The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks

Friedemann Zenke[1,2,*] and Tim P. Vogels[1]

1 Centre for Neural Circuits and Behaviour, University of Oxford, Oxford, UK.
2 Friedrich Miescher Institute for Biomedical Research, Basel, Switzerland.

* friedemann.zenke@fmi.ch

Abstract

Brains process information in spiking neural networks. Their intricate connections shape the diverse functions these networks perform. In comparison, the functional capabilities of models of spiking networks are still rudimentary. This shortcoming is mainly due to the lack of insight and practical algorithms to construct the necessary connectivity. Any such algorithm typically attempts to build networks by iteratively reducing the error compared to a desired output. But assigning credit to hidden units in multi-layered spiking networks has remained challenging due to the non-differentiable nonlinearity of spikes. To avoid this issue, one can employ surrogate gradients to discover the required connectivity in spiking network models. However, the choice of a surrogate is not unique, raising the question of how its implementation influences the effectiveness of the method. Here, we use numerical simulations to systematically study how essential design parameters of surrogate gradients impact learning performance on a range of classification problems. We show that surrogate gradient learning is robust to different shapes of underlying surrogate derivatives, but the choice of the derivative's scale can substantially affect learning performance. When we combine surrogate gradients with a suitable activity regularization technique, robust information processing can be achieved in spiking networks even at the sparse activity limit. Our study provides a systematic account of the remarkable robustness of surrogate gradient learning and serves as a practical guide to model functional spiking neural networks.

Introduction

The computational power of deep neural networks [1, 2] has reinvigorated interest in using in-silico systems to study information processing in the brain [3, 4]. For instance, performance-optimized artificial neural networks bear striking representational similarity with the visual system [5–11] and serve to formulate hypotheses about their mechanistic underpinnings. Similarly, the activity of artificial recurrent neural networks optimized to solve cognitive tasks resembles cortical activity in prefrontal [12, 13], medial frontal [14], and motor areas [15, 16] and inspire comparison and lively discussion.

All of these studies rely on conventional artificial neural networks with graded activation functions as commonly used in machine learning. The recipe for building a deep neural network is straight-forward. The value of a scalar loss function defined at the output of the network is decreased through gradient descent. They differ from biological neural networks in important respects. For instance, they lack cell type diversity and do not obey Dale's law while ignoring the fact that the brain uses spiking neurons. We generally accept these flaws because we do not know how to construct more complicated networks. For instance, gradient descent only works when the involved system is differentiable. This is not the case for spiking neural networks (SNNs).

Surrogate gradients have emerged as a solution to build functional SNNs capable of solving complex information processing problems [17–24]. To that end, the actual derivative of a spike, which appears in the analytic expressions of the gradients, is replaced by any well-behaved function. There are many possible choices of such surrogate derivatives, and consequently, the resulting surrogate gradient is, unlike the true gradient of a system, not unique. A number of studies have successfully applied different instances of surrogate derivatives to various problem sets [19, 21, 22, 25–27]. While this suggests that the method does not crucially depend on the specific choice of surrogate derivative, we know relatively little about how the choice of surrogate gradient impacts the effectiveness and whether some choices are better than others. Previous studies did not address this question because they solved different computational problems, thus precluding a direct comparison. In this article, we address this issue by providing benchmarks for comparing the trainability of SNNs on a range of supervised learning tasks and systematically vary the shape and scale of the surrogate derivative used for training networks on the *same* task.

Results

To systematically evaluate the performance of surrogate gradients, we sought to repeatedly train the same network on the same problem while changing the surrogate gradient. Towards this end, we required a demanding spike-based classification problem with a small computational footprint to serve as benchmark. There are only few established benchmarks for SNNs. One approach is to use analog-valued machine learning datasets as input currents directly [17] or to first convert them to Poisson input spike trains [18, 20]. These input paradigms, however, do not fully capitalize on the ability to encode information in spike timing, an important aspect of spiking processing. Gütig [28] addressed this point with the Tempotron by classifying randomly generated spike timing patterns in which each input neuron fires a single spike. Yet, completely random timing, precludes the possibility to assess generalization performance, i.e., the ability to generalize to previously unseen inputs. To assess if SNNs could learn to categorize spike patterns and generalize to unseed patterns, we created a number of synthetic classification datasets with added temporal structure. Specifically, we created spike rasters for a given set of input afferents. Each afferent only fired one spike, and the spike times of all afferents were constrained to lie on a low dimensional smooth random manifold in the space of all possible spike timings. All data points from the same manifold were defined as part of the same input class, whereas different manifolds correspond to different classes.

The spike-timing manifold approach has several advantages: First, the temporal structure in the data permits studying generalization, a decisive advantage over using purely random spike patterns. Second, the task complexity is seamlessly adjustable by tuning the number of afferents, manifold's smoothness parameter α (Fig. 1a), the intrinsic manifold dimension D, and the number of classes n (Fig. 1b). Third, we ensure that each input neuron spikes exactly once (Fig. 1c), guaranteeing that, the resulting datasets are purely spike-timing-dependent and thus cannot be classified from firing rate information. Finally, sampling an arbitrary number of data points from each class is computationally cheap and it is equally easy to generate an arbitrary number of different datasets with comparable properties.

To demonstrate the validity of our approach, we tested it on a SNN with a single hidden layer on a simple two-way classification problem (Fig. 2a; Methods). We

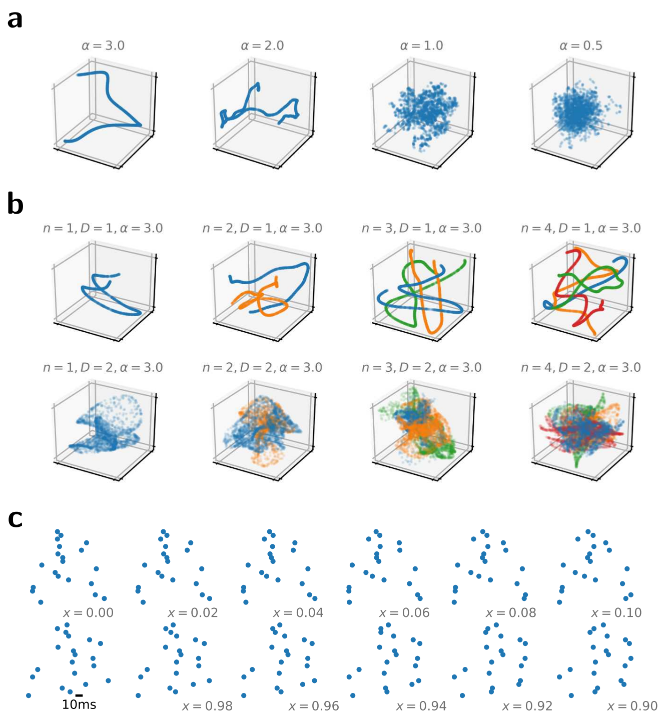

Fig 1. Smooth random manifolds provide a flexible way of generating synthetic spike-timing datasets. **(a)** Four one-dimensional example manifolds for different smoothness parameters α in a three-dimensional embedding space. From each manifold, we plotted 1000 random data points**.** **(b)** Same as in (a), but keeping $\alpha = 3$ fixed while changing the manifold-dimension D and the number of random manifolds (different colors). By sampling different random manifolds, it is straight forward to build synthetic multi-way classification tasks. **(c)** Spike raster plots corresponding to twelve samples along the intrinsic manifold coordinate x of a one-dimensional smooth random manifold ($\alpha = 3$) whereby we interpreted the embedding space coordinates as firing times of the individual neurons.

modeled the units of the hidden layer as current-based leaky integrate-and-fire neurons. Between layers, the connectivity was strictly feed-forward and all-to-all. The output layer consisted of two leaky integrators that did not spike, allowing us to compute the maximum of the membrane potential [29] and to interpret these values as the inputs for a standard classification loss function for supervised learning (Methods). In this setup, the readout unit with the highest activity level signals the putative class-membership of each input [23].

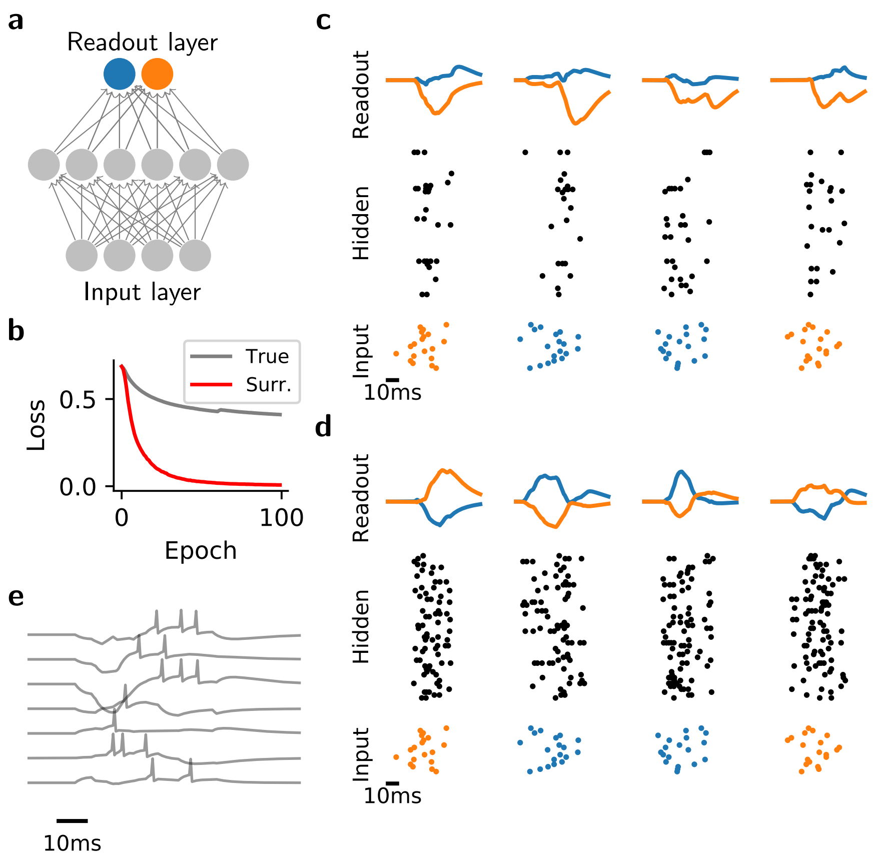

Fig 2. Surrogate gradient descent allows building functional SNNs.
(a) Sketch of the network model with two readout units at the top. **(b)** Learning curves of the network when using the actual gradient ("true", gray) or a surrogate gradient (red) to train an SNN on a binary random manifold classification problem.
(c) Snapshot of network activity before training. Bottom: Spike raster of the input layer activity. Four different inputs corresponding to two different classes are plotted in time (orange/blue). Middle: Spike raster of the hidden layer activity. Top: Readout unit membrane potential. The network erroneously classifies the two "orange" inputs as belonging to the "blue" class, as can be read off from the maximum activity of its readout units. **(d)** Same as in c, but following training of the network with surrogate gradient descent. **(e)** Example membrane potential traces from 7 randomly selected hidden layer neurons during a single trial.

We first confirmed that learning is poor when we used the actual gradient. To that end, we computed it using the derivative of the hard threshold nonlinearity of the spikes. As expected, the hard threshold nonlinearity prevented gradient flow into the hidden layer [24], and consequently lead to poor performance (Fig. 2b,c). In contrast, when we used surrogate gradients to train the same network, the problem disappeared. Learning took place in both hidden and output layer and resulted in a substantial reduction of the loss function (Fig. 2b–e).

Surrogate gradient learning is robust to the shape of the surrogate derivative

A necessary ingredient of surrogate grading learning is a suitable surrogate derivative. To study the effect of the surrogate derivative comparably, we generated a random manifold dataset with ten classes. We chose the remaining parameters, i.e., the number of input units, the manifold dimension, and the smoothness α, to make the problem impossible to solve for a network without hidden layer while at the same time keeping the computational burden minimal. We trained multiple instances of the same network with a single hidden layer ($n_{\mathrm{h}} = 1$) using the derivative of a fast sigmoid as a surrogate derivative (Fig. 3a "SuperSpike"; [25]) on this dataset. In each run, we kept both the dataset and the initial parameters of the model fixed, but varied the slope parameter β of the surrogate. For each value of β we performed a parameter sweep over the learning rate η. Following training, we measured the classification accuracy on held-out data. This search revealed an extensive parameter regime of β and η in which the system was able to solve the problem with high accuracy (Fig. 3b). The addition of a second hidden layer only marginally improved upon this result, whereas, as expected, a network without hidden layer performed poorly (Fig. 3c). The extent of the parameter regime yielding high performance suggests remarkable robustness of surrogate gradient learning to changes in the "steepness" of the surrogate derivative. While a steep approach to the threshold could be seen as a closer, hence better approximation of the actual derivative of the spike, the surrogate gradient remains largely unaffected by how *closely* the function resembles the exact derivative as long as it is not a constant.

Next we tested different surrogate derivative shapes, namely a standard sigmoid (Sigmoid′) and piece-wise linear function (Esser et al.; Fig. 3a; [19, 22]). This manipulation led to a reduction of the size of the parameter regime in β in which the network was able to perform the task, which is presumably due to vanishing gradients [30]. However, there was no substantial reduction in maximum performance (Fig. 3b,c). Using a piece-wise linear surrogate derivative (Esser et al.) led to a further reduction of viable parameters β (Fig. 3b), but did not affect maximum performance, regardless of whether we used one or two hidden layers (Fig. 3c). To check whether a surrogate derivative was required at all for solving the random manifold problem, we assayed the learning performance for $\beta = 0$, which corresponds to setting the function to 1. This change resulted in a significant drop in performance comparable to a network without hidden units (Fig. 3c) suggesting that a nonlinear voltage-dependence is crucial to learn useful hidden layer representations. Finally, we confirmed that these findings were robust to different initial network parameters and datasets (Fig. 3d) apart from only a few outliers with low performance in the two hidden layer case ($n_h = 2$). These outliers point at the vital role of proper initialization [31, 32].

Surrogate gradient learning in recurrent networks is sensitive to the scale of the surrogate derivative

In most studies that rely on surrogate gradients, the surrogate derivative is normalized to 1 [19, 21, 22, 24, 25] (cf. Fig. 3a), markedly different from the actual derivative of a spiking threshold which is infinite (Fig. 5a). The scale of the substituted derivative has a strong effect on the surrogate gradient due to both explicit and implicit forms of recurrence in SNNs (Fig. 4; [24]). Most notably, the scale may determine whether

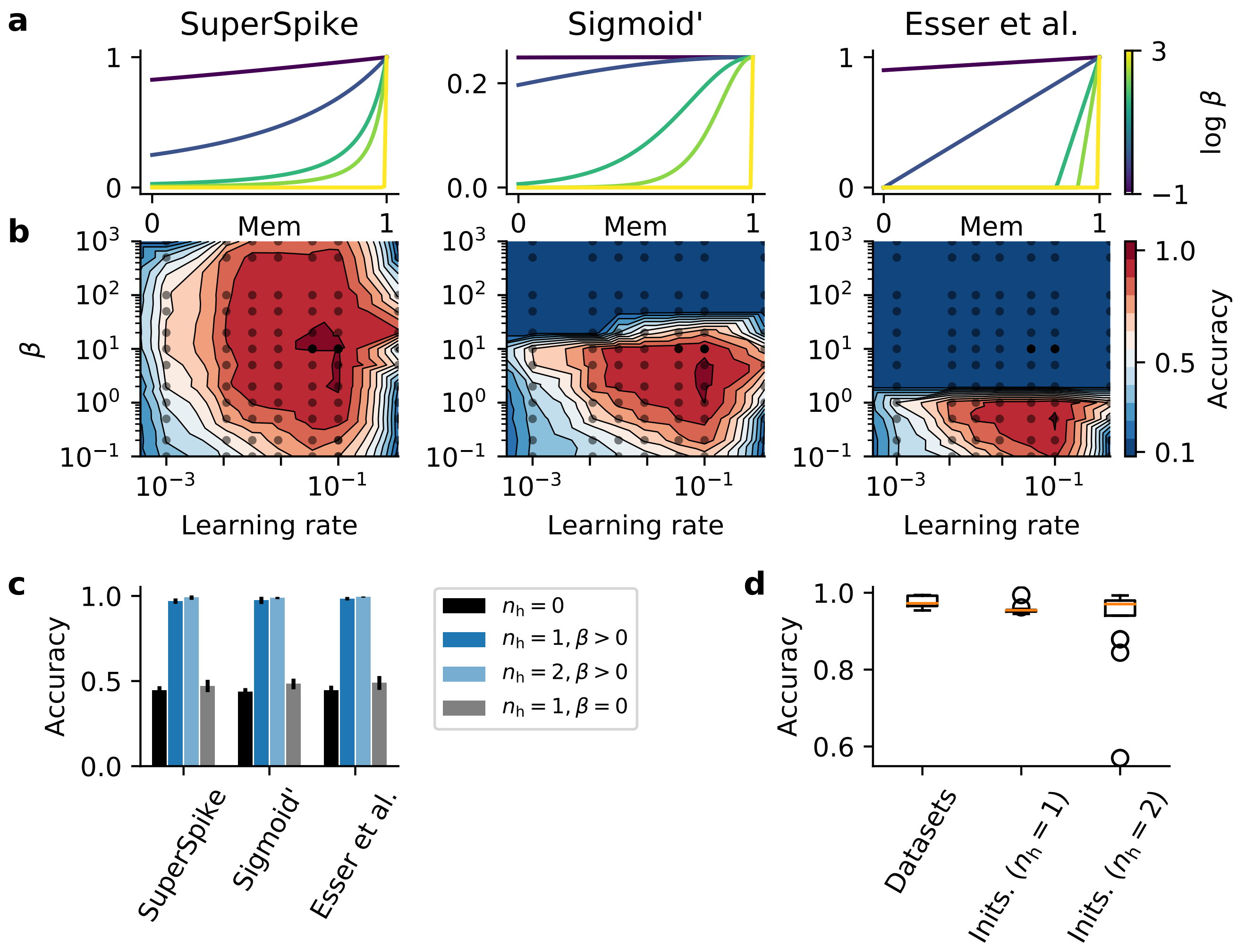

Fig 3. Surrogate gradient learning is robust to the shape of the surrogate derivative. **(a)** Three different surrogate derivative shapes that have been used for training on the synthetic smooth random manifold spiking dataset. From left to right: SuperSpike [25], the derivative of a fast sigmoid function, Sigma′, the derivative of a standard sigmoid function, and "Esser et al.", a piece-wise linear function [19, 22]. Colors correspond to different values of the slope parameter β. **(b)** Accuracy on held-out data as a function of the learning rate η and the slope β for the corresponding surrogates in (a) for a network with one hidden layer ($n_h = 1$). **(c)** Test accuracy for the five best parameter combinations obtained from a grid search, as shown in (b) for different surrogates and numbers of hidden layers n_h. While a network without hidden layer is unable to solve the classification problem (black), networks trained with a wide range of different surrogates and slope parameters ($\beta > 0$) have no problem solving the task with high accuracy (shades of blue). However, the problem is not solved with high accuracy by a network with a hidden layer in which the surrogate derivative was a constant during training ($\beta = 0$; gray). Error bars correspond to the standard deviation ($n = 5$). **(d)** Whisker plot of classification accuracy for a network with one hidden layer over five different realizations of the random manifold datasets ("Datasets") and for the same dataset, but using different weight initializations ("Inits.") in networks with either one ($n_\mathrm{h} = 1$) or two ($n_\mathrm{h} = 2$) hidden layers.

gradients vanish or explode [30]. However, it is unclear to what extent such differences translate into performance changes of the trained networks.

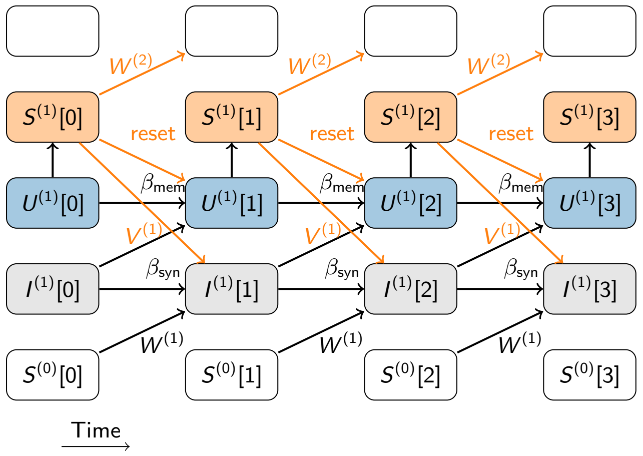

Fig 4. SNNs can be have both implicit and explicit recurrence. Schematic of the computational graph of a single SNN layer composed of leaky integrate-and-fire (LIF) neurons (see Methods). Input spike trains $S^{(0)}$ enter at the bottom and affect the synaptic current variable $I^{(1)}$ through the feed-forward weights $W^{(1)}$. Time flows from left to right. Any link that connects temporally adjacent nodes in the graph constitute a form of recurrence in the computation whereby the synaptic connections $V^{(1)}$ contribute explicit recurrence to the graph. Implicit recurrence is contributed, for instance, by the decay of synaptic current variables and the membrane potentials $U^{(1)}$. Additionally, the spike reset contributes another form of implicit recurrence by coupling the future states to the output spike train $S^{(1)}$. Recurrences involving the surrogate derivative, e.g. the reset, depend on both the shape and the scale of the surrogate chosen and can substantially alter the surrogate gradient.

To gain a better understanding of how derivative scales larger than 1 affect surrogate gradient learning, we trained networks on a specific random manifold task, an *asymptotic* version of the SuperSpike surrogate (aCtl; Fig. 3) and our well-tested standard SuperSpike function (sCtl; Fig. 3) as a control (sCtl). As we expected the difference in scale to manifest itself primarily in the presence of recurrence, we compared networks in which we treated the spike reset as differentiable (DR) with networks in which its contribution was ignored by *detaching* it from the computational graph. Technically speaking, we prevented PyTorch's auto-differentiation routines [33] from considering connections in the computational graph that correspond to the spike reset when computing the gradient with back-propagation through time (BPTT) (see Methods Eq. (1)). While both the normalized (sCtl) as well as the asymptotic surrogate (aCtl), performed equally well when the reset was detached, combining a differentiable reset with an asymptotic surrogate derivative (aDR) lead to impaired performance (Fig. 5b,c). This adverse effect on learning was amplified in deeper networks (Fig. 5d). Thus, the scale of the surrogate derivative plays indeed an important role for learning success if implicit recurrence, as contributed by the spike reset, is present in the network.

Since the spike reset constitutes a specific form of implicit recurrence (cf. Fig. 4), we were wondering whether we would observe a similar phenomenon for explicit recurrence through recurrent synaptic connections. To that end, we repeated the above performance measurements in networks with recurrent connections, but kept the spike reset term detached to prevent gradient flow. We observed a small but measurable reduction in accuracy for the best performing networks (Fig. 5c,e). Importantly,

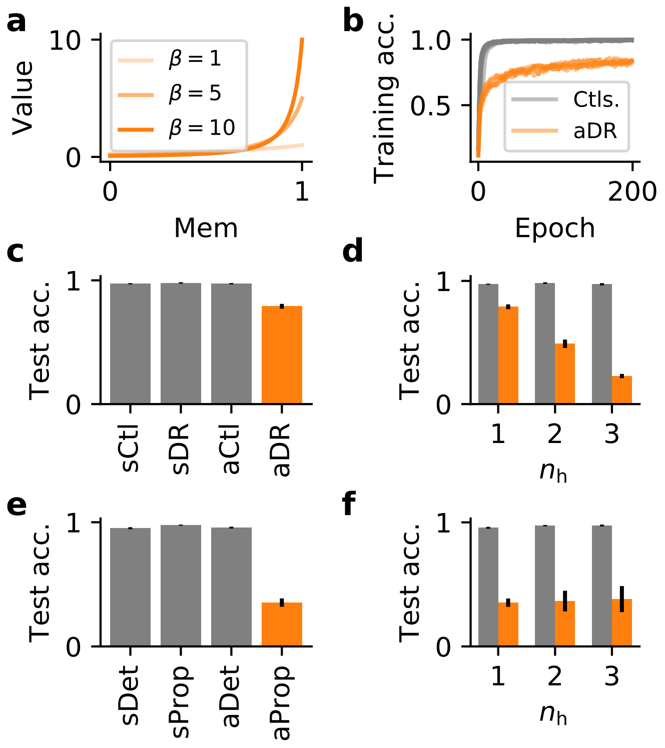

Fig 5. Surrogate gradient learning is sensitive to the scale of the surrogate derivative. **(a)** Illustration of pseudo derivatives σ' that converge toward the actual derivative of a hard spike threshold $\beta \to \infty$. Note that in contrast to Fig. 3a, their maximum value grows as β increases. **(b)** Training accuracy of several spiking networks $(n_h = 1)$ during training on a synthetic classification task. The gray curves comprise control networks in which the surrogate derivative was either normalized to one or in which we used an asymptotic surrogate derivative, but prevented surrogate gradients from flowing through the spike reset. Orange curves correspond to networks with asymptotic pseudo derivatives with differentiable spike reset (aDR). In all cases, we plot the five best performing learning curves obtained from an extensive grid-search over β and the learning rate η (cf. Fig. 3). **(c)** Quantification of the test accuracy of the different learning curves shown in (b). We trained all networks using a SuperSpike nonlinearity. The reset term was either ignored (sCtl), or in which a differentiable reset was used (vDR). Similarly, we considered an asymptotic variant of SuperSpike that does converge toward the exact derivative of a step function for $\beta \to \infty$, without (aCtl) or with a differentiable reset term (aDR). The results shown correspond to the ten best results from a grid search. The error bars denote the standard deviation. **(d)** A similar comparison of control cases in which reset terms were ignored (gray) or could contribute to the surrogate gradient (orange) for different numbers of hidden layers. **(e)** Test accuracy as in (c), but comparing SuperSpike (v) and the asymptotic (a) case in which gradients can flow through recurrent connections (Prop) versus the detached case (Ctl). **(f)** Test accuracy for asymptotic SuperSpike as a function of the number of hidden layers for networks in which gradients were flowing through recurrent connections (orange) versus the detached case (gray).

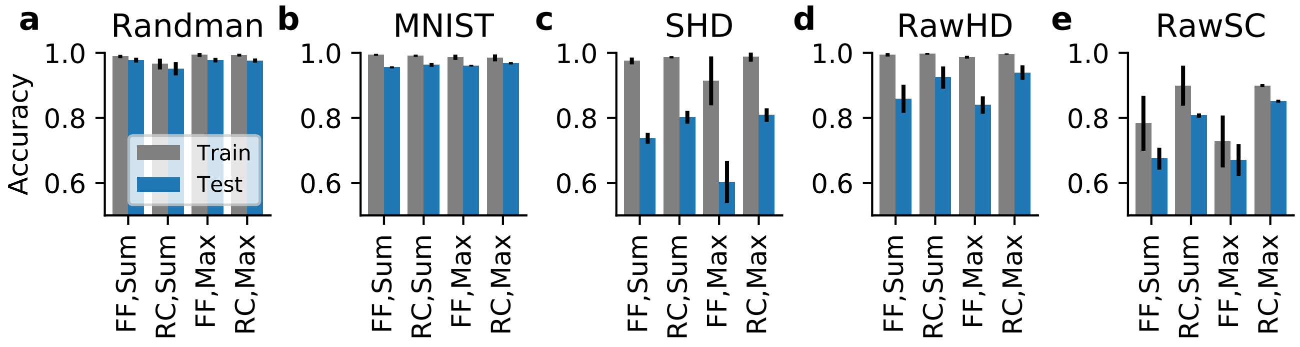

Fig 6. Surrogate gradient learning is effective on different loss functions, input paradigms, and datasets. Bar plots showing the test classification accuracy for different datasets (a–e). The plots further distinguish models by their readout configuration (Sum vs. Max) and whether they use purely feed-forward (FF) or explicitly recurrent (RC) synaptic connectivity. Each bar corresponds to the mean over the ten best performing models on held-out validation data and error bars signify the standard deviation.

however, there was a substantial decrease in classification performance when gradients were allowed to flow through recurrent connections, *and* the asymptotic SuperSpike variant was used (Fig. 5e). Unlike in the differentiable reset case, the effect was severe enough to drop network performance to chance level even for a network with a single hidden layer (Fig. 5f). In summary, surrogate gradients are sensitive to the scale of the surrogate derivative. More specifically, when the scale of the surrogate derivative is too large, while either implicit or explicit recurrence are present in the network, the effect on learning can be detrimental.

Surrogate gradient learning is robust to changes in the loss functions, input paradigms, and datasets

So far we investigated synthetic random manifold datasets in strictly feed-forward networks trained with loss functions that were defined on the maximum over time (Max) of the readout units. Next, we performed additional simulations in which the loss was computed by summation over time ("Sum"; see Methods). Based on our findings above, we limited our analysis to the SuperSpike surrogate with $\beta = 10$ and detached reset terms. As before, we performed a grid search over the learning rate η and selected the ten best performing models using held-out validation data. We then computed their classification performance on a separate test set. We repeated our simulation experiments on the above random manifold task and did not observe any substantial difference in the accuracy for the Max and Sum type readout heads (Fig. 6a).

To check the validity of these findings on a different dataset, we trained networks on MNIST by converting pixel values into spike latencies (Fig. 7a,b; Methods). In this paradigm, each input neuron fires either a single or no spike for each input. The networks reached a test accuracy of $(97.4 \pm 0.2)\,\%$, which is comparable to a conventional artificial neural network with the same number of neurons and hidden layers and to previous SNN studies using temporal coding [34]. We did not observe any discernible performance differences between the two readout types we tested (Fig. 6b).

Further, to study the effect of explicit recurrence, we ran separate experiments for recurrently connected networks (RC). Importantly, we did not observe any substantial performance differences between strictly feed-forward and recurrently connected networks either (Fig. 6a,b).

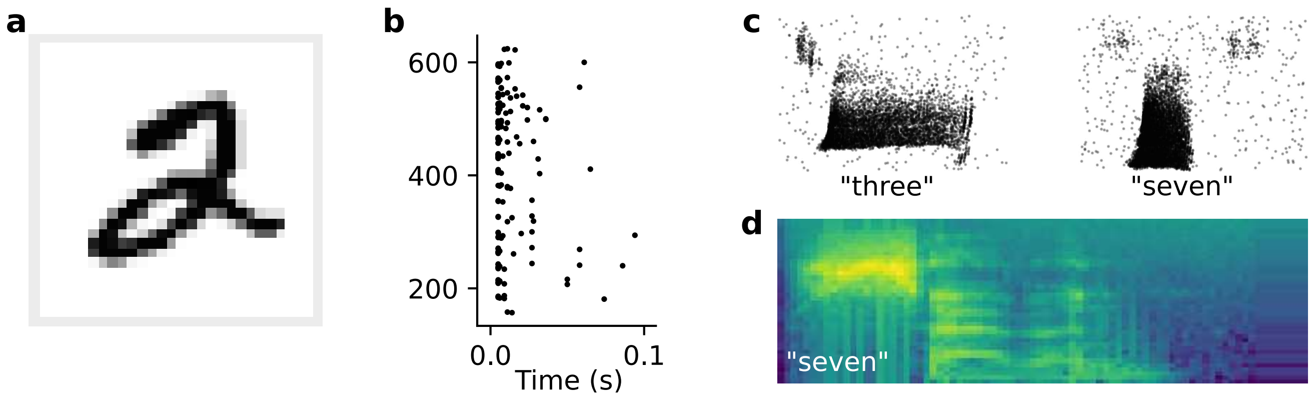

Fig 7. Examples of different input paradigms. **(a)** One example image from MNIST hand-written digit dataset. **(b)** Spike raster plot of the corresponding spike latency encoding for the $28 \times 28 = 784$ input neurons. **(c)** Spike raster of two example inputs for the spoken digits "three" and "seven" taken from the SHD dataset [23]. **(d)** Mel-scaled spectrogram of an utterance of the number "seven" as used for simulations using raw audio input.

We speculated that the absence of an effect may be due to the short duration of the input spike trains considered so far ($\sim$ 50ms). Recurrent connections are typically thought to provide neural networks with longer timescale dynamics, effectively giving the network a working memory. Thus, the beneficial effects of recurrent connections may only emerge when using stimuli of longer duration, with potentially multiple spikes from any given input neuron.

To test this hypothesis, we trained networks on the Spiking Heidelberg Digits (SHD) dataset [23], which consists of simulated input spikes from the auditory pathway of varied duration between 0.6–1.4s (Fig. 7c). Indeed, we found that the best performing models in this case were recurrent and achieving state-of-the-art classification accuracy of (0.82 ± 0.02) % (Fig. 6c). These data are consistent with the notion that working memory plays a vital role for the classification of longer input patterns.

Surrogate gradient learning in networks with current-based input

Until now, we considered spike-based datasets. While spiking inputs are arguably the most natural input to SNNs, they come with an important caveat. All spiking datasets assume a specific encoding model, which is used to convert analog input data into a spiking representation. The chosen model, however, may not be optimal and thus adversely affect classification performance. To avoid this issue, we sought to *learn* the spike encoding by directly feeding current-based input to a set of spiking units [35]. To test this idea, we converted the raw audio data of the Heidelberg Digits [23] to Mel-spaced spectrograms (7d; Methods). To reduce overfitting, we decreased the number of channels and time steps to values commonly used in artificial speech recognition systems. Specifically, we used 40 channels and 80 time frames, corresponding to compression in time by a factor of about five (Methods). The networks trained on this RawHD dataset showed reduced overfitting, yet still benefited from recurrent connections compared to strictly feed-forward networks (Fig. 6d). Concretely, recurrent networks reached (94 ± 2) % test accuracy, whereas the feed-forward networks reached only (84 ± 3) %. In agreement with the results on Randman and MNIST, there were no significant differences between the Sum and Max readout configurations (Fig. 6).

We wanted to know whether the discrepancy between feed-forward and recurrent networks would increase for more challenging data sets. This was made possible by the performance gain from reducing the input dimension to 40 channels, which allowed us to train SNNs on the larger Speech Command dataset ([36]; Methods). This dataset comprises over 100k utterances from 35 classes, including "yes", "no", and "left". In contrast to the original intended use for keyword spotting, here we assayed its top-1 classification accuracy over all classes, a more challenging problem than accurate detection of only a subset of words. The best SNNs achieved $(85.3 \pm 0.3)\,\%$ on this challenging benchmark (Fig. 6e). On the same task, the spiking feed-forward network performed at $(70 \pm 2)\,\%$. There was a clear benefit from adding recurrent connections to the network, with the performance of the Max ($(85.3 \pm 0.3)\,\%$) being slightly better than the Sum ($(80.7 \pm 0.4)\,\%$) readout configuration.

These findings illustrate that surrogate gradient learning is robust to changes in the input paradigm, including spiking and non-spiking datasets. For more complex datasets, recurrently connected networks performed better than strictly feed-forward networks. Finally, in the majority of cases, surrogate gradient learning was robust to the details of how the output loss was defined.

Optimal sparse spiking activity levels in SNNs

Up to now, we have focused on maximizing classification accuracy while ignoring the emerging activity levels in the resulting SNNs. However, we found that for some solutions the neurons in these networks displayed implausibly high firing rates (Fig. 8a). Experimental results suggest that most biological networks exhibit sparse spiking activity, a feature that is presumed to underlie their superior energy-efficiency [37–42].

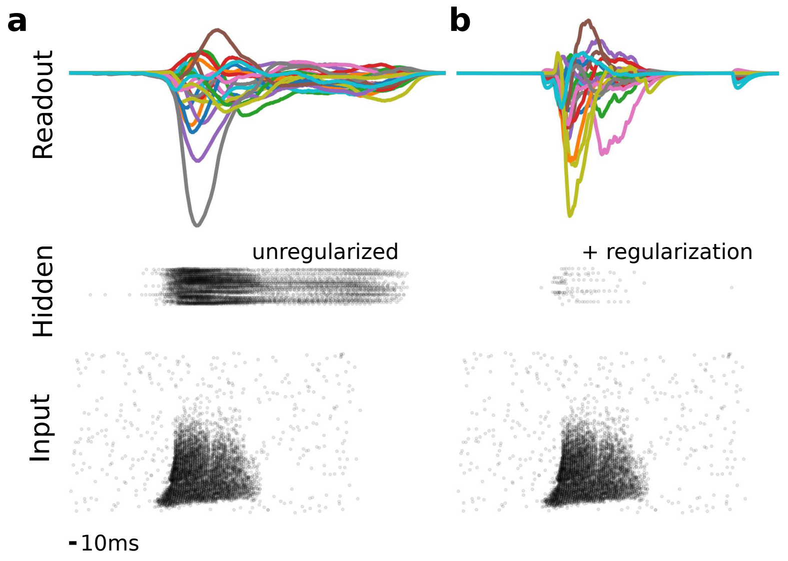

Fig 8. Activity regularization renders hidden layer activity sparse while maintaining functionality. (a) Activity snapshot of one example input from SHD in a trained network. Spike raster plots of the input and hidden layer units are shown at the bottom and in the middle. The activity of the twenty readout units is plotted at the top with the brown line corresponding to the correct output for this example. Without any specific regularization, spiking activity in the hidden layer is pathologically high. **(b)** As in (a), but for a network trained with a penalty term for high spiking activity. This form of activity regularization drastically alters the hidden layer activity for the same input, while leaving the winning output of the network unchanged (brown line).

We investigated whether surrogate gradients could instantiate SNNs in this biologically plausible, sparse activity regime. To that end, we trained SNNs with added

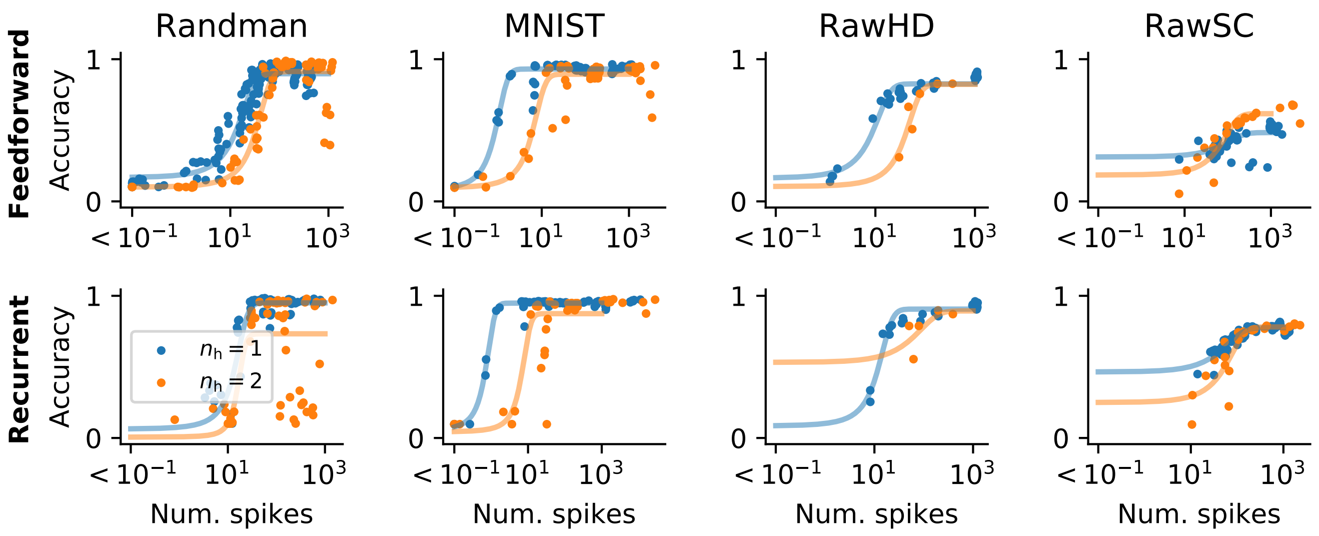

Fig 9. Classification accuracy degrades below a critical number of hidden-layer spikes. Plots showing classification accuracy as a function of the average number of hidden layer spikes per input. The different columns correspond to the different data sets (cf. Fig. 6 and Fig. 7). Top row: Networks with feed-forward connectivity. Bottom row: Networks with recurrent synapses. Blue data points correspond to networks with one hidden layer, whereas orange data points come from networks with two hidden layers. The solid lines correspond to fitted Sigmoid functions.

activity regularization that penalized high spiking activity (Fig. 8b; Methods) and measured the average hidden-layer firing rates and the classification accuracy.

The regularisation term dramatically reduced the number of spikes that were necessary for successful learning. In most cases, the number of spikes could be reduced by approximately two orders of magnitude before there was a notable decline in performance and we found a critical transition in the average number of hidden layer spikes below which networks performed poorly (Fig. 9). For example, in the random manifold task, the transition occurred at ≈ 36 hidden layer spikes per input in a single hidden layer and ≈ 76 spikes in a two hidden layer feed-forward network. In the recurrent network, this number was reduced to 26 ($n_\mathrm{h} = 1$).

In the case of MNIST fewer than ten spikes were sufficient on average to achieve the point of diminishing returns beyond which additional spiking activity did not improve classification performance.

This trend could be replicated for all other data sets, with varying degrees of spike reduction. Adding an additional hidden layer generally required more spikes for the same performance. Recurrency generally did not have a great effect on the minimum number of spikes and did not improve performance except on RawSC. On RawSC, 80% classification accuracy was achieved only by a few feed-forward networks with > 2000 spikes on average. In the recurrent network, this level was already attained with ≈ 150 spikes (Fig. 9).

In all cases, the transition from chance level to maximum accuracy occurred in less than one order of magnitude change in the mean number of spikes. On all datasets we tested, the addition of a second hidden layer led to an overall increase in mean spiking activity, which did not yield a notable performance change on Randman and MNIST, but resulted in small improvements on RawHD and RawSC.

These results illustrate that activity regularized SNNs can perform with high accuracy down to some critical activity threshold at which their performance degrades rapidly. Importantly, we found several network configurations that showed competitive performance with an average number of spikes substantially lower than the number of hidden units. For instance, to classify MNIST with high accuracy, an average of 10-20 action potentials was sufficient. Such low activity levels are more consistent with the sparse neuronal activity observed in biological neural circuits and illustrate that surrogate gradients are well-suited to build SNNs which use such lausibly sparse activity levels for information processing.

Discussion

Surrogate gradients offer a promising way to instill complex function in artificial models of spiking networks. In this article, we focused on two aspects of surrogate gradient learning in SNNs. First, we showed, using a range of supervised classification problems, that surrogate gradient learning in SNNs is robust to different shapes of surrogate derivatives. In contrast, inappropriate choice of scale adversely affected learning performance. Our results imply that for practical applications, surrogate derivatives should be appropriately normalized. Second, by constraining their activity through regularization, we showed that surrogate gradients can produce SNNs capable of efficient information processing with sparse spiking activity.

Surrogate gradients have been used by a number of studies to train SNNs[24], to solve small-scale toy problems with fractionally predictive neurons [43], to train convolutional SNNs on challenging neuromorphic [44, 45] and vision benchmarks [19], or to train recurrent SNNs on temporal problems requiring working memory [21, 22]. These studies used a range of different surrogate derivatives ranging from exponential [21], piece-wise linear [22], or tanh [27], sometimes with a non-standard neuron model with a constant leak term [19], but due to the different function choices and datasets they are not easily comparable. Here we provide such a comprehensive comparison.

Towards this end, we had to make some compromises. Like previous studies, our study is limited to supervised classification problems, because supervised learning offers a well-defined and intuitive quantification of computational performance. To keep the number of model parameters tractable, we focused on current-based LIF neurons. Moreover, we entirely dispensed with Dale's law and relied solely on all-to-all connectivity. However, we believe that most of our findings will carry over to more realistic neuronal, synaptic, and connectivity models. The present study thus provides a set of blueprints and benchmarks to accelerate the design of future studies.

Although considered biologically implausible [46], we have limited our study to training SNNs with backpropagation, the de-facto standard for computing gradients in systems involving recurrence and hidden neurons. Although there exist more plausible forward-in-time algorithms like real-time recurrent learning (RTRL) [47] they are prohibitively expensive or else require additional approximations that affect learning performance [24, 25, 48, 49]. Instead, using BPTT enabled the targeted manipulation of gradient flow through different elements of the network which allowed us to dis-entwine some of complexity of surrogate gradient learning. Finally, the present study was purely numerical. It will require future work to establish a rigorous theoretical understanding of surrogate gradient learning dynamics to understand optimal derivatives and algorithms for learning in SNNs.

In summary, surrogate gradients allow translating the success of deep learning to biologically inspired SNNs by optimizing their connectivity toward functional complexity through end-to-end optimization. The in-depth study of both surrogate gradients and the resulting functional SNNs will occupy scientists for years to come and will likely prove transformative for neural circuit modeling.

Methods

Supervised learning tasks

In this study we used a number of synthetic and real-world learning tasks with the overarching aim to balance computational feasibility and practical relevance. To that end, we focused on synthetic datasets generated from random manifolds and real-world auditory datasets.

Smooth random manifold datasets

We generated a range of synthetic classification data sets based on smooth random manifolds. Suppose we want to generate a smooth random manifold of dimension D in an embedding space of dimension M. We are looking for a smooth random function $f : \mathbb{R}^D \to \mathbb{R}^M$ defined over the finite interval $0 \leq x < 1$ in each of intrinsic manifold coordinate axis. Moreover, we would like to keep its values bounded in a similar $(0 \leq x < \tau_{\text{randman}})^M$ box in the embedding space. To achieve this we first generate M smooth random functions $f_i : \mathbb{R}^D \to \mathbb{R}$ and then combine them to $f : \mathbb{R}^D \to \mathbb{R}^M$. Specifically, we generate the f_i from the Fourier basis as follows:

$$f_i(\vec{x}) = \prod_{j \in D} \left[\sum_{k=1}^{n_{\text{cutoff}}} \frac{1}{k^\alpha} \theta_{ijk}^{\text{A}} \sin\left(2\pi\left(k x_j \theta_{ijk}^{\text{B}} + \theta_{ijk}^{\text{C}}\right)\right) \right]$$

where the parameters θ_{ijk}^L for $L \in \{\text{A}, \text{B}, \text{C}\}$ were drawn i.i.d. from a uniform distribution $\mathcal{U}(0, 1)$. We set $n_{\text{cutoff}} = 1000$ which leaves α as a parameter that controls the smoothness of the manifold. Larger values of α lead to more slowly varying manifolds whereas smaller values increase the high frequency content (Fig. 1a). In addition to α, the complexity of the learning problem can be seamlessly adjusted by either increasing the intrinsic dimension D of the manifold or the number of random manifolds whereby each random manifold corresponds to a separate class of the classification problem (Fig. 1b). We generated spike trains by randomly sampling the resulting random manifolds and standardizing their numerical values to lie between 0 and τ_{randman} in all embedding dimensions. Finally, the resulting values were interpreted as the firing times from one class of the classification problem (Fig. 1c).

We chose a default parameter set for most of our experiments that struck a good balance between minimizing both the embedding dimension and the number of samples needed to solve the problem while simultaneously not being solvable by a two layer network without hidden units. Specifically, we chose a 10-way problem with $D = \alpha = 1$, $M = 20$, and $\tau_{\text{randman}} = 50\text{ms}$ and fixed the random seed in situations in which we reused the same dataset. We generated 1000 data points for each class out of which we used 800 for training and two sets of 100 each for validation and testing purposes.

Spike latency MNIST dataset

To convert the analog valued MNIST dataset [50] to firing times we proceeded as follows. We first standardized all pixel values x_i^μ to lie within the interval $0 \leq x < \tau_{\text{eff}}$. We then computed the time to first spike latency T as the time to reach firing threshold of a leaky integrator

$$T(x) = \begin{cases} \tau_{\text{eff}} \log\left(\frac{x}{x - \vartheta}\right) & x > \vartheta \\ \infty & \text{otherwise} \end{cases}$$

in our simulations we used $\vartheta = 0.2$ and $\tau_{\text{eff}} = 50\text{ms}$ (Fig. 7a,b).

Auditory datasets

We used both spiking and non-spiking auditory datasets of digit and word utterances. Specifically, we used the SHDs without any further preprocessing [23]. For performance reasons and to dispense with the spike conversion process, we ran additional simulations with non-spiking auditory inputs. Specifically, we worked with the raw Heidelberg Digits (RawHD) and Pete Warden's Speech Commands dataset (RawSC) [36] which were preprocessed as follows: We first applied a pre-emphasis filter to the raw audio signal $x(t)$ by computing $y(t) = x(t) - 0.95x(t-1)$. We then computed 25ms frames with a 10ms stride from the resulting signal and applied a Hamming window to each frame. For each frame we computed the 512-point fast Fourier transform to obtain its power spectrum. From the power spectrum we further computed the filter banks by applying 40 triangular filters on a Mel-scale [51]. After cropping or padding to 80 (RawHD) or 100 (RawSC) steps by repeating the last frame, the analog valued filter banks were fed directly to the SNNs.

Network models

To train SNN models with surrogate gradients we implemented them in PyTorch [33]. To that end, all models were explicitly formulated in discrete time with time step Δt.

Neuron model

We used leaky integrate-and-fire neurons with current-based exponential synapses [52, 53]. The membrane dynamics of neuron i in layer l were characterized by the following update equations

$$U_i^{(l)}[n+1] = \left(\beta_{\text{mem}} U_i^{(l)}[n] + (1-\beta_{\text{mem}})\, I_i^{(l)}[n]\right)\left(1 - S_i^{(l)}[n]\right) \tag{1}$$

where $U_i^{(l)}[n]$ corresponds to the membrane potential of neuron i in layer l at time step n and is its $S_i^{(l)}[n]$ the associated output spike train defined via the Heaviside step function Θ as $S_i^{(l)}[n] \equiv \Theta\left(U_i^{(l)}[n] - 1\right)$. Note that in this formulation, the membrane dynamics are effectively re-scaled such that the resting potential corresponds to 0 and the firing threshold of 1. This choice simplifies the implementation of the neuronal reset dynamics through the factor on the right hand side. During the backward pass of gradient computation with BPTT, the derivative of the step function is approximated using a surrogate as explained later. In situations in which we ignored the reset term this was done differently for the output spike train and the spike train underlying the reset term, by detaching it from the computational graph. The membrane decay variable β_{mem} is associated with the membrane time constant τ_{mem} through $\beta_{\text{mem}} \equiv \exp\left(-\frac{\Delta t}{\tau_{\text{mem}}}\right)$. Finally, the variable $I_i^{(l)}[n]$ is the synaptic current defined as follows

$$I_i^{(l)}[n+1] \quad = \quad \beta_{\text{syn}} I_i^{(l)}[n] + \sum_j W_{ij}^{(l)} S_j^{(l-1)}[n] + \sum_j V_{ij}^{(l)} S_j^{(l)}[n]$$

with feed-forward afferent weights W_{ij} and the optional recurrent weights V_{ij}. In analogy to the membrane decay constant, β_{syn} is defined as $\beta_{\text{syn}} \equiv \exp\left(-\frac{\Delta t}{\tau_{\text{syn}}}\right)$. We set $\tau_{\text{mem}} = 10\text{ms}$ and $\tau_{\text{syn}} = 5\text{ms}$. Together the computations involved in each time step can be summarized in the computational graph of the model (cf. Fig. 4).

Readout layer

The readout units in our models are identical to the above neuron model, but without the spike and associated reset. Additionally, we allow for a separate membrane time constant $\tau_{\text{readout}} = 20\text{ms}$ with $\beta_{\text{out}} \equiv \exp\left(-\frac{\Delta t}{\tau_{\text{readout}}}\right)$. Overall their dynamics are described by

$$U_i^{(\text{out})}[n+1] = \beta_{\text{out}} U_i^{(\text{out})}[n] + (1-\beta_{\text{out}})\, I_i^{(\text{out})}[n]$$

Connectivity and initialization

We used all-to-all connectivity in all simulations without bias terms unless mentioned explicitly. The weights were initialized from a uniform distribution $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ with $k = \frac{1}{n_{\text{inputs}}}$ where n_{inputs} is the number of afferent connections.

Readout heads and supervised loss function

We trained all our networks by minimizing a standard cross entropy loss $\mathcal{L}_{\text{sup}}$ defined as

$$\mathcal{L}_{\text{sup}} = -\frac{1}{N} \sum_{\mu=1}^{N} \sum_{i=1}^{C} y_i^\mu \log\left(p_i^\mu\right)$$

where y_i^μ is the one-hot encoded target for input μ, N is the number of input samples, and C is the number of classes. The output probabilities p_i^μ were given by the Softmax function

$$p_i^\mu = \frac{e^{a_i^\mu}}{\sum_{k=1}^{C} e^{a_k^\mu}}$$

in which the logits a_i^μ for each input μ were given by either defined as the maximum $a_i^\mu = \max_n \left(U_i^{(\text{out})}[n]\right)$, a notion inspired by the Tempotron [29], or the sum over all time steps $a_i^\mu = \sum_n \left(U_i^{(\text{out})}[n]\right)$ depending on the readout configuration we used.

Activity regularization

To control spiking activity levels in the hidden layers we employed two forms of activity regularization. First, to prevent quiescent units in the hidden layers, we introduced a lower activity threshold ν_{lower} at the neuronal level defined as

$$g_{\text{lower}}^\mu = -\frac{\lambda_{\text{lower}}}{M} \sum_{i}^{M} \left(\left[\nu_{\text{lower}} - \zeta_i^{(l),\mu}\right]_+\right)^2$$

with the neuronal spike count $\zeta_i^{(l)} \equiv \left(\sum_n S_i^{(l)}[n]\right)$ and the number of neurons M in hidden layer l. Similarly, we defined an upper threshold at the population level as

$$g_{\text{upper}}^\mu = -\lambda_{\text{upper}} \left(\left[\frac{1}{M} \sum_{i}^{M} \zeta_i^{(l),\mu} - \nu_{\text{upper}}\right]_+\right)^L$$

for which we explored both values of $L \in \{1, 2\}$. The overall regularization loss was computed by summing and averaging $\mathcal{L}_{\text{reg}} = \frac{1}{N}\sum_\mu \left(g_{\text{lower}}^\mu + g_{\text{upper}}^\mu\right)$. Finally we optimized the total loss $\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{sup}} + \mathcal{L}_{\text{reg}}$ by dint of surrogate gradient descent.

Surrogate gradient descent

We minimized the loss $\mathcal{L}_{\text{tot}}$ by adjusting the parameters W and V in direction of the negative surrogate gradients using Adam with default parameters [54]. Surrogate gradients were computed with back-propagation through time using PyTorch's automatic differentiation capabilities. To deal with the non-differential spiking nonlinearity of the hidden layer neurons, we approximated their derivatives $S'\left(U_i^\mu[n]\right) = \Theta'\left(U_i^\mu[n] - 1\right)$ with suitable surrogates $\sigma'\left(U_i^\mu[n]\right) = h\left(U_i^\mu[n] - 1\right)$. Throughout, we used the following functions $h(x)$:

SuperSpike: $h(x) = \frac{1}{(\beta|x|+1)^2}$

Sigmoid′: $h(x) = s(x)\,(1 - s(x))$ with the Sigmoid function $s(x) = \frac{1}{1+\exp(-\beta x)}$

Esser et al.: $h(x) = \max(0, 1.0 - \beta\,|x|)$

where β is a parameter that controls the slope of the surrogate derivative. In Figure 5, we additionally considered an asymptotic variant of SuperSpike defined as $h(x) = \frac{\beta}{(\beta|x|+1)^2}$. Unless mentioned otherwise we set $\beta = 10$. Table 1 specifies the relevant hyperparameters used in the different simulations. An example of how these manipulations can be achieved easily with PyTorch can be found at `https://github.com/fzenke/spytorch` [55]. All simulations and parameter sweeps were performed on compute nodes equipped with Nvidia Quadro RTX 5000 and V100 GPUs.

Acknowledgments

FZ was supported by the Wellcome Trust (110124/Z/15/Z) and the Novartis Research Foundation. TPV was supported by a Wellcome Trust Sir Henry Dale Research fellowship (WT100000), a Wellcome Trust Senior Research Fellowship (214316/Z/18/Z) and an ERC Consolidator Grant SYNAPSEEK.

References

1. Schmidhuber J. Deep learning in neural networks: An overview. *Neural Netw*, 61: 85–117, January 2015. doi: 10.1016/j.neunet.2014.09.003.

2. LeCun Y, Bengio Y, and Hinton G. Deep learning. *Nature*, 521(7553):436–444, May 2015. doi: 10.1038/nature14539.

3. Barrett DG, Morcos AS, and Macke JH. Analyzing biological and artificial neural networks: challenges with opportunities for synergy? *Current Opinion in Neurobiology*, 55:55–64, April 2019. doi: 10.1016/j.conb.2019.01.007.

4. Richards BA, Lillicrap TP, Beaudoin P, Bengio Y, Bogacz R, Christensen A, Clopath C, Costa RP, Berker Ad, Ganguli S, Gillon CJ, Hafner D, Kepecs A, Kriegeskorte N, Latham P, Lindsay GW, Miller KD, Naud R, Pack CC, Poirazi P, Roelfsema P, Sacramento J, Saxe A, Scellier B, Schapiro AC, Senn W, Wayne G, Yamins D, Zenke F, Zylberberg J, Therien D, and Kording KP. A deep learning framework for neuroscience. *Nat Neurosci*, 22(11):1761–1770, November 2019. doi: 10.1038/s41593-019-0520-2.

5. McIntosh L, Maheswaranathan N, Nayebi A, Ganguli S, and Baccus S. Deep Learning Models of the Retinal Response to Natural Scenes. *NeurIPS*, pages 1369–1377, 2016.

6. Maheswaranathan N, McIntosh LT, Kastner DB, Melander J, Brezovec L, Nayebi A, Wang J, Ganguli S, and Baccus SA. Deep learning models reveal internal structure and diverse computations in the retina under natural scenes. *bioRxiv*, page 340943, June 2018. doi: 10.1101/340943.

7. Tanaka H, Nayebi A, Maheswaranathan N, McIntosh L, Baccus S, and Ganguli S. From deep learning to mechanistic understanding in neuroscience: the structure of retinal prediction. *NeurIPS*, pages 8535–8545, 2019.

8. Yamins DLK, Hong H, Cadieu CF, Solomon EA, Seibert D, and DiCarlo JJ. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proc Natl Acad Sci U S A*, 111(23):8619–8624, October 2014. doi: 10.1073/pnas.1403112111.

9. Yamins DLK and DiCarlo JJ. Using goal-driven deep learning models to understand sensory cortex. *Nat Neurosci*, 19(3):356–365, March 2016. doi: 10.1038/nn.4244.

10. McClure P and Kriegeskorte N. Representational Distance Learning for Deep Neural Networks. *Front Comput Neurosci*, 10, 2016. doi: 10.3389/fncom.2016.00131.

11. Pospisil DA, Pasupathy A, and Bair W. 'Artiphysiology' reveals V4-like shape tuning in a deep network trained for image classification. *eLife*, 7:e38242, December 2018. doi: 10.7554/eLife.38242.

12. Mante V, Sussillo D, Shenoy KV, and Newsome WT. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84, November 2013. doi: 10.1038/nature12742.

13. Cueva CJ, Marcos E, Saez A, Genovesio A, Jazayeri M, Romo R, Salzman CD, Shadlen MN, and Fusi S. Low dimensional dynamics for working memory and time encoding. *bioRxiv*, page 504936, March 2019. doi: 10.1101/504936.

14. Wang J, Narain D, Hosseini EA, and Jazayeri M. Flexible timing by temporal scaling of cortical responses. *Nat Neurosci*, 21(1):102–110, January 2018. doi: 10.1038/s41593-017-0028-6.

15. Michaels JA, Schaffelhofer S, Agudelo-Toro A, and Scherberger H. A neural network model of flexible grasp movement generation. *bioRxiv*, page 742189, August 2019. doi: 10.1101/742189.

16. Stroud JP, Porter MA, Hennequin G, and Vogels TP. Motor primitives in space and time via targeted gain modulation in cortical networks. *Nature Neuroscience*, 21(12):1774, December 2018. doi: 10.1038/s41593-018-0276-0.

17. Hunsberger E and Eliasmith C. Spiking Deep Networks with LIF Neurons. *arXiv:1510.08829 [cs]*, October 2015.

18. Lee JH, Delbruck T, and Pfeiffer M. Training Deep Spiking Neural Networks Using Backpropagation. *Front Neurosci*, 10, 2016. doi: 10.3389/fnins.2016.00508.

19. Esser SK, Merolla PA, Arthur JV, Cassidy AS, Appuswamy R, Andreopoulos A, Berg DJ, McKinstry JL, Melano T, Barch DR, di Nolfo C, Datta P, Amir A, Taba B, Flickner MD, and Modha DS. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc Natl Acad Sci U S A*, 113(41): 11441–11446, October 2016. doi: 10.1073/pnas.1604850113.

20. Pfeiffer M and Pfeil T. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Front Neurosci*, 12, 2018. doi: 10.3389/fnins.2018.00774.

21. Shrestha SB and Orchard G. SLAYER: Spike Layer Error Reassignment in Time. *NeurIPS*, pages 1419–1428, 2018.

22. Bellec G, Salaj D, Subramoney A, Legenstein R, and Maass W. Long short-term memory and Learning-to-learn in networks of spiking neurons. *NeurIPS*, pages 795–805, 2018.

23. Cramer B, Stradmann Y, Schemmel J, and Zenke F. The Heidelberg spiking datasets for the systematic evaluation of spiking neural networks. *arXiv:1910.07407 [cs, q-bio]*, December 2019.

24. Neftci EO, Mostafa H, and Zenke F. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-based optimization to spiking neural networks. *IEEE Signal Process Mag*, 36(6):51–63, November 2019. doi: 10.1109/MSP.2019.2931595.

25. Zenke F and Ganguli S. SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Comput*, 30(6):1514–1541, April 2018. doi: 10.1162/neco_a_01086.

26. Huh D and Sejnowski TJ. Gradient Descent for Spiking Neural Networks. *NeurIPS*, pages 1440–1450, 2018.

27. Woźniak S, Pantazi A, Bohnstingl T, and Eleftheriou E. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6):325–336, June 2020. doi: 10.1038/s42256-020-0187-0.

28. Gütig R. Spiking neurons can discover predictive features by aggregate-label learning. *Science*, 351(6277):aab4113, March 2016. doi: 10.1126/science.aab4113.

29. Gütig R and Sompolinsky H. The tempotron: a neuron that learns spike timing-based decisions. *Nat Neurosci*, 9(3):420–428, March 2006. doi: 10.1038/nn1643.

30. Hochreiter S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *Int J Unc Fuzz Knowl Based Syst*, 06(02): 107–116, April 1998. doi: 10.1142/S0218488598000094.

31. He K, Zhang X, Ren S, and Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

32. Mishkin D and Matas J. All you need is a good init. *arXiv:1511.06422 [cs]*, February 2016.

33. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, and Chintala S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*, pages 8026–8037, 2019.

34. Mostafa H. Supervised Learning Based on Temporal Coding in Spiking Neural Networks. *Trans Neural Netw Learn Syst*, 29(7):3227–3235, July 2018. doi: 10.1109/TNNLS.2017.2726060.

35. Zimmer R, Pellegrini T, Singh SF, and Masquelier T. Technical report: supervised training of convolutional spiking neural networks with PyTorch. *arXiv:1911.10124 [cs, stat]*, November 2019.

36. Warden P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv:1804.03209 [cs]*, April 2018.

37. Schemmel J, Briiderle D, Griibl A, Hock M, Meier K, and Millner S. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1947–1950. IEEE, 2010.

38. Sterling P and Laughlin S. *Principles of Neural Design*. The MIT Press, reprint edition edition, June 2017. ISBN 978-0-262-53468-0.

39. Boahen K. A neuromorph's prospectus. *Comput Sci Eng*, 19(2):14–28, March 2017. doi: 10.1109/MCSE.2017.33.

40. Neftci EO. Data and Power Efficient Intelligence with Neuromorphic Learning Machines. *iScience*, 5:52–68, July 2018. doi: 10.1016/j.isci.2018.06.010.

41. Roy K, Jaiswal A, and Panda P. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, November 2019. doi: 10.1038/s41586-019-1677-2.

42. Cramer B, Billaudelle S, Kanya S, Leibfried A, Grübl A, Karasenko V, Pehle C, Schreiber K, Stradmann Y, Weis J, Schemmel J, and Zenke F. Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate. *arXiv:2006.07239 [cs, q-bio, stat]*, June 2020.

43. Bohte SM. Error-Backpropagation in Networks of Fractionally Predictive Spiking Neurons. In *Artificial Neural Networks and Machine Learning – ICANN 2011*, Lecture Notes in Computer Science, pages 60–68. Springer, Berlin, Heidelberg, June 2011. ISBN 978-3-642-21734-0 978-3-642-21735-7. doi: 10.1007/978-3-642-21735-7_8.

44. Orchard G, Jayawant A, Cohen GK, and Thakor N. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Front. Neurosci.*, 9, 2015. doi: 10.3389/fnins.2015.00437.

45. Amir A, Taba B, Berg D, Melano T, McKinstry J, Di Nolfo C, Nayak T, Andreopoulos A, Garreau G, Mendoza M, and others. A Low Power, Fully Event-Based Gesture Recognition System. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017.

46. Crick F. The recent excitement about neural networks. *Nature*, 337(6203): 129–132, January 1989. doi: 10.1038/337129a0.

47. Williams RJ and Zipser D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

48. Murray JM. Local online learning in recurrent networks with random feedback. *eLife*, 8:e43299, May 2019. doi: 10.7554/eLife.43299.

49. Bellec G, Scherr F, Hajek E, Salaj D, Legenstein R, and Maass W. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv:1901.09049 [cs]*, January 2019.

50. LeCun Y, Cortes C, and Burges CJ. *The MNIST database of handwritten digits.* 1998.

51. Huang X, Acero A, Hon HW, and Reddy R. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development.* Prentice Hall, 2001. ISBN 978-0-13-022616-7.

52. Vogels TP and Abbott LF. Signal propagation and logic gating in networks of integrate-and-fire neurons. *J Neurosci*, 25(46):10786, 2005. doi: 10.1523/JNEUROSCI.3508-05.2005.

53. Gerstner W, Kistler WM, Naud R, and Paninski L. *Neuronal dynamics: from single neurons to networks and models of cognition.* Cambridge University Press, Cambridge, 2014. ISBN 978-1-107-06083-8.

54. Kingma D and Ba J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014.

55. Zenke F. SpyTorch, March 2019. URL `https://github.com/fzenke/spytorch`.

Parameter	Randman	MNIST	SHD	RawHD	RawSC
Number of input units	20	784	700	40	40
Number of hidden units	100	100	256	256	256
Number of readout units	10	10	20	20	35
Δt / Number of steps	1ms / 100	1ms / 100	2ms / 500	2ms / 80	2ms / 100
Minibatch size	128, 250	256	256	128	512
Number of epochs	50–100	100	200	50	50
Dataset (train/valid./test)	8k/2k/2k	45k/5k/10k	7498/833/2088	7498/833/2088	84849/9981/11005
Number of hidden layers n_h	≤ 2	≤ 3	≤ 2	1	1
Learning rate sweep	$10^{-3} \leq \eta \leq 0.1$	$2 \times 10^{-4} \leq \eta \leq 1^{-1}$	$2 \times 10^{-4} \leq \eta \leq 1^{-1}$	1×10^{-3}	1×10^{-3}, 5×10^{-3}, 0.01
Best η	0.05	0.01	1×10^{-3}	1×10^{-3}	1×10^{-3}
Lower L2: λ_lower / ν_lower	$100.0/10^{-3}$	—	$100.0/10^{-3}$	$100.0/10^{-3}$	$100.0/(0.01,10^{-3})$
Upper L1 strength $\lambda_\mathrm{upper,1}$	1,100	0–100	0.06	0, 10, 20, 100	0.06, 1, 10
Upper L1 threshold $\nu_\mathrm{upper,1}$	0–1000	0–1000	0–1000	0, 0.1, 0.2, 0.5, 1, 100	0–1000
Upper L2 strength $\lambda_\mathrm{upper,2}$	0,1,100	0, 0.001, 0.1 , 1	—	—	0,1
Upper L2 threshold $\nu_\mathrm{upper,2}$	0–100	0.1, 100	—	—	10,100

Table 1. Parameter values of network simulations.