Reducing the spike rate of deep spiking neural networks based on time-encoding

IOP Publishing Neuromorph. Comput. Eng. 4 (2024) 034004 https://doi.org/10.1088/2634-4386/ad64fd

NEUROMORPHIC
Computing and Engineering

OPEN ACCESS

RECEIVED
10 April 2024

ACCEPTED FOR PUBLICATION
18 July 2024

PUBLISHED
29 July 2024

PAPER

Reducing the spike rate of deep spiking neural networks based on time-encoding

Riccardo Fontanini, Alessandro Pilotto, David Esseni and Mirko Loghi*

DPIA, Università di Udine, Udine, Italy
* Author to whom any correspondence should be addressed.

E-mail: mirko.loghi@uniud.it

Keywords: spiking neural networks, spike rate, learning strategy, loss function

Abstract

A primary objective of Spiking Neural Networks is a very energy-efficient computation. To achieve this target, a small spike rate is of course very beneficial given the event-driven nature of such a computation. A network that processes information encoded in spike timing can, by its nature, have such a sparse event rate, but, as the network becomes deeper and larger, the spike rate tends to increase without any improvements in the final accuracy. If, on the other hand, a penalty on the excess of spikes is used during the training, the network may shift to a configuration where many neurons are silent, thus affecting the effectiveness of the training itself. In this paper, we present a learning strategy to keep the final spike rate under control by changing the loss function to penalize the spikes generated by neurons after the first ones. Moreover, we also propose a 2-phase training strategy to avoid silent neurons during the training, intended for benchmarks where such an issue can cause the switch off of the network.

1. Introduction

Conventional Artificial Neural Networks (ANNs) have reported great success in application domains that are difficult to tackle with an algorithmic approach, such as text and speech recognition [1, 2], classification and segmentation of images [3], and robotic control [4]. The inter-neural communication in ANNs occurs through continuous activation values produced by non-linear but differentiable functions, and the training of the networks targets the minimization of a loss function that is also differentiable. Consequently, the back-propagation approach is a gradient-based optimization of the computational graph that is both well-established and very effective [5].

Spiking Neural Networks (SNNs) have been labeled as the third generation of neural networks [6], and behave as an event-driven computational system, where asynchronous spikes are used for communication among neurons. SNNs can solve the same classes of problems addressed by ANNs, but they can mimic much better than ANNs the behavior of the brain, and hopefully emulate also the outstanding energy efficiency of the computation in biological systems. The energy efficiency of SNNs stems from the sparsity of events in time and on the asynchronous and local nature of the computation [7], which makes the networks very promising for energy-constrained, edge computing applications, such as internet of things, autonomous vehicles, as well as wearable and implantable devices [8, 9].

For SNNs with information encoded in the spike rate, several approaches have been proposed, as Spatio-Temporal Backpropagation [10], activation check points [11], and local Backpropagation Through Time [12]. Such approaches, however, for the intrinsic nature of the information encoding, do not consider the minimization of the spike rate as a figure of merit.

On the contrary, if the information is encoded in the timing of each spike, a sparse rate can in principle be achieved without penalizing the network operation, since the rate and the information encoding are no longer tightly correlated. In such cases, however, the computation of the gradients would require the differentiation of the activation potential over time. Nevertheless, since the spikes in SNNs can be formally described as Dirac delta functions, a straightforward application of differential calculus to compute gradients

is not feasible, thus complicating the learning strategies compared to the well-assessed solutions employed in ANNs.

Such a challenge has been tackled by resorting to different strategies, including (a) the conversion of weights computed for an ANN to weights suitable for an SNN [13]; (b) the use of a differentiable surrogate function (and the corresponding surrogate gradient) [14]; (c) the computation of the exact gradients for some special cases [15, 16].

Thanks to the relatively simple formulation of the Leaky Integrate-and-Fire (LIF) model for neurons, exact gradient computation has been mainly derived for the LIF model. However, because it has been theoretically proven that spiking networks have a Turing-equivalent computing power even for very basic mechanisms of spike generation [17], the use of LIF neurons should not be perceived as an actual limitation, even if more biophysically accurate neuron models have been proposed, such as the Hodgkin and Huxley model [18].

For a feed-forward network with a single hidden layer consisting of leaky integrate-and-fire neurons, which employs a spike-timing encoded input and has a loss function that depends on the times of the spikes, exact gradients have been computed with EventProp in [19]. This was accomplished by using adjoint functions, which are evaluated and recorded only at the time instants corresponding to spikes. This approach leverages the sparseness of spikes to obtain a computation of the gradient that is effective in terms of both computation energy and memory requirements.

In deep SNNs, however, a mere gradient descent approach has limitations even when the exact gradient can be calculated. In fact the average spike rate tends to accelerate for an increasing number of layers, which deteriorates the energy efficiency of both the learning and the inference phase. In this respect, if measures are introduced in order to control the spike rate, one should carefully avoid inducing a large number of silent neurons, namely neurons that do not emit spikes. Such neurons, in fact, do not practically contribute to the loss function, so a gradient descent strategy is no longer effective in optimizing their input weights. This hazard is similar to the 'vanishing' gradient issue that in ANNs precludes the convergence toward the actual minimum of the loss function.

In this work, we propose a modification of the loss function used in EventProp [19], associated with a possible change in the learning strategy, in order to reduce the spike rate and, at the same time, avoid the silent neuron issue. A preliminary report of this research has been presented in a conference paper [20]. Here, we extend that work with a more detailed analysis and more thorough experimental validations that involve additional benchmarks and networks.

2. Related work

For a feed-forward network of LIF neurons, the gradient of a first-spike-time-based loss function has been computed in [15], by resorting to a linear approximation of the thresholding function near the spike time. In [16] such an approach has been extended to a loss dependent on an arbitrary number of output spikes, whereas [21] addressed the case of recurrent networks.

Different approaches have been developed by computing exact gradients using methods from the optimal control theory. In [22], the sensitivity analysis has been applied to a recurrent network of LIF neurons that aims to exhibit a given oscillatory behavior. Algorithms for recurrent SNNs have been derived in [23], by leveraging adjoint equations and accounting for the hybrid dynamic of neurons, as well as in [24], where the threshold for neuron firing has been replaced by a gate function that smoothens the transitions and facilitates the implementation of the adjoint methods.

As it has been already mentioned, exact gradient computation for SNNs has been also developed in EventProp [19], where the discontinuity on the adjoint equations enforced by spikes are computed going backward in time in a topological order. Here the gradient is expressed in terms of the values of the adjoint functions at the spike times only, thus leveraging the sparsity of the events to improve the effectiveness of the computation.

None of the aforementioned approaches, however, implies any target for the possible spikes generated after the classification has been completed. Such spikes can occur when neurons are over-excited from a large number of afferents, which is statistically more probable when the network becomes deeper. While an excess of spikes is innocuous for the network operation, it is clearly detrimental for energy consumption and, moreover, it is also biologically implausible because the refractory mechanism can effectively limit the spike rate in biological neurons. In this latter respect, we here argue that a refractory period appears indispensable in fully-connected recurrent SNNs to prevent an unlimited spike rate [25], but this also implies an energy cost, because the refractoriness is obtained with a dissipative configuration in most circuital implementations [26]. In a feed-forward network, instead, the excess of spikes can also be avoided by a judicious choice of the synaptic weights, which makes the implementation of the refractory period not strictly necessary and thus

saves the corresponding energy cost. The targets of this work are in fact deep feed-forward networks with a time encoding of the information, and we will present a training strategy that can optimize the network performance and, at the same time, keep the spike rate under control.

3. Method

3.1. Models

3.1.1. Spiking neurons and gradients

The constitutive equations of the LIF neurons are the same used in EventProp [19]:

$$\begin{cases} \tau_m \frac{\partial V}{\partial t} = -V + R_m I \\ \tau_s \frac{\partial I}{\partial t} = -I \end{cases} \tag{1}$$

where V and I are the neural membrane potential and the input synaptic current respectively. τ_m and τ_s are the time constants governing the dynamics of the membrane potential and the synaptic current respectively, while R_m is the membrane resistance. The state variables V and I of each neuron are initialized to zero:

$$\begin{cases} V(t=0) = 0 \\ I(t=0) = 0 \end{cases} \tag{2}$$

The times of spikes are implicitly defined by:

$$\left(\mathbf{V}^{-}\right)_{n(k)} = \Theta \tag{3}$$

where the vectors $\mathbf{V}$ and $\mathbf{I}$ gather all the membrane potentials and currents of the network, and Θ is the constant threshold potential. The notation $\left(\mathbf{V}^{\pm}\right)_{n(k)}$ denotes, throughout this work, the n_{th} component of the membrane potential vector $\mathbf{V}$ an instant before $(-)$, or after $(+)$, the emission of the k_{th} spike. When the membrane potential of a neuron reaches the threshold, such a neuron emits a spike and resets to the rest potential ($\left(\mathbf{V}^{+}\right)_{n(k)} = 0$ V), though the current does not change ($\left(\mathbf{I}^{-}\right)_{n(k)} = \left(\mathbf{I}^{+}\right)_{n(k)}$) since a node does not act on itself. However, the k_{th} spike produces a jump on the current of the receiving neuron given by the synaptic weight $\left(\mathbf{I}^{+}\right)_{m} = \left(\mathbf{I}^{-}\right)_{n(k)} + W_{m,n}$, where $W_{m,n}$ links the nth spiking neuron to the mth neuron. Here it should be noticed that this differential model is similar to the Spike-Response Model (SRM) [25] and that, moreover, the dynamic of the current I is necessary to maintain information in the spike timing. In fact, with a finite τ_s value the weight of an afferent synapse can affect the delay between the output and the input spike, whereas this modulation becomes progressively impossible when τ_s tends to zero.

To train the network we compute the exact gradient, leveraging adjoint variables and back-propagation as in EventProp [19], to perform a mini-batch stochastic gradient-descent. The equations of adjoint variables can be derived from equation (1) and are:

$$\begin{cases} \tau_m \frac{\partial \lambda_V}{\partial t} = \lambda_V \\ \tau_s \frac{\partial \lambda_I}{\partial t} = \lambda_I - R_m \lambda_V \end{cases} \tag{4}$$

$$\begin{cases} \left(\lambda_V{}^{-}\right)_{n(k)} & = \frac{1}{R_m \left(I^{-}\right)_{n(k)} - \theta} \left[\left(\lambda_V{}^{+}\right)_{n(k)} R_m \left(I^{-}\right)_{n(k)} + \frac{\partial \mathcal{L}(t_k)}{\partial t_k} + \sum_{m \neq n}^{N_n} \left(R_m \left(\lambda_V{}^{+}\right)_m - \left(\lambda_I{}^{+}\right)_m \right) W_{m,n} \right] \\ \left(\lambda_I{}^{-}\right)_{n(k)} & = \left(\lambda_I{}^{+}\right)_{n(k)} \end{cases} . \tag{5}$$

The adjoint variables have discontinuities (ruled by equation (5)) in correspondence with spikes and are evaluated backward in time, starting from the initial condition:

$$\begin{cases} \lambda_V(t=T) = 0 \\ \lambda_I(t=T) = 0 \end{cases} . \tag{6}$$

An example of the neural state functions and the adjoint functions is represented in figure 1(a).

All the loss functions considered in this work only depend on the spike times; we do not take into account losses that depend on the integral of the membrane potential, since they are not consistent with an event-driven computation. Hence, the gradient of the loss function can be written as:

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = \sum_{k=1}^{N_{spk}} \frac{d\,\mathcal{L}(t_k)}{dt_k} \frac{dt_k}{dW_{i,j}} \tag{7}$$

where N_{spk} is the number of spikes generated by the network.

By applying the implicit function theorem to equation (3) and exploiting the adjoint variables, the gradient of the loss becomes:

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = -\tau_s \sum_{k \in \{\text{spikes from } j\}} (\boldsymbol{\lambda_I})_i (t_k). \tag{8}$$

Hence, to calculate the gradient of $\mathcal{L}$, we only need to sample λ_I when a spike happens. Moreover, since the state functions and their adjoints can be expressed in closed form between two discontinuities, all the evaluations can be performed with an event-driven computation.

3.1.2. Loss functions

We consider deep feed-forward networks for classification tasks. The network receives input spikes, related to input data through an application-specific encoding, and propagates spikes up to the output neurons in the last layer. The output node that emits the first spike indicates the predicted class.

The first loss function taken into account is [19]:

$$L_W = CE + \alpha \cdot CS \tag{9}$$

where CE (cross-entropy) and CS (classification spike) are defined as:

$$\begin{cases} CE = -\frac{1}{N_b} \sum_{b=1}^{N_b} \log \left(\frac{\exp\left(-t_{b,1(T)}/\tau_0\right)}{\sum_{a=1}^{N_O} \exp\left(-t_{b,1(a)}/\tau_0\right)} \right) \\ CS = \frac{1}{N_b} \sum_{b=1}^{N_b} \left[\exp\left(\frac{t_{b,1(T)}}{\tau_1} \right) - 1 \right] \end{cases} \tag{10}$$

with N_b and N_O being the number of samples of the batch and output neurons, respectively. τ_0, τ_1 are normalization constants, and α is a hyper-parameter. $t_{b,1(T)}$ is the time of the first spike emitted by the output neuron T corresponding to the correct classification (the *target* neuron), and $t_{b,1(a)}$ is, instead, the time of the first spike of the generic output neuron a.

CE is the actual target of the minimization since it is directly related to the average classification error: it is in fact a measure of how much the real distribution of the dataset differs from the distribution estimated by the network [27]. Since CE depends only on the *difference among times* of the first output spikes, we can add terms to the loss function to drive the minimization towards points with convenient characteristics. Here CS aims to minimize the delay between the input and the output spikes and, as a by-product, to reduce the probability that neurons remain silent.

For shallow networks, L_W provides good results [19], albeit for deep networks a good level of accuracy is obtained at the cost of a large spike rate, as discussed in section 4. Trying to reduce the spike rate by decreasing the hyper-parameter α leads to a relevant reduction of the accuracy since many neurons become silent.

Therefore, to improve the control of the spike rate, we defined a second loss function by adding a term that, for each neuron, reduces the excess of spikes without penalizing the first one:

$$L = CE + \alpha \cdot CS + \eta \cdot SP \tag{11}$$

SP (spike penalty) is defined as:

$$SP = \frac{1}{N_b} \sum_{b=1}^{N_b} \sum_{a=1}^{N_n} \sum_{p=2}^{P(a,b)} \frac{1}{\Delta t_{p,a,b}}. \tag{12}$$

N_n is the number of neurons of the network, $P(a,b)$ is the total number of spikes emitted by the neuron a during the inference of sample b, and η is another hyper-parameter. $\Delta t_{p,a,b} = t_{b,p(a)} - t_{b,1(a)}$ denotes the time difference between each spike produced by the neuron a during the inference of the sample b ($t_{b,p(a)}$) and the first one ($t_{b,1(a)}$) (for the sake of clarity, figure 1(b) provides a representation of the spiking notation). If a given neuron emits only a single spike, its contribution to SP is null, thus the spike penalty tends to enlarge the delay between the first emitted spike and the following ones for each neuron of the network.

With the loss function L, the network could, in principle, reach the same CE performance obtained with L_W also reducing the average spike rate. However, such a result is not always easily achievable, because of the detrimental influence of silent neurons. As it will be discussed in more detail in section 4, a low spike rate is obtained with small values of α that, as a drawback, may lead to an early switch off of nodes and,

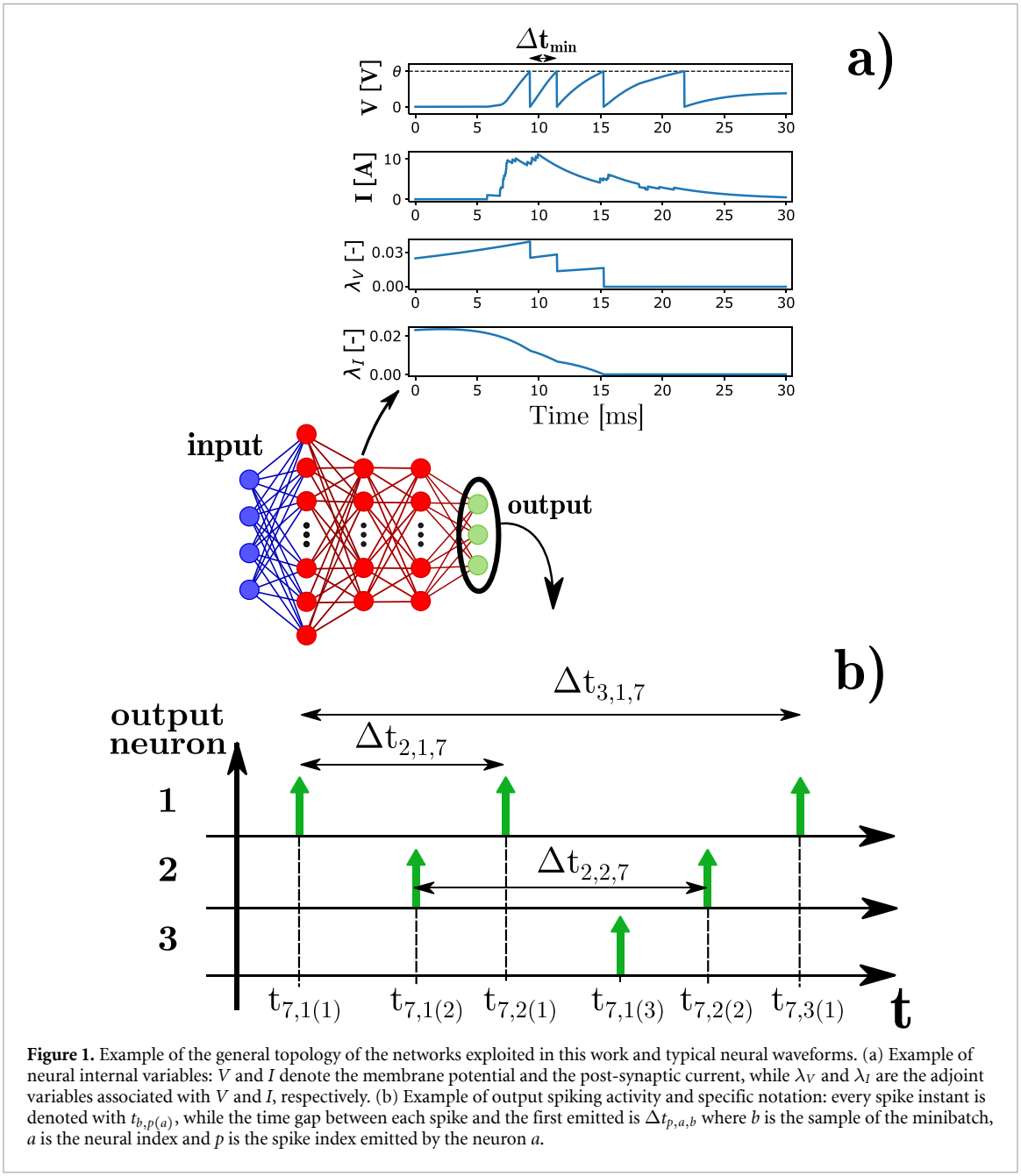

Figure 1. Example of the general topology of the networks exploited in this work and typical neural waveforms. (a) Example of neural internal variables: V and I denote the membrane potential and the post-synaptic current, while λ_V and λ_I are the adjoint variables associated with V and I, respectively. (b) Example of output spiking activity and specific notation: every spike instant is denoted with $t_{b,p(a)}$, while the time gap between each spike and the first emitted is $\Delta t_{p,a,b}$ where b is the sample of the minibatch, a is the neural index and p is the spike index emitted by the neuron a.

consequently, to poor accuracy. Silent neurons, in fact, hamper the minimization process because they do not contribute to the gradient of the loss (equation (8)). On the contrary, to guarantee an acceptable final accuracy, large values of α are needed, but such a choice offsets the spike rate reduction we are aiming for. Nevertheless, we observed that in some cases the minimization does converge to good values of CE even for small values of α, thus suggesting that, with the proper choice of the hyper-parameters, L should be able to yield both a low CE and a low spike rate. Finding suitable values for α and η, however, is not at all trivial, because the influence of hyper-parameters strongly depends on the starting point of the minimization and the network topology.

Therefore, for those cases where a strategy to reach a convenient starting point is required, we introduce a third loss function, L_A:

$$L_A = CE + \alpha \cdot AS + \eta \cdot SP \tag{13}$$

where AS (additional spikes) is an augmentation term, defined as:

$$AS = \frac{1}{N_O N_b} \sum_{b=1}^{N_b} \sum_{a=1}^{N_O} \left[\exp\left(\frac{t_{b,1(a)}}{\tau_1} \right) - 1 \right] \tag{14}$$

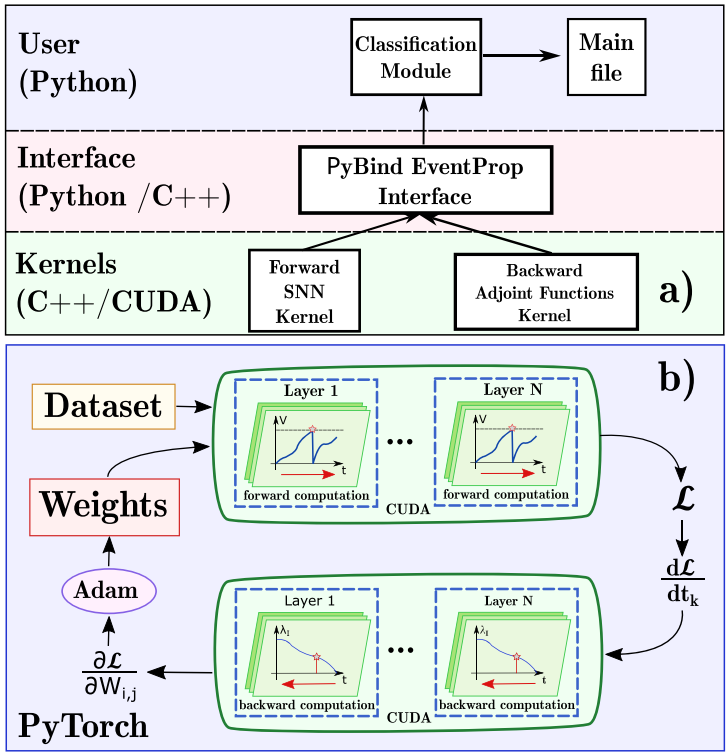

Figure 2. Sketch of the implemented algorithm. (a) The architecture of the simulator. (b) Computation flow of the algorithm. The simulator is based on Pytorch with a CUDA-implemented extension for the calculation of the neural signals and the adjoint functions.

that focuses on the first spike event of *all* output neurons.

Notice that this term can be written as:

$$\begin{aligned} AS = &\frac{1}{N_O N_b}\sum_{b=1}^{N_b}\left[\exp\left(\frac{t_{b,1(T)}}{\tau_1}\right)-1\right] \\ &+\frac{1}{N_O N_b}\sum_{b=1}^{N_b}\sum_{a\in NT}\left[\exp\left(\frac{t_{b,1(a)}}{\tau_1}\right)-1\right] \\ =&\frac{CS}{N_O}+\frac{1}{N_O N_b}\sum_{b=1}^{N_b}\sum_{a\in NT}\left[\exp\left(\frac{t_{b,1(a)}}{\tau_1}\right)-1\right] \end{aligned} \tag{15}$$

where, again, T is the target neuron and NT is the set of the not-target nodes.

Hence, L_A contains the same terms as L (up to a constant pre-factor which can be considered as part of the hyper-parameter) plus the extra addends related to other spikes. An important consideration, however, is that every minimum of L_W and L is also a minimum of CE, but this is not necessarily the case for L_A because of the term AS. Nevertheless, a minimum of L_A is close enough to a minimum of CE to be considered a good starting point for a successive minimization and, moreover, it can be reached without incurring in the switch-off issue, as it has been empirically demonstrated by our experimental results.

In the following, we propose the use of the loss function L to minimize both the cross-entropy and the number of spikes emitted by the network during inference. Moreover, for those cases where the use of L leads to the switch off of the network, we propose a 2-phase strategy for learning. In this latter strategy, during the first phase, the training aims at the minimization of L_A and thus provides a good starting point for the second phase, in which the loss L finally leads to a minimization of the cross-entropy.

3.2. SNN simulator

We simulated the networks using the PyTorch [28] framework. The simulator is implemented in Python, to interact with PyTorch, and in CUDA/C++, to accelerate the computation by exploiting the parallelism exposed by an Nvidia GPU. Figure 2(a) shows the architecture of the simulator, with the Python modules in charge of loading the dataset and of orchestrating the simulation. The CUDA code computes the spike times (finding the roots of equation (3)) and updates the adjoint variables for each discontinuity. The final

Table 1. Model's parameters.

Description	Symbol	Value
Membrane time constant	τ_m	20 ms
Synaptic time constant	τ_s	5 ms
Membrane resistance	R_m	1 Ω
Threshold	Θ	1 V
Min input time spike	t_{min}	0 ms
Max input time spike	t_{max}	20 ms
Simulation time	T	30 ms
Adam parameter	β_1	0.9
Adam parameter	β_2	0.999
Adam parameter	ϵ	1×10^{-8}
Learning rate decay factor	—	0.95
Learning rate decay step	—	1 epoch
CE normalization constant	τ_0	0.5 ms
CS, AS normalization constant	τ_1	6.4 ms
Yin-Yang 1^{st} phase Learning rate	—	5×10^{-3}
Yin-Yang 2^{nd} phase Learning rate	—	2×10^{-4}
Yin-Yang batch size	N_b	32
MNIST Learning rate (*both phases*)	—	2×10^{-3}
MNIST batch size	N_b	20

computation of the gradient and the update of synaptic weights is performed by the PyTorch framework through its implementation of the Adam optimizer [29]. The whole flow of the computation is represented in figure 2(b).

The network hyper-parameters are listed in table 1, with values in the upper portion common to all the simulated networks.

3.3. Yin-Yang benchmark

The first benchmark taken into account is the Yin-Yang dataset [30, 31] composed of points in a 2D space. Each point lies in a region, defined by non-linear boundaries, which is the target of the classification (there are three regions). A point is translated to spikes by means of a linear space-time conversion within a fixed range $[t_{min}, t_{max}]$ of its coordinates (x, y) and their duals $(1-x, 1-y)$. The dual spikes are added to keep the average spike times of the inputs constant for the whole dataset. Moreover, an input spike at fixed time $t = t_{min}$ is added as a bias for a total of 5 inputs for the classifying SNN. Figure 3(a) shows the structure of the dataset and an example of the space-time conversion.

This benchmark is small enough, still not trivial, to allow the fast preliminary evaluations of the learning strategies that we discussed above. To achieve fair and meaningful comparisons, we use a network with almost the same number of synapses as the network used in [19], but deeper and, thus, with fewer neurons: for this benchmark we employ a 5/40/25/13/3 feed-forward network with synaptic weights initialized with a uniform distribution in the range reported in table 2.

3.4. MNIST benchmark

The second benchmark considered is the MNIST dataset [32]. MNIST is composed of images (28x28 pixel large, gray-scale) that depict the hand-written digits to be recognized by the network. The training set contains 60 000 images, while the images in the test set are 10 000. Each pixel of the input image is translated to an input spike: the spiking time is a linear mapping of the pixel intensity to the range $[t_{min}, t_{max}]$ as shown in figure 3(b). Since the majority of the pixels has the background color and the input data are large enough (784 pixels), there is no need to add dual inputs or a bias.

For this benchmark, we use a 784/280/160/70/10 network, that requires more than $2 \cdot 10^5$ synapses. The number of synapses is too large, compared to the number of images in the training set, to avoid overfitting. Therefore, we augmented the training set by introducing random shifts (horizontal and vertical, up to 10%) and random rotations (up to 20°) of the images. Even for this benchmark, the synaptic weights are initialized with a uniform distribution: the ranges are reported in table 2.

The starting weight ranges of both benchmarks are tuned to initially ignite all neurons of the network and produce a sparse enough spiking activity to prevent an over-excitation of the output layer's neurons. When the 2-phase learning strategy is adopted, the learning rate for the Yin-Yang analysis has been changed between the first and the second phase of training to better follow the distribution of the average weights of the network, as reported in table 1. Conversely, the MNIST preliminary exploration reported in the next

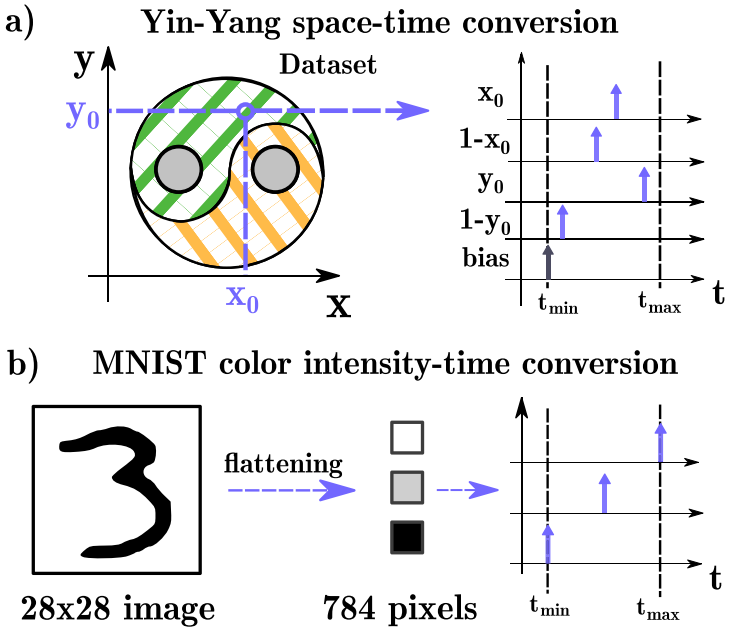

Figure 3. Datasets conversion and example of neural signals. (a) Representation of the Yin-Yang Dataset and the linear transformation of spatial coordinates into spiking time events. (b) Example of conversion of a MNIST image: the pixel color intensity is translated into a spike timing with a linear transformation.

Table 2. Ranges for the initialization of the synaptic weights. Layer 0 is the input layer.

Layer	# Neurons	Minimum	Maximum
		Yin-Yang	
1	40	1.0	3.0
2	25	0.2	1.0
3	13	0.0	1.0
4	3	0.0	1.0
		MNIST	
1	280	−0.01	0.03
2	160	−0.05	0.15
3	70	−0.2	0.6
4	10	−0.4	0.8

paragraph reveals good behavior in the training process even by keeping the same learning rate for both phases.

4. Results and discussion

4.1. Yin-Yang simulation results

The first evaluation is the behavior of our network with three hidden layers, applied to the Yin-Yang benchmark, with the reference loss L_W and by varying the hyper-parameter α. Figure 4 reports (a) the accuracy and the mean arrival time of the first classification spike, and (b) the average number of spikes per neuron ($\langle N_S \rangle$) during inference for each layer. $\langle N_S \rangle$ is computed by summing all the spikes emitted by each neuron of a given layer of the network, and then by dividing such a sum by the number of neurons in that layer). Data are related to one inference performed by a trained network. For low values of α (the dashed area) the accuracy is quite poor since at least one layer (in particular the output layer 4) has $\langle N_S \rangle < 1$. This implies that some neurons are silent hence, at some point, the gradient descent lost the control to change the weights of synapses that are afferent to such neurons. Larger values of α provide better accuracy and, as a by-product, a faster response, but with the penalty of an excess of spikes. As stated above, too many spikes are harmful in terms of energy efficiency and should be thus avoided. The ideal condition is no more than a single spike per neuron per inference.

As an example of the behavior of the network *during* the training, we report two cases in figure 5 for 100 epochs. The first case (dashed lines: $\alpha = 5 \times 10^{-3}$, large enough to provide a good accuracy) shows how the

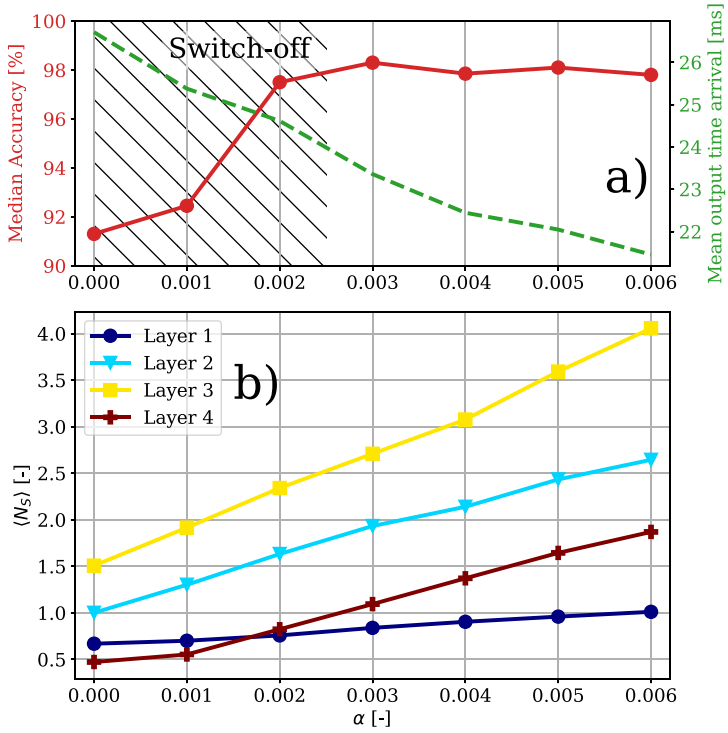

Figure 4. Yin-Yang—Exploration of the parameter α for loss L_W. (a) Comparison between the accuracy of the network (solid line) and the mean first spike arrival at the output layer (dashed line). (b) Dependency of the mean number of spikes emitted during inference by each layer of the network (note that Layer 0 is the input layer). The learning rate is set to 0.003.

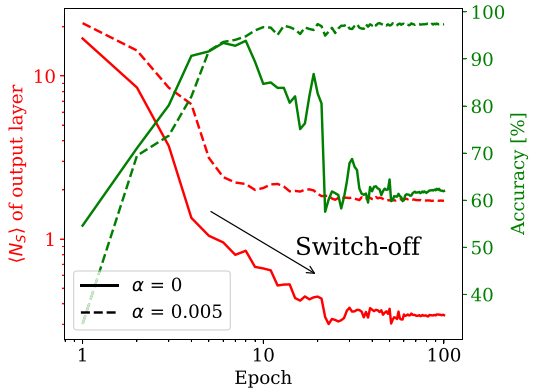

Figure 5. Yin-Yang—Example of switch off. Comparison between the average number of spikes per neuron $\langle N_S \rangle$ of the output layer and the accuracy of the network for two 100-epoch trainings, with the same starting point and using L_W as loss function.

spike rate, albeit reduced while the training is in progress, saturates at a value that is close to 2. The second case (solid lines: $\alpha = 0$) shows that the accuracy improves only for the first few epochs; when the spike rate becomes too small, too many nodes become quiescent, and the network accuracy drops to uselessness.

This first exploration shows that, even for a simple benchmark, keeping the spike rate under control while training a deep network is challenging, if not impossible, and it is the driving motivation for the changes in the loss function proposed in this work.

As mentioned above, the loss functions L and L_A introduce a new hyper-parameter (η). Figure 6 shows the impact of α on the average number of spikes of the output layer, for some values of η and at the end of a 100 epochs training, while figure 7 shows the accuracy for the same configurations. In both plots, dashed lines are for L and continuous lines are for L_A. Figure 6 points out how the excess of spikes can be totally eliminated by using L as the loss function, hence remarking the effectiveness of the added penalty term. The accuracy, however, is negatively impacted (figure 7), and, to reach acceptable values, large values of the hyper-parameter α are required, with the resulting increase in the number of spikes. Furthermore, the minimization of L is very dependent on the starting point and in general, it is not very reliable because, in

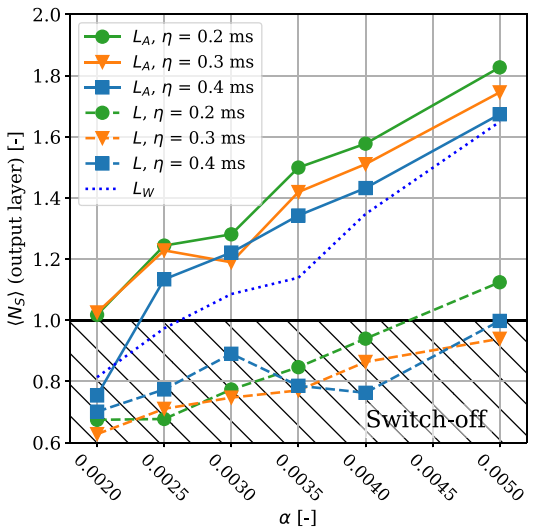

Figure 6. Yin-Yang—Average number of output spikes per neuron during inference for different α, η in the L and L_A cases. As a comparison, it is also shown the output spike rate of figure 4(b) (dotted line). The learning rate is set to 0.003.

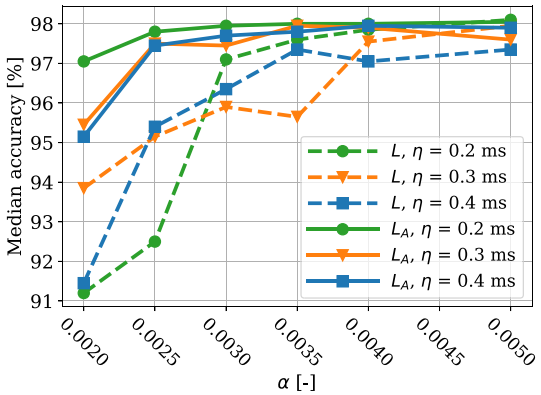

Figure 7. Yin-Yang—Median accuracy after 100 epochs of training. This plot shows the impact of α and η on the network's training for two different loss functions: L and L_A. The learning rate is set to 0.003.

many cases, some neurons become quiescent, hence adversely affecting the optimization. This is the reason for the nonmonotonic behavior of the accuracy for L shown in figure 7.

Conversely, the minimization with L_A keeps all the neurons active, because of the *AS* term, and provides an appreciable accuracy (but for the smallest values of α). This indicates that the minimization does not suffer from the switch off condition, as confirmed by the number of spikes of the output layer, hence the minimum of L_A is approached without halting in some plateau. However, as discussed in section 3.1, L_A cannot provide the actual minimum of the cross-entropy for the very nature of *AS*, hence a better accuracy may be achievable. The purpose of L_A, in fact, is allowing a *surrogate* minimization of L while forcing the nodes to stay active. Then, using the result of the minimization through L_A as the starting point of a second minimization, that uses L, we can avoid the aforementioned instability (because the initial point is quite close to the final minimum) and reach the real minimum of *CE*.

Figure 8 shows an example of 2-phase training, where the first phase (100 epochs, plot on the left) is performed minimizing L_A, while in the second phase (60 epochs, plot on the right) the target of the minimization is L. In the left plot, we also show the behavior of a training that only uses L and that successfully achieves a valid minimum; here it should be noticed that this is a selected lucky case, chosen among many unsuccessful tests, while all the training minimizations performed with L_A converge to a minimum. In the right plot, we can observe the improvement of the second phase, which brings the actual minimum of *CE* (dotted line), when L is used while L_A cannot provide any further improvement (continuous line).

The overall results of the 2-phase training, in terms of accuracy and average number of spikes during inference (for the whole network, that means by averaging over all the neurons in the SNN, and for the

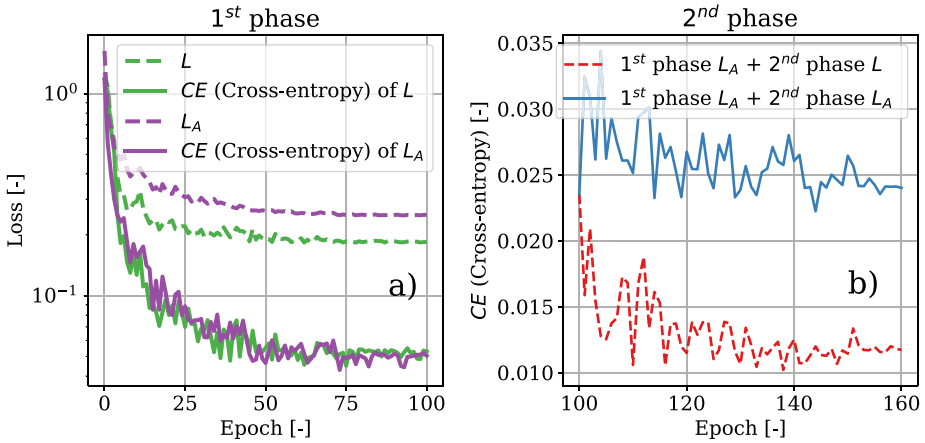

Figure 8. Yin-Yang - (a) Comparison between the total loss and the CE component of L and L_A during training and for $\alpha = 0.004$ and $\eta = 0.4$ ms. (b) CE component trends during the 2nd phase of training for L and L_A, started from the final result achieved for L_A in (a). The 2nd phase was performed with the same losses and parameters except for the learning rate, set to 2×10^{-4}. The final accuracy in the 2nd training phase improves, for the L case, from ∼98.4% to ∼98.7%.

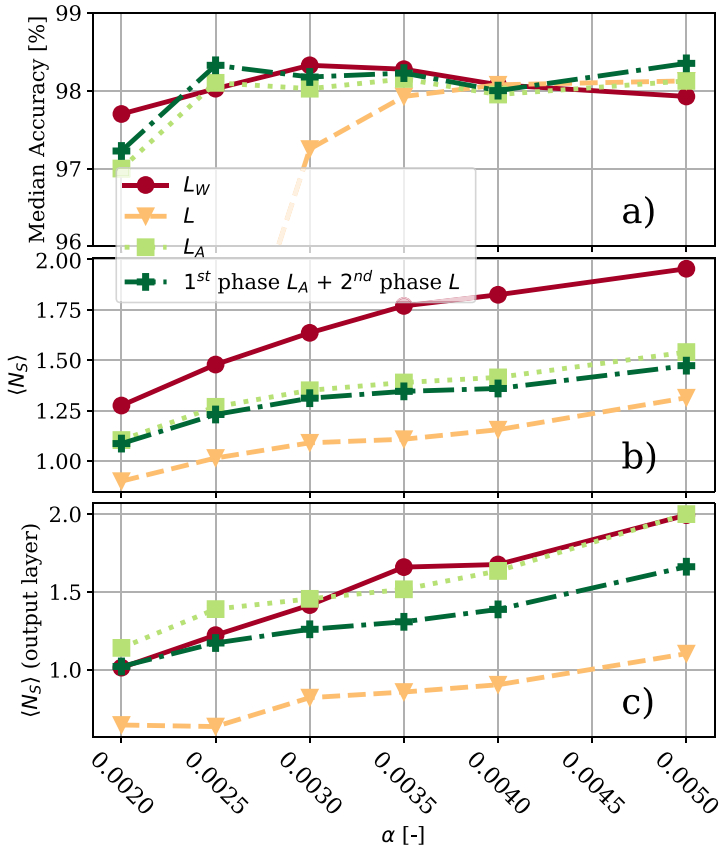

Figure 9. Yin-Yang—Summary of the training methodologies of this work. (a) shows the median accuracy of the network, (b) shows the average number of spikes per neuron of the whole network, and (c) shows the average number of spikes emitted during inference by the output layer. These results were obtained with $\eta = 0.3$ ms and averaged over 10 seeds per point. The learning rate is set to 0.005, except for the 2-phase learning, where it is set to 0.005 during the first phase and to 2×10^{-4} during the second phase, as mentioned in table 1.

output layer only), are depicted in figure 9 and are compared with the same results for training that use only L_W, L or L_A.

We notice that, for the configuration of hyper-parameters chosen in figure 9, the resulting accuracy of the 2-phase training is consistently better than the one obtained by using only L or L_A. Moreover, figure 9 highlights the fact that the 2-phase learning strategy allows us to achieve an accuracy that is similar to the one obtained by using L_W, also in terms of the standard deviation of the accuracy and of the cross-entropy

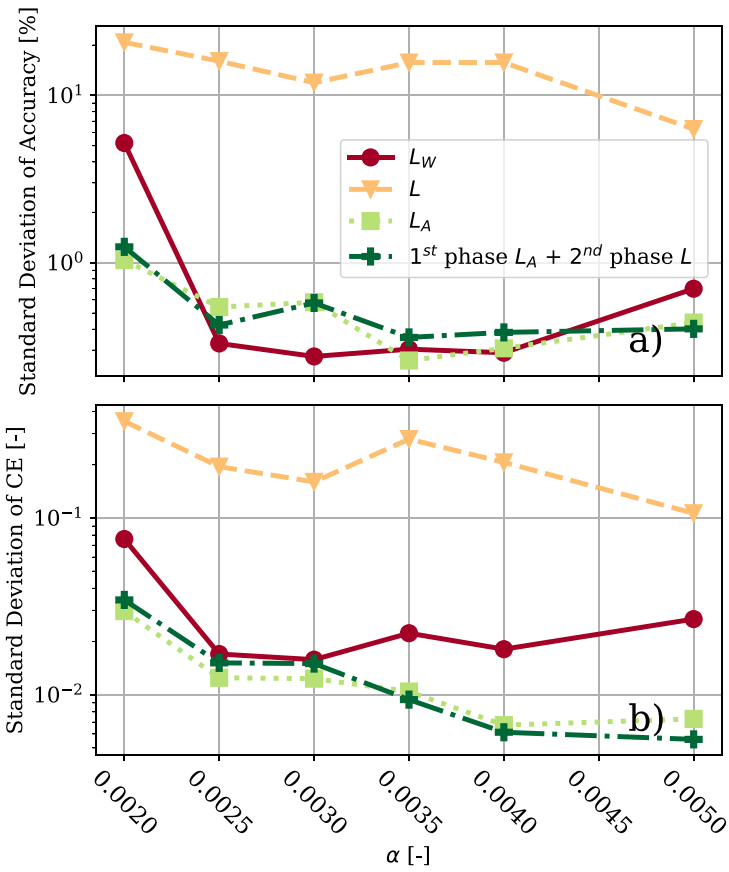

Figure 10. Yin-Yang—Standard deviation of the (a) accuracy and (b) the cross-entropy of the results reported in figure 9.

Table 3. Best results in terms of median accuracy for Yin-Yang.

Training	*Accuracy: Median, Mean (STD) (%)*	*Cross-Entropy (STD)*	$\langle N_S \rangle$ *(STD)*	$\alpha \times 10^{-3}$	η[ms]
L_W [19]	−,98.10 (2.00)	—	—	3.0	—
L_W	98.33, 98.28 (0.27)	0.036 (0.016)	1.64 (0.20)	3.0	—
L	98.25, 98.28 (0.27)	0.039 (0.012)	1.43 (0.11)	5.0	0.2
L_A	98.38, 98.18 (0.61)	0.042 (0.021)	1.42 (0.17)	3.5	0.4
2 — *phase*	**98.35, 98.30 (0.51)**	**0.033 (0.020)**	**1.36 (0.16)**	**3.5**	**0.4**

(figure 10), while the spike rate is kept at a lower value. Finally, we see that the lowest spike rate can only be found with the training that uses L. We remark that this is due to the switch off of some neurons that, in turn, precludes the achievement of good accuracy, as it is highlighted by the large standard deviations reported for L in figure 10.

Table 3 reports the accuracy, the cross-entropy, and the average number of spikes per neuron and per inference that are obtained in the best case for the three loss functions adopted and for the 2-phase strategy. We also reported the values of the hyper-parameters that lead to such best cases. The best results are obtained for L_W, L, and for the 2-phase strategy with a better performance in terms of spike rate for the 2-phase learning. However, it must be considered that for the 2-phase learning the results in the table are easily obtained (all the training phases have converged to a good minimum of the cross-entropy), while for L only a small fraction of the possible combinations of the hyper-parameters α and η allowed us not to incur in the switch off issue.

Finally, to study the effect of the network's depth on the performance, we report in figure 11 the results obtained during inference, after training three networks with different numbers of layers on the Yin-Yang dataset. In particular, the configuration of the three networks that we have studied is the following:

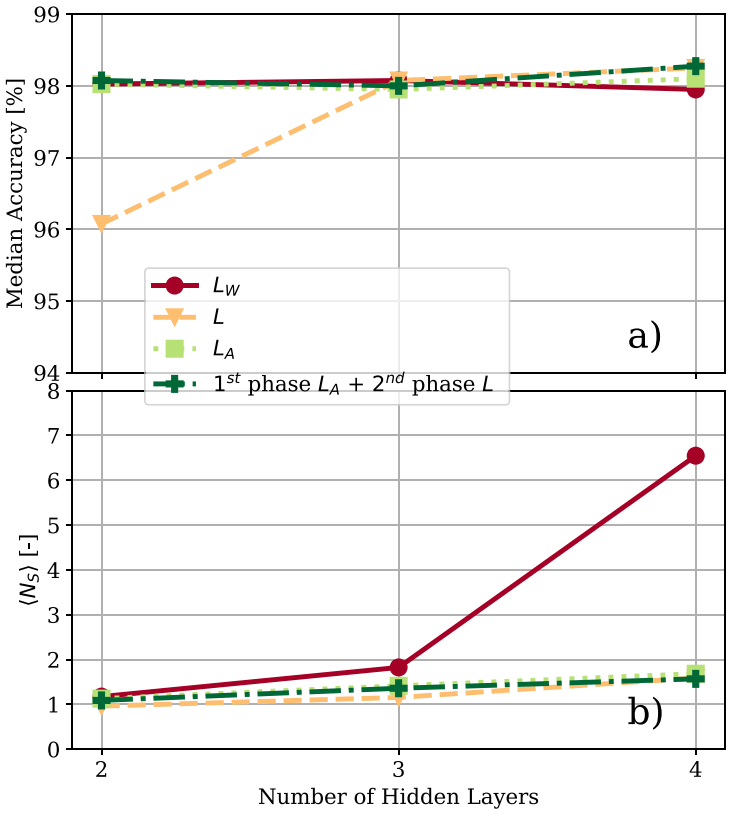

Figure 11. Yin-Yang—(a) Median accuracy and (b) average number of spikes per neuron ($\langle N_s \rangle$) emitted during inference as a function of the number of hidden layers in the network. The results have been obtained by setting the hyper-parameters $\alpha = 0.004$, and $\eta = 0.3$ ms.

- Input layer + 2 hidden layers + output layer, 5/40/20/3 neurons;
- Input layer + 3 hidden layers + output layer, 5/40/25/13/3 neurons (i.e. the default configuration of the network);
- Input layer + 4 hidden layers + output layer, 5/40/33/25/13/3 neurons.

We notice in figure 11 that, while the accuracy is only slightly affected by the network's depth, the average number of spikes per neuron emitted during inference, $\langle N_s \rangle$, increases with the number of layers, and such an increase is super-linear when we employ the loss function L_W, namely the loss function originally used in EventProp [19]. On the other hand, the two-phase learning strategy allows us to keep under control the value of $\langle N_s \rangle$, even for the deepest networks explored in figure 11.

As already stated in section 2, the increase of $\langle N_s \rangle$ with the number of layers is an expected trend, because the number of spikes received by a neuron increases with the number of upstream neurons (not only those in the previous layer and directly connected to the specific neuron), thus exciting the neuron to emit more spikes. A neuron emitting many spikes, in turn, excites the downstream neurons, so that the spike rate tends to increase with the number of layers unless specific measures are undertaken to counteract this behavior.

4.2. MNIST simulation results

For the MNIST benchmark a larger network, compared to the Yin-Yang benchmark, is needed to obtain a sufficient accuracy. In this respect, while the scope of this work is not a record-breaking performance on the MNIST dataset, we still believe that any accuracy below 97% is not meaningful enough to be considered. Therefore, we adopted a 5-layer network with a significantly larger number of hidden neurons, as already mentioned in section 3.4. Training such a network with the L_W loss function exacerbates the issues already discussed for the Yin-Yang benchmark. Since from the first epochs of learning, when a small value of α is used, the switch-off involves so many neurons that the whole last layer becomes inactive. In particular we observe that, when a small value of the hyper-parameter α is used to avoid the switch off of the network, the SNN tends to an over-excitation of neurons resulting in a large number of emitted spikes.

We also verified that, regardless of the loss function that is used, the simulations show a behavior similar to the one reported in figure 12, where data about training are shown over time. Such a plot reports the accuracy and the CE computed on the training set and on the test set for the first 60 epochs ($\alpha = 0.003$

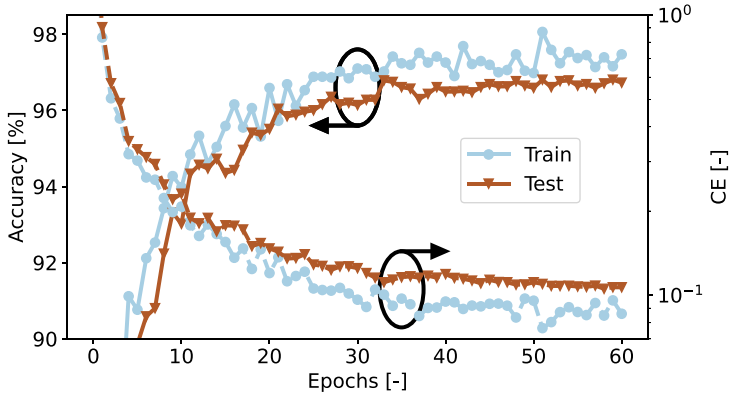

Figure 12. MNIST—General behavior of accuracy and cross-entropy during training. In particular, it can be observed the saturation of the two curves after 60 training epochs, without any anomaly on the test set, meaning that no overfitting has occurred.

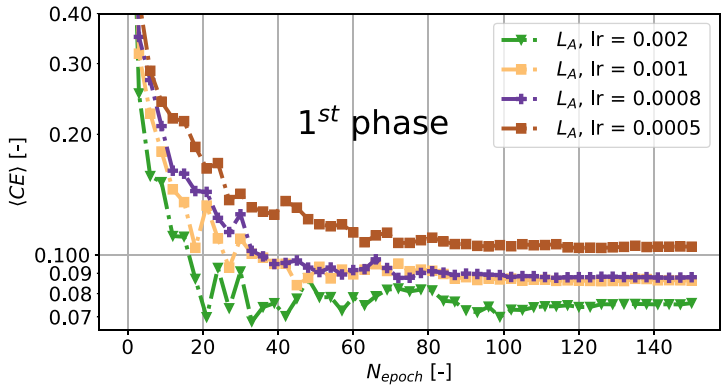

Figure 13. Preliminary analyses on MNIST: behavior of the cross-entropy during the initial 150 training epochs for several learning rates.

and $\eta = 3$ ms are used). Both accuracy and CE behave as expected, with a smooth saturation towards the minimum of the loss function.

As it can be seen, for the MNIST dataset we observe that: (i) the dimension of the network and the choice of the hyper-parameters that we made allowed us to avoid the switch off of the network during training; (ii) the results that we obtained with the loss function L and with the two-phase learning strategy are quite similar in terms of accuracy and $\langle N_S \rangle$. Therefore, in the following, we will always compare (unless otherwise mentioned) the results obtained by training the network with the loss function L_A (that cannot provide the true minimum of the cross-entropy), with those obtained with the two-phase learning strategy. In figure 13, the CE over time is shown for the first phase and for some learning rate up to epoch 150. Furthermore, figure 14 illustrates the CE behavior during the second phase, for the same learning rate. We can notice how the CE, which has saturated in the first phase, can be further reduced when L is adopted as the loss function. Moreover, the nice decrement of CE in that second phase is also an indication that the neuronal switch off has been avoided and that the training can improve during the second phase. This behavior is consistent for all the values of the learning rate we adopted. Therefore, to speed up the training in our further exploration, we choose to keep the same value ($lr = 2 \times 10^{-3}$) for both phases.

Figure 15 shows the accuracy and the cross-entropy reached by the network at the end of the training for some values of the hyper-parameters α and η. Such a plot also reports the results for a network trained using L_A only, showing how the 2-phase learning can always converge to good results in terms of accuracy and cross-entropy, that are consistently better than those obtained by using solely L_A. The standard deviations of the results shown in figure 15 are plotted in figure 16. We also recall that neither the L nor the L_W alone can be used to train this network because they do not reach proper convergence during the training.

Figure 17 shows the average number of spikes per neuron that are required to perform a single inference. In the plot on the right, the average is computed over all the nodes, while in the plot on the left, it is reported

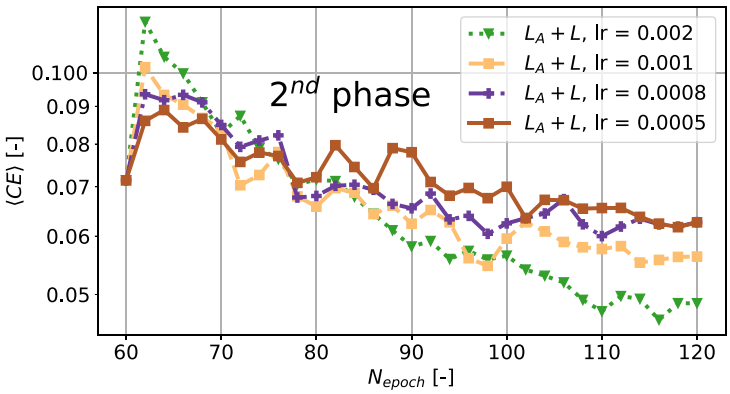

Figure 14. Preliminary analyses on MNIST: behavior of the cross-entropy in the second phase of training starting from the same initial condition (1st phase with L_A and $lr = 0.001$). The initial increment of the CE is due to the CE minimum discontinuity from L_A to L.

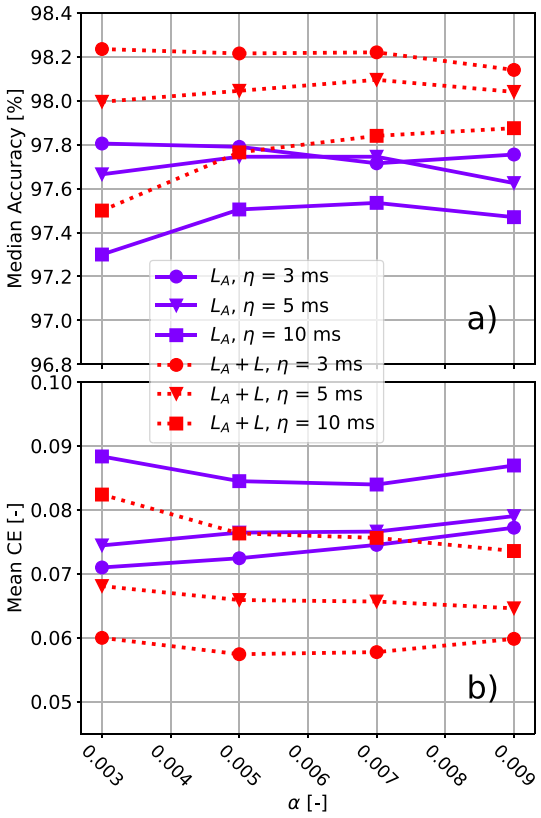

Figure 15. MNIST—Final training results of MNIST. (a) Median accuracy of the network and (b) mean cross-entropy for both the training phases and some α values.

separately for each layer (for a single value of α). In both cases, the 2-phase learning keeps the spike rate well below the rate that is reached using L_A only, showing its effectiveness on both metrics, namely accuracy and spike rate. We also notice that when the loss function L_W is used, the average number of spikes emitted by the network during inference is way larger than in those cases where a spike penalty term is taken into account.

Even for MNIST, we report, in table 4, the accuracy, the cross-entropy, and the average number of spikes per neuron and per inference that are obtained in the best case for the usable learning strategies. We notice that the use of the loss function L or of the 2-phase strategy allows us to achieve better results in terms of accuracy and CE, and at the same time keep the spike rate under control.

Finally, we also considered different topologies of the network, to ensure that the effectiveness of our learning strategy is not restricted to a single case. We trained three other networks, after choosing the α and η

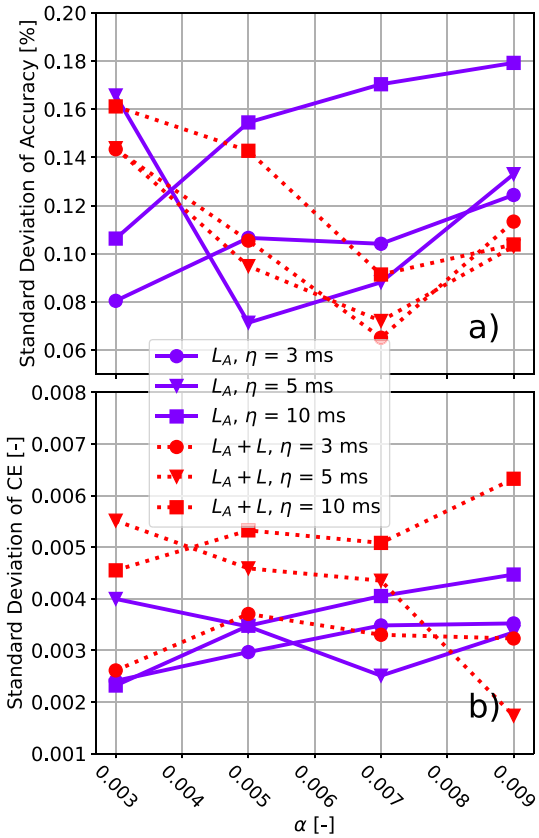

Figure 16. MNIST—Standard deviation of (a) the accuracy and of (b) the cross-entropy of the results reported in figure 15.

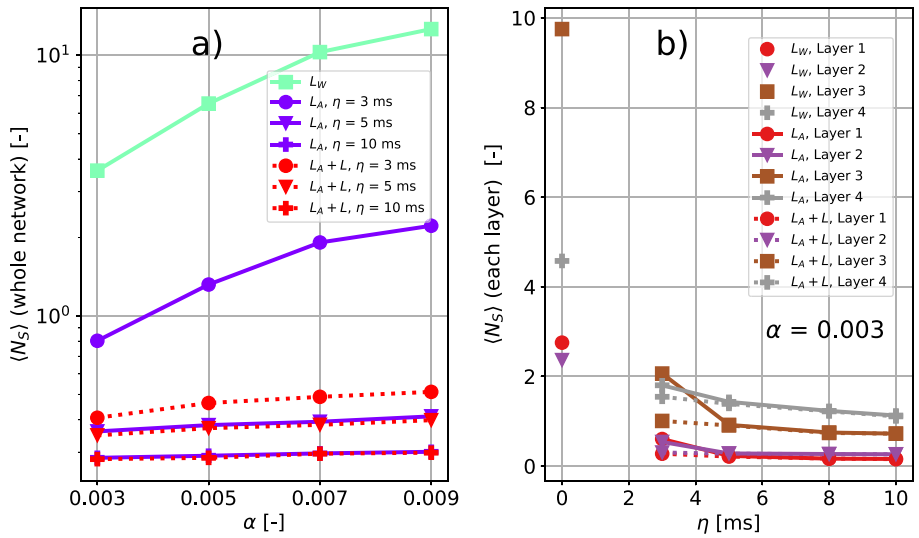

Figure 17. MNIST—Comparison between the results obtained by using L_W, L_A, and the two-phase learning strategy for (a) the overall average number of spikes produced by the network during inference and (b) the single layer components for the $\alpha = 0.003$ case (note that Layer 0 is the input layer). (a) Shows an average number of spikes produced by the whole network much below the threshold value of 1 spike per neuron due to the strong dependence on the average behavior of the first (280 neurons) and the second layers (160 neurons). The network switch off is prevented by the excitation of the last layer (layer 3, plot (b)) which does not affect too much the results of (a).

that provided the best results, and compared the accuracy of the 2-phase learning against the one obtained using L_A only (Figure 18). It turned out that the proposed strategy is consistently effective in reaching a good accuracy for all the cases that we have explored.

Table 4. Best results in terms of median accuracy for MNIST.

Training	*Accuracy: median, mean (STD) (%)*	*Cross-Entropy (STD)*	$\langle N_S \rangle$ *(STD)*	$\alpha \times 10^{-3}$	η[ms]
L_W [19]	−,97.60 (1.00)	—	—	0.0	—
L_W	98.17, 98.17 (0.11)	0.059 (0.003)	6.53 (1.14)	5.0	—
L	97.91, 97.88 (0.12)	0.067 (0.003)	0.60 (0.03)	7.0	3.0
L_A	97.81, 97.82 (0.08)	0.071 (0.002)	0.58 (0.02)	3.0	3.0
2 — *phase*	**98.24, 98.19 (0.14)**	**0.060 (0.003)**	**0.41 (0.05)**	**3.0**	**3.0**

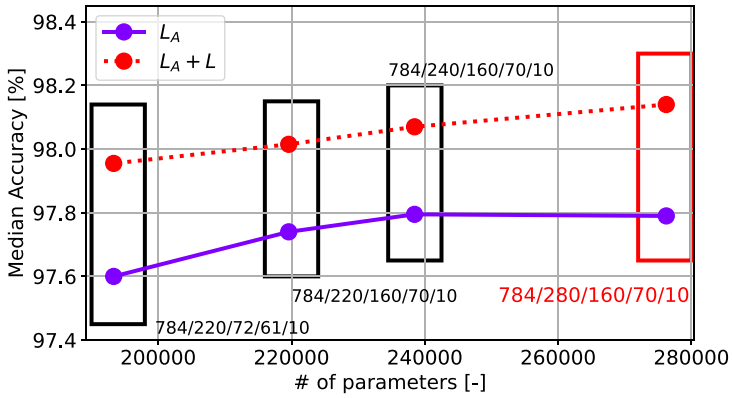

Figure 18. MNIST—Final accuracy after the first and the second phase of training for multiple topologies of the network and, hence, for different numbers parameters. The average accuracy results obtained for the reference network are highlighted in red. In this figure, the hyper-parameters are fixed for all networks, and in particular $\alpha = 0.003$ and $\eta = 3$ ms.

5. Conclusions

In this paper, we proposed a learning strategy that leverages the exact computation of the gradient and, at the same time, keeps the overall spike rate under control. This was accomplished thanks to a change of the original loss function in [19], to which we added a new term introducing a penalty for the spikes after the first one of the neurons in the output layer. We found that a gradient descent strategy alone is not always sufficient to obtain a small spike rate and good accuracy, because a challenging trade-off between the spike rate and the minimization of the loss function may emerge during the training, and this is especially true for smaller networks. Acting on the spike rate only, in fact, can cause many neurons to go silent, hence resulting in a gradient vanishing problem. On the other hand, focusing only on the network accuracy almost invariably results in a large spike rate. The neural over-excitation issue gains more relevance as the network becomes deeper or the number of neurons becomes larger. To achieve successful training even in those cases where the aforementioned trade-off is a serious threat to convergence, we proposed a learning strategy composed of two phases. The first phase uses an 'augmented' loss function (L_A) that is built to avoid silent neurons. While the minimization of L_A does not converge to the minimum of the real metric of interest (the cross-entropy), it tends to converge to a point that is close to such a minimum and it is thus an effective starting point for the second training phase. Here, a 'correct' loss (L) is used that still pursues also a minimization of the spike rate, thus leading to the convergence of the network towards the minimum of the cross-entropy, while maintaining at the same time a small average spike rate.

In an alternative perspective, the first phase can also be considered as an initialization strategy for the *true* minimization (i.e. the second phase). In fact, the local minimum reached with L_A seems to be an effective starting point to subsequently minimize the cross-entropy by using the loss L, that at the same time also keeps the spike rate under control.

A final consideration is related to the additional terms used in the losses L_A and L with respect to the *vanilla* loss L_W. Since such terms only aim at reducing the global spike rate (SP) while keeping neurons active (AS), it is reasonable to surmise that they can be used even with a different main loss which, for example, is not focused on cross-entropy. However, this hypothesis is only driven by intuition. Therefore, the effectiveness of the augmentation terms or instead the need for adjustments or major changes in the learning strategy are interesting questions that remain to be addressed in future investigations.

Data availability statement

No new data were created or analysed in this study.

Acknowledgments

This work was supported by the Departmental Strategic Plan (PSD) of the University of Udine - Interdepartmental project of Artificial Intelligence AI.

ORCID iDs

Riccardo Fontanini https://orcid.org/0000-0002-4246-946X
Alessandro Pilotto https://orcid.org/0000-0002-2860-3714
David Esseni https://orcid.org/0000-0002-3468-5197
Mirko Loghi https://orcid.org/0000-0001-7876-3612

References

[1] Sutskever I, Vinyals O and Le Q V 2014 Sequence to sequence learning with neural networks *Proc. 27th Int. Conf. on Neural Information Processing Systems -Volume 2, NIPS'14* (MIT Press) pp 3104–12
[2] Hinton G *et al* 2012 Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups *IEEE Signal Process. Mag.* **29** 82–97
[3] Krizhevsky A, Sutskever I and Hinton G E 2012 Imagenet classification with deep convolutional neural networks *Advances in Neural Information Processing Systems* vol 25, ed F Pereira, C J Burges, L Bottou and K Q Weinberger (Curran Associates, Inc)
[4] Liu R, Nageotte F, Zanne P, de Mathelin M and Dresp-Langley B 2021 Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review *CoRR* (arXiv:2102.04148)
[5] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (MIT Press) (available at: http://www.deeplearningbook.org)
[6] Maass W 1997 Networks of spiking neurons: the third generation of neural network models *Neural Netw.* **10** 1659–71
[7] Mead C 1990 Neuromorphic electronic systems *Proc. IEEE* **78** 1629–36
[8] Covi E, Donati E, Liang X, Kappel D, Heidari H, Payvand M and Wang W 2021 Adaptive extreme edge computing for wearable devices *Front. Neurosci.* **15**
[9] Joung Y-H 2013 Development of implantable medical devices: from an engineering perspective *Int. Neurourol. J.* **17** 98–106
[10] Wu Y, Deng L, Li G, Zhu J and Shi L 2018 Spatio-temporal backpropagation for training high-performance spiking neural networks *Front. Neurosci.* **12**
[11] Singh S, Sarma A, Lu S, Sengupta A, Kandemir M T, Neftci E, Narayanan V and Das C R 2022 Skipper: enabling efficient snn training through activation-checkpointing and time-skipping *2022 55th IEEE/ACM Int. Symp. on Microarchitecture (MICRO)* pp 565–81
[12] Guo W, Fouda M E, Eltawil A M and Nabil Salama K 2023 Efficient training of spiking neural networks with temporally-truncated local backpropagation through time *Front. Neurosci.* **17**
[13] Rueckauer B, Lungu I-A, Hu Y, Pfeiffer M and Liu S-C 2017 Conversion of continuous-valued deep networks to efficient event-driven networks for image classification *Front. Neurosci.* **11**
[14] 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
[15] Bohté S M, Kok J N and La Poutré H 2000 Spikeprop: backpropagation for networks of spiking neurons *ESANN* 419–24
[16] Booij O and tat Nguyen H 2005 A gradient descent rule for spiking neurons emitting multiple spikes *Inf. Process. Lett.* **95** 552–8
[17] Maass W 1996 Lower bounds for the computational power of networks of spiking neurons *Neural Comput.* **8** 1–40
[18] Hodgkin A L and Huxley A F 1952 A quantitative description of membrane current and its application to conduction and excitation in nerve *J. Physiol.* **117** 500–44
[19] Wunderlich T C and Pehle C 2021 Event-based backpropagation can compute exact gradients for spiking neural networks *Sci. Rep.* **11** 12829
[20] Fontanini R, Esseni D and Loghi M 2022 Reducing the spike rate in deep spiking neural networks *Proc. Int. Conf. on Neuromorphic Systems 2022, ICONS '22*, (*New York, NY, USA*) (Association for Computing Machinery)
[21] Xu Y, Zeng X, Han L and Yang J 2013 A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks *Neural Netw.* **43** 99–113
[22] Selvaratnam K, Kuroe Y and Mori T 2000 Learning methods of recurrent spiking neural networks *Trans. Inst. Syst. Control Inf. Eng.* **13** 95–104
[23] Kuroe Y and Ueyama T 2010 Learning methods of recurrent spiking neural networks based on adjoint equations approach *The 2010 Int. Joint Conf. on Neural Networks (IJCNN)* pp 1–8
[24] Huh D and Sejnowski T J 2018 Gradient descent for spiking neural networks *Advances in Neural Information Processing Systems* vol 31, ed S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi and R Garnett (Curran Associates, Inc)
[25] Gerstner W and Kistler W M 2002 *Spiking Neuron Models: Single Neurons, Populations, Plasticity* (Cambridge University Press)
[26] Indiveri G *et al* 2011 Neuromorphic silicon neuron circuits *Front. Neurosci.* **5**
[27] Kullback S and Leibler R A 1951 On information and sufficiency *Ann. Math. Stat.* **22** 79–86
[28] Paszke A *et al* 2019 Pytorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32, ed H Wallach, H Larochelle, A Beygelzimer, F d' Alché-Buc, E Fox and R Garnett (Curran Associates, Inc) pp 8024–35

[29] Kingma D P and Ba J 2015 Adam: a method for stochastic optimization *3rd Int. Conf. on Learning Representations, ICLR 2015* (*San Diego, CA, USA, May7-9 2015 Conf. Track Proc.*) ed Y Bengio and Y LeCun
[30] Li L, Pratap A, Lin H-T and Abu-Mostafa Y S 2005 Improving generalization by data categorization *Knowledge Discovery in Databases: Pkdd 2005* ed A M'ario Jorge, L Torgo, P Brazdil, R Camacho and J ao Gama (Springer) pp 157–68
[31] Kriener L, Göltz J and Petrovici M A 2021 The yin-yang dataset *CoRR* (arXiv:2102.08211)
[32] Lecun Y, Bottou L, Bengio Y and Haffner P 1998 Gradient-based learning applied to document recognition *Proc. IEEE* **86** 278–324