International Journal of Neural Systems, Vol. 30, No. 6 (2020) 2050027 (16 pages)

DOI: 10.1142/S0129065720500276

World Scientific
www.worldscientific.com

Temporal Backpropagation for Spiking Neural Networks with One Spike per Neuron

Saeed Reza Kheradpisheh*
Department of Computer and Data Sciences, Faculty of Mathematical Sciences
Shahid Beheshti University, Tehran, Iran
s_kheradpisheh@sbu.ac.ir

Timothée Masquelier
CERCO UMR 5549, CNRS – Université Toulouse 3, Toulouse, France
timothee.masquelier@cnrs.fr

Accepted 8 March 2020
Published Online 28 May 2020

We propose a new supervised learning rule for multilayer spiking neural networks (SNNs) that use a form of temporal coding known as rank-order-coding. With this coding scheme, all neurons fire exactly one spike per stimulus, but the firing order carries information. In particular, in the readout layer, the first neuron to fire determines the class of the stimulus. We derive a new learning rule for this sort of network, named S4NN, akin to traditional error backpropagation, yet based on latencies. We show how approximated error gradients can be computed backward in a feedforward network with any number of layers. This approach reaches state-of-the-art performance with supervised multi-fully connected layer SNNs: test accuracy of 97.4% for the MNIST dataset, and 99.2% for the Caltech Face/Motorbike dataset. Yet, the neuron model that we use, nonleaky integrate-and-fire, is much simpler than the one used in all previous works. The source codes of the proposed S4NN are publicly available at https://github.com/SRKH/S4NN.

Keywords: Spiking neural network; supervised learning; temporal backpropagation; single spike coding.

1. Introduction

Biological neurons communicate via short stereotyped electrical impulses called "spikes", or "action potentials". Each neuron integrates incoming spikes from the presynaptic neurons and whenever its membrane potential reaches a certain threshold, it also sends an outgoing spike to the downstream neurons. In the brain, the spike times, in addition to the spike rates, are known to play an important role in how neurons process information.[1,2] Spiking neural networks (SNNs) are thus more biologically realistic than the artificial neural networks (ANNs),[3–6] and as SNNs use sparse and asynchronous binary signals processed in a massively parallel fashion, they are one of the best available options to study how the brain computes at the neuronal description level. But SNNs are also appealing for artificial intelligence technology, especially for edge computing, since their implementations on the so-called neuromorphic chips can be far less energy–hungry than ANN implementations (typically done on GPUs or similar hardware), mostly because they can leverage efficient event-based computations.[4,7–12]

Recently, an extensive effort has been made by numerous researchers to develop direct supervised learning algorithms for SNNs.[7] The main challenge for this is the nondifferentiability of the thresholding activation function of spiking neurons at firing times. One solution to this problem is to consider spike rates instead of exact firing times.[13–15] The second approach is to use smoothed spike functions that are differentiable with respect to time.[16] The third

*Corresponding author.

set of methods use surrogate gradients at the firing times.[8,17–22] The last approach, known as latency learning, is the main focus of this paper. In this approach, the firing time of the neuron is defined as a function of its membrane potential or the firing time of presynaptic neurons.[23–25] In this way, the derivation of the thresholding activation function is no longer required.

More specifically, our goal is to classify static inputs (e.g. images), with an SNN in which neurons fire once at most, but the most strongly activated neurons fire first.[24–36] Thus, the spike latencies, or firing order, carry information. Here, we used simple nonleaky integrate-and-fire neurons[37] in all the layers of the proposed SNN. Indeed, each neuron simply integrates weighted input spikes (received from instantaneous synapses) through time with no leak and emits only one spike right after crossing its threshold for the first time, or zero spike if this threshold is never reached. In the readout layer, there is one neuron per category. As soon as one of these neurons fires, the network assigns the corresponding category to the input, and the computations can stop when only a few neurons have fired. This coding scheme is thus extremely economical in the number of spikes.

In this work, we adapted the well-known backpropagation algorithm,[38] originally designed for ANNs, to this sort of SNNs. Backpropagation has been shown to solve extremely difficult classification problems in ANNs with many layers, leading to the so-called "deep learning" revolution.[39] The *tour de force* of backpropagation is to solve the multilayer credit assignment problem.[40] That is, it finds what the hidden layers should do to minimize the loss in the readout layer. This motivated us, and others,[23–25,34] to adapt backpropagation to single-spike SNNs, by using the latencies instead of the firing rates. The main strength of our approach with respect to the above-mentioned ones is the use of a much simpler neuron model: A nonleaky integrate-and-fire neuron with instantaneous synapses. Yet, it reaches a comparable accuracy on the MNIST dataset.[41]

2. Methods

The proposed *single-spike supervised spiking neural network* (S4NN) is comprised of an input layer converting input data into a spike train and feeding it into the network, followed by one or more hidden layers of nonleaky integrate-and-fire (IF) neurons processing the input spikes, and finally, an output layer of nonleaky IF neurons with one neuron per category. Figure 1 demonstrates an S4NN with two hidden layers. Here, we use a temporal (i.e. rank-order) coding called time-to-first-spike in the input layer which is very sparse and produces at most one spike for each

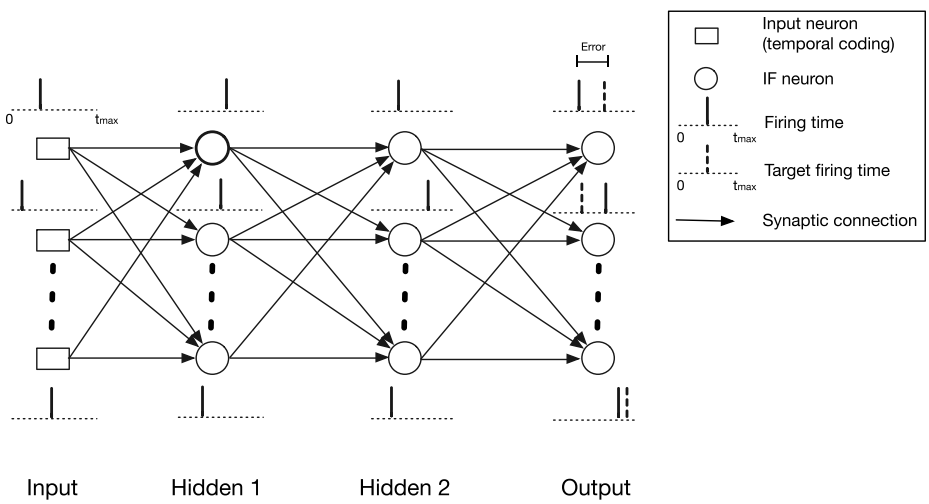


Fig. 1. An S4NN with two hidden layers. The input layer converts the input data into a spike train (using the temporal time-to-first-spike coding) and sends it to the next layer. Spikes are propagated through the network and reach the output layer. The output layer computes the errors with respect to the target firing times, and then, synaptic weights are updated using the temporal error backpropagation.

input value. The subsequent neurons are also limited to fire exactly once.

To train the network, a temporal version of the backpropagation algorithm is used. We assume an image categorization task with several images per category. First, the network decision on the category of the input image is made by considering the first output neuron to fire. Then, the error of each output neuron is computed by comparing its actual firing time with a target firing time (see Sec. 2.5). Finally, these errors are backpropagated through the layers and weights get updated through stochastic gradient descent. Meanwhile, the temporal backpropagation confronts two challenges: defining the target firing time and computing the derivative of the neuron firing time with respect to its membrane potential. To overcome these challenges, the proposed learning algorithm uses relative target firing times and approximated derivations.

2.1. *Time-to-first-spike coding*

The first step of an SNN is to convert the analog input signal into a spike train representing the same information. The neural processing in the following neurons should be compatible with this coding scheme to be able to decipher the information encoded in the input spikes. Here, we use a time-to-first-spike coding for the entry layer (in which a larger input value corresponds to an earlier spike) and IF neurons in subsequent layers that fire once.

Consider a gray image with the pixel intensity values in range $[0, I_{\max}]$, each input neuron encodes its corresponding pixel value in a single spike time in range $[0, t_{\max}]$. The firing time of the ith input neuron, t_i, is computed based on the ith pixel intensity value, I_i, as follows:

$$t_i = \left\lfloor \frac{I_{\max} - I_i}{I_{\max}} t_{\max} \right\rfloor. \tag{1}$$

Therefore, the spike train of the ith neuron in the input layer (layer zero) is defined as

$$S_i^0(t) = \begin{cases} 1, & \text{if } t = t_i, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Notably, this simple intensity-to-latency code does not need any preprocessing steps like applying Gabor or DoG filters that are commonly used in SNNs, especially, in those with STDP learning rule which can not handle homogeneous surfaces.[30,31,42] Also, it produces only one spike per pixel and hence the obtained spike train is way sparser than what is common in rate codes.

Neurons at the subsequent layers fire as soon as they reach their threshold, and the first neuron to fire in the output layer determines the network decision. Hence, the network decision depends on the earliest spikes throughout the network. In other words, neural information in all the layers is encoded in the spike times of the earliest neurons to fire. Therefore, one can say that the time-to-first-spike information coding is at work in subsequent layers as well.

2.2. *Forward path*

S4NN consists of multiple layers of nonleaky IF neurons and there is no limitation on the number of the layers, hence, one can implement S4NN with any arbitrary number of hidden layers. The membrane potential of the jth neuron in the lth layer at time point t, $V_j^l(t)$, is computed as

$$V_j^l(t) = \sum_i w_{ji}^l \sum_{\tau=1}^{t} S_i^{l-1}(\tau), \tag{3}$$

where S_i^{l-1} and w_{ji}^l are, respectively, the input spike train and the input synaptic weight from the ith presynaptic neuron in the previous layer to neuron j. The IF neuron emits a spike the first time its membrane potential reaches the threshold, θ_j^l,

$$S_j^l(t) = \begin{cases} 1 & \text{if } V_j^l(t) \geq \theta_j^l \;\&\; S_j^l(<t) \neq 1, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $S_j^l(<t) \neq 1$ checks if the neuron has not fired at any previous time step.

As explained in the previous section, the input image is transformed into a spike train, $S^0(t)$, in which each input neuron will emit a spike with a delay, in the range $[0, t_{\max}]$, negatively proportional to the corresponding pixel value. These spikes are propagated toward the first layer of the network, where each neuron receives incoming spikes and updates its membrane potential until it reaches its threshold and sends a spike to the neurons in the next layer. For each input image, the simulation starts by resetting all the membrane voltages to zero and continues for $t_{\max}$ time steps. Note that during

a simulation, each neuron at any layer is allowed to fire once at most. In the training phase, we need to know the firing time of all neurons (see Eqs. (15) and (9)), hence if a neuron was silent, we assume that it fires a fake spike at the last time step, $t_{\max}$. During the test phase, neurons can be silent or fire once at most. Finally, regarding the time-to-first-spike coding deployed in our network, the output neuron which fires earlier than others determines the category of the input stimuli.

2.3. *IF approximating ReLU*

In traditional ANNs with Rectified Linear Units (ReLU)[43] activation function, the output of a neuron in layer l with index j is computed as

$$y_j^l = \max\left(0, z_j^l = \sum_i w_{ji}^l x_i^{l-1}\right), \tag{5}$$

where x_i^{l-1} ($x_i^{l-1} > 0$) and w_{ji}^l are the ith input and connection weight, respectively. Thus, the ReLU neuron with a larger z_j^l has a larger output value, y_j^l. Generally, the main portion of this integration value is due to the large inputs with large connection weights. In our time-to-first-spike coding, larger values correspond to earlier spikes, and hence, if an IF neuron receives these early spikes through strong synaptic weights, it will also fire earlier. Note, as the network decision is based on the first spike in the output layer, thus earlier spikes carry more information. In this way, the time-to-first-spike coding is preserved in the hidden and output layers. Therefore, for the same inputs and synaptic weights, we can assume an equivalence relation between the output of the ReLU neuron, y_j^l, and the firing time of the corresponding IF neuron, t_j^l,

$$y_j^l \sim t_{\max} - t_j^l, \tag{6}$$

and we know that

$$\frac{\partial y_j^l}{\partial w_{ji}^l} = \frac{\partial y_j^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{ji}^l} = \begin{cases} x_i^{l-1}, & \text{if } y_j^l > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

where $\partial y_j^l / \partial z_j^l = 1$ if $y_j^l > 0$.

Regarding the fact that in the IF neuron, t_j^l is not a function of w_{ji}^l, we cannot compute $\partial t_j^l / \partial w_{ji}^l$. Therefore, according to Eq. (6), we assume that $\partial t_j^l / \partial V_j^l = -1$ if $t_j^l < t_{\max}$ (see Eq. (7)). Note that according to Eq. (3), we have $\partial V_j^l / \partial w_{ji}^l = \sum_{\tau=1}^{t_j^l} S_i^{l-1}(\tau)$. Thus, we have

$$\frac{\partial t_j^l}{\partial w_{ji}^l} = \frac{\partial t_j^l}{\partial V_j^l}\frac{\partial V_j^l}{\partial w_{ji}^l} = \begin{cases} -\sum_{\tau=1}^{t_j^l} S_i^{l-1}(\tau), & \text{if } t_j^l < t_{\max}, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where $\sum_{\tau=1}^{t_j^l} S_i^{l-1}(\tau) = 1$ if $t_i^{l-1} \le t_j^l$.

2.4. *Backward path*

We assume that in a categorization task with C categories, each output neuron is assigned to a different category. After completing the forward path over the input pattern, each output neuron may fire at a different time point. As mentioned before, the category of an input image is predicted as the category assigned to the winner output neuron (the output neuron which has fired earlier than others).

Hence, to be able to train the network, we define a temporal error function as

$$e = [e_1, \ldots, e_C] \quad \text{s.t.} \quad e_j = (T_j^o - t_j^o)/t_{\max}, \tag{9}$$

where T_j^o and t_j^o are the target and actual firing times of the jth output neuron, respectively. The target firing times should be defined in a way that the correct neuron fires earlier than others. We use a relative target firing calculation that is fully explained in Sec. 2.5. Here, we assume that T_j^o is known.

During the learning phase, we use the stochastic gradient descent[38] (SGD)and backpropagation algorithms to minimize the "squared error" loss function. For each training sample, the loss is defined as

$$L = \frac{1}{2}\|e\|^2 = \frac{1}{2}\sum_{j=1}^{C} e_j^2, \tag{10}$$

and, hence, we need to compute its gradient with respect to each synaptic weight. To update w_{ji}^l, the synaptic weight between the ith neuron of layer $l-1$ and the jth neuron of layer l, we have

$$w_{ji}^l = w_{ji}^l - \eta\frac{\partial L}{\partial w_{ji}^l}, \tag{11}$$

where η is the learning rate parameter.

Let's define

$$\delta_j^l = \frac{\partial L}{\partial t_j^l}, \tag{12}$$

therefore, by considering Eqs. (8) and (12), we have

$$\begin{aligned}\frac{\partial L}{\partial w_{ji}^l} &= \frac{\partial L}{\partial t_j^l}\frac{\partial t_j^l}{\partial w_{ji}^l} \\ &= \begin{cases} -\delta_j^l \sum_{\tau=1}^{t_j^l} S_i^{l-1}(\tau) & \text{if } t_j^l < t_{\max} \\ 0 & \text{otherwise,} \end{cases}\end{aligned} \tag{13}$$

where for the output layer (i.e. $l = o$) we have

$$\delta_j^o = \frac{\partial L}{\partial e_j}\frac{\partial e_j}{\partial t_j^o} = -e_j, \tag{14}$$

and for the hidden layers (i.e. $l \neq o$), according to the backpropagation algorithm, we have

$$\begin{aligned}\delta_j^l &= \sum_k \frac{\partial L}{\partial t_k^{l+1}}\frac{\partial t_k^{l+1}}{\partial V_k^{l+1}}\frac{\partial V_k^{l+1}}{\partial t_j^l} \\ &= \sum_k \delta_k^{l+1} w_{kj}^{l+1}[t_j^l \leq t_k^{l+1}],\end{aligned} \tag{15}$$

where k iterates over neurons in layer $l+1$. Note that regarding Eq. (12) we have $\partial L/\partial t_k^{l+1} = \delta_k^{l+1}$, and as explained in Sec. 2.3 we approximate $\partial t_k^{l+1}/\partial V_k^{l+1} = -1$. To compute $\partial V_k^{l+1}/\partial t_j^l$ we should note that reducing t_j^l will increase V_k^{l+1} by w_{kj}^{l+1} earlier in time, hence we approximate $\partial V_k^{l+1}/\partial t_j^l = -w_{kj}^{l+1}$ if and only if $[t_j^l \leq t_k^{l+1}]$.

To avoid the exploding and vanishing gradient problems during backpropagation, we use normalized gradients. Literally, at any layer l, we normalize the backpropagated gradients before updating the weights

$$\delta_j^l \leftarrow \frac{\delta_j^l}{\sum_i \delta_i^l}. \tag{16}$$

To avoid over-fitting, we added an L_2-norm regularization term $\lambda \sum_l \sum_{i,j} (w_{ji}^l)^2$ (over all the synaptic weights in all the layers) to the "squared error" loss function in Eq. (10). The parameter λ is the regularization parameter accounting for the degree of weigh penalization.

2.5. *Relative target firing time*

As the proposed network works in the temporal domain, for each input image, we need to define the target firing time of the output neurons regarding its category label.

One possible scenario is to define a fixed and predefined vector of target firing times for each category, in a way that the correct neuron has a shorter target firing time than others. For instance, if the input image belongs to the ith category, then, one can define $T_i^o = \tau$ and $T_j^o = t_{\max}$ for $j \neq i$, where $0 < \tau < t_{\max}$ is the desired firing time for the winner neuron. In this way, the correct output neuron is encouraged to fire early at time τ, while others are forced to block firing until the end of the simulation.

Such strict approaches have several drawbacks. For instance, let's assume an input image belonging to the ith category with $t_i^o < \tau$, in this way, the correct neuron has a negative error (see Eq. (9)). The backward path will update the weights to make this neuron fire later which means the network should forget what has helped the correct neuron to fire quickly. It is not desirable as we want the network to respond as quickly as possible.

The other scenario is to use a dynamic method to determine the target firing times for each input image, independently. Here, we propose a relative method that takes the actual firing times into account. Let's assume an input image of the ith category is fed to the network and the firing time of the output neurons are obtained. First, we compute the minimum output firing time as $\tau = \min\{t_j^o | 1 < j < C\}$ and then we set the target firing time of the jth output neuron as

$$T_j^o = \begin{cases} \tau & \text{if } j = i, \\ \tau + \gamma & \text{if } j \neq i \ \&\ t_j^o < \tau + \gamma, \\ t_j^o & \text{if } j \neq i \ \&\ t_j^o \geq \tau + \gamma, \end{cases} \tag{17}$$

where, γ is a positive constant term penalizing output neurons with firing times close to τ. Other neurons which have fired quite after τ are not penalized and the correct output neuron is encouraged to fire earlier than others at the minimum firing time, τ.

In a special case where all output neurons are silent during the simulation and their firing time is manually set to $t_{\max}$, we compute the target firing times as

$$T_j^o = \begin{cases} t_{\max} - \gamma & \text{if } j = i, \\ t_{\max} & \text{if } j \neq i, \end{cases} \tag{18}$$

to encourage the correct output neuron to fire during the simulation.

Table 1. The structural, initialization, and model parameters used for the Caltech face/motorbike and MNIST datasets.

	Layer size			Initial weights		Model parameters				
Dataset	Input	Hidden	Output	Hidden	Output	$t_{\max}$	θ	η	γ	λ
Caltech face/motorbike	160×250	4	2	$[0, 1]$	$[0, 50]$	256	100	0.1	3	10^{-6}
MNIST	28×28	400	10	$[0, 5]$	$[0, 50]$	256	100	0.2	3	10^{-6}

2.6. *Learning procedure*

As mentioned before, the proposed network employs a temporal version of SGD and backpropagation to train the network. During a training epoch, images are converted into input spike trains by the time-to-first-spike coding (see Sec. 2.1) and fed to the network one by one. Through the forward path, each IF neuron at any layer receives incoming spikes and emits a spike when it reaches its threshold (see Sec. 2.2). Then, after computing the relative target output firing times (encouraging correct output neuron to fire earlier, see Sec. 2.5), we update the synaptic weights in all the layers using temporal error backpropagation (see Sec. 2.4). Note that we force neurons to fire a fake spike at the last time step if they could not reach the threshold during the simulation (it is necessary for the learning rule). After completing the forward and backward processes on the current input image, the membrane potentials of all the IF neurons are reset to zero and the network gets ready to process the next input image. Notably, each neuron is allowed to fire only once during the processing of each input image.

As stated before, except for the fake spikes, IF neurons fire if and only if they reach their threshold. Let us consider an IF neuron that has decreased its weights (during the weight update process) in a way that it can not reach its threshold for any of the training images. Now, it is a dead neuron and only emits fake spikes. Hence, if a neuron dies, and does not fire real spikes during a training epoch, we reuse it by resetting its synaptic weights to a new set of random values drawn from a uniform distribution in the same range as the initial weights. Although it happens rarely, it helps the network to use all its learning capacity.

3. Results

We first use the Caltech 101 face/motorbike dataset to better demonstrate the learning process in S4NN and its capacity to work on large-scale and natural images. Afterward, we evaluate S4NN on the MNIST dataset which is one of the widely used benchmarks in the area of spiking neural networks (SNNs)[7] to demonstrate its capability to handle large and multi-class problems. The parameter settings of the S4NN models used for the Caltech face/motorbike and MNIST datasets are provided in Table 1.

3.1. *Caltech face/motorbike dataset*

We evaluated S4NN on the Caltech 101 face/motorbike dataset available at http://www.vision.caltech.edu. Some sample images are provided in Fig. 2. We trained the network on 200 randomly selected images per category. Also, we selected 50 random images from each category as the validation set. The remaining images were used in the test phase. We grayscaled all images and rescaled them to be of size 160×250 pixels.

In the first experiment, we use a fully connected architecture with a hidden layer of four IF neurons. The input layer has the same size as the input images (i.e. 160×250) and the firing time of each input neuron is calculated by the time-to-first-spike coding explained in Sec. 2.1. The output layer is comprised of two output IF neurons (the face and the motorbike neurons) corresponding to the image categories. We set the maximum simulation time as $t_{\max} = 256$ and initialize the input-hidden and hidden-output synaptic weights with random values drawn from uniform distributions in range $[0, 1]$ and $[0, 50]$, respectively. We also set the learning rate as $\eta = 0.1$, the penalty term in the target firing time calculation as $\gamma = 3$, and the regularization parameter as $\lambda = 10^{-6}$. The threshold of all neurons in all layers, θ_i^l, is set to 100.

Figure 3 shows the trajectory of the mean sum-of-squared-error (MSSE) for the training and validation samples through the training epochs. The sudden jumps in the early part of the MSSE curves are mainly due to the enormous weight changes in the

Fig. 2. Some sample images from Caltech face/motorbike dataset.

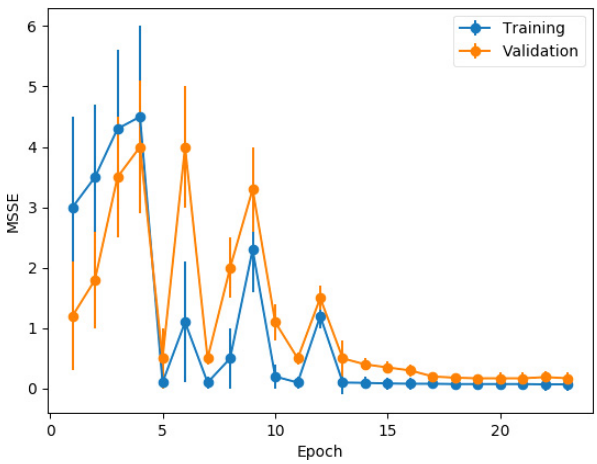


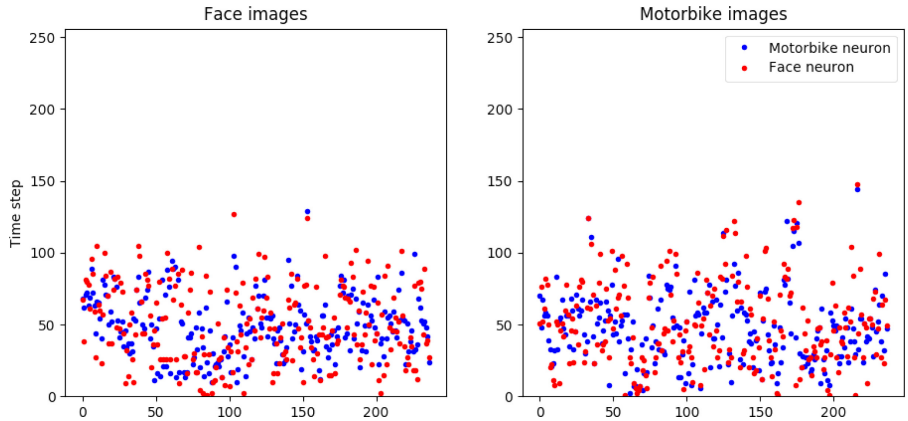
Fig. 3. The mean and the standard deviation of the sum-of-squared-error of the proposed S4NN over the training and validation samples through the training epochs. MSSE fluctuates at the beginning of the learning but gets stable after 15 epochs and remains below 0.1.

first training epochs that may keep any of the output neurons silent (emitting fake spikes only) for a while, however, it is being resolved during the next epoch. Finally, after some epochs, the network overcomes this challenge and decreases the MSSE below 0.1.

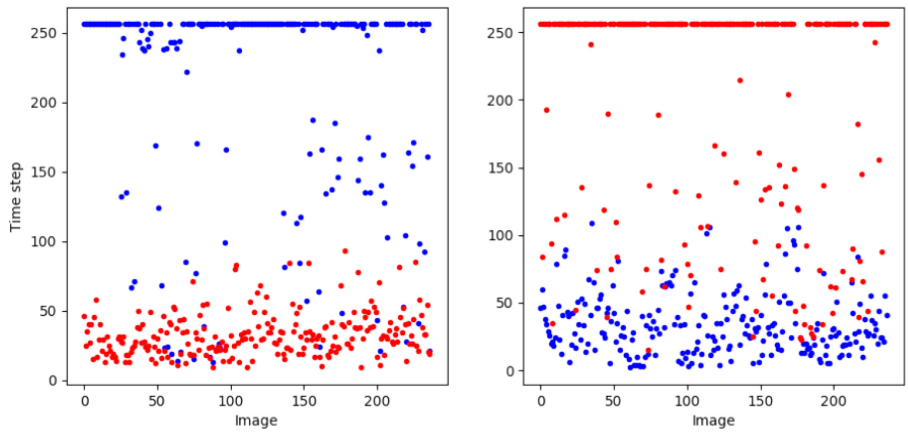
The proposed S4NN could reach $99.75\% \pm 0.1\%$ recognition accuracy (i.e. the percentage of correctly classified samples) on training samples and $99.2\% \pm 0.2\%$ recognition accuracy on testing samples which outperforms previously reported SNN results on this dataset (see Table 2). In Masquelier *et al.*,[28] a two-layer convolutional SNN trained by unsupervised STDP followed by a supervised potential-based radial basis functions (RBFs) classifier reached 99.2% accuracy on this dataset. This network uses four Gabor filters and four scales in the first layer and extracts ten different filters for the second layer. Also, it does not make decisions by the spike times, rather it uses neurons' membrane potential to do the classification. In Kheradpisheh *et al.* (2018),[30] an STDP-based SNN with three convolutional layers (respectively consisting of 4, 20, and 10 filters) and a SVM classifier could reach to 99.1% accuracy on this dataset. This model has also used the membrane potentials of neurons in the last layer to do the classification. To do a spike-based classification, authors in Mozafari *et al.*[31] proposed a two-layer convolutional network with four Gabor filters in the first layer and 20 filters learned by reward-modulated STDP in the second layer. Each of the 20

Table 2. The recognition accuracy of different SNN models on the Caltech face/motorbike dataset along with their learning and classification methods. Note that models with spike-based classification do not need an external classifier and make their decision based on the spiking activity of their last layer.

Model	Learning method	Classifier	Accuracy (%)
Masquelier *et al.*[28]	Unsupervised STDP	RBF	99.2
Kheradpisheh *et al.*[30]	Unsupervised STDP	SVM	99.1
Mozafari *et al.*[31]	Reward modulated STDP	Spike-based	98.2
S4NN (This paper)	Backpropagation	Spike-based	99.2



(a) Before Training



(b) After Training

Fig. 4. The firing times of the face and motorbike output neurons over the face and motorbike images at (a) the beginning and (b) the end of the learning phase. The left (right) plots show the firing times of both neurons over the face (motorbike) images.

filters was assigned to a specific category and a decision was made by the first neuron to fire. It reached 98.2% accuracy on Caltech face/motorbike dataset. The important feature of this network was the spike-time-based decision-making achieved through reinforcement learning. The proposed S4NN also makes decisions by the spike times and could reach 99.2% accuracy only by using four hidden and two output neurons.

As explained in Sec. 2.2, each output neuron is assigned to a category and the network decision is made based on the first output neuron to fire. During the learning phase, regarding the relative target firing time (see Sec. 2.5), the network adjusts its weights to make the correct output neuron to fire first (see Sec. 2.4). Figure 4 provides the firing time of both face and motorbike output neurons (over the training and validation images) at the beginning and ending of the learning phase. As seen in Fig. 4(a), at the beginning of the learning, the distributions of the firing time of both output neurons (regardless of the image category) are interleaved which leads to a poor classification accuracy around the chance level. But as the learning phase proceeds and the network learns to solve the task, the correct output neuron tends to fire earlier.

As shown in Fig. 4(b), at the end of the learning phase, for each image category, its corresponding output neuron fires at the early time steps while the other neuron fires long after. Note that, during the training phase, we force neurons to emit a fake spike at the last time step if they have not fired during

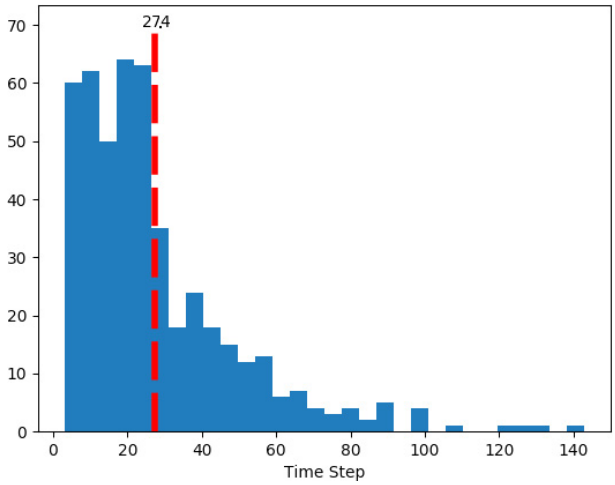
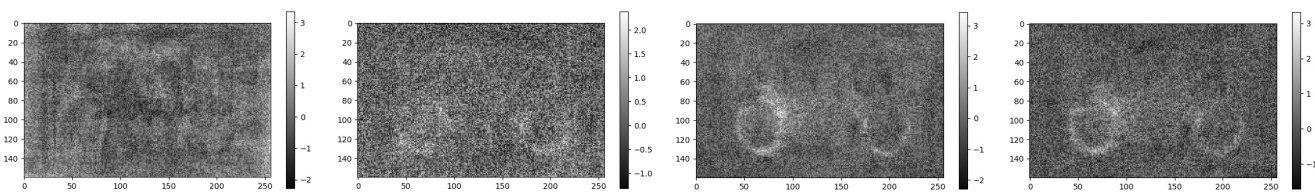


Fig. 5. The histogram of the firing time of the winner neuron (regardless of its category) over the training images. The red dashed line shows the mean firing time of the winner neuron.

the simulation. Hence, in the test phase, we do not need to continue the simulation after the emission of the first spike in the output layer. Figure 5 shows the distributions of the firing time of the winner neurons. The mean firing time for winner neuron is 27.4 (shown by the red line) wherein 78% of the images, the winner neuron has fired within the first 40 time steps. It means that the network makes its decision very quickly (compared to the maximum possible simulation time, $t_{\max} = 256$) and accurately (with only 0.8% error rate).

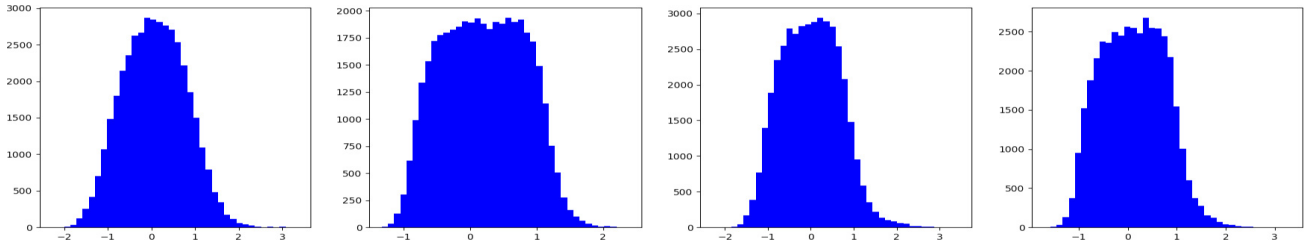
As the employed network has only one hidden layer of fully connected neurons, we can simply reconstruct the pattern learned by each hidden neuron by plotting its synaptic weights. Figure 6 depicts the synaptic weights of the four hidden neurons at the end of the learning phase. As seen, neurons #2 to #4 became selective to different shapes of motorbikes covering the shape variety of motorbikes. Neuron #1 has learned a combination of faces appearing at different locations and consequently responds only to face images. Because of the competition held between the output neurons to fire first, hidden and output neurons should learn and rely on the early spikes received from the input layer (not all of them). And this is the reason why the learned features in the hidden layer are not visually well detectable. The distribution of synaptic weights for each of the four hidden neurons are plotted in Fig. 7. As seen, the initial uniform distribution of the weights is transformed into the normal distribution with the zero mean. Here, positive weights encourage neurons to fire for their learned patterns and negative weights prevent them from firing for other patterns. Negative weights help the network to decrease the chance of unwanted spikes. For instance, a negative synaptic weight from a motorbike selective hidden neuron to the face output neuron significantly decreases the chance of an unwanted spike by the face neuron.

Furthermore, we evaluated the robustness of the trained S4NN to jitter noise. To this end, during the test phase, we add random integers drawn from a uniform distribution in range [−J, J] to the pixels of the input images. We changed the jitter parameter, J, from 0 to 240 with a step size of 20. Figure 8 shows the recognition accuracy of the S4NN trained on face/motorbike dataset over the test samples contaminated by different levels of jitter. Interestingly, even for $J = 240$, the S4NN accuracy drops by at most 5%. It shows that S4NN is robust to even intense noise levels. Indeed, neurons in the hidden layer has strong (positive or negative) synaptic weights only to those input neurons that contribute in the face/motorbike categorization task (see Fig. 6) while the rest majority of inputs have very small synaptic weights (see Fig. 7) and do not contribute much in the neural processing. Hence, because the



(a) Neuron #1 (b) Neuron #2 (c) Neuron #3 (d) Neuron #4

Fig. 6. The pattern (input-hidden weight matrix) learned by each of the four hidden neurons. The first neuron responds to face images while the other three are selective to the motorbikes variants.



(a) Neuron #1 (b) Neuron #2 (c) Neuron #3 (d) Neuron #4

Fig. 7. The histogram of the input-hidden synaptic weighs for each of the four hidden neurons.

jitter noise just changes the order of spikes, it can not much affect the behavior of IF neurons. Note that IF neurons are perfect integrators without leak and are less sensitive to the order of inputs than leaky neurons.

To assess the capacity of the proposed temporal backpropagation algorithm to be used in deeper architectures, we did another experiment on Caltech face/motorbike dataset with a three-layer network. The deep network is comprised of two hidden layers each of which consists of four IF neurons followed by an output layer with two IF neurons. We initialized the input-hidden1, hidden1-hidden2, and hidden2-output weights with random values drawn from uniform distributions in range $[0, 1]$, $[0, 50]$, and $[0, 50]$, respectively. Other parameters are the same as the aforementioned network with one hidden layer. After 25 training epochs, the network reached 99.1±0.2% accuracy on testing images with the mean firing time of 32.1 for the winner neuron. Although the accuracy of the network is 0.1% higher than the deeper network on average, this difference is not statistically significant (paired t-test on the accuracies of 10 different runs for each network; p-value < 0.05).

3.2. *MNIST dataset*

MNIST[41] is a benchmark dataset that has been widely used in the SNN literature.[7] We also evaluated the proposed S4NN on the MNIST dataset which contains 60,000 training and 10,000 test handwritten single-digit images. Each image is of size 28×28 pixels and contains one of the digits 0–9. To this end, we used a S4NN with one hidden and one output layer containing 400 and 10 IF neurons, respectively. The input layer is of the same size as the input images where the firing time of each input neuron is determined by the time-to-first-spike coding

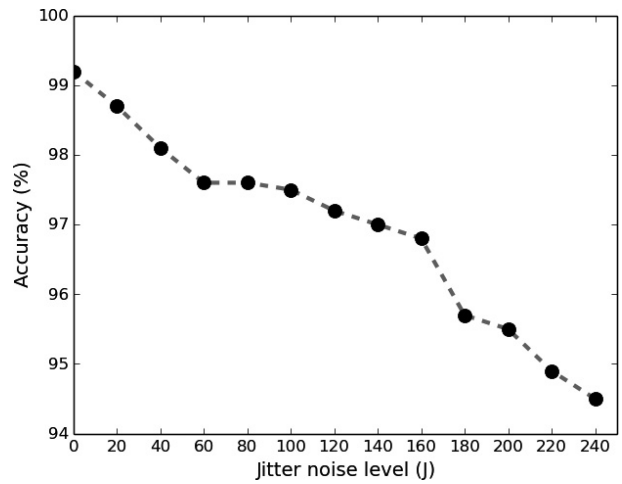


Fig. 8. The recognition accuracy of S4NN trained on the normal face/motorbike images and evaluated on test images contaminated by different amounts of jitter noise.

explained in Sec. 2.1 with the maximum simulation time of $t_{\max} = 256$. The input-hidden and hidden-output layers' synaptic weights are randomly drawn from uniform distributions in ranges $[0, 5]$ and $[0, 50]$, respectively. The threshold for all the neurons in all the layers was set to $\theta_i^l = 100$. We set the learning rate as $\eta = 0.2$, the penalty term in the target firing time calculation as $\gamma = 3$, and the regularization parameter as $\lambda = 10^{-6}$.

Table 3 provides the categorization accuracies of the proposed S4NN (97.4±0.2%) and other recent SNNs with spike-time-based supervised learning rules on the MNIST dataset. In Mostafa,[24] the use of 800 IF neurons with alpha functions complicates the neural processing and the learning procedure of the network. In Tavanaei *et al.*,[44] the network computational cost is quite large due to the use of rate coding and 1000 hidden neurons. In Comsa *et al.*,[25] the use of complicated SRM neuron model with

Table 3. The recognition accuracies of recent supervised SNNs with time-based backpropagation on the MNIST dataset. The details of each model including its input coding scheme, neuron model, learning method, and the number of hidden neurons are provided.

Model	Coding	Neuron model	Learning method	Hidden neurons	Acc. (%)
Mostafa[24]	Temporal	IF (exponential synaptic current)	Temporal backpropagation	800	97.2
Tavanaei *et al.*[44]	Rate	IF (instantaneous synaptic current)	STDP-based backpropagation	1000	96.6
Comsa *et al.*[25]	Temporal	SRM (exponential synaptic current)	Temporal backpropagation	340	97.4
ANN	—	ReLU	Backpropagation with Adam	400	98.1
S4NN (This paper)	Temporal	IF (instantaneous synaptic current)	Temporal backpropagation	400	97.4

Table 4. The mean firing time-step of the correct output neuron along with the mean required number of spikes (in all the layers) until the emission of the first spike at the output layer, for each digit category.

Digit	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
Mean firing time-step	97.2 ±40.0	44.1 ±24.4	75.3 ±33.9	98.1 ±40.3	118.5 ±34.7	81.2 ±38.4	90.9 ±36.7	100.1 ±36.2	115.6 ±36.9	75.6 ±34.1
Mean required spikes	221.0 ±42.8	172.6 ±43.2	226.4 ±42.7	220.5 ±41.5	233.2 ±40.5	220.7 ±43.3	224.0 ±42.7	224.6 ±43.0	233.6 ±40.6	213.4 ±43.6

the exponential synaptic current makes it difficult for event-based implementation. Comsa *et al.* implemented their model in two versions, where their fast model, similar to ours, decides by the first spike at the output layer and reached 97.4% accuracy on MNIST. While the slow version of their model needs to wait for all the hidden neurons to fire before making its decision. The slow version could reach 97.9% accuracy on MNIST. The advantage of S4NN is the use of simple neuron model (IF with an instantaneous synaptic current), temporal coding with at most one spike per neuron, and simple supervised temporal learning rule. Also, we used only 400 neurons in the hidden layer which makes it lighter than other networks.

We have also implemented a three-layer ANN (input-hidden-output) with 400 hidden units. We used the ReLU activation function for both hidden and output layers and employed mean squared error (MSE) as the loss function. We trained the network with Adam optimizer and reached 98.1% accuracy on MNIST. Although the ANN outperforms all the SNN models in Table 3, the advantage of SNNs is their energy efficiency and hardware friendliness.

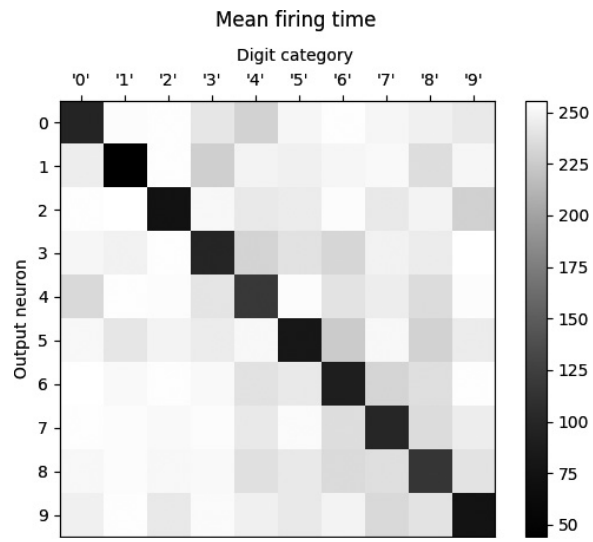


Fig. 9. The mean firing time of each output neuron (rows) over the images of different digit categories (columns).

Figure 9 shows the mean firing time of each output neuron on images of different digit categories. As seen, for each digit category, there is a huge gap between the mean firing time of the correct output

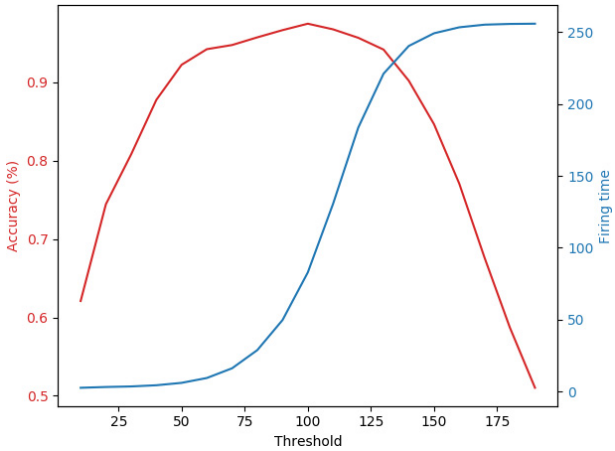


Fig. 10. The speed-accuracy trade-off. The network is pre-trained by the threshold of 100 for all hidden and output neurons, then the model is evaluated on test set with the threhold of 10 to 150. For each threshold value, the accuracy and the mean firing time of the winner ouput nueron is computed.

neuron and others. Digits '1' and '4' with the firing times of 44.1 and 118.5 have the minimum and maximum mean firing times, respectively. Hypothetically, recognition of digit '1' relies on much fewer spikes than other digits and would have a much faster response. While digit '4' (or digit '8' with the mean firing time of 101.5) needs much more input spikes to be correctly recognized from other (and similar) digits. Interestingly, on average, the network needs 172.69 spikes to recognize digit '1' and 233.22 spikes for digit '4'. Table 4 presents the mean firing time of the correct output neurons along with the mean required number of spikes. Note that the required spikes are obtained by counting the number of spikes in all the three layers (input, hidden, and output) until the emission of the first spike at the output layer.

On average, the proposed S4NN makes its decisions with 97.4% precision in 89.7 time steps (35.17% of maximum simulation time) with only 218.3 spikes (18.22% of 784+400+10 possible spikes). Note that, on average, hidden neurons emit 132.2±6.7 until the network makes its decision. Therefore, the proposed network works in a fast, accurate, and sparse manner.

In a further experiment, we assessed the speed-accuracy trade-off in S4NN. To do so, we first trained S4NN (with the threshold 100 for all neurons) on MNIST and frizzed its weights, then we changed the threshold of all of its hidden and output neurons from 10 to 150 and evaluated it on the test set. Figure 10 shows the accuracy and the mean firing time of the winner output neurons (i.e. response-time) over different threshold values. As seen, by increasing the threshold, the accuracy increases, goes above 94% after threshold 70, and peaks at the threshold 100. Also, it can be seen that the mean response-time fastly grows after threshold 70. The mean response-time is around 15 time steps for threshold 70 and around 89 time steps for threshold 100. Hence, one can get a faster but a bit less accurate response from S4NN by lowering the threshold of a pre-trained network.

4. Discussion

SNNs are getting more and more popular these days[45–50] and it is one of the best tools to study computations in the brain.[51–59] In this paper, we proposed a SNN (called S4NN) comprised of multiple layers of nonleaky IF neurons with time-to-first-spike coding and temporal error backpropagation. Regarding the fast processing of objects in visual cortex (often in range 100–150 ms) and the fact that there are at least 10 synapses from photoreceptors in retina to object responsive neurons in inferotemporal (IT) cortex, each neuron has only about 10–15 ms to perform its computation which is not enough for rate coding.[60] Also, it is shown that the first wave of spikes in IT cortex around 100 ms after the image presentation carries enough information for object recognition,[61] indicating the importance of early spikes. In addition, there are many other neurophysiological[62,63] and computational[26,27] evidence supporting the importance of first-spike-coding.

According to our employed temporal coding, input neurons emit a spike with a latency negatively proportional to the corresponding pixel value and upstream neurons are allowed to fire only once at most. The proposed temporal error backpropagation, pushes the correct output neuron to fire earlier than others. It forces the network to make quick and accurate decisions with few spikes (high sparsity). Our experiments on Caltech face/motorbike (99.2% accuracy) and MNIST (97.4% accuracy) datasets show the merits of S4NN to accurately solve object recognition tasks with a simpler neuron model (i.e.

nonleaky IF) compared to other recent supervised SNNs with temporal learning rules.

Let's assume an S4NN model with l layers, where n is the number of neurons in the largest layer of the network. In a clock-based implementation, for any layer, the membrane potential of all neurons at any time step can be updated in $O(n^2)$. Therefore, the feedforward path of S4NN can be performed in $O(l * n^2 * t)$, where t is the time step of the first spike in the output layer. Note that the proposed temporal backpropagation forces the network to respond as accurate and early as possible. Hence, the required time steps, t, would be much smaller than the maximum simulation time. Note that the actual computational time of S4NN could be shorter when the time step period is shorter.

Hardware implementations are out of the scope of this paper. However, S4NN has some important features that might make it more (digital) hardware friendly. First, computation is restricted to at most one spike per neuron, and in practice, a decision is made before most neurons have fired. Conversely, spike-rate-based SNNs require a longer time to have enough output spikes to make a confident decision. Our approach is thus advantageous in terms of latency, but also in terms of energy, since on most neuromorphic chips energy consumption is mainly caused by spikes.[10] Second, our approach is memory efficient, as we can forget the state of a neuron as soon as it has fired, and re-use the corresponding memory for other neurons. Note that other approaches with at most one spike per neuron also share these three advantages.[24,25,34,64] Yet our neuron model is much simpler: there is no leak, and the synapses are instantaneous, which, as explained below, make it more hardware-friendly. Here we have shown for the first time that backpropagation can be adapted to this simple neuron model, even if this requires some approximation (Eq. (6)).

If a leak can be efficiently implemented in analog hardware using the physics of transistors or capacitors,[9] it is always costly in digital hardware. Two approaches have been proposed. Either the potential of all neurons is decreased periodically, for example, every millisecond (see e.g. Ref. 65). Obviously, this approach is energy–hungry. The leak can also be handled in an event-based manner: leakage is taken into account when an input spike is received, based on the elapsed time since the last input spike (see e.g. Refs. 66 and 67). But this requires storing the last input spike time for each neuron, which increases the memory footprint. Finally, instantaneous synapses are by far the most simple synapses to handle: each input spike causes a punctual potential increment. Current-based, or conductance-based synapses, require a state parameter, and each input spike causes the potential to be updated on several consecutive time steps.

Due to the nondifferentiability of the thresholding activation function of spiking neurons at their firing times, applying gradient descent and backpropagation algorithms to SNNs has always been a big challenge. Different studies proposed different techniques including rate-based differentiable activation functions,[13–15] smoothed spike generators,[16] and surrogate gradients.[8,17–21] All these approaches do not deal with spike times. In the last approach, known as latency learning, neuronal activity is defined based on its firing time (usually the first spike) and contrary to the three previous approaches, the derivation of the thresholding activation function is not needed. However, they need to define the firing time of the neuron as a function of its membrane potential or the firing time of presynaptic neurons and use its derivation in the backpropagation process. For instance, in Spikeprop,[23] authors use a linear approximation function that relies on the changes of the membrane potential around the firing time (hence, they can not use the IF neuron model). Also, in Mostafa,[24] by using exponentially decaying synapses, the author has defined the firing time of a neuron directly based on the firing times of its presynaptic neurons. Here, by assuming a monotonically increasing linear relation between the firing time and the membrane potential, we could use IF neurons with instantaneous synapses in the proposed S4NN model.

SNNs with latency learning use single-spike-time coding, and hence, there is a problem if neurons do not reach their threshold, because then the latency is not defined. There are different approaches to deal with this problem. In Mostafa,[24] the author uses nonleaky neurons and makes sure that the sum of the weights is more than the threshold or in Comsa,[25] authors use fake input "synchronization pulses" to push neurons over the threshold. In the proposed S4NN, we assume that if a neuron has not fired during the simulation it will fire sometime after the

simulation, thus, we force it to emit a fake spike at the last time step.

Here, we just tested the S4NN on image categorization tasks, future studies can test S4NN on other data modalities. As shown on the Caltech face/motorbike dataset, the proposed learning rule is scalable and can be used in deeper S4NN architectures. Also, it can be used in convolutional spiking neural networks (CSNNs). Current CSNNs are mainly converted from traditional CNNs with rate[68–71] and temporal coding.[72] Although these networks are well in terms of accuracy, they might not work efficiently in terms of computational cost or time. Recent efforts to develop CSNNs with spike-based backpropagation have led to impressive results on different datasets,[73,74] however, they use costly neuron models and rate coding schemes. Hence, extending the proposed S4NN to convolutional architectures can provide large computational benefits. The most important challenge in this way is to prevent vanishing/exploding gradients and learning under the weight-sharing constraint in convolutional layers. But contrary to the rate-based CSNNs, the max-pooling operation can be simply done by propagating the first spike emerging inside the receptive field of each pooling neuron.

Moreover, although SNNs are more hardware friendly than traditional ANNs, the backpropagation process in supervised SNNs is not easy to be implemented in hardware. Recently, efforts are made to approximate backpropagation using spikes[75] that can be used in S4NN and make it more suitable for hardware implementation.

Acknowledgments

This research was partially supported by the French Agence Nationale de la Recherche (Grant: Beating Roger Federer ANR-16-CE28-0017-01). The authors would like to thank Dr. A. Yousefzadeh for his valuable comments and discussions and Dr. J. P. Jaffrézou for proofreading this paper.

References

1. R. VanRullen, R. Guyonneau and S. J. Thorpe, Spike times make sense. *Trends Neurosci.* **28**(1) (2005) 1–4.
2. R. Brette, Philosophy of the spike: Rate-based versus spike-based theories of the brain, *Front. Syst. Neurosci.* **9** (2015) 151.
3. A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire and T. M. McGinnity, A review of learning in biologically plausible spiking neural networks, *Neural Netw.* **122** (2020) 253–272.
4. M. Pfeiffer and T. Pfeil, Deep learning with spiking neurons: Opportunities and challenges, *Front. Neurosci.* **12** (2018) 774.
5. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *Int. J. Neural Syst.* **19**(4) (2009) 295–308.
6. B. Illing, W. Gerstner and J. Brea, Biologically plausible deep learning–but how far can we go with shallow networks? *Neural Netw.* **118** (2019) 90–101.
7. A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier and A. Maida, Deep learning in spiking neural networks, *Neural Netw.* **111** (2019) 47–63.
8. E. O. Neftci, H. Mostafa and F. Zenke, Surrogate gradient learning in spiking neural networks, *IEEE Signal Processing Magazine* **36** (2019) 61–63.
9. K. Roy, A. Jaiswal and P. Panda, Towards spike-based machine intelligence with neuromorphic computing, *Nature* **575** (2019) 607–617.
10. M. Oster, R. Douglas and S.-C. Liu, Quantifying input and output spike statistics of a winner-take-all network in a vision system, in *2007 IEEE Int. Symp. Circuits and Systems* (IEEE, 2007), pp. 853–856.
11. R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas-Pérez, T. Delbruck *et al.*, Caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing–learning–actuating system for high-speed visual object recognition and tracking, *IEEE Trans. Neural Netw.* **20**(9) (2009) 1417–1438.
12. C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco and T. Delbruck, Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output, *Proc. IEEE* **102**(10) (2014) 1470–1484.
13. E. Hunsberger and C. Eliasmith, Spiking deep networks with life neurons, arXiv:1510.08829.
14. J. H. Lee, T. Delbruck and M. Pfeiffer, Training deep spiking neural networks using backpropagation, *Front. Neurosci.* **10** (2016) 508.
15. E. O. Neftci, C. Augustine, S. Paul and G. Detorakis, Event-driven random back-propagation: Enabling neuromorphic deep learning machines, *Front. Neurosci.* **11** (2017) 324.
16. D. Huh and T. J. Sejnowski, Gradient descent for spiking neural networks, *Adv. Neural Inf. Process. Syst.* **31** (2018) 1433–1443.
17. S. M. Bohte, Error-backpropagation in networks of fractionally predictive spiking neurons, *Int. Conf. Artificial Neural Networks* (Springer, 2011), pp. 60–68.
18. S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswama, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. D. Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner and D. S. Modha, Convolutional networks for

fast energy-efficient neuromorphic computing, *Proc. Natl. Acad. Sci. USA* **113**(41) (2016) 11441–11446.
19. S. B. Shrestha and G. Orchard, Slayer: Spike layer error reassignment in time, *Adv. Neural Inf. Process. Syst.* **31** (2018) 1412–1421.
20. F. Zenke and S. Ganguli, Superspike: Supervised learning in multilayer spiking neural networks, *Neural Comput.* **30**(6) (2018) 1514–1541.
21. G. Bellec, D. Salaj, A. Subramoney, R. Legenstein and W. Maass, Long short-term memory and learning-to-learn in networks of spiking neurons, *Adv. Neural Inf. Process. Syst.* **31** (2018) 787–797.
22. R. Zimmer, T. Pellegrini, S. Singh Fateh and T. Masquelier, Technical report: Supervised training of convolutional spiking neural networks with PyTorch, (2019), arXiv: 1911.10124.
23. S. M. Bohte, H. La Poutré and J. N. Kok, Error-backpropagation in temporally encoded networks of spiking neurons, *Neurocomput.* **48** (2000) 17–37.
24. H. Mostafa, Supervised learning based on temporal coding in spiking neural networks, *IEEE Trans. Neural Netw. Learning Syst.* **29**(7) (2017) 3227–3235.
25. I. M. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo and J. Alakuijala, Temporal coding in spiking neural networks with alpha synaptic function, *IEEE International Conference on Acoustics, Speech, and Signal Processing* (Barcelona, Spain, 2020).
26. S. J. Thorpe and J. Gautrais, Rank order coding, in *Computational Neuroscience: Trends in Research*, ed. J. M. Bower (Plenum Press, New York, 1998), pp. 113–118.
27. S. Thorpe, A. Delorme and R. V. Rullen, Spike-based strategies for rapid processing, *Neural Netw.* **14**(6–7) (2001) 715–725.
28. T. Masquelier and S. J. Thorpe, Unsupervised learning of visual features through spike timing dependent plasticity, *PLoS Comput. Biol.* **3**(2) (2007) e31.
29. S. R. Kheradpisheh, M. Ganjtabesh and T. Masquelier, Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition, *Neurocomput.* **205** (2016) 382–392.
30. S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe and T. Masquelier, Stdp-based spiking deep convolutional neural networks for object recognition, *Neural Netw.* **99** (2018) 56–67.
31. M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini and M. Ganjtabesh, First-spike-based visual categorization using reward-modulated stdp, *IEEE Trans. Neural Netw. Learning Syst.* **29**(12) (2018) 6178–6190.
32. M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe and T. Masquelier, Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks, *Pattern Recognit.* **94** (2019) 87–95.
33. M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini and T. Masquelier, SpykeTorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron, *Front. Neurosci.* **13** (2019) 1–12.
34. J. Göltz, A. Baumbach, S. Billaudelle, O. Breitwieser, D. Dold, L. Kriener, A. F. Kungl, W. Senn, J. Schemmel, K. Meier and M. A. Petrovici, Fast and deep neuromorphic learning with time-to-first-spike coding (2019), arXiv: 1911.10124.
35. R. Vaila, J. Chiasson and V. Saxena, Feature extraction using spiking convolutional neural networks, in *Proc. Int. Conf. Neuromorphic Systems — ICONS '19* (ACM Press, New York, USA, 2019), pp. 1–8.
36. P. Falez, P. Tirilly, I. M. Bilasco, P. Devienne and P. Boulet, Multi-layered spiking neural network with target timestamp threshold adaptation and stdp, *2019*, *Int. Joint Conf. Neural Netw.* (*IJCNN*) (IEEE, 2019), pp. 1–8.
37. A. N. Burkitt, A review of the integrate-and-fire neuron model: Ii. inhomogeneous synaptic input and network properties, *Biol. Cybernet.* **95**(2) (2006) 97–112.
38. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, Chap. 6.5 (MIT Press, 2016), pp. 200–220.
39. Y. LeCun, Y. Bengio and G. Hinton, Deep learning, *Nature* **521**(7553) (2015) 436–444.
40. J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Netw.* **61** (2015) 85–117.
41. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, Gradient-based learning applied to document recognition, *Proc. IEEE* **86**(11) (1998) 2278–2324.
42. R. Vaila, J. Chiasson and V. Saxena, Deep convolutional spiking neural networks for image classification, arXiv:1903.12272.
43. A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Adv. Neural Inf. Process. Syst.* **25** (2012) 1097–1105.
44. A. Tavanaei and A. Maida, Bp-stdp: Approximating backpropagation using spike timing dependent plasticity, *Neurocomput.* **330** (2019) 39–47.
45. T. Wu, F.-D. Bîlbîe, A. Păun, L. Pan and F. Neri, Simplified and yet turing universal spiking neural p systems with communication on request, *Int. J. Neural Syst.* **28**(8) (2018) 1850013.
46. M. Bernert and B. Yvert, An attention-based spiking neural network for unsupervised spike-sorting, *Int. J. Neural Syst.* **29** (2018) 1850059.
47. F. Galán-Prado, A. Morán, J. Font, M. Roca and J. L. Rosselló, Compact hardware synthesis of stochastic spiking neural networks, *Int. J. Neural Syst.* **29** (2019) 1950004.
48. R. Hu, Q. Huang, H. Wang, J. He and S. Chang, Monitor-based spiking recurrent network for the

representation of complex dynamic patterns, *Int. J. Neural Syst.* **29** (2019) 1950006.
49. A. Geminiani, C. Casellato, A. Antonietti, E. D'Angelo and A. Pedrocchi, A multiple-plasticity spiking neural network embedded in a closed-loop control system to model cerebellar pathologies, *Int. J. Neural Syst.* **28**(5) (2018) 1750017.
50. X. Zhang, G. Foderaro, C. Henriquez and S. Ferrari, A scalable weight-free learning algorithm for regulatory control of cell activity in spiking neuronal networks, *Int. J. Neural Syst.* **28**(2) (2018) 1750015.
51. G. Antunes, S. F. Faria da Silva and F. M. Simoes de Souza, Mirror neurons modeled through spike-timing-dependent plasticity are affected by channelopathies associated with autism spectrum disorder, *Int. J. Neural Syst.* **28**(5) (2018) 1750058.
52. A. Antonietti, J. Monaco, E. D'Angelo, A. Pedrocchi and C. Casellato, Dynamic redistribution of plasticity in a cerebellar spiking neural network reproducing an associative learning task perturbed by tms, *Int. J. Neural Syst.* **28**(9) (2018) 1850020.
53. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural Netw.* **22**(10) (2009) 1419–1431.
54. S. Ghosh-Dastidar and H. Adeli, Improved spiking neural networks for EEG classification and epilepsy and seizure detection, *Integr. Comput.-Aided Eng.* **14**(3) (2007) 187–212.
55. H. Adeli and S. Ghosh-Dastidar, *Automated EEG-Based Diagnosis of Neurological Disorders*: *Inventing the Future of Neurology* (CRC Press, 2010).
56. H. Peng, J. Yang, J. Wang, T. Wang, Z. Sun, X. Song, X. Luo and X. Huang, Spiking neural p systems with multiple channels, *Neural Netw.* **95** (2017) 66–71.
57. T. Wu, A. Păun, Z. Zhang and L. Pan, Spiking neural p systems with polarizations, *IEEE Trans. Neural Netw. Learn. Syst.* **29**(8) (2017) 3349–3360.
58. L. Pan, G. Păun, G. Zhang and F. Neri, Spiking neural p systems with communication on request, *Int. J. Neural Syst.* **27**(8) (2017) 1750042.
59. H. Peng, J. Wang, P. Shi, M. J. Pérez-Jiménez and A. Riscos-Núñez, An extended membrane system with active membranes to solve automatic fuzzy clustering problems, *Int. J. Neural Syst.* **26**(3) (2016) 1650004.
60. S. J. Thorpe, Spike arrival times: A highly efficient coding scheme for neural networks, *Parallel Process. Neural Syst.* (1990) 91–94.
61. C. P. Hung, G. Kreiman, T. Poggio and J. J. DiCarlo, Fast readout of object identity from macaque inferior temporal cortex, *Science* **310**(5749) (2005) 863–866.
62. F. Bengtsson, R. Brasselet, R. S. Johansson, A. Arleo and H. Jörntell, Integration of sensory quanta in cuneate nucleus neurons *in vivo*, *PloS One* **8**(2) (2013) e56630.
63. R. Brasselet, R. S. Johansson and A. Arleo, Quantifying neurotransmission reliability through metrics-based information analysis, *Neural Comput.* **23**(4) (2011) 852–881.
64. C. Stöckl and W. Maass, Recognizing images with at most one spike per neuron, (2019), pp. 1–14, arXiv: 2001.01682.
65. A. Yousefzadeh, T. Masquelier, T. Serrano Gotarre dona and B. Linares-Barranco, Hardware implementation of convolutional STDP for on-line visual feature learning, *2017 IEEE International Symposium on Circuits and Systems* (*ISCAS*) (Baltimore, MD, USA, 2017), pp. 1–4.
66. A. Yousefzadeh, T. Serrano-Gotarredona and B. Linares-Barranco, Fast Pipeline 128 × 128 pixel spiking convolution core for event-driven vision processing in FPGAs, in *2015 International Conference on Event-Based Control*, *Communication*, *and Signal Processing* (*EBCCSP*), (IEEE, 2015), pp. 1–8.
67. G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor and R. Benosman, HFirst: A temporal approach to object recognition, *IEEE Trans. Pattern Analy. Mach. Intell.* **37** (2015) 2028–2040.
68. Y. Cao, Y. Chen and D. Khosla, Spiking deep convolutional neural networks for energy-efficient object recognition, *Int. J. Comput. Vis.* **113**(1) (2015) 54–66.
69. P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni and E. Neftci, Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware, in *2016 IEEE Int. Conf. Rebooting Computing* (*ICRC*) (IEEE, 2016), pp. 1–8.
70. A. Sengupta, Y. Ye, R. Wang, C. Liu and K. Roy, Going deeper in spiking neural networks: Vgg and residual architectures, *Front. Neurosci.* **13** (2019) 95.
71. B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer and S.-C. Liu, Conversion of continuous-valued deep networks to efficient event-driven networks for image classification, *Front. Neurosci.* **11** (2017) 682.
72. B. Rueckauer and S.-C. Liu, Conversion of analog to spiking neural networks using sparse temporal coding, *2018 IEEE Int. Symp. Circuits and Systems* (*ISCAS*), (IEEE, 2018), pp. 1–5.
73. Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie and L. Shi, Direct training for spiking neural networks: Faster, larger, better, *Proc. AAAI Conf. Artif. Intell.* **33** (2019) 1311–1318.
74. C. Lee, S. S. Sarwar and K. Roy, Enabling spike-based backpropagation in state-of-the-art deep neural network architectures (2019), arXiv:1903.06379.
75. J. C. Thiele, O. Bichler and A. Dupret, Spikegrad: An ann-equivalent computation model for imple menting backpropagation with spikes (2019), arXiv: 1906.00851.