A gradient descent rule for spiking neurons emitting multiple spikes

Olaf Booij [a,*] Hieu tat Nguyen [a]

[a] *Intelligent Sensory Information Systems, University of Amsterdam, Faculty of Science, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands*

Abstract

A supervised learning rule for Spiking Neural Networks (SNNs) is presented that can cope with neurons that spike multiple times. The rule is developed by extending the existing SpikeProp algorithm which could only be used for one spike per neuron. The problem caused by the discontinuity in the spike process is counteracted with a simple but effective rule, which makes the learning process more efficient. Our learning rule is successfully tested on a classification task of Poisson spike trains. We also applied the algorithm on a temporal version of the XOR problem and show that it is possible to learn this classical problem using only one spiking neuron making use of a hairtrigger situation.

Key words: Spiking neural networks, Temporal pattern recognition, Error-backpropagation, Parallel processing

1 Introduction

In the last ten years the aim of neural network research has shifted to models that are more biologically plausible, such as Spiking Neural Networks (SNN) [1,8,3]. These SNNs model the precise time of the spikes fired by a neuron, as opposed to conventional neural networks, which model only the average firing rate of the neurons [9]. Due to this temporal coding, SNNs are believed to be especially useful for the processing of temporal patterns, such as speech recognition [1,8,10]. These temporal patterns consist of correlating temporal events, such as a series of phonemes in the case of speech. The fact that

* Corresponding author.
Email addresses: obooij@science.uva.nl (Olaf Booij), hieu@science.uva.nl (Hieu tat Nguyen).

these temporal events can happen more than once, makes temporal patterns actually different from static patterns. So in order to process such patterns with an SNN, they have to be encoded in multiple spikes per input neuron and the neurons have to be able to spike more than once. Although there have been various studies introducing learning rules for networks of spiking neurons that spike only once [1,13], no practical supervised algorithm has been developed that can cope with neurons that fire multiple spikes.

In [6] a learning method is designed for one spike per neuron, which is shown to be extendable to cope with input neurons spiking multiple times. However, using this extension, the classification is more influenced by the first spikes than by later spikes, making the method less practical. A principle often used to classify multiple spikes per input neuron, is by first clustering these multiple spikes to single spikes using an SNN in combination with an unsupervised learning rule and then feeding these single spikes to a second SNN which uses a supervised learning rule [11,12]. This classification method is far from optimal because the clustering method does not distinguish between salient and invariant information in the data for a particular classification task.

It would be beneficial to have a completely supervised learning algorithm that is designed to cope with multiple spikes per neuron and could learn to classify temporal data like the conventional backpropagation algorithm can learn static data. In this study we developed such a learning rule based on the SpikeProp algorithm, which is successful for the classification of patterns coded in single spikes [1]. We extend this rule to make it applicable to neurons that spike multiple times.

This paper is organized as follows. In Section 2 we describe the neural model we use. Then, in Section 3, we derive our learning rule and discuss the problems caused by the discontinuity of the spiking process. We test our algorithm in Section 4 and finally draw some conclusions in Section 5.

2 Spiking neuron model

We model the behavior of the spiking neurons according to the Spike Response Model [8]. In this model the output of a neuron j is completely described by the set of precise firing times of the spikes it produced, the so called spike train:

$$\mathcal{F}_j = \{t_j^f; 1 \leq f \leq n\}, \tag{1}$$

where n denotes the number of spikes and the spike train is chronologically ordered: $1 \leq f < g \leq n \;\rightarrow\; t_j^f < t_j^g$. The neuron fires a spike when its internal

state, the membrane potential u_j, reaches a threshold ϑ from below:

$$u_j(t^f) = \vartheta \ \wedge \ \frac{du_j(t)}{dt}|_{t=t^f} > 0. \tag{2}$$

We have set the threshold ϑ to 1.0.

This potential is influenced not only by the spikes of its presynaptic neurons Γ_j (see figure 1), but also by the spikes it produced itself:

$$u_j(t) = \sum_{t_j^f \in \mathcal{F}_j} \eta(t - t_j^f) + \sum_{i \in \Gamma_j} \sum_{t_i^f \in \mathcal{F}_i} \sum_k w_{ji}^k \varepsilon(t - t_i^f - d_{ji}^k), \tag{3}$$

where variable w_{ji}^k denotes the weight of synapse k from neuron i to neuron j and d_{ji}^k denotes the axonal delay. The spike response function ε describes the contribution of one synaptic terminal on the potential of the postsynaptic neuron caused by a presynaptic spike. In our simulations this so called postsynaptic potential (PSP) is modeled by the difference between two exponential decays:

$$\varepsilon(s) = \left[\exp\left(-\frac{s}{\tau_m}\right) - \exp\left(-\frac{s}{\tau_s}\right)\right] \mathcal{H}(s), \tag{4}$$

where $\mathcal{H}(s)$ denotes the Heavyside step function: $\mathcal{H}(s) = 0$ for $s \leq 0$ and $\mathcal{H}(s) = 1$ for $s > 0$. The time constants τ_m and τ_s which determine the rise and decay of the function are set to 10 ms and 5 ms respectively.

After a neuron emitted a spike, the potential decreases instantly to the resting potential, which is defined as zero, and it is more difficult for the neuron to generate a second spike shortly afterwards, the so called refractoriness. This is modeled by the η function, for which we use simple exponential decay:

$$\eta(s) = -\vartheta \exp\left(-\frac{s}{\tau_r}\right) \mathcal{H}(s), \tag{5}$$

where the time constant τ_r is set to 10 ms and ϑ is again the membrane threshold. In the learning process the derivative of both functions will be used. For the special case of $s = 0$, for which these derivatives are undetermined, we will define them to be zero. Both these functions can be replaced by other functions without invalidating the learning rule, as long as the derivatives exist.

3 Extending SpikeProp to multiple spikes

In the following we will derive the learning rule for multiple spiking neurons by extending the existing SpikeProp algorithm [1]. The SpikeProp algorithm was specifically designed for networks of spiking neurons where each neuron emitted at most one spike each. The tasks on which it could be applied were thus no different than that of conventional neural networks. In this section we extend the algorithm so it can cope with spiking neurons that emit more than one spike and can thus learn to classify sets of spike trains.

The SpikeProp rule is based on the error-backpropagation rule for conventional neural networks which in turn is based on the gradient descent method [9]. This method requires the derivative of the network error with respect to the free parameters to exist. For SNNs this is not always the case due to the discontinuity in the threshold function (equation (1)). A small change of a parameter does not only cause a shift of the spike times, but can also cause a neuron to emit less or more spikes. If such a generation or removal of spikes occurs, the network error makes a jump. We do not take this effect into account during the derivation of the algorithm, but assume that the number of spikes stays the same. In some cases this approximation causes problems, for which we propose some extra rules in subsection 3.2.

3.1 Gradient descent

We now derive a gradient descent learning rule in a similar way as SpikeProp [1]. However we take into account that the neurons can fire multiple spikes. In SpikeProp every neuron backpropagates a single error value, namely the error with respect to its one spike. In our derivation the neuron needs to determine and backpropagate an error value for each spike it fired.

The network architecture on which the SpikeProp algorithm and our extension of it can be applied is in principle not constrained to feedforward connections only, as was the case with the conventional backpropagation algorithm [9]. So the network could be recurrent as long as there are no connections leading to input neurons or coming from output neurons, see figure 1. Although we must note that we have only tested the algorithm on feedforward networks.

We define the network error to be the sum squared error of the first spike of the output neurons O, so later spikes of these neurons are ignored:

$$E = \frac{1}{2} \sum_{o \in O} (t_o^1 - \widehat{t}_o^1)^2, \tag{6}$$

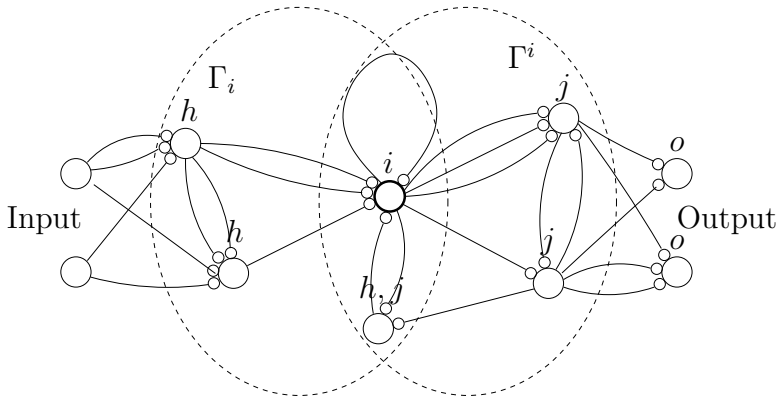

Figure 1. A network architecture with recurrent connections on which the learning rule can be applied.

where $\widehat{t}_o^1$ denotes the desired spike time. Other error functions, including functions over more than only the first output spike, are also possible, as long as the derivative of the error to one spike exists. In our case the error with respect to one output spike is determined by

$$\frac{\partial E}{\partial t_o^1} = t_o^1 - \widehat{t}_o^1. \tag{7}$$

The parameters that are tuned to reduce this error are the weights of the synapses w_{ji}^k. Other parameters, like the axonal delays, could also be learned [14]. Like SpikeProp we calculate the derivative of the network error with respect to the weight in order to calculate the appropriate weight change [1]:

$$\Delta w_{ih}^k = -\kappa \frac{\partial E}{\partial w_{ih}^k}, \tag{8}$$

where κ denotes the learning rate. Because neuron i can fire multiple times and all these firing times depend on the weight, this equation can be expanded with regard to these spikes:

$$\Delta w_{ih}^k = -\kappa \sum_{t_i^f \in \mathcal{F}_i} \frac{\partial E}{\partial t_i^f} \frac{\partial t_i^f}{\partial w_{ih}^k}. \tag{9}$$

First the second factor on the right hand side, the derivative of a firing time with respect to the weight, is calculated using the results of the study by Bohte et al. for the SpikeProp algorithm; see [1] or [2] for a complete derivation.

$$\frac{\partial t_i^f}{\partial w_{ih}^k} = \frac{\partial u_i(t_i^f)}{\partial w_{ih}^k} \frac{-1}{\frac{\partial u_i(t_i^f)}{\partial t_i^f}}. \tag{10}$$

Both the derivatives on the right hand side can be derived using equation (3). When calculating the partial derivative of the potential during a spike t_i^f with respect to the weight, we must keep in mind that earlier spikes $t_i^g < t_i^f$ also depend on the same weight and influence the potential. Hence, the function becomes recursive:

$$\frac{\partial u_i(t_i^f)}{\partial w_{ih}^k} = -\sum_{t_i^g \in \mathcal{F}_i} \eta'(t_i^f - t_i^g) \frac{\partial t_i^g}{\partial w_{ih}^k} + \sum_{t_h^l \in \mathcal{F}_h} \varepsilon(t_i^f - t_h^l - d_{ih}^k). \tag{11}$$

We will not explicitly require $t_i^g < t_i^f$, while this condition is already met due to the fact that $\varepsilon'(s) = \eta'(s) = 0$ for $s \leq 0$.

The second derivative of equation (10) evaluates to:

$$\frac{\partial u_i(t_i^f)}{\partial t_i^f} = \sum_{t_i^g \in \mathcal{F}_i} \eta'(t_i^f - t_i^g) + \sum_{h \in \Gamma_i} \sum_{t_h^l \in \mathcal{F}_h} \sum_k w_{ih}^k \varepsilon'(t_i^f - t_h^l - d_{ih}^k). \tag{12}$$

We now calculate the first factor on the right hand side of equation (9), the derivative of the network error with respect to a spike. This is derived for every spike of non-output neuron $i \notin O$, which depends on all spikes by all its neural successors Γ^i (see figure 1):

$$\frac{\partial E}{\partial t_i^f} = \sum_{j \in \Gamma^i} \sum_{t_j^g \in \mathcal{F}_j} \frac{\partial E}{\partial t_j^g} \frac{\partial t_j^g}{\partial t_i^f}. \tag{13}$$

Thus, in order to calculate the derivative of the error with respect to a spike, we already have to know the derivative of the error with respect to spikes of its neuronal successors that happened later in time. In practice this means that the error with respect to the last spike of the network has to be computed first, then to the spike preceding the last, and so on. This is the temporal equivalent of the conventional backpropagation technique, where the error was propagated back spatially through the network [9].

The derivative of a postsynaptic spike with respect to a presynaptic spike can be further expanded as in equation (10):

$$\frac{\partial t_j^g}{\partial t_i^f} = \frac{\partial u_i(t_j^g)}{\partial t_i^f} \frac{-1}{\frac{\partial u_j(t_j^g)}{\partial t_j^g}}. \tag{14}$$

The second factor already appeared in equation (10) and is calculated in equation (12). The first factor, the derivative of the potential during a postsynaptic

spike with respect to a presynaptic spike, can again be derived from equation (3):

$$\frac{\partial u_j(t_j^g)}{\partial t_i^f} = -\sum_{t_j^l \in \mathcal{F}_j} \eta'(t_j^g - t_j^l)\frac{\partial t_j^l}{\partial t_i^f} - \sum_k w_{ji}^k \varepsilon'(t_j^g - t_i^f - d_{ji}^k). \tag{15}$$

As can be seen this is again a recursive function, so the order in which the computations are done is important.

With this set of formulas it is now possible to compute the necessary weight changes that minimize a certain error measure of the spikes of the output neurons given a pattern of input spike trains.

3.2 Handling of discontinuity

We now address the problems the discontinuity of the spike process cause for the derived gradient descent rule. There are two circumstances that need special attention.

A troublesome situation, which is overlooked in all other studies regarding SpikeProp [1,15,14,10], occurs when a spike is produced by a potential that barely reaches the threshold. In this case the gradient of the potential during the spike is very small which in turn causes the derivative of the error to be very high resulting in a very large weight change, see equation (10) and (14). These large weight changes will almost always lead to a large increase in the network error, after which learning has to start over again. We used a simple but effective solution to overcome this chaotic behavior by putting a lower bound on the slope of the potential when computing the error. If the gradient of the potential during the spike turned out to be smaller than 0.1, we used a value of 0.1 instead, which prevented large weight changes.

Secondly, if a weight of a connection is so small that the postsynaptic neuron will not spike, then the error with respect to that weight will be zero, see equation (9). In this case the weight will never increase and the neuron will never be able to spike. A way to circumvent this, is to introduce an extra rule that increases the weight with a small amount if the postsynaptic neuron did not fire. An equivalent method, used in [14] and [10] is to lower the threshold of the postsynaptic neuron. Another approach is to start learning with such high weights that the neurons will initially fire for every input pattern. Using a small enough learning rate the weights will generally be lowered to values that minimize the network error but which do not cause an output neuron to stop firing. In our experiments we used the second approach, while it proved to be sufficient to avoid non-spiking output neurons.

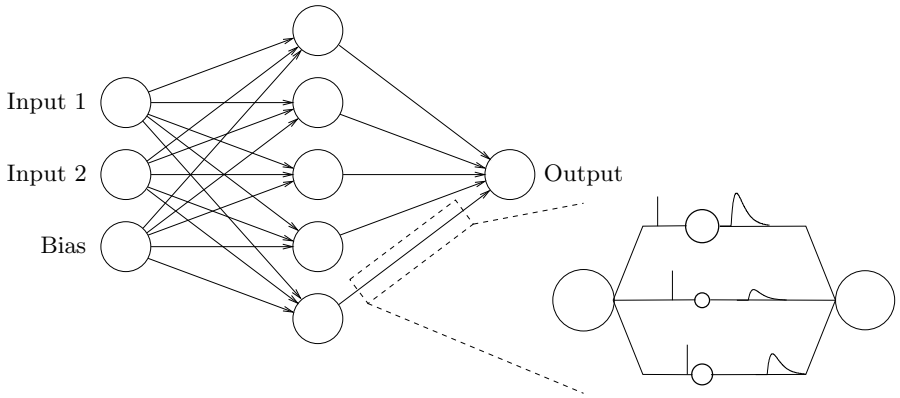

Figure 2. The layered feedforward architecture used in our experiments. Every connection consists of multiple synapses with different delays.

4 Experiments

In this section we illustrate the capabilities of the proposed learning rule to learn artificially generated tasks. First the effect of adding the extra rule, as described in subsection 3.2, is shown by applying the algorithm on a temporal version of the eXclusive-OR (XOR) benchmark. Then we show the capability of the algorithm to classify multiple spikes by applying it to Poisson spike trains.

The network architecture we use is a simple layered feedforward network with multiple delays per connection, as described in [5] and used to test the SpikeProp algorithm [1], see figure 2. Due to the lack of recurrent connections, such a network is easy to comprehend, but also has the disadvantage that the timescale on which it can recognize spikes is determined by the delay interval and the time constants of the connections, which are generally small. We use the same delay configuration as in [1] and [12]: every connection consists of 16 synapses with a delay of $\{1, 2, \ldots, 16\}$ ms.

4.1 The XOR benchmark

The original SpikeProp algorithm was tested on the classical XOR problem [1]. We attempt to replicate that experiment with and without the rule that prevents large weight changes. Before the network is applied to the function, it is encoded into spike time coding as shown in table 1.

This encoding is fed to a network with 3 input neurons, one hidden layer with 5 neurons, which are only allowed to spike once, and 1 output neuron, see figure 2. Before learning, the weights leading to the hidden neurons are set to random values between -0.5 and 1.0, while the weights leading to the output

Input 1	Input 2	Bias		Output
0	0	0	→	16
0	6	0	→	10
6	0	0	→	10
6	6	0	→	16

Table 1
A spike time encoding of the temporal XOR problem [1]. The numbers represent the precise time of the spikes in milliseconds.

neurons are set in such a way that connections from 4 hidden neurons only have positive weights, making them purely excitatory, and from the remaining neuron only negative weights, making it inhibitory. The weight initialization is thus bounded for these excitatory and inhibitory neurons to a range of respectively $[0, 1]$ and $[-0.5, 0]$. Like in [1], we noticed that the network could not converge if the positive and negative weights were mixed. The learning rate κ is set to 10^{-2} and learning is applied after each pattern, the so called on-line learning method. Learning is halted when the sum squared error (SSE) drops below 1.0. For the simulations we used an iterative approach with a simulation step of 0.01 ms.

Without the rule to prevent chaotic weight changes 4 out of 100 trials did not converge to a correct weight configuration within a 1000 training cycles. In the successful trials the learning algorithm needed on average 196 training cycles to learn the problem. When applying the rule, bounding the slope of the potential, the algorithm always converged and needed 164 cycles on average, which suggests that the simple rule is sufficient to counteract the problems of the discontinuity.

While doing further experiments with the temporally encoded XOR function, we discovered something quite surprising. It turns out that a network without the hidden layer can also learn to solve the XOR problem. This was only possible using the extra rule and a very small learning rate of 10^{-4}. For this special task we switch to batch learning, as opposed to on-line learning, to ensure that the resulting solution was valid for all 4 patterns. The algorithm had trouble to converge and needed more than 10^6 cycles. With smaller time constants for the PSP, $\tau_m = 4$ ms and $\tau_s = 2$ ms, the learning is more stable and faster, but still needs more than 10^5 cycles. In [2] we investigate these results further and describe a one-layered SNN with only 7 synapses that can solve the temporal XOR. The important fact to conclude from this result is that a single spiking neuron can represent the XOR-function, as encoded in table 1; something that is impossible with a conventional neuron because of the non-linearity of XOR. It must be noted that the solutions found always use a so called hair-trigger situation: for some input patterns the potential just reaches the threshold, while for other patterns it stays slightly under

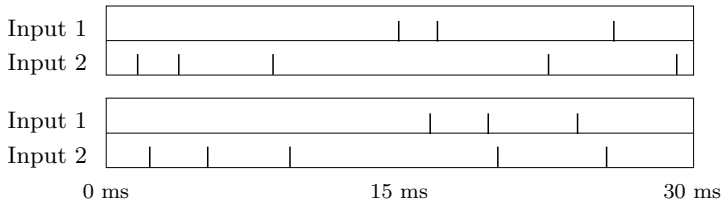

Figure 3. Two patterns consisting of two Poisson spike trains derived from the same template pattern by shifting the individual spikes.

it [7]. These situations should be avoided because the network will not be robust against noise and as experienced the learning algorithm has difficulty to converge to the right weight settings.

4.2 The Poisson Benchmark

In this experiment the algorithm will be tested on its capability to classify two sets of spike train patterns produced by Poisson processes [4]. For both the classes a template pattern is created consisting of two Poisson spike trains with a duration of 30 ms and on average 1 spike per 10 ms. From both these templates 10 noisy patterns are created by randomly shifting each spike by an amount drawn from a normal distribution with an SD of 2 ms, see figure 3. The resulting set of 20 patterns is then split in half to produce a training and a testing set. The output is again coded in the time-to-first-spike: for one class the output-neuron is designated to fire an early spike at 31 ms and for the other a later spike at 36 ms.

The same network architecture as used for the XOR benchmark is applied to this classification task (see figure 2). Also, the same parameter settings are used with the exception of the stopping criterion, which is set to an SSE of 20.

The algorithm learned the training set in on average 17 cycles and was then fed with the testing set. In the testing procedure the output was evaluated as follows: an output spike before 33.5 ms was seen as an early spike, while all other spikes were seen as a late spike. The network achieved an accuracy of 89 %, which shows that the algorithm is indeed capable of learning multiple spikes and can generalize well over temporal noise.

5 Conclusion and future work

We have developed a supervised learning rule for SNNs which is equivalent to the famous error-backpropagation rule for conventional neural networks and

proposed a simple but effective rule to counteract the problems of discontinuity. The algorithm does not restrict the spiking neurons to fire only once like other supervised rules, but can deal with multiple spikes per neuron. The capabilities of this new algorithm are shown by successfully learning Poisson spike trains. In [2] it was also successfully applied to a lipreading task, classifying small pieces of video. This demonstrates that the algorithm can be used for real-world applications requiring the processing of temporal data.

In our study we solely used feedforward networks, which can not be used to process temporal data with large time scales. The learning rule is in principle also applicable to more general architectures including recurrent connections, which do not have this restriction. Further research should be conducted to investigate this possibility.

In [7] it is shown that spiking neurons are more powerful than conventional neurons. The unexpected success of the one-layered SNN on the temporal version of the XOR benchmark emphasizes this. New research even indicates that one spiking neuron with sufficient delayed axons can represent all possible mappings of a finite set of spike trains. Nevertheless, most of the solutions will use hairtrigger situations, like that for the XOR benchmark, and are thus not very suitable for networks of noisy neurons, and in addition can not be used for real-world applications which are inherently noisy.

Acknowledgments

We would like to thank Sander Bohte for a critical reading of the manuscript and helpful suggestions. Research was carried out for the partial fulfillment of the Masters of Science degree at the Intelligent Sensory Information Systems, University of Amsterdam.

References

[1] S. M. Bohte, J. N. Kok, and J. A. La Poutré. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.

[2] O. Booij. Temporal pattern classification using spiking neural networks. Master's thesis, Intelligent Sensory Information Systems, University of Amsterdam, August 2004. Available from http://www.xs4all.nl/˜obooij/study.

[3] W. Gerstner and W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[4] D. Heeger. Poisson model of spike generation, 2000. Available from http://www.cns.nyu.edu/~david/ftp/handouts/poisson.pdf.

[5] J. J. Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376:33–36, 1995.

[6] J.J. Hopfield and C.D Brody. What is a moment? transient synchrony as a collective mechanism for spatiotemporal integration. *PNAS*, 98(3):1282–1287, 2001.

[7] W. Maass. Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. In M. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems, volume 9*, pages 211–217. MIT Press, Cambridge, MA, 1997.

[8] W. Maass and C. M. Bishop, editors. *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.

[9] J. L. McClelland, D. E. Rumelhart, and the PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*. MIT Press, Cambridge, MA, 1986.

[10] S. M. Moore. Backpropagation in spiking neural networks. Master's thesis, University of Bath, 2002. Available from http://www.simonchristianmoore.co.uk/back.htm.

[11] T. Natschläger and W. Maass. Information dynamics and emergent computation in recurrent circuits of spiking neurons. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[12] T. Natschläger and B. Ruf. Spatial and temporal pattern analysis via spiking neurons. *Network: Computation in Neural Systems*, 9(2):319–332, 1998.

[13] B. Ruf and M. Schmitt. Hebbian learning in networks of spiking neurons using temporal coding. In J. Mira, R. Moreno-Diaz, and J. Cabestany, editors, *Biological and artificial computation: From neuroscience to technology*, pages 380–389. Springer, Berlin, 1997. Volume 1240 of Lecture Notes in Computer Science.

[14] B. Schrauwen and J. Van Campenhout. Extending spikeprop. In *Proceedings of the International Joint Conference on Neural Networks*, pages 471–476, Budapest, July 2004.

[15] J. Xin and M. J. Embrechts. Supervised learning with spiking neuron networks. In *Proceedings IEEE International Joint Conference on Neural Networks, IJCNN01*, Washington D.C., July 2001.