude of the weight change between neurons j and i is proportional to the value of the PSP at the moment neuron i fires its first spike. In this example the output neuron does not correspond to the class of the input pattern, therefore the direction of the weight change is negative to discourage early spiking.

where we neglect the contribution from the refractory term. Evaluating the gradient of a spike train poses a challenge given its discontinuous nature when generated by LIF neurons. One approach to resolving this might instead just consider the first spike contributed by hidden layer neurons, as used for SpikeProp (Bohte et al., 2002), although this loses information about neural firing frequency and typically requires the addition of multiple subconnections with the next layer to support sufficient downstream activation. There have been extensions of SpikeProp to allow for multiple spikes in the hidden layers (Booij and Nguyen, 2005; Ghosh-Dastidar and Adeli, 2009), although these methods rely on small learning rates and constrained weight gradients to allow for convergence. To circumvent this issue, we treat hidden layer neurons as being probabilistic in order to provide smoother gradients (Gardner et al., 2015). Specifically, we introduce stochastic spike generation for hidden neurons using the Escape Noise model (Gerstner et al., 2014). By this mechanism, hidden neuron firing events are distributed according to an inhomogeneous Poisson process with a time-dependent rate parameter that is a function of the neuron's membrane potential: $\rho_i^l(t) = g(u_i^l(t))$. This can be interpreted as the neuron's instantaneous firing rate, or firing density, where the probability of the neuron firing a spike over an infinitesimal time window δt is given by $\rho_i^l(t)\delta t$. Here we take an exponential dependence of the firing density on the distance between the neuron's membrane potential and threshold (Gerstner et al., 2014):

$$g(u_i^l(t)) = \rho_0 \exp\left(\frac{u_i^l(t) - \vartheta}{\Delta u}\right), \tag{23}$$

where $\rho_0 = 0.01\,\mathrm{ms}^{-1}$ is the instantaneous rate at threshold, and $\Delta u = 1\,\mathrm{mV}$ controls the variability of generated spikes. Hence, following our previous method in Gardner et al. (2015), we can substitute the gradient of the spike train in Equation (22) with the gradient of its expected value, conditioned on spike trains in the previous layer of the network, such that

$$\frac{\partial S_i^{L-1}(t)}{\partial w_{ij}^{L-1}} \to \frac{\partial \left\langle S_i^{L-1}(t)\right\rangle_{S_i^{L-1}|\{S_j^{L-2}\}}}{\partial w_{ij}^{L-1}}. \tag{24}$$

If we also condition the expected spike train on the neuron's most recently fired spike, $\hat{t}_i < t$, then we can express Equation (24) as the gradient of the instantaneous value of the spike train, distributed according to its firing density (Frémaux et al., 2013):

$$\frac{\partial \left\langle S_i^{L-1}(t) \right\rangle_{S_i^{L-1}|\{S_j^{L-2}\},\hat{t}_i}}{\partial w_{ij}^{L-1}} = \frac{\partial}{\partial w_{ij}^{L-1}} \sum_{q\in\{0,\delta(t)\}} q(t)\rho_i^{L-1}(t|\{S_j^{L-2}\},\hat{t}_i) = \delta_D(t-\hat{t})\frac{\partial \rho_i^{L-1}(t|\{S_j^{L-2}\},\hat{t}_i)}{\partial w_{ij}^{L-1}} , \quad (25)$$

where $\delta_D(t-\hat{t})$ is the Dirac-delta function centered on some most recent spike time $\hat{t}$. Using Equations (4) and (23), and denoting "$|\{S_j^{L-2}\},\hat{t}_i$" as "$|L-2,i$" for brevity, we obtain

$$\frac{\partial \left\langle S_i^{L-1}(t) \right\rangle_{S_i^{L-1}|L-2,i}}{\partial w_{ij}^{L-1}} = \frac{1}{\Delta u}\delta_D(t-\hat{t})\rho_i^{L-1}(t|L-2,i)\left(\epsilon * S_j^{L-2}\right)(t) = \frac{1}{\Delta u}\left\langle S_i^{L-1}(t)\left(\epsilon * S_j^{L-2}\right)(t)\right\rangle_{S_i^{L-1}|L-2,i} . \quad (26)$$

We can estimate the expected value of the spike train's gradient through samples generated by the network during simulations, hence the above can be approximated as

$$\frac{\partial \left\langle S_i^{L-1}(t) \right\rangle_{S_i^{L-1}|L-2,i}}{\partial w_{ij}^{L-1}} \approx \frac{1}{\Delta u} S_i^{L-1}(t)\left(\epsilon * S_j^{L-2}\right)(t) . \quad (27)$$

Combining Equations (22), (24), and (27) provides an estimate for the gradient of the k-th output neuron's membrane potential, evaluated at the time of its first fired spike:

$$\left.\frac{\partial u_k^L(t)}{\partial w_{ij}^{L-1}}\right|_{t=\tau_k} = \frac{1}{\Delta u} w_{ki}^L \left(\epsilon * \left[S_i^{L-1}\left(\epsilon * S_j^{L-2}\right)\right]\right)(\tau_k) . \quad (28)$$

Hence, combining the above with Equations (20) and (21) gives the second-last layer weight update rule:

$$\Delta w_{ij}^{L-1} = -\frac{\eta}{\Delta u}\sum_{k\in\Gamma^{i,L-1}} \delta_k^L w_{ki}^L \left(\epsilon * \left[S_i^{L-1}\left(\epsilon * S_j^{L-2}\right)\right]\right)(\tau_k) . \quad (29)$$

From examining the terms in the above equation, we note that the double convolution term, $\left(\epsilon * \left[S_i^{L-1}\left(\epsilon * S_j^{L-2}\right)\right]\right)(\tau_k)$, captures the correlation between sequences of spikes fired by neurons i and j in layers $L-1$ and $L-2$, respectively. Furthermore, this correlation trace is evaluated at the moment of an output neuron k generating its first spike, τ_k, thereby signifying the joint contribution of these upstream layer neurons in supporting this final layer response. Similarly to Equation (18), this correlation term is combined with the resulting error signal due to output neuron k, but additionally summed over all output error signals in order to obtain the total contribution to the network error; as part of this summation, each weight value w_{ki}^L takes into account whether this upstream activity elicits either an excitatory or inhibitory effect on the corresponding downstream response. Hence, the weight w_{ij}^{L-1} is changed in the opposite direction to these summed terms, according to the gradient descent procedure. For a visualization of this process, an example of a weight update taking place between the input and hidden layers of an SNN containing a single hidden layer is shown in **Figure 2**.

Integrated formula. Integrating out the spike trains in Equation (29) gives

$$\Delta w_{ij}^{L-1} = -\frac{\eta}{\Delta u}\sum_{k\in\Gamma^{i,L-1}} \delta_k^L w_{ki}^L \sum_{t_i^f\in\mathcal{F}_i^{L-1}} \epsilon(\tau_k - t_i^f) \sum_{t_j^g\in\mathcal{F}_j^{L-2}} \epsilon(t_i^f - t_j^g) , \quad (30)$$

where $\mathcal{F}_i^{L-1}$ and $\mathcal{F}_j^{L-2}$ are the list of spike times from neurons i and j in layers $L-1$ and $L-2$, respectively. The weight update formulae described by Equations (19) and (30) determine the supervised learning process of our first-to-spike neural classifier rule, as applied to multilayer SNNs containing a single hidden layer of stochastic neurons. The above procedure is not restricted to SNNs containing one hidden layer, however: as demonstrated in the **Supplementary Material**, it is also possible to extend this approach to networks containing more than one hidden layer.

2.3. Temporal Encoding

For demonstrative purposes, this article studies the performance of the proposed multilayer learning rule as applied to a selection of benchmark classification datasets. To this end, it was necessary to first convert input features into spike-based representations: to be conveyed by the input layer of an SNN for downstream processing. Therefore, we made use of three distinct encoding strategies to achieve this, including: latency-based, receptive fields and scanline encoding. An overview of each strategy is described as follows.

2.3.1. Latency Encoding

A straightforward means to forming a temporal representation of an input feature is to signal its intensity based on the latency of a spike. Specifically, if we consider an encoding LIF neuron that is injected with a fixed input current, then the time taken for it to respond with a spike can be determined as a function of the current's intensity: by interpreting the feature's value as a current, it is therefore possible for it to be mapped to a unique firing time. For an encoding LIF neuron i with a fixed firing threshold that only receives a constant current I_i, its first response time is given by (Gerstner and Kistler, 2002):

$$t_i^1 = \begin{cases} \tau_m \log\left(\frac{RI_i}{RI_i-\vartheta}\right) & \text{if } RI_i > \vartheta , \\ \infty & \text{otherwise} , \end{cases} \quad (31)$$

where we use the same parameter selections for τ_m and ϑ as used in section 2.1, and the resistance is set to $R = 4M\Omega$. In terms of relating feature values to current intensities, we take one of two different approaches. In the first approach we arbitrarily associate each feature value with a unique intensity value, which is ideally suited to the case where features are limited to a small number of

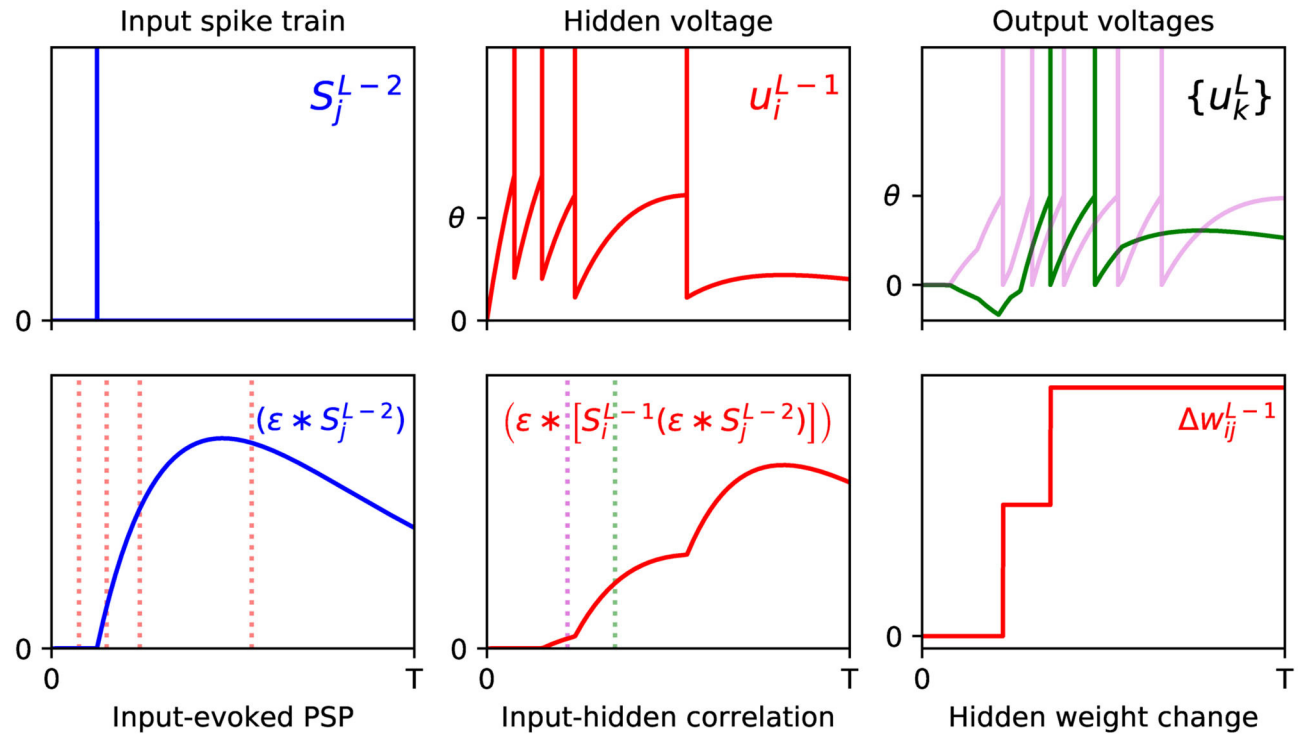

FIGURE 2 | Example of the weight update process with respect to a hidden layer, $l = L - 1$, of a multilayer SNN, according to Equation (29). **(Top)** The left panel shows the spike train of a presynaptic neuron j in the first layer, $l = L - 2$, observed over time T (arbitrary units). As shown in the middle panel, a postsynaptic neuron i in layer $l = L - 1$ receives this input spike, among others, and responds by firing a sequence of hidden spikes. The right panel shows the response of all output neurons in the network, each of which are stimulated in part by neuron i. **(Bottom)** The left panel shows the PSP evoked by input neuron j at its postsynaptic target i. The red dotted lines indicate the firing times of i, which capture input-to-hidden spike correlations as shown in the middle panel. Since the first hidden spike occurs before the input spike, this spike makes no contribution to the input-hidden correlation. The middle panel also indicates the first firing times of all neurons in the output layer, marked by the magenta (first output to respond) and green dotted lines. The values of the input-hidden correlation trace at these two moments are used to inform the candidate weight change between input and hidden neurons j and i, as shown in the right panel. In this example, the output neuron corresponding to the correct class label (magenta) is the first to fire a spike, as desired. The hidden neuron i projects positive- and negative-valued synaptic weights to the magenta and green output neurons, respectively, hence this translates into a positive change in w_{ij}^{L-1} at each moment.

discrete values. In the second approach, and in the case where features take real values, we devise a more direct association; specifically, each value x_i belonging to a feature vector $\mathbf{x}$ is normalized to fall within the unit range before being scaled by a factor $I_{\max}$, providing the current intensity I_i. The specific choice of $I_{\max}$ used depends on the studied dataset. Regardless of the approach we take, and in order to maintain a tight distribution of early spike arrivals, we disregard spikes with timings >9 ms by setting them to infinity.

2.3.2. Receptive Fields

An alternative, population-based approach to encoding real-valued variables relies on the concept of receptive fields. This biologically-inspired strategy describes a type of rank-order code (Thorpe et al., 2001), whereby each encoding neuron is constrained to fire at most one spike in response to presented input values. In the context of SNNs, a method has been described by Bohte et al. (2002) which involves setting up a population of neurons with overlapping, graded response curves which are individually sensitive to a certain subset of values an encoded feature can take. Typically, a Gaussian-shaped response curve (or receptive field) is assumed, where an early (late) spike fired by a neuron corresponds to strong (weak) overlap with its encoded feature. For the datasets that are encoded in this way, we assign q neurons with Gaussian receptive fields to each feature. For the i-th feature, with values existing in the range $[x_i^{\min}, \ldots, x_i^{\max}]$, its encoding neural fields are centered according to $x_i^{\min} + (2j-3)/2 \cdot (x_i^{\max} - x_i^{\min})/(q-2)$, for encoder indices $1 \leq j \leq q$, and using the width parameter $\sigma_i = 2/3 \cdot (x_i^{\max} - x_i^{\min})/(q-2)$ (Bohte et al., 2002). Hence, a data sample consisting of n_f features results in a matrix of first-layer neural activations: $\mathbf{a}^1 \in \mathbb{R}^{n_f \times q}$, with values in the range $(0, 1)$. As in Bohte et al. (2002), these activations are then mapped to a matrix of single spike time encodings according to $\mathbf{t}^1 = 10 \cdot (1 - \mathbf{a}^1)$, where values >9 are discarded since they are deemed insufficiently activated. For the datasets transformed by receptive fields in this article, we used a different number of encoding neurons to give the best performance.

2.3.3. Scanline Encoding

A promising strategy for transforming visual data into spike patterns is to apply "scanline encoding," a technique that has

been described in Lin et al. (2018). Scanline encoding is a method inspired by human saccadic eye movements, and works to detect object edges within images when scanned across multiple line projections; when an increase in pixel intensity is detected along one of these scanlines, an associated, encoding spike is generated. The efficiency of this method derives from its subsampling of image pixels using a limited number of scanlines, as well as its invariance to small, local image distortions; in this way, it is possible to perform dimensionality reduction on images with spatial redundancy. There are several ways in which scanline encoding can be implemented, and the specific approach taken by Lin et al. (2018) represents just one possible choice. In general, the first step involves setting up a number of scanlines with certain orientations that are fixed with respect to all training and test images; ideally, the directions of these scanlines should be selected to capture the most informative pixels, as determined through preliminary parameter sweeps. The next step is to then preprocess the images by reading the values of pixels along these directions. In Lin et al. (2018), these scanlines are additionally split into several segments, where each segment maps to a channel that injects spikes into an SNN: if a contrast shift is detected anywhere along one of these segments, then a repeating spike train is generated for the corresponding channel. Although using segmented-scanlines provided the authors with a test accuracy of 96.4 % on MNIST, a large number of neurons was required to encode the images in this way and the significance of individual spike timings was disregarded.

In our approach we wish to fully utilize the timings of individual spikes to maintain sparse image representations, and also to avoid artificially segmenting scanlines in the first instance. To this end, we modify the spike generation process by instead interpreting the read-in pixel values underneath each line as a series of sequentially-occurring current stimulus values. Hence, if we assume these values are injected over some duration to an encoding LIF neuron, then we arrive at a sequence of precisely-timed spikes representing each scanline. In terms of parameter selection, the encoding LIF neurons are designed to be relatively fast responders: with their membrane time constants set to 3 ms. The resistance of each neuron is set to $R = 10\,\mathrm{M}\,\Omega$, with a firing threshold of just one millivolt to elicit a rapid response. Immediately after firing a spike an encoding neuron's membrane potential is reset to 0 mV, and the neuron is forced to remain quiescent for 1 ms. With respect to the scanlines, we first decide on a number n_s according to the experimental design. Each scanline is then setup as follows. First, the orientation of each line is selected according to a uniform distribution over the range $[0, \pi)$. Each line is then set to intercept through a position that is normally-distributed about the image center, where the scale parameter of this distribution is a quarter of the image width. These scanlines remain fixed across all training and test images. Hence, when an image is encoded, the pixels lying underneath a scanline are injected as current stimulus values into a corresponding LIF encoder, after first normalizing pixels to exist in the unit range. Pixels are always scanned-in from the bottom of an image upwards, over a duration of 9 ms. An example of this encoding strategy is illustrated in **Figure 3**.

2.4. Network Structure

In all of the experiments we considered fully-connected, feedforward SNNs containing a single hidden layer of spiking neurons, such that $L = 3$. Data samples presented to a network were encoded by the collective firing activity of input layer neurons, according to one of the temporal encoding strategies described above; hidden layer neurons were free to perform computations on these input patterns, and learn features useful for downstream processing. Neurons in the last, or output, layer of a network were tasked with forming class predictions on these data samples according to a first-to-spike mechanism, where the predicted class label was determined according to which one of $N_{l=3} = c$ output neurons was the first to respond with an output spike. The number of neurons implemented in the input and hidden layers, N_1 and N_2, respectively, depended on the type of input data and the run experiment, although we generally aimed to design a minimalistic setup for efficiency reasons. As described in section 2.2, stochastic and deterministic SRM_0 neurons were simulated in the hidden and output layers, respectively, and, unless otherwise stated, all neurons within a layer shared the same cellular parameters. For all experiments, the internal simulation time step was set to $\delta t = 0.1$ ms.

In terms of network connections, we initialized hidden and output weights to drive the initial firing rates of neurons to approximately one spike in response to presented data samples. Initial weight values were drawn from a uniform distribution, as detailed in the description of each experiment. Unless otherwise stated, each pre- and post-synaptic neuron pair had a single connection with no conduction delay in spike propagation.

2.5. Synaptic Plasticity

Synaptic weights projecting onto the hidden and output layers of multilayer SNNs were modified during training via a combination of synaptic plasticity rules, while subject to certain constraints. This process is described in detail as follows.

2.5.1. Learning Procedure

During training, data samples were presented to a network iteratively as mini-batches, where computed weight gradients were accumulated over individual samples before actually being applied at the end of each mini-batch; this procedure was selected in order to reduce the variance of weight changes in a network to provide smoother convergence, as well as to obtain more reliable estimates of network activity as needed for regularization. The order in which data samples were assigned as mini-batches was random, and, unless otherwise specified, the number of samples in each batch was 150. Furthermore, weight gradients were only computed after observing the entire response of a network when stimulated by a data sample, which in most cases was completed after duration $T = 40$ ms given input spikes arriving within 10 ms of pattern onset. Hence, if the network was presented with a data sample described by an input vector $\mathbf{x}$ and a one-hot encoded class label $\mathbf{y}$, then, after applying a suitable temporal encoding strategy, a synaptic weight gradient in the l-th layer was

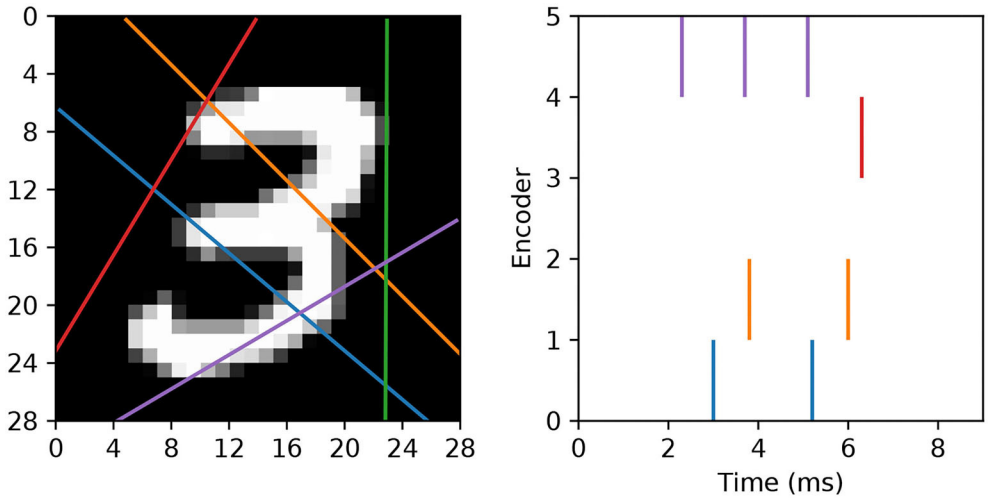

FIGURE 3 | Example of an MNIST digit being transformed into five spike trains using "scanline encoders." In this case, the orientations of the scanlines were randomly predetermined, and pixel intensities lying underneath each line were read-in working from the bottom upwards over 9 ms. Encoder 4 (purple line) overlapped the digit for the longest duration, thereby prompting the highest number of encoding spikes, whereas encoder 2 (green line) did not cross any part of the digit, and therefore resulted in no spikes.

determined as

$$\Delta w_{ij}^{l} = -\eta \left(\frac{\partial C(\mathbf{y}, \mathbf{a}^{L})}{\partial w_{ij}^{l}} + \lambda(S_i^l) - \gamma(S_i^l) \right) , \tag{32}$$

where the first term in brackets on the RHS is the gradient of the cost function, which is evaluated following the steps of Equations (15) or (20) for the output and hidden layers, respectively. The second term, $\lambda(S_i^l)$, is a regularization function which depends on the postsynaptic neuron's firing activity, and the final term, $\gamma(S_i^l)$, is a synaptic scaling function. These two functions are defined as follows.

2.5.2. Regularization Term

As a means to encourage network generalization we enforced an L2 penalty term with respect to hidden and output layer weight gradients during training. Additionally, we also included a factor penalizing high neuronal firing rates: a strategy that has been demonstrated in Zenke and Ganguli (2018) to provide increased network stability as well as boosted performance. The regularization term is defined by

$$\lambda(S_i^l) = \lambda_0 w_{ij}^l \zeta(S_i^l) , \tag{33}$$

where λ_0 is a scaling factor that is optimized for each experiment and the function $\zeta(S_i^l) = \left[\int_{t=0}^{T} S_i^l(t) \mathrm{d}t \right]^2$ is an activity-dependent penalty term that depends on the number of spikes fired by a neuron i in layer l. Since data samples were iteratively presented to the network, and the observation period T was set sufficiently large, integrating over a spike train accurately reflected a neuron's firing rate. Through preliminary simulations we found that selecting an exponent greater than one with respect to the dependence of ζ on a neuron's firing rate gave the best results, consistent with that found in Zenke and Ganguli (2018).

2.5.3. Synaptic Scaling Term

It was necessary to include a synaptic scaling term as part of the weight gradient computation in order to sustain at least a minimum of activity in the network during training. This is because the weight update formulae described by Equations (18) and (29) both depend on presynaptic spikes in order to compute output and hidden weight gradients, which would result in non-convergent learning if no spikes could be acted upon. Adapting the synaptic scaling rule described in our previous work (Gardner et al., 2015), as well as taking inspiration from the scaling procedure used in Mostafa (2017), we define the synaptic scaling term as follows:

$$\gamma(S_i^l) = \begin{cases} \gamma_0 |w_{ij}^l| & \text{if } \int_{t=0}^{T} S_i^l(t) \mathrm{d}t = 0 , \\ 0 & \text{otherwise} , \end{cases} \tag{34}$$

where $\gamma_0 = 0.1$ is a scaling parameter. From a biological perspective, synaptic scaling can be interpreted as a homoeostatic learning factor that assists with maintaining desired activity levels (van Rossum et al., 2000).

2.5.4. Learning Schedule

The learning procedure used to compute weight gradients, defined by Equation (32), was accumulated over all data samples assigned to a mini-batch before weights were actually updated in a trained network. However, rather than directly using these computed gradients, we took the additional step of applying synapse-specific, adaptive learning rates to modulate the magnitude of the weight updates. As found in Zenke and Ganguli (2018), and through preliminary simulations, we found that a technique referred to as RMSProp (Hinton et al., 2012) was more effective in providing convergent performance than applying a global, non-adaptive learning rate, and proved less sensitive to the experimental design. Specifically, an accumulated

weight gradient Δw_{ij}^l, as determined using Equation (32), was used to inform a weight update via RMSProp according to

$$w_{ij}^l \leftarrow w_{ij}^l + \frac{\eta_0}{\sqrt{m_{ij}^l + \varepsilon}} \Delta w_{ij}^l$$
$$m_{ij}^l \leftarrow \beta m_{ij}^l + (1-\beta)\left(\Delta w_{ij}^l\right)^2 , \quad (35)$$

where $\eta_0 > 0$ is a constant coefficient that was specific to each experiment, m_{ij}^l is an auxiliary variable that keeps track of the recent weight gradient magnitudes, $\varepsilon = 1 \times 10^{-8}$ is a small offset that was included for numerical stability and $\beta = 0.9$ is a decay factor. The initial value of m_{ij}^l was taken to be zero. Additionally, weights were constrained to a range, $w_{ij}^l \in [w_{\min}, w_{\max}]$, in order to prevent overlearning during training. The weight limits were specific to each of the studied experiments.

3. RESULTS

3.1. Solving the XOR Task

As a first step in assessing the performance of the first-to-spike multilayer classifier, we tested its ability to classify data samples that were linearly non-separable. A classic benchmark for this is the exclusive-or (XOR) classification task, a non-trivial problem for which a hidden layer of spiking neurons is indicated to be necessary for its solution (Grüning and Sporea, 2012; Gardner et al., 2015).

An XOR computation takes as its input two binary-valued input variables, and maps them to a single binary target output in the following way: $\{0,0\} \rightarrow 0$, $\{0,1\} \rightarrow 1$, $\{1,0\} \rightarrow 1$, and $\{1,1\} \rightarrow 0$, where 1 and 0 correspond to Boolean True and False, respectively. To make this scheme compatible with SNN processing, we first transformed the input values into spike-based representations using an appropriate temporal encoding strategy. In this case, each binary value was encoded by the latency of an input spike, such that values of 1 and 0 corresponded to early and late spike timings, respectively. For simplicity, we selected the associated current intensities, as defined by Equation (31), such that an input value of 1 resulted in a spike latency of 0 ms, and an input value of 0 resulted in a spike latency of 6 ms. Furthermore, in order to make the task non-trivial to solve, and to allow the network to discriminate between input patterns presenting both True or False values, we also included an input bias neuron that always fired a spike at 0 ms to signal pattern onset (Bohte et al., 2002). Hence, as illustrated in **Figure 4A**, we setup an SNN which contained three input neurons (one bias and two encoders), five hidden neurons and two output neurons to signal the two possible class labels (True/False). At the start of each experiment run, hidden and output weights were initialized by drawing their values from uniform distributions over the ranges: $[0, 16]$ and $[0, 6.4]$, respectively. The softmax scale parameter defining output activations was set to $\nu = 2$. In terms of network training, results were gathered from runs lasting 500 epochs, where each epoch corresponded to the presentation of all four input patterns. Regularization was not required on this task, so we set $\lambda_0 = 0$. The RMSProp coefficient was set to $\eta_0 = 0.5$, and throughout training hidden and output weights were constrained between $[-30, 30]$.

As shown in **Figure 4B**, the network was successful in learning the XOR task: reaching a final training loss of 0.02 ± 0.01 as obtained from 100 independent runs [error reported as standard error of the mean (SEM)]. This reflected a final classification accuracy of $99.8 \pm 0.2\%$, which didn't reach precisely 100% due to the stochastic nature of hidden layer spike generation. In terms of the final weight distributions of the network (**Figures 4C,D**), systematic trends were observed for certain connections in the hidden and output layers. With respect to the hidden layer, incoming connections received from the encoder neurons were widely distributed, with just over 70% being excitatory. By comparison, the bias neuron tended to project positively-skewed weights, with almost 90% being excitatory; the relatively large fraction of excitatory connections indicated its role in sustaining hidden layer activity, irrespective of the input pattern statistics. With respect to the final distribution of output layer weights, the False- and True-signaling neurons differed from each other by assuming a greater proportion of weight values saturating toward their lower and upper limits, respectively. The result of this distribution was to suppress the erroneous output neuron until the desired one received sufficient input activation to fire; this process can be inferred from the spike rasters depicted in **Figure 5**, showing the hidden and output layer responses of a post-trained network when presented with each of the four input patterns. For example, when presented with patterns labeled as "False" the output neuron signaling "True" responded with a small number of delayed spikes, whereas the correct neuron promptly responded with multiple, rapid spikes.

3.2. Classifying the Iris and Wisconsin Datasets

A key determinant of a classifier system's performance is its ability to generalize from training data to previously unseen test samples. In order to assess the generalization ability of the proposed learning rule we applied it to classifying two benchmark datasets: Iris and Wisconsin.

The Iris dataset (Fisher, 1936) presents a multi-class classification problem, containing 150 data samples evenly split between three different classes: two of which are not linearly separable from each other. Each sample is described by four real-valued features measuring some of the attributes of three types of Iris flower. The Wisconsin breast cancer dataset (Wolberg and Mangasarian, 1990) is a binary classification problem, containing 699 samples split between two different classes. Each sample consists of nine discrete-valued features measuring the likelihood of malignancy based on their intensity, and is labeled as either benign or malignant. We note that of these 699 original samples 16 contained missing values, which we discarded to provide a revised total of 683. In terms of our strategy for transforming these two datasets into spike-based representations, we followed the approach of Bohte et al. (2002); specifically, Gaussian receptive fields were applied as a means to converting the input features into first-layer spike latencies, resulting in spike-timings distributed between 0 and 9 ms (see section 2.3).

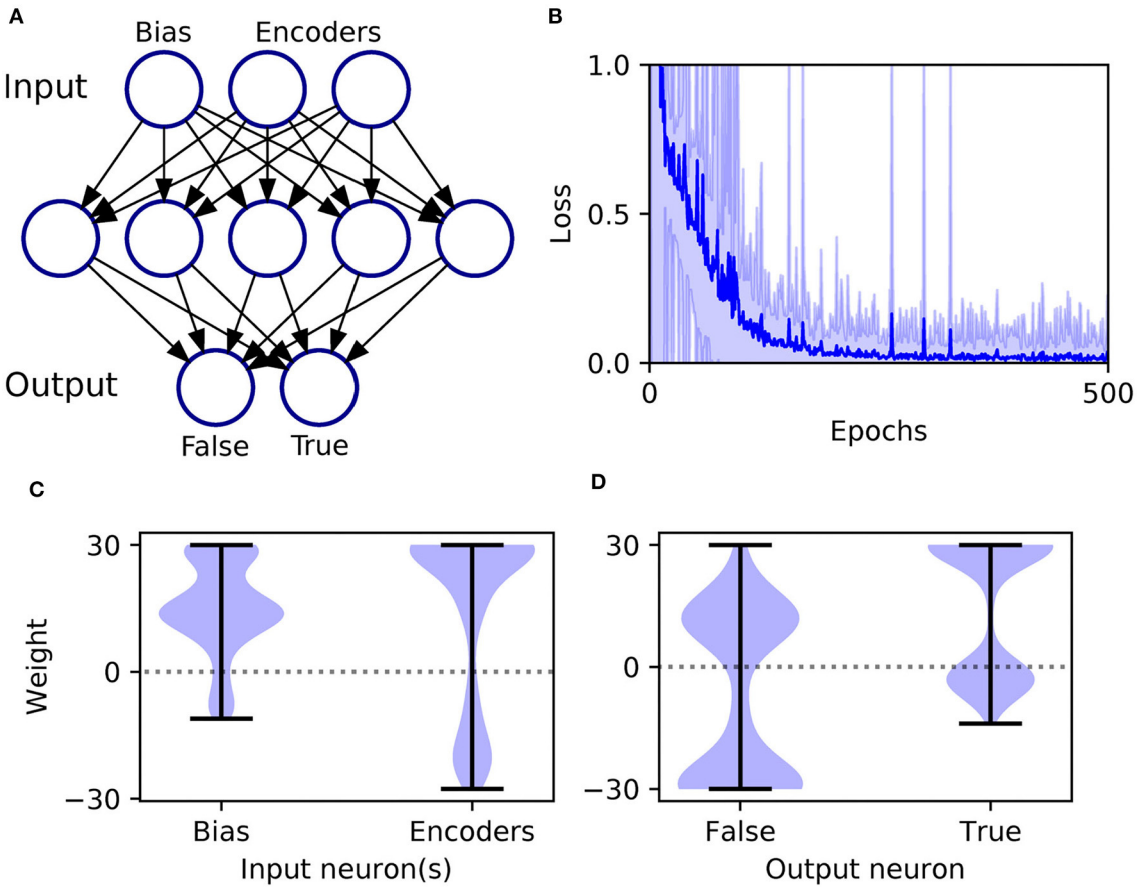

FIGURE 4 | Solving the XOR computation in a multilayer SNN using first-to-spike decoding. **(A)** Network setup: consisting of three, five, and two neurons in the input, hidden and output layers, respectively. Input patterns were encoded by the latencies of spikes in the input layer, and their class labels were predicted based on which one of the two output neurons was the first to respond with a spike. **(B)** Network loss as a function of the number of training epochs, averaged over 100 independent runs. The loss is defined as the cross-entropy, given by Equation (6), and is taken as the average value with respect to all of training samples per epoch. The units of loss is measured in nats. The shaded region shows the standard deviation. **(C)** Post-training hidden layer weight distributions. This panel breaks down the overall distribution of hidden weights into two components: one with respect to connections projecting from the input bias neuron, and the other due to input encoder neurons. The shaded width corresponds to the probability density of weights. **(D)** Post-training output layer weight distributions. This panel shows the distribution of output weights with respect to connections projecting from the hidden layer onto False- or True-signaling output neurons.

For Iris, consisting of $n_f = 4$ features, we assigned a population of $q = 12$ input neurons to encode each feature, resulting in a total input layer size $N_1 = 48$. For Wisconsin, with $n_f = 9$ features, a population of $q = 7$ neurons per feature was assigned, resulting in $N_1 = 63$ input neurons. As usual, one output neuron was assigned to each input class, and for both Iris and Wisconsin we implemented a hidden layer size of 20. Hence, the SNN structures were described by $48 \times 20 \times 3$ and $63 \times 20 \times 2$ for Iris and Wisconsin, respectively. At the start of each experimental run for Iris and Wisconsin, output weights were initialized with values drawn from a uniform distribution over $[0, 2]$. Hidden weights were initialized according to uniform distributions over $[0, 4]$ and $[0, 2.2]$ for Iris and Wisconsin, respectively. For both datasets the softmax scale parameter of Equation (5) was set to $\nu = 2$. With respect to network training, stratified three-fold cross-validation was used to obtain more accurate estimates for the network performance. Data samples were presented to the network as mini-batches, and one epoch of training corresponded to a complete sweep over all unique training samples presented in this way. The regularization parameter was set to $\lambda_0 = 10^{-3}$ and the RMSProp coefficient: $\eta_0 = 0.1$. In all cases, hidden and output weights were constrained to values in the range $[-15, 15]$.

For both Iris and Wisconsin the trained SNNs demonstrated success in fitting the data (**Figures 6A,B**), with final training accuracies of 99.88 ± 0.04 and $98.04 \pm 0.04\%$ after 100 epochs, respectively. In terms of their generalization to test data, it was necessary to impose early-stopping to prevent overfitting. From multiple runs of the experiment, we determined the ideal training cut-off points to be approximately 30 and 6 epochs for Iris and Wisconsin, respectively. Since the number of weight update iterations/mini-batches per epoch was just one for Iris and four for Wisconsin, the equivalent number of iterations to early-stopping were 30 and 24, respectively. From the networks' confusion matrices (**Figures 6C,D**), evaluated at the point of early-stopping, the test accuracies were 95.2 ± 0.2 and $97.12 \pm 0.08\%$ for Iris and Wisconsin, respectively. As expected,

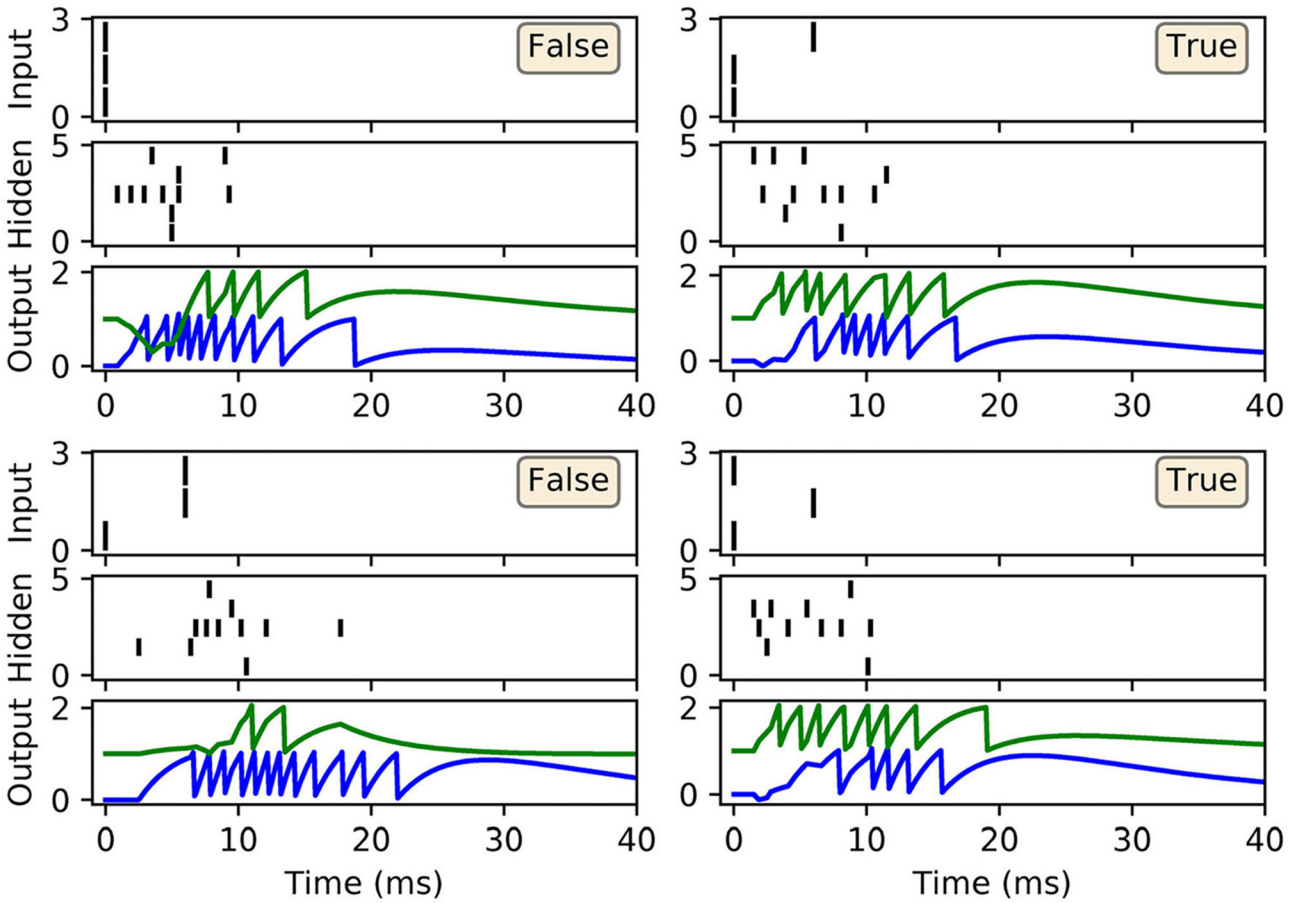

FIGURE 5 | Spike rasters depicting network activity on the XOR task, with the same setup as in **Figure 4**, after 200 epochs of training. Each of these four main panels depicts network activity in response to one of the four possible XOR input patterns, where the associated class label is indicated as either "True" or "False." Each main panel consists of three subplots: the top and middle subplots correspond to input and hidden spike patterns, respectively, where vertical lines indicate spikes. The bottom subplot displays the output layer voltage traces, where firing times are indicated by sharp discontinuities in these traces. Of the two output neurons, the first (blue trace) signals the class label "False," and the second (green trace) signals "True." In this example, all of the input patterns were correctly identified.

the matrix for Iris indicated the relative challenge in separating the latter two, linearly non-separable classes. Furthermore, and as desired, the incidence of null predictions formed by the trained networks was kept to a minimum; in most cases a null prediction corresponded to a lack of firing activity in the network, typically preventing weight gradient computations, however this issue was mitigated by the addition of synaptic scaling in order to drive sufficient neuronal activation.

For comparison purposes, we also evaluated the performance of shallow multilayer perceptrons (MLPs) on these two datasets using the Python package Scikit-Learn (Pedregosa et al., 2011): containing the same number of hidden and output layer neurons as the corresponding SNNs, but instead four and nine input neurons to represent standardized Iris and Wisconsin features, respectively. Hence, using logistic hidden activations and minimizing cross-entropy loss via the "adam" adaptive learning procedure, the resulting MLP test accuracies were 95.9 ± 0.2 and $97.0 \pm 0.1\%$ for Iris and Wisconsin, respectively, on par with our first-to-spike classifier model. The MLP classifiers were trained for a maximum of 1,600 epochs, and results averaged from 40 repetitions of stratified three-fold cross-validation. The number of epochs required for convergence were around 1,000 and 100 for Iris and Wisconsin, respectively. By comparison with existing spike-based learning algorithms, our test accuracies are competitive: falling within 1 % of several reported in the literature (Bohte et al., 2002; Sporea and Grüning, 2013; Tavanaei and Maida, 2019) but achieved in a fewer number of epochs.

In terms of the dynamics of our first-to-spike model, the spiking activity of the SNNs in response to selected data samples from the two datasets, upon early-stopping during training, is shown in **Figure 7**. It is clear for each of the presented samples that input spikes were confined to the first 10 ms, which prompted phasic activity in the hidden layer. With respect to the Iris samples (**Figure 7A**), the network formed rapid, and correct first-spike responses in the output layer: in this case within just 10 ms. Due to the parameterization of the learning rule, firing responses generated by the remaining neurons in the output layer were not completely eliminated: since these other neurons fired with sufficiently delayed onset, their resulting contribution to the output error signals used to inform weight updates were

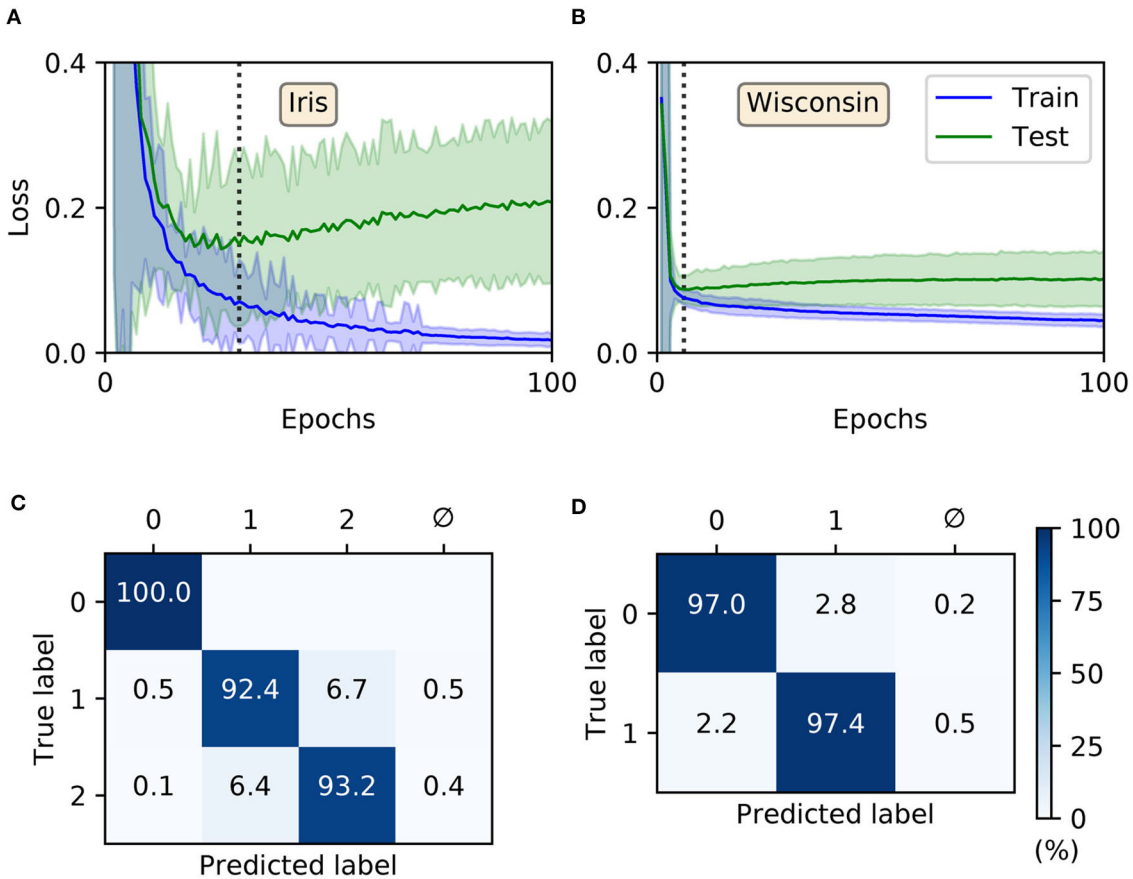

FIGURE 6 | Training multilayer SNNs to classify the Iris and Wisconsin datasets using first-to-spike decoding. Iris and Wisconsin contain three and two pattern classes, respectively, with associated network structures: $48 \times 20 \times 3$ and $63 \times 20 \times 2$. **(A,B)** Evolution of the average training/test loss of the network on the two datasets, where the shaded region indicates the standard deviation. The vertical dashed line in each panel indicates the point at which early-stopping was applied, corresponding to the minimum recorded value in the test loss. Early-stopping was taken at 30 and 6 epochs for Iris and Wisconsin, respectively, which is equivalent to 30 and 24 (weight update) iterations. These results were obtained from 40 independent runs. **(C,D)** Confusion matrices of the SNNs, post-training on Iris **(C)** and Wisconsin **(D)**. The values report the percentage test accuracy, evaluated at the moment when early-stopping was applied. Iris data samples belonging to classes labeled "1" and "2" are linearly non-separable from each other, and for Wisconsin the labels "0" and "1" correspond to "benign" and "malignant," respectively. The null symbol, ϕ, indicates that no clear prediction was formed by the network, being a consequence of either no output spiking or more than one output neuron sharing the same first spike response time (within the time resolution, δt).

minimal. This behavior was encouraged, given that it minimized data overfitting and prevented unstable dynamics arising due to competition between the backpropagation and synaptic scaling components of the learning rule (c.f. Equation 32). As with Iris, the network generated desired, rapid first-spike responses in the output layer when responding to Wisconsin data samples (**Figure 7B**). In this example the Wisconsin-trained network formed correct predictions on the two selected samples, and interestingly a ramp-up in both hidden and output layer activity was observed for the malignant-labeled sample in order to shift the desired first-spike response earlier.

3.3. Sensitivity to the Learning Schedule

The previous experiments have demonstrated the performance of the first-to-spike classifier rule using optimal parameter selections of the learning coefficient η_0, used as part of the definition of RMSProp (see Equation 35). The sensitivity of the rule to its learning schedule is an important consideration regarding its versatility and application to unfamiliar data, therefore we tested its robustness when swept over a wide range of η_0 values.

As the test case we used the Iris dataset, with the same temporal encoding procedure and network setup as described in the previous section. As previously, stratified three-fold cross-validation was used to estimate the test loss during training. The regularization parameter was set to $\lambda_0 = 10^{-3}$, and weights were constrained to values between $[-15, 15]$. Each epoch corresponded to one iterative weight update procedure, using a mini-batch size of 100. The network was trained for a total of 150 epochs for each η_0 selection, where at the end of each run we identified the minimum value in the recorded test loss and its associated number of epochs. The minimum number of epochs was determined by finding the first point at which the average test loss fell within 1% of its lowest value, and its error

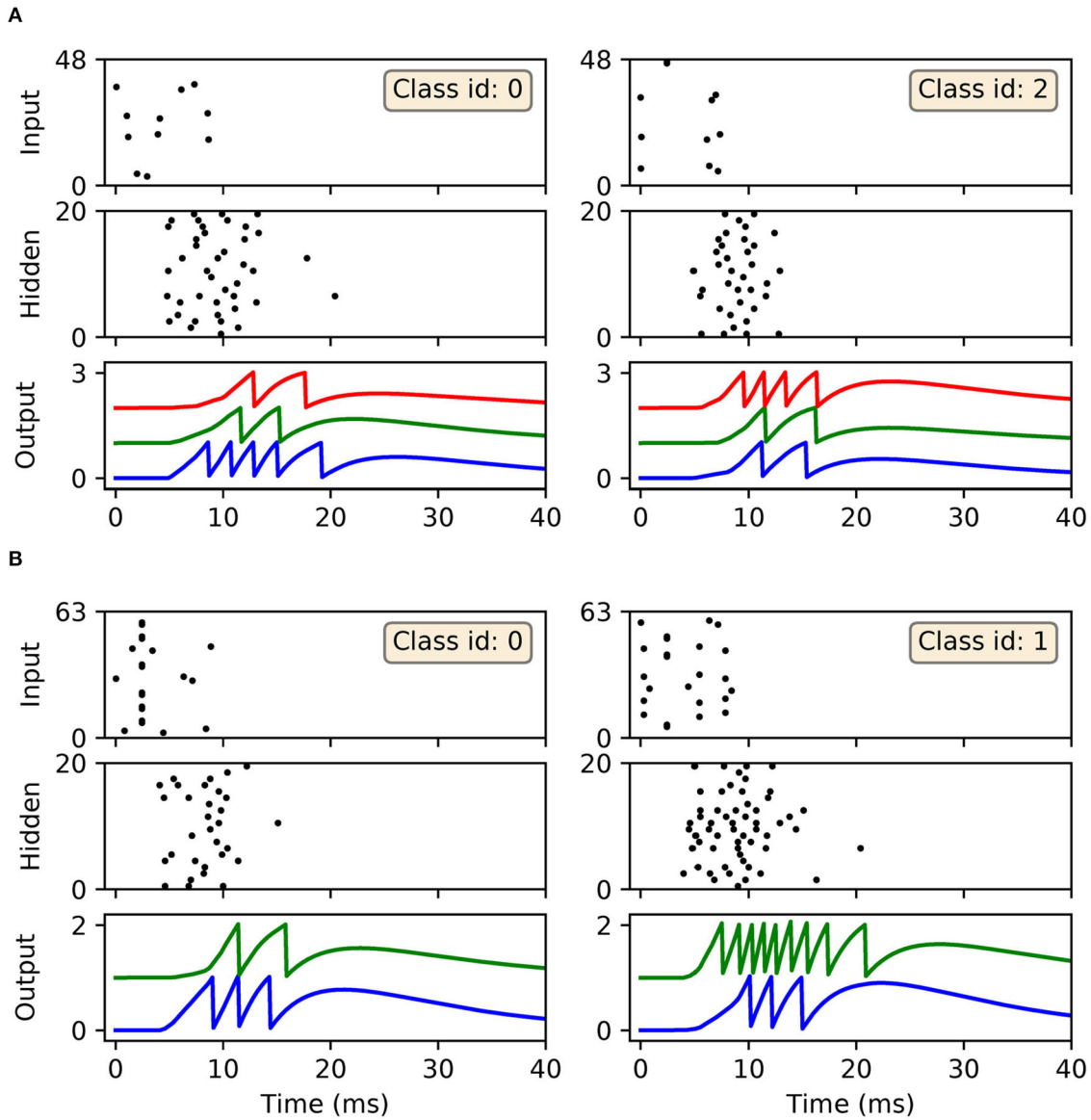

FIGURE 7 | Spike rasters depicting network activity in response to selected Iris and Wisconsin data samples, corresponding to the experiment in **Figure 6** for networks trained with early-stopping applied. **(A)** The left and right main panels depict typical network responses to Iris samples belonging to the first and last classes, respectively. Both panels show spiking activity in the input, hidden and output layers of a trained network. **(B)** The left and right main panels depict network responses to benign (class id: 0) and malignant (class id: 1) Wisconsin samples, respectively. For both **(A,B)** desired first-spike responses in the output layer were observed, resulting in correct input classifications.

was estimated based on the margin from falling within 10 % of the lowest value.

The minimum test loss attained, including the associated number of training epochs, is shown in **Figure 8** for selections of η_0 between 10^{-2} and 10^{1}. From these results, it follows that a learning coefficient with a value of around 10^{-1} provided a reasonable trade-off between network performance and learning speed: larger η_0 values returned sub-optimal test loss minima, while smaller values led to an exponential increase in the training time with little change in the performance. Extended parameter sweeps also indicated similar behavior with respect to the Wisconsin dataset as the test case. For these reasons we were motivated to select an optimal value of $\eta_0 = 10^{-1}$ for both Iris and Wisconsin.

These observations support our selection of RMSProp, as opposed to a fixed learning rate which demonstrated greater sensitivity to its parameter choice. In addition to the above, we found that RMSProp's learning coefficient exhibited a

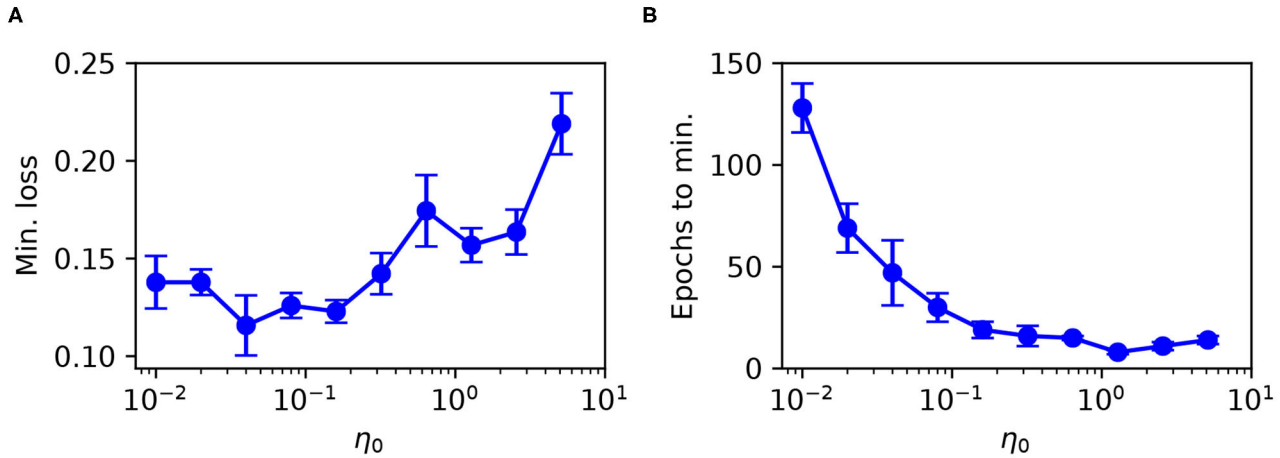

FIGURE 8 | Parameter sweep over the RMSProp learning coefficient η_0, evaluated for an SNN trained on Iris with the same network structure and training procedure as in **Figure 6**. **(A)** Minimum recorded test loss as a function of η_0, taken as the median value, where error bars indicate the SEM. **(B)** The associated number of training epochs needed to reach the minimum test loss. These results were gathered from 20 independent runs.

dependence on the number of input synapses per network layer, such that more optimal performance was attained by adjusting η_0 proportional to $1/N_l$. This was used to inform our choice of η_0 used for the MNIST dataset.

3.4. Classifying MNIST: Latency Encoding

The MNIST dataset of handwritten digits (LeCun et al., 1998) is commonly used to benchmark a classifier system's performance, owing to the relatively high structural complexity of the data and the large number of training and test samples. Although this problem is largely solved using deep convolutional neural networks, MNIST still poses a challenge to solve using spike-based computational approaches. For this reason, we apply our first-to-spike classifier rule to solving MNIST in order to get an indication of its potential for real world data classification.

MNIST consists of 60 and 10k training and test samples, respectively, where each sample corresponds to a 28×28, 8-bit grayscale image depicting a handwritten digit between 0 and 9. In order to make these real-valued images compatible with spike-based processing we applied a latency encoding strategy: forming a one-one association between each input pixel and an encoding LIF neuron. In this way, each image, consisting of 784 pixels, was transformed into 784 single-spike latencies presented by the first layer of a multilayer SNN. Specifically, and as described in section 2.3, the pixel values were transformed into current intensities using the scaling factor $I_{\max} = 20\,\text{nA}$, resulting in the following pixel-to-latency mapping: $[84, 256] \mapsto [9, 2]\text{ms}$, where pixel values less than 84 were insufficient to elicit an encoded response. In terms of network structure, the simulated SNNs consisted of $784 \times N_2 \times 10$ neurons, where the number of hidden neurons, N_2, was varied, and the number of output neurons was matched to the 10 digit classes. According to the first-to-spike decoding strategy, the first output neuron to respond with a spike predicted the input sample's class. The network weights were initialized by drawing hidden and output values from uniform distributions over the ranges: $[0, 0.4]$ and $[0, 32/N_2]$, respectively. The softmax scale parameter was set to $\nu = 4$ in order to tighten the conditional probability distribution of class label predictions formed by the output layer: this choice was supported by preliminary simulations, where it was found to boost the discriminative power of the SNN when handling a larger number of input classes. In terms of network training, at the start of each run 600 of the MNIST training samples were set aside in a stratified manner for validation. The remaining training samples were then iteratively presented to the network as mini-batches, with a total of 4,000 iterations. To get an indication of the network's performance during training the loss on the validation data was computed every 20 iterations. The regularization parameter and RMSProp coefficient were set to $\lambda_0 = 10^{-4}$ and $\eta_0 = 0.01$, respectively. Throughout training, all weights were constrained to values in the range $[-2, 2]$ to avoid overfitting.

As shown by **Figure 9**, the trained SNNs were capable of generalizing from the MNIST training samples to the withheld test samples, with a highest recorded test accuracy of $89.4 \pm 0.4\%$ for a network containing 160 hidden layer neurons. With the given selection of regularization parameters and weight constraints, model overfitting was minimized and smooth convergence was observed within the maximum number of training iterations (**Figure 9A**). Moreover, as the hidden layer size was increased, a speedup in the learning rate became apparent. As indicated by **Figure 9B**, the accuracy of the network approached an asymptotic value of just under 90 % when containing up to 160 neurons. The confusion matrix depicted in **Figure 9C** corresponds to a network containing 160 neurons, and provides some insight into the robustness of network classifications with respect to each of the presented MNIST test digits. As expected, digits "zero" and "one" were least challenging for the network to identify by virtue of their distinctive features,

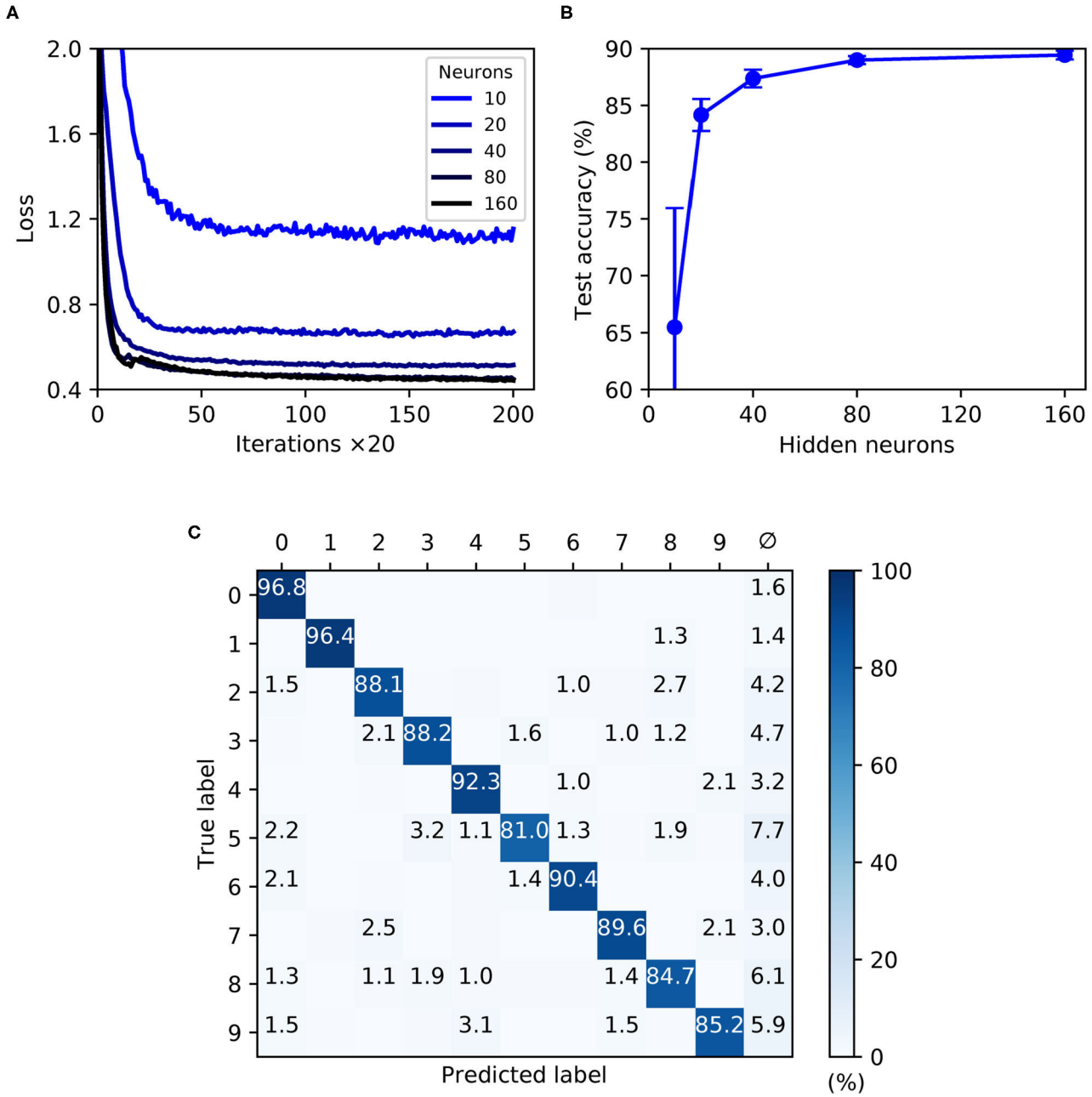

FIGURE 9 | Multilayer SNNs trained on the MNIST dataset, using a latency encoding strategy to transform pixels into single, precisely-timed input spikes. **(A)** Network loss as a function of the number of training iterations, evaluated using a validation set. Each curve corresponds to a different number of hidden layer neurons, N_2, from between 10 (lightest color) and 160 (darkest color). **(B)** The final accuracy of the network after 4,000 training iterations, as evaluated on the 10 k test samples. **(C)** Confusion matrix of an SNN containing 160 hidden neurons, as evaluated on the test samples at the end of training. Values <1 % are not indicated. These results were averaged from 10 independent runs.

whereas the digit "five," for example, tended to share a greater feature overlap with several other digits, making it somewhat more difficult to discriminate. Furthermore, in the event of a digit not being recognized by the network there was a tendency toward a null prediction (no output spikes) being returned rather than an erroneous digit; technically this is a preferable outcome, since it reduces the likelihood of false-positives by the network.

In summary, the first-to-spike classifier rule has demonstrated generalization capability on the MNIST dataset to a reasonable degree of accuracy. Although the best performing shallow MLP, containing 800 hidden units, is capable of 98.4 % accuracy

(Simard et al., 2003), MNIST presents more of a challenge with spike-based implementations. Despite not reaching a level that is state-of-the-art, our accuracy nevertheless approaches that obtained by another probabilistic, spike-based formulation that achieved around 92 % accuracy (Neftci et al., 2014). We also indicate that the results reported here do not reflect an upper bound on the classifier's MNIST performance, and with further parameter tuning and feature preprocessing further accuracy gains would be expected; as demonstrated by Mostafa (2017) and Kheradpisheh and Masquelier (2020), it is possible to attain high accuracies of around 97 % using a first-to-spike decoding scheme, although currently this comes with the caveat of only considering single spike responses of hidden neurons. For simplicity, this experiment considered a straightforward one-one mapping between each input pixel and encoding neuron in order to transform the data, although such a scheme is computationally prohibitive for spike-based processing and fails to fully exploit the precise timings of individual spikes. Utilizing a fully temporal encoding strategy presents the next challenge, and is addressed in the following section.

3.5. Classifying MNIST: Scanline Encoding

So far, the technical capability of the first-to-spike classifier rule has been demonstrated on MNIST when encoded using a one-one association between input pixels and single-spike latencies. This encoding strategy is somewhat simplistic, however, and fails to take full advantage of the precise timings of multiple spikes as a means to perform dimensionality reduction on the data. To address this we consider an alternative encoding strategy, termed scanline encoding, that extends on the work of Lin et al. (2018), and enables more compact feature representations of the MNIST digits using substantially fewer encoding neurons.

In order to transform the real-valued features of MNIST into sequences of precisely-timed spikes, we applied the scanline encoding strategy as described in 2.3. In summary, at the start of each simulation run we implemented a variable number of scanlines, n_s, ranging between 8 and 64. Each of these scanlines had an arbitrarily determined orientation, independent of the others, and was constrained to intersect through a point close to the center of the image space. For the duration of each run these scanlines were held fixed, and in response to each presented sample a scanline read-in its sequence of input pixels and returned a time-varying current; this current in turn acted as the stimulus for an encoding LIF neuron in the first layer of an SNN, driving a spike train response (**Figure 3** illustrates this scanning process for an example image). Hence, the strategy we employed here was capable of transforming high-dimensional images into compact spatiotemporal spike patterns, whilst still retaining characteristics of their spatial organization. With respect to the network structure, SNNs consisting of $N_1 \times N_2 \times 10$ neurons were simulated, where the number of input neurons was matched to the number of scanlines used, $N_1 = n_s$, and the number of hidden neurons N_2 was varied. As previously, the number of output neurons was matched to the 10 different digit classes of MNIST, and first-to-spike decoding was used to classify the data samples. In terms of network connections, two distinct modeling approaches were considered regarding input-to-hidden layer spike propagation: "delayless" and "delayed." In the delayless case, spikes were instantaneously transmitted from input to hidden neurons, as has been implemented so far for all the previous experiments. In the delayed case, however, spikes transmitted from input to hidden neurons were subject to propagation delay: ranging from between 1 and 10 ms, rounded to the nearest millisecond. At the start of each experimental run, these propagation, or conduction, delays were randomly drawn from a uniform distribution for each input-to-hidden connection, and held fixed thereafter. In all cases hidden-to-output layer spike conduction was delayless. The purpose of simulating conduction delays was to determine if this could assist a network in linking early/late spike arrival times arising from scanline-encoded digits. With respect to weight initialization, hidden and output weights were initialized according to uniform distributions with values ranging between $[0, 40/N_1]$ and $[0, 32/N_2]$, respectively. The softmax scale parameter was set to $\nu = 4$. For each run, a network was trained and validated in a similar way to the latency encoding experiment: 600 MNIST training samples were set aside for validation every 20*th* iteration, and the remaining samples were iteratively presented as mini-batches for 1,600 iterations. The regularization parameter and RMSProp coefficient were set to $\lambda_0 = 10^{-4}$ and $\eta_0 = 0.05$, respectively. All of the network weights were constrained to values in the range $[-6, 6]$.

Shown in **Figure 10** are results obtained from trained SNNs, with and without conduction delays between the input and hidden layers, for scanline-encoded MNIST samples. It can be seen that in all cases the trained networks converged in learning within the maximum 1,600 iterations, with an additional small speedup as the number of hidden neurons was increased from 10 to 160 (**Figures 10A,B**). It is also apparent that the inclusion of conduction delays reduced the final loss values, enabling the network to approach the same values as reported for latency-encoded MNIST (compare with **Figure 9A**). In terms of the post-training performance on the withheld test samples, the highest attained accuracies were $76 \pm 6\%$ and $87 \pm 2\%$ for delayless and delayed networks, respectively, as evaluated using 160 hidden neurons and between 32 and 64 scanlines (**Figures 10C,D**). From the gathered results, there was diminishing returns in the final test accuracy as the number of neurons and scanlines were increased beyond 160 and 32, respectively. Comparatively, the test accuracy returned by the best performing network with conduction delays fell within just 3% of that obtained based on a latency encoding strategy, but with the advantage of using an order of magnitude fewer input neurons.

The confusion matrix of a post-trained $32 \times 160 \times 10$ network with conduction delays, as evaluated on the withheld MNIST test samples, is depicted in **Figure 11**. These results correspond to a high performing case from **Figure 10**. As expected, the network demonstrated the least difficulty in recognizing the digits "zero" and "one," closely followed by "six." However, the network tended to confuse the digits "four" and "nine" with relatively high frequency, owing to their closer similarities. By comparison with the confusion matrix from **Figure 9C**, the overall percentage of null predictions by the networks for both encoding strategies were consistent.

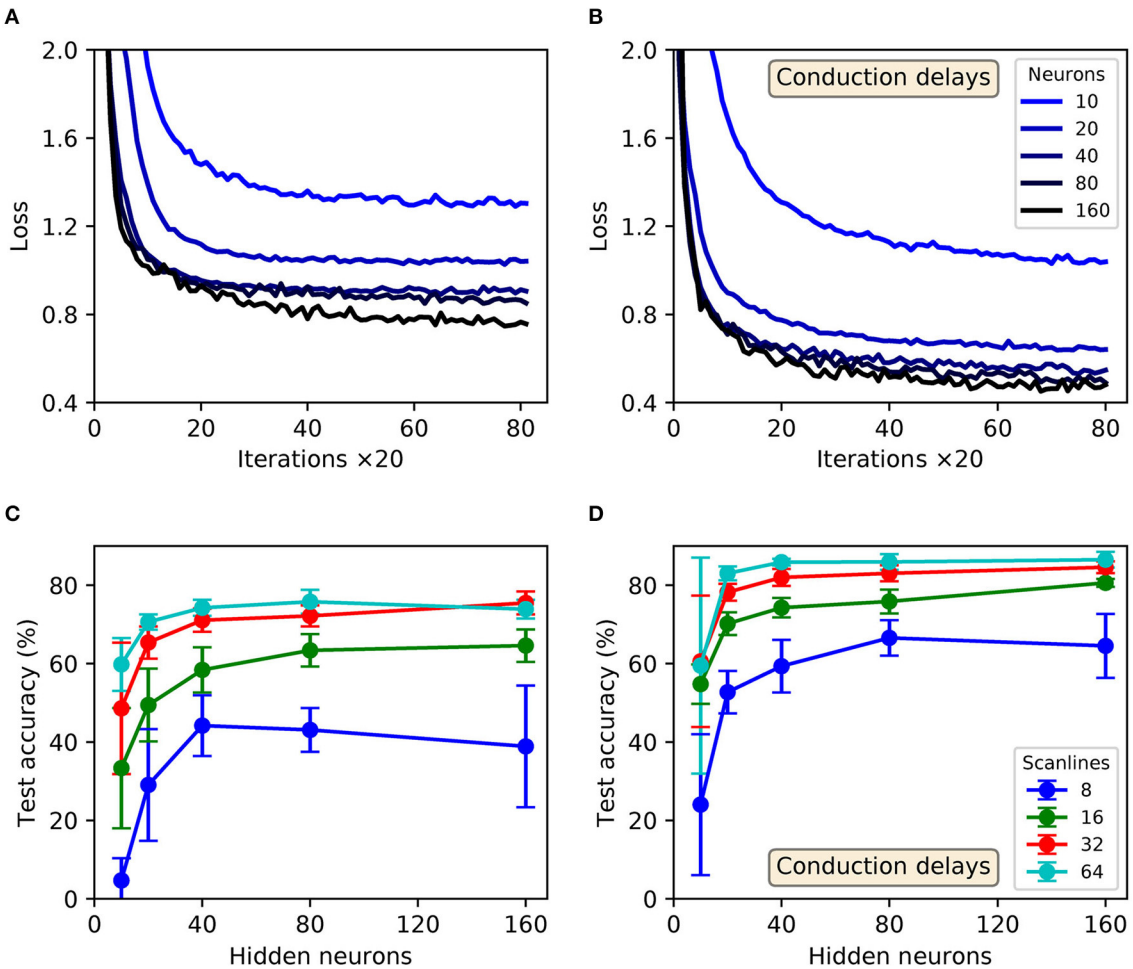

FIGURE 10 | Multilayer SNNs, with or without input-to-hidden layer spike-propagation (conduction) delays, trained on the MNIST dataset. Scanline encoding was used to transform images into compact spatiotemporal spike patterns. **(A,B)** Network losses for 32 scanlines as a function of the number of training iterations, evaluated using a validation set. **(A,B)** Correspond to "delayless" and "delayed" networks, respectively: referring to the inclusion of conduction delays between the input and hidden layers. For each panel, the curve color corresponds to a certain number of hidden layer neurons, N_2: from between 10 (lightest color) and 160 (darkest color). **(C,D)** The final accuracies of the network after 1,600 training iterations, as evaluated on the 10k test samples. **(C,D)** Correspond to delayless and delayed networks, respectively. These results were averaged from 10 independent runs.

Despite this, networks utilizing scanline encoding gave rise to more variable predictions between experiment runs: the coefficients of variation with respect to correct predictions was 0.08 ± 0.03 and 0.02 ± 0.01 for scanline- and latency-based encoding, respectively; this discrepancy was attributed to the random selection of scanline orientations between runs, whereas latency-based representations of the same input samples remained fixed.

For illustrative purposes, an example of a scanline-encoded digit and the response it elicited in a post-trained SNN is shown in **Figure 12**. A sample of the digit "one," which was withheld during network training, was transformed into sequences of precisely-timed spikes, and represented by the first layer of neurons in a minimal $32 \times 40 \times 10$ network with input-to-hidden layer conduction delays. In this example, the network correctly identified the input sample by driving the corresponding output neuron to respond first with a spike. As indicated by **Figure 12A**, most of the feature space was covered by the 32 scanlines: an increase in this number resulted in diminishing returns, relating to feature redundancy. In terms of the spike raster subplots in **Figure 12B**, there is a relatively large offset in the emergence of hidden spiking with respect to the onset of input spikes, whereas there is a large degree of overlap between hidden-and-output spiking. This reflects the delayed propagation of input-to-hidden spikes, but which affords the network time to link early and late input spikes in order to inform its final decision. The activity of neurons in the output layer tended to be greater than that observed with the other datasets (compare with **Figure 7**), indicative of the increased complexity in learning to discriminate between a larger number of classes with more feature overlap.

To summarize, this section has demonstrated a novel methodology to training SNNs on MNIST that are more

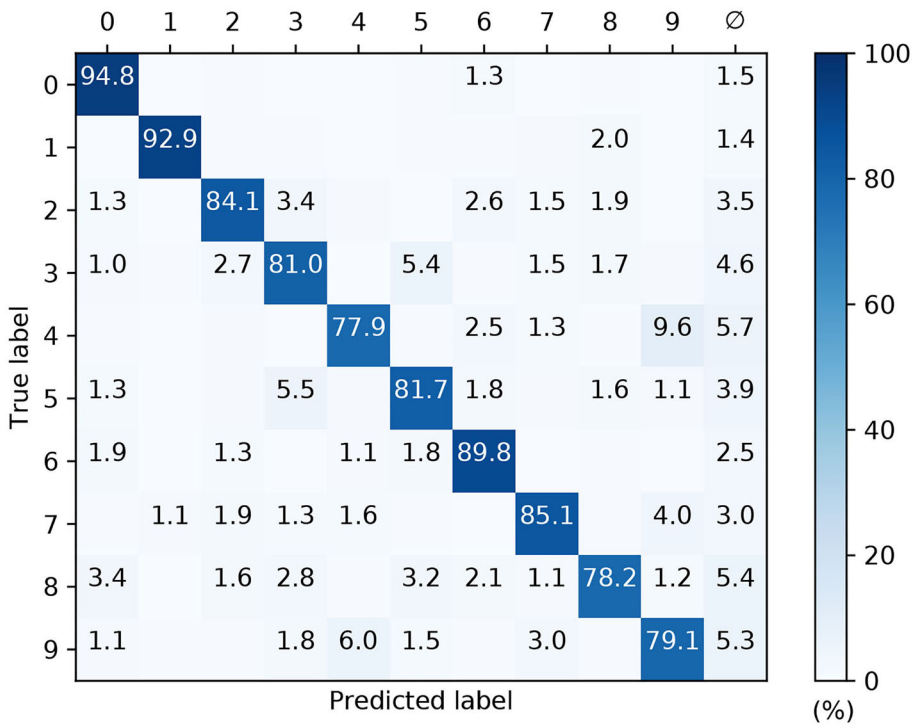

FIGURE 11 | Confusion matrix of an SNNs with input-to-hidden layer conduction delays, after 1,600 training iteration on MNIST encoded using 32 scanlines. This was evaluated on the withheld test samples. The network contained 160 hidden neurons. Values <1 % are not indicated. These results were averaged from 10 independent runs.

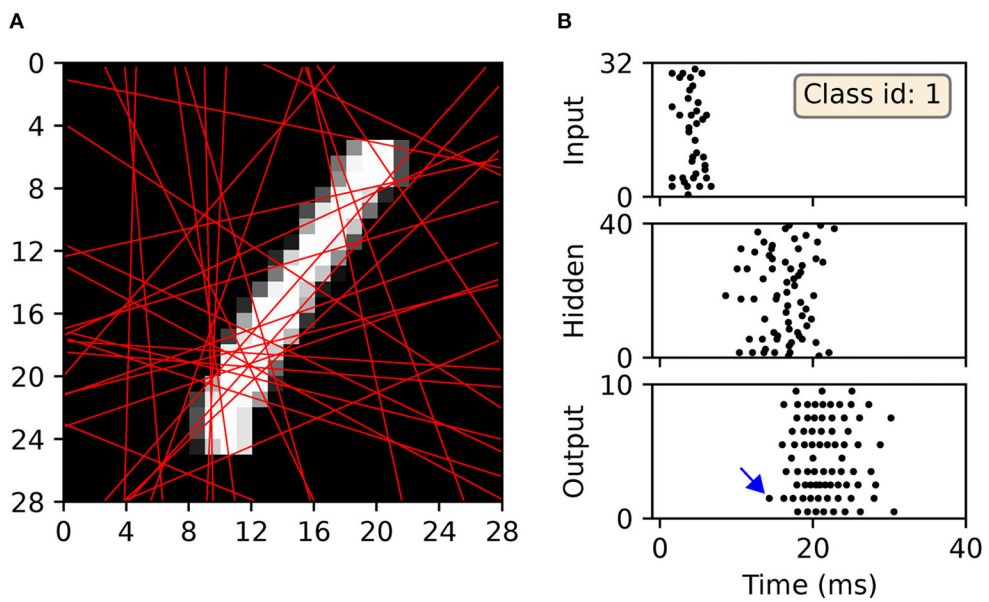

FIGURE 12 | Illustration of the encoding and subsequent classification of the MNIST digit "one" based on scanlines. In this example, the digit is first transformed into spike trains via scanline encoding, before being processed by a previously trained SNN containing 40 hidden neurons and input-to-hidden layer conduction delays. **(A)** The 32 scanlines encoding the digit "one" (red lines). **(B)** Spike raster of the network's response to the encoded digit. The top, middle, and bottom subplots correspond to input, hidden and output spike patterns, respectively. In this case, the first neuron to respond with a spike (indicated by the blue arrow) corresponds to the desired class label, resulting in a correct classification.

constrained in their size, and yet more efficient in terms of their spike-based processing. This has been realized by the application of scanline encoding: a feature preprocessing method that can transform high dimensional images into compact spatiotemporal spike patterns. In particular, the results obtained using this method provided a test accuracy of almost 90 %: close to what we obtained for the more computationally expensive one-one encoding scheme. Relying more on the precise timings of individual spikes for data classification massively reduces the number of encoding neurons required, and could find important applications in constrained network architectures, for example in neuromorphic systems like Heidelberg's HICANN-DLS device (Friedmann et al., 2017). It is expected that the performance of this method could be improved upon by making scanline encoding more domain specific: for example by optimizing scanline orientations prior to network training, rather than setting them arbitrarily. Our choice of a random initialization, however, indicates the potential in transferring this method to unfamiliar problem domains.

4. DISCUSSION

In this article we have introduced a new supervised approach to training multilayer spiking neural networks, using a first-to-spike decoding strategy, with the proposed learning rule capable of providing robust performance on several benchmark classification datasets. The learning rule extends on our previous formulation in Gardner et al. (2015) by redefining the network's cost function to depend on the distribution of first spike arrival times in the output layer, rather than entire spike trains, and specifying the target signal according to which one of c output neurons should be driven to fire first; this redefinition of the cost function is particularly advantageous for data classification purposes since it places much less of a constraint on the network's parameters during training, thereby avoiding overfitting of the data. Furthermore, restricting our focus to just first-spike arrival times in the output layer has allowed us to largely reduce the runtime of simulations: an important consideration when taking into account the relatively high computational cost in simulating spike-based models. Based on first-to-spike decoding, pattern classification was rapid: with predictions in some cases made within just 10 ms of pattern onset. Such a decoding strategy has similarly been used to good effect in Mostafa (2017), Bagheri et al. (2018), and Kheradpisheh and Masquelier (2020), and moreover avoids the ambiguity of decision making based on comparisons between entire target and actual output spike trains as used in Bohte et al. (2002), Florian (2012), Sporea and Grüning (2013), and Gardner et al. (2015). We highlight the novel, hybrid nature of our learning model: which implements both deterministic, LIF-type output neurons for more reliable network responses, and stochastic hidden layer neurons that should aid with its regularization.

The formulation of our learning rule combines several different techniques, as found in Bohte et al. (2002), Pfister et al. (2006), Gardner et al. (2015), and Mostafa (2017). As our first step, we selected the network's cost function as the cross-entropy, dependent on the distribution of first-spike arrival times in the output layer, and set the target signal according to the index of the neuron associated with the correct class (Mostafa, 2017). Subsequently, and in order to apply the technique of spike-based backpropagation, we estimated the gradients of deterministically-generated output firing times by following the linear approximation used in Bohte et al. (2002); our choice here was motivated by simplicity reasons, since only single, first-spikes in the output layer were required for parameter optimization. We then applied our previously described probabilistic method to solving hidden layer spike gradients for stochastic neurons (Gardner et al., 2015), which supports multiple firing times and is analytically tractable (Pfister et al., 2006). Our decision to combine first-to-spike decoding with probabilistic, multi-spike trains was chiefly driven by the novelty of this approach; by comparison, other first-to-spike multilayer learning algorithms have sacrificed full sequences of hidden spikes and selected simplified neuron models in order to establish backpropagation rules which can be applied recursively for deep learning purposes (Mostafa, 2017; Kheradpisheh and Masquelier, 2020). Although our choice of increased complexity comes at a performance cost on the more challenging MNIST dataset, we still indicated the merits of our approach when implemented using a constrained network architecture, which has potential application for low-energy neuromorphic processing tasks.

In terms of the considered experiments, we first demonstrated the learning rule to be capable of solving the non-trivial XOR classification task: establishing its ability to classify linearly non-separable data, for which a hidden layer is required. Subsequently, the rule was found to be highly accurate in classifying more challenging data samples belonging to the Iris and Wisconsin datasets, and for which two of the Iris classes are linearly non-separable. In particular, we found that implementing the RMSProp learning schedule (Hinton et al., 2012) made the network less sensitive to the choice of learning coefficient η_0, suggesting it as an effective mechanism to minimizing the process of parameter fine-tuning; this in turn increased the flexibility of the rule as applied to different datasets. Additionally, we found in general that regularizing the network by penalizing high neuronal firing rates resulted in improved generalization ability and accuracy, confirming the observations of Zenke and Ganguli (2018). Suppressing high firing rates also reduced the number of computational operations in the run simulations, since fewer hidden spikes were integrated over when computing the iterative weight updates, thereby enabling a large speedup in runtime. With respect to the rule's performance on MNIST, the test accuracy didn't reach state-of-the art: with our rule achieving around 90 %. It is noted, however, that attaining high accuracy on MNIST, including other structurally complex datasets such as Fashion-MNIST and ImageNET (Deng et al., 2009; Xiao et al., 2017), via spike-based processing still poses more of a challenge compared with traditional ANN approaches. Many existing spike-based supervised learning methods have achieved accuracies ranging between 90 and almost 99 % on MNIST, with the highest levels relating to deeper SNN architectures and convolutional filtering optimized for image processing tasks

(O'Connor et al., 2013; Neftci et al., 2014; Diehl et al., 2015; Lee et al., 2016; Mostafa, 2017; Tavanaei and Maida, 2019; Kheradpisheh and Masquelier, 2020; Zenke and Vogels, 2020). In our approach, we tested minimal SNN architectures with their flexibility and transferability to constrained neuromorphic hardware platforms such as HICANN-DLS (Friedmann et al., 2017) in mind. Intriguingly, however, we found that our rule was still capable of achieving almost 90 % accuracy on MNIST when using our novel scanline encoding method, as inspired by Lin et al. (2018), and relying on as little as 32 encoding input neurons. We also note that these results represent a lower bound on what is achievable using our learning method: with model refinement in the form of extended hyper-parameter fine-tuning and deeper network architectures, it is expected that the final accuracies could be further increased.

We note that learning in recurrent SNN architectures is emerging as an active focus of research, and recent work has managed to achieve high performance on a challenging phoneme recognition task based on spiking computations with state-holding behavior (Bellec et al., 2020). This work introduced the e-prop learning algorithm for recurrent SNNs, and approximates backpropagation through time (BPTT) by modulating local, candidate weight changes with top-down learning signals in order to efficiently learn temporal processing tasks; BPTT refers to the typical process by which a recurrent network is trained, and involves "unrolling" the network into an equivalent feedforward one for the purposes of determining the gradients at each time step. Potentially, the learning rule we proposed here might be extended to work with recurrent network structures: for instance by combining BPTT with our weight-gradient calculations for hidden neurons, but using a simplified approximation of these gradients, or pseudo-gradients, for analytical tractability. This would be an interesting challenge to address, and might indicate a role for first-to-spike decoding in the context of learning sequence classification tasks.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The code and examples of our implementation are available on GitHub at https://github.com/BCGardner/snn-classifier. The datasets analyzed in this article are accessible from https://archive.ics.uci.edu and http://yann.lecun.com/exdb/mnist.

AUTHOR CONTRIBUTIONS

BG and AG conceptualized the study, developed the theoretical formalism, planned the experiments, and analyzed the results. BG wrote the software and carried out the experiments. BG wrote the article in consultation with AG. All authors contributed to the article and approved the submitted version.

FUNDING

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2).

ACKNOWLEDGMENTS

This article has been released as a preprint at: https://arxiv.org/ (Gardner and Grüning, 2020).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fncom.2021.617862/full#supplementary-material

REFERENCES

Albers, C., Westkott, M., and Pawelzik, K. (2016). Learning of precise spike times with homeostatic membrane potential dependent synaptic plasticity. *PLoS ONE* 11:e0148948. doi: 10.1371/journal.pone.0148948

Bagheri, A., Simeone, O., and Rajendran, B. (2018). "Training probabilistic spiking neural networks with first-to-spike decoding," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Calgary: IEEE), 2986–2990. doi: 10.1109/ICASSP.2018.8462410

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* 11:3625. doi: 10.1038/s41467-020-17236-y

Bi, G.-Q., and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Bohte, S. M., Kok, J. N., and Poutré, H. L. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Booij, O., and Nguyen, H. T. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Inform. Process. Lett.* 95, 552–558. doi: 10.1016/j.ipl.2005.05.023

Brea, J., Senn, W., and Pfister, J.-P. (2013). Matching recall and storage in sequence learning with spiking neural networks. *J. Neurosci.* 33, 9565–9575. doi: 10.1523/JNEUROSCI.4098-12.2013

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "ImageNet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL: IEEE), 248–255. doi: 10.1109/CVPR.2009.5206848

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, (Killarney: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280696

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Ann. Eugen.* 7, 179–188. doi: 10.1111/j.1469-1809.1936.tb02137.x

Florian, R. V. (2012). The chronotron: a neuron that learns to fire temporally precise spike patterns. *PLoS ONE* 7:e40233. doi: 10.1371/journal.pone.0040233

Frémaux, N., Sprekeler, H., and Gerstner, W. (2013). Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS Comput. Biol.* 9:e1003024. doi: 10.1371/journal.pcbi.1003024

Friedmann, S., Schemmel, J., Grübl, A., Hartel, A., Hock, M., and Meier, K. (2017). Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Trans. Biomed. Circ. Syst.* 11, 128–142. doi: 10.1109/TBCAS.2016.2579164

Gardner, B., and Grüning, A. (2016). Supervised learning in spiking neural networks for precise temporal encoding. *PLoS ONE* 11:e0161335. doi: 10.1371/journal.pone.0161335

Gardner, B., and Grüning, A. (2020). Supervised learning with first-to-spike decoding in multilayer spiking neural networks. *arXiv [preprint]. arXiv:2008.06937*.

Gardner, B., Sporea, I., and Grüning, A. (2015). Learning spatiotemporally encoded pattern transformations in structured spiking neural networks. *Neural Comput.* 27, 2548–2586. doi: 10.1162/NECO_a_00790

Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511815706

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9781107447615

Ghosh-Dastidar, S., and Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Netw.* 22, 1419–1431. doi: 10.1016/j.neunet.2009.04.003

Gollisch, T., and Meister, M. (2008). Rapid neural coding in the retina with relative spike latencies. *Science* 319, 1108–1111. doi: 10.1126/science.1149639

Grüning, A., and Bohte, S. M. (2014). "Spiking neural networks: principles and challenges," in *Proceedings of the 22nd European Symposium on Artificial Neural Networks (ESANN 2014). Computational Intelligence and Machine Learning* (Bruges: ESANN).

Grüning, A., and Sporea, I. (2012). Supervised learning of logical operations in layered spiking neural networks with spike train encoding. *Neural Process. Lett.* 36, 117–134. doi: 10.1007/s11063-012-9225-1

Gütig, R. (2014). To spike, or when to spike? *Curr. Opin. Neurobiol.* 25, 134–139. doi: 10.1016/j.conb.2014.01.004

Gütig, R., and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci.* 9, 420–428. doi: 10.1038/nn1643

Gütig, R., and Sompolinsky, H. (2009). Time-warp-invariant neuronal processing. *PLoS Biol.* 7:e1000141. doi: 10.1371/journal.pbio.1000141

Hinton, G., Srivastava, N., and Swersky, K. (2012). *Neural Networks for Machine Learning*. Coursera, video lectures.

Hung, C. P., Kreiman, G., Poggio, T., and DiCarlo, J. J. (2005). Fast readout of object identity from macaque inferior temporal cortex. *Science* 310, 863–866. doi: 10.1126/science.1117593

Jang, H., Simeone, O., Gardner, B., and Gruning, A. (2019). An introduction to probabilistic spiking neural networks: probabilistic models, learning rules, and applications. *IEEE Signal Process. Mag.* 36, 64–77. doi: 10.1109/MSP.2019.2935234

Jang, H., Skatchkovsky, N., and Simeone, O. (2020). VOWEL: A local online learning rule for recurrent networks of probabilistic spiking winner-take-all circuits. *arXiv [preprint]. arXiv:2004.09416*.

Jimenez Rezende, D., and Gerstner, W. (2014). Stochastic variational learning in recurrent spiking networks. *Front. Comput. Neurosci.* 8:38. doi: 10.3389/fncom.2014.00038

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kheradpisheh, S. R., and Masquelier, T. (2020). S4NN: Temporal backpropagation for spiking neural networks with one spike per neuron. *Int. J. Neural Syst.* 30:2050027. doi: 10.1142/S0129065720500276

Kiani, R., Esteky, H., and Tanaka, K. (2005). Differences in onset latency of macaque inferotemporal neural responses to primate and non-primate faces. *J. Neurophysiol.* 94, 1587–1596. doi: 10.1152/jn.00540.2004

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Lin, C.-K., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M., et al. (2018). Programming spiking neural networks on Intel92s Loihi. *Computer* 51, 52–61. doi: 10.1109/MC.2018.157113521

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955

Memmesheimer, R.-M., Rubin, R., Ölveczky, B. P., and Sompolinsky, H. (2014). Learning precisely timed spikes. *Neuron* 82, 925–938. doi: 10.1016/j.neuron.2014.03.026

Mohemmed, A., Schliebs, S., Matsuda, S., and Kasabov, N. (2012). SPAN: spike pattern association neuron for learning spatio-temporal spike patterns. *Int. J. Neural Syst.* 22:1250012. doi: 10.1142/S0129065712500128

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biol. Cybern.* 98, 459–478. doi: 10.1007/s00422-008-0233-1

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., and Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Front. Neurosci.* 7:272. doi: 10.3389/fnins.2013.00272

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T., and Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Front. Neurosci.* 7:178. doi: 10.3389/fnins.2013.00178

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* 12, 2825–2830. Available online at: http://jmlr.org/papers/v12/pedregosa11a.html

Pfister, J.-P., Toyoizumi, T., Barber, D., and Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Comput.* 18, 1318–1348. doi: 10.1162/neco.2006.18.6.1318

Ponulak, F., and Kasiński, A. (2010). Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Comput.* 22, 467–510. doi: 10.1162/neco.2009.11-08-901

Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). "Best practices for convolutional neural networks applied to visual document analysis," in *International Conference on Document Analysis and Recognition (ICDAR), Vol. 2* (Edinburgh: IEEE), 958. doi: 10.1109/ICDAR.2003.1227801

Sporea, I., and Grüning, A. (2013). Supervised learning in multilayer spiking neural networks. *Neural Comput.* 25, 473–509. doi: 10.1162/NECO_a_00396

Tavanaei, A., and Maida, A. (2019). BP-STDP: approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* 330, 39–47. doi: 10.1016/j.neucom.2018.11.014

Thorpe, S., Delorme, A., and Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Netw.* 14, 715–725. doi: 10.1016/S0893-6080(01)00083-1

Urbanczik, R., and Senn, W. (2009). A gradient learning rule for the tempotron. *Neural Comput.* 21, 340–352. doi: 10.1162/neco.2008.09-07-605

van Rossum, M. C., Bi, G. Q., and Turrigiano, G. G. (2000). Stable Hebbian learning from spike timing-dependent plasticity. *J. Neurosci.* 20, 8812–8821. doi: 10.1523/JNEUROSCI.20-23-08812.2000

VanRullen, R., Guyonneau, R., and Thorpe, S. J. (2005). Spike times make sense. *Trends Neurosci.* 28, 1–4. doi: 10.1016/j.tins.2004.10.010

Wolberg, W. H., and Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proc. Natl. Acad. Sci. U.S.A.* 87, 9193–9196. doi: 10.1073/pnas.87.23.9193

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv [preprint]. arXiv:1708.07747*.

Yu, Q., Tang, H., Tan, K. C., and Li, H. (2013). Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *PLoS ONE* 8:e78318. doi: 10.1371/journal.pone.0078318

Zenke, F., and Ganguli, S. (2018). Superspike: Supervised learning in multilayer spiking neural networks. *Neural Comput*. 30, 1514–1541. doi: 10.1162/neco_a_01086

Zenke, F., and Vogels, T. P. (2020). The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *bioRxiv*. doi: 10.1101/2020.06.29.176925

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The reviewer SBF declared a past collaboration with one of the authors AG to the handling Editor.