ELSEVIER

Neurocomputing 48 (2002) 17–37

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Error-backpropagation in temporally encoded networks of spiking neurons

Sander M. Bohte[a,*], Joost N. Kok[a,c], Han La Poutré[a,b]

[a] *CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands*
[b] *School of Technology Management, Eindhoven University of Technology, The Netherlands*
[c] *LIACS, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands*

Received 27 October 2000; accepted 6 June 2001

Abstract

For a network of spiking neurons that encodes information in the timing of individual spike times, we derive a supervised learning rule, *SpikeProp*, akin to traditional error-backpropagation. With this algorithm, we demonstrate how networks of spiking neurons with biologically reasonable action potentials can perform complex non-linear classification in fast temporal coding just as well as rate-coded networks. We perform experiments for the classical XOR problem, when posed in a temporal setting, as well as for a number of other benchmark datasets. Comparing the (implicit) number of spiking neurons required for the encoding of the interpolated XOR problem, the trained networks demonstrate that temporal coding is a viable code for fast neural information processing, and as such requires less neurons than instantaneous rate-coding. Furthermore, we find that reliable temporal computation in the spiking networks was only accomplished when using spike response functions with a time constant longer than the coding interval, as has been predicted by theoretical considerations.

Keywords: Spiking neurons; Temporal coding; Error-backpropagation

1. Introduction

Due to its success in artificial neural networks, the sigmoidal neuron is reputed to be a successful model of biological neuronal behavior. By modeling the rate at

* Corresponding author.
E-mail addresses: s.m.bohte@cwi.nl (S.M. Bohte), joost@liacs.nl (J.N. Kok), han.la.poutre@cwi.nl (H. La Poutré).

0925-2312/02/$ - see front matter
PII: S0925-2312(01)00658-0

which a single biological neuron discharges action potentials (spikes) as a monotonically increasing function of input-match, many useful applications of artificial neural networks have been build [22,7,37,34] and substantial theoretical insights in the behavior of connectionist structures have been obtained [41,27].

However, the spiking nature of biological neurons has recently led to explorations of the computational power associated with temporal information coding in single spikes [32,21,13,26,20,17,49]. In [29] it was proven that networks of spiking neurons can simulate arbitrary feedforward sigmoidal neural nets and can thus approximate any continuous function, and that neurons that convey information by individual spike times are computationally more powerful than neurons with sigmoidal activation functions [30].

As spikes can be described by "event" coordinates (place, time) and the number of active (spiking) neurons is typically sparse, artificial spiking neural networks allow for efficient implementations of large neural networks [48,33]. Single-spike-time computing has also been suggested as a new paradigm for VLSI neural network implementations [28] which would offer a significant speed-up.

Network architectures based on spiking neurons that encode information in the individual spike times have yielded, amongst others, a self-organizing map akin to Kohonen's SOM [39] and networks for unsupervised clustering [35,8]. The principle of coding input intensity by relative firing time has also been successfully applied to a network for character recognition [10]. For applications of temporally encoded spiking neural networks however, no practical supervised algorithm has been developed so far. Even in [10] the authors resort to traditional sigmoidal backpropagation to learn to discriminate the histograms of different spike-time responses.

To enable useful supervised learning with the temporal coding paradigm, we develop a learning algorithm for single spikes that keeps the advantages of spiking neurons while allowing for at least equally powerful learning as in sigmoidal neural networks. We derive an error-backpropagation-based supervised learning algorithm for networks of spiking neurons that transfer the information in the timing of a single spike. The method we use is analogous to the derivation by Rumelhart et al. [40]. To overcome the discontinuous nature of spiking neurons, we approximate the thresholding function. We show that the algorithm is capable of learning complex non-linear tasks in spiking neural networks with similar accuracy as traditional sigmoidal neural networks. This is demonstrated experimentally for the classical XOR classification task, as well as for a number of real-world datasets.

We believe that our results are also of interest to the broader connectionist community, as the possibility of coding information in spike times has been receiving considerable attention. In particular, we demonstrate empirically that networks of biologically reasonable spiking neurons can perform complex non-linear classification in a fast temporal encoding just as well as rate-coded networks. Although this paper primarily describes a learning rule applicable to a class of artificial neural networks, spiking neurons are significantly closer to biological neurons than sigmoidal ones. For computing with a rate-code on a very short time scale, it is generally believed that in biological neural systems the responses of a large num-

ber of spiking neurons are pooled to obtain an instantaneous measure of average firing rate. Although expensive in neurons, such a pooled response has the advantage that it is robust *because* of the large number of participating neurons. When comparing such an instantaneous rate-code to temporal coding with single spikes, it is well known that significantly less spiking neurons are required, albeit at the cost of robustness. In this paper, we present, to the best of our knowledge, the first spiking neural network that is trainable in a supervised manner and as such demonstrates the viability and efficiency of a functional spiking neural network as a function approximator.

We also present results that support the prediction that the length of the rising segment of the post-synaptic potential needs to be longer than the relevant temporal structure in order to allow for reliable temporal computation [28]. For spiking neurons, the post-synaptic potential describes the dynamics of a spike impinging onto a neuron, and is typically modeled as the difference of two exponentially decaying functions [19]. The effective rise and decay time of such a function is modeled after the membrane potential time constants of biological neurons. As noted, from a computational point of view, our findings support the theoretical predictions in [28]. From a biological perspective, these findings counter the common opinion among neuroscientists that fine temporal processing in spiking neural networks is prohibited by the relatively long time constants of cortical neurons (as noted for example in [15]).

The rest of this paper is organized as follows: in Section 2 the spiking neural network is formally defined. In Section 3, we derive the error-backpropagation algorithm. In Section 4 we test our algorithm on the classical XOR example, and we also study the learning behavior of the algorithm. By encoding real-valued input dimensions into a temporal code by means of receptive fields, we show results for a number of other benchmark problems in Section 5. The results of the experiments are discussed in Section 6. In a separate subsection, we consider the relevance of our findings for biological systems.

2. A network of spiking neurons

The network architecture consists of a feedforward network of spiking neurons with multiple delayed synaptic terminals (Fig. 1, as described by Natschläger and Ruf [35]). The neurons in the network generate action potentials, or spikes, when the internal neuron state variable, called "membrane potential", crosses a threshold ϑ. The relationship between input spikes and the internal state variable is described by the spike response model (SRM), as introduced by Gerstner [18]. Depending on the choice of suitable spike-response functions, one can adapt this model to reflect the dynamics of a large variety of different spiking neurons.

Formally, a neuron j, having a set Γ_j of immediate predecessors ("pre-synaptic neurons"), receives a set of spikes with firing times t_i, $i \in \Gamma_j$. Any neuron generates at most one spike during the simulation interval, and fires when the internal state variable reaches a threshold ϑ. The dynamics of the internal state variable

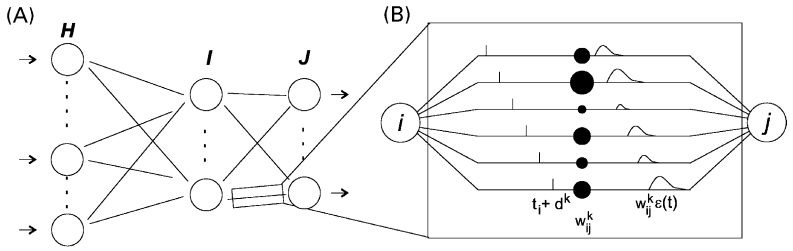

Fig. 1. (A). Feedforward spiking neural network, (B) connection consisting of multiple delayed synaptic terminals.

$x_j(t)$ are determined by the impinging spikes, whose impact is described by the spike-response function $\varepsilon(t)$ weighted by the synaptic efficacy ("weight") w_{ij}:

$$x_j(t) = \sum_{i \in \Gamma_j} w_{ij} \varepsilon(t - t_i). \tag{1}$$

The spike-response function in (1) effectively models the unweighted post-synaptic potential (PSP) of a single spike impinging on a neuron. The height of the PSP is modulated by the synaptic weight w_{ij} to obtain the effictive post-synaptic potential. The spike-response function as used in our experiments is defined in (3).

In the network as introduced in [35], an individual connection consists of a fixed number of m synaptic terminals, where each terminal serves as a sub-connection that is associated with a different delay and weight (Fig. 1B). The delay d^k of a synaptic terminal k is defined by the difference between the firing time of the pre-synaptic neuron, and the time the post-synaptic potential starts rising (Fig. 2A). We describe a pre-synaptic spike at a synaptic terminal k as a PSP of standard height with delay d^k. The unweighted contribution of a single synaptic terminal to the state variable is then given by

$$y_i^k(t) = \varepsilon(t - t_i - d^k) \tag{2}$$

with $\varepsilon(t)$ a spike-response function shaping a PSP, with $\varepsilon(t) = 0$ for $t < 0$. The time t_i is the firing time of pre-synaptic neuron i, and d^k the delay associated with the synaptic terminal k. The spike-response function describing a standard PSP is of the form

$$\varepsilon(t) = \frac{t}{\tau} e^{1 - t/\tau} \tag{3}$$

modeling a simple α-function for $t > 0$ (else 0), and τ models the membrane potential decay time constant that determines the rise and decay time of the PSP.

Extending (1) to include multiple synapses per connection and inserting (2), the state variable x_j of neuron j receiving input from all neurons i can then be described as the weighted sum of the pre-synaptic contributions

$$x_j(t) = \sum_{i \in \Gamma_j} \sum_{k=1}^{m} w_{ij}^k y_i^k(t), \tag{4}$$

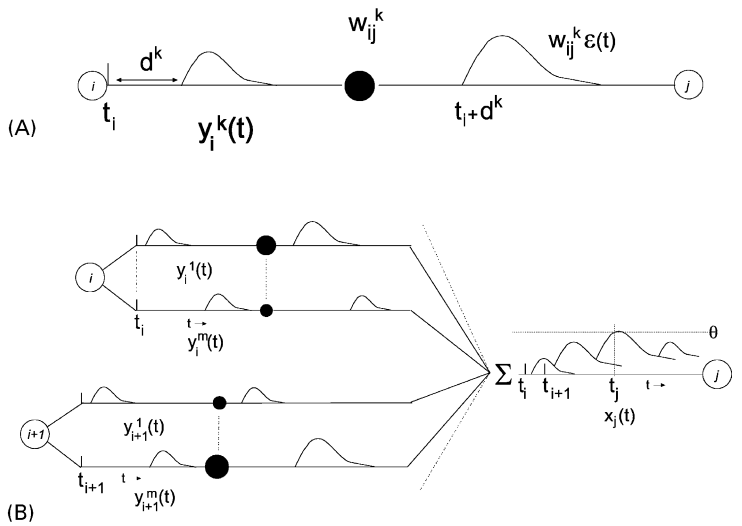

Fig. 2. Connections in a feedforward spiking neural network: (A) single synaptic terminal: the delayed pre-synaptic potential is weighted by the synaptic efficacy to obtain the post-synaptic potential; (B) two multi-synapse connections. Weighted input is summed at the target neuron.

where w_{ij}^k denotes the weight associated with synaptic terminal k (Fig. 2B). The firing time t_j of neuron j is determined as the first time when the state variable crosses the threshold ϑ: $x_j(t) \geqslant \vartheta$. Thus, the firing time t_j is a non-linear function of the state variable x_j: $t_j = t_j(x_j)$. The threshold ϑ is constant and equal for all neurons in the network.

3. Error-backpropagation

We derive error-backpropagation, analogous to the derivation by Rumelhart et al. [40]. Equations are derived for a fully connected feedforward network with layers labeled H(input), I(hidden) and J(output), where the resulting algorithm applies equally well to networks with more hidden layers.

The target of the algorithm is to learn a set of target firing times, denoted $\{t_j^{\mathrm{d}}\}$, at the output neurons $j \in J$ for a given set of input patterns $\{P[t_1 \ldots t_h]\}$, where $P[t_1 \ldots t_h]$ defines a single input pattern described by single spike times for each neuron $h \in H$. We choose as the error-function the least mean squares error-function, but other choices like entropy are also possible. Given desired spike times $\{t_j^{\mathrm{d}}\}$ and actual firing times $\{t_j^{\mathrm{a}}\}$, this error-function is defined by

$$E = \frac{1}{2} \sum_{j \in J} (t_j^{\mathrm{a}} - t_j^{\mathrm{d}})^2. \tag{5}$$

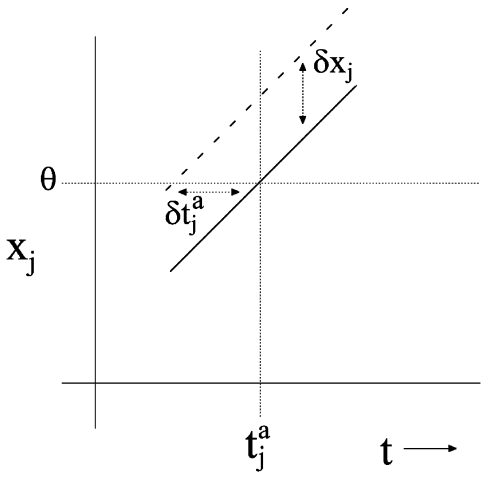

Fig. 3. Relationship between δx_j and δt_j for an ε space around t_j.

For error-backpropagation, we treat each synaptic terminal as a separate connection k with weight w_{ij}^k. Hence, for a backprop rule, we need to calculate

$$\Delta w_{ij}^k = -\eta \frac{\partial E}{\partial w_{ij}^k} \tag{6}$$

with η the learning rate and w_{ij}^k the weight of connection k from neuron i to neuron j. As t_j is a function of x_j, which depends on the weights w_{ij}^k, the derivative on the right-hand part of (6) can be expanded to

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial t_j}(t_j^a)\frac{\partial t_j}{\partial w_{ij}^k}(t_j^a) = \frac{\partial E}{\partial t_j}(t_j^a)\frac{\partial t_j}{\partial x_j(t)}(t_j^a)\frac{\partial x_j(t)}{\partial w_{ij}^k}(t_j^a). \tag{7}$$

In the last two factors on the right, we express t_j as a function of the thresholded post-synaptic input $x_j(t)$ around $t = t_j^a$. We assume that for a small enough region around $t = t_j^a$, the function x_j can be approximated by a linear function of t, as depicted in Fig. 3. For such a small region, we approximate the threshold function $\delta t_j(x_j) = -\delta x_j(t_j)/\alpha$, with $\partial t_j/\partial x_j(t)$ the derivative of the inverse function of $x_j(t)$. The value α equals the local derivative of $x_j(t)$ with respect to t, that is $\alpha = \partial x_j(t)/\partial t(t_j^a)$.

The second factor in (7) evaluates to

$$\frac{\partial t_j}{\partial x_j(t)}(t_j^a) = \left.\frac{\partial t_j(x_j)}{\partial x_j(t)}\right|_{x_j=\vartheta} = \frac{-1}{\alpha} = \frac{-1}{\partial x_j(t)/\partial t(t_j^a)} = \frac{-1}{\sum_{i,l} w_{ij}^l(\partial y_i^l(t)/\partial t)(t_j^a)}. \tag{8}$$

In further calculations, we will write terms like $\partial x_j(t_j^a)/\partial t_j^a$ for $\partial x_j(t)/\partial t(t_j^a)$.

We remark that this approximation only holds when the weights to a neuron are not altered such that the membrane potential no longer reaches threshold, and the neuron hence no longer fires. This is a potential problem, but can be countered by encoding input into the network in such a way that early spikes are automatically "more important" than later spikes. The encoding we outline in Section 5 is consistent with such spike-time evolution and should in general alleviate the problem: in our experiments it proved an effective solution. Note however that once a neuron no longer fires for *any* input pattern, there is no mechanism to "prop-up" the weights again. In our experiments, we set the initial weights such that each neuron in the network responded to at least part of the input pattern. With this additional provision, we did not experience any problems with "silent" neurons.

Note also that the approximation might imply that for large learning rates the algorithm can be less effective. We will consider this issue in the application of the algorithm in Section 4.1.

The first factor in (7), the derivative of E with respect to t_j, is simply

$$\frac{\partial E(t_j^a)}{\partial t_j^a} = (t_j^a - t_j^d). \tag{9}$$

We have

$$\frac{\partial x_j(t_j^a)}{\partial w_{ij}^k} = \frac{\partial\{\sum_{n\in\Gamma_j}\sum_l w_{nj}^l y_n^l(t_j^a)\}}{\partial w_{ij}^k} = y_i^k(t_j^a). \tag{10}$$

When we combine these results, (6) evaluates to

$$\Delta w_{ij}^k(t_j^a) = -\eta\frac{y_i^k(t_j^a)(t_j^d - t_j^a)}{\sum_{i\in\Gamma_j}\sum_l w_{ij}^l(\partial y_i^l(t_j^a)/\partial t_j^a)}. \tag{11}$$

For convenience, we define δ_j:

$$\delta_j \equiv \frac{\partial E}{\partial t_j^a}\,\frac{\partial t_j^a}{\partial x_j(t_j^a)} = \frac{(t_j^d - t_j^a)}{\sum_{i\in\Gamma_j}\sum_l w_{ij}^l(\partial y_i^l(t_j^a)/\partial t_j^a)} \tag{12}$$

and (7) can now be expressed as

$$\frac{\partial E}{\partial w_{ij}^k} = y_i^k(t_j^a)\frac{\partial E}{\partial t_j^a}\,\frac{\partial t_j^a}{\partial x_j(t_j^a)} = y_i^k(t_j^a)\delta_j \tag{13}$$

yielding

$$\Delta w_{ij}^k = -\eta y_i^k(t_j^a)\delta_j. \tag{14}$$

Eqs. (12) and (14) yield the basic weight adaptation function for neurons in the output layer.

We continue with the hidden layers: for error-backpropagation in other layers than the output layer, the generalized delta error in layer I is defined for $i \in I$ with actual firing times t_i^a:

$$\begin{aligned}\delta_i &\equiv \frac{\partial t_i^a}{\partial x_i(t_i^a)} \frac{\partial E}{\partial t_i^a} \\ &= \frac{\partial t_i^a}{\partial x_i(t_i^a)} \sum_{j \in \Gamma^i} \frac{\partial E}{\partial t_j^a} \frac{\partial t_j^a}{\partial x_j(t_j^a)} \frac{\partial x_j(t_j^a)}{\partial t_i^a} \\ &= \frac{\partial t_i^a}{\partial x_i(t_i^a)} \sum_{j \in \Gamma^i} \delta_j \frac{\partial x_j(t_j^a)}{\partial t_i^a}, \end{aligned} \tag{15}$$

where Γ^i denotes the set of immediate neural successors in layer J connected to neuron i.

As in [7], in (15) we expand the local error $\partial E(t_i^a)/\partial t_i^a$ in terms of the weighted error contributed to the subsequent layer J. For the expansion, the same chain rule as in (7) is used under the same restrictions, albeit for $t = t_i$.

The term $\partial t_i^a / \partial x_i(t_i^a)$ has been calculated in (8), while for $\partial x_j(t_j^a)/\partial t_i^a$:

$$\begin{aligned}\frac{\partial x_j(t_j^a)}{\partial t_i^a} &= \frac{\partial \sum_{l \in I} \sum_k w_{lj}^k y_l^k(t_j^a)}{\partial t_i^a} \\ &= \sum_k w_{ij}^k \frac{\partial y_i^k(t_j^a)}{\partial t_i^a}. \end{aligned} \tag{16}$$

Hence

$$\delta_i = \frac{\sum_{j \in \Gamma^i} \delta_j \{\sum_k w_{ij}^k (\partial y_i^k(t_j^a)/\partial t_i^a)\}}{\sum h \in \Gamma_i \sum_l w_{hi}^l (\partial y_h^l(t_i^a)/\partial t_i^a)}. \tag{17}$$

Thus, for a hidden layer and by (14)–(17), the weight adaptation rule compounds to

$$\Delta w_{hi}^k = -\eta y_h^k(t_i^a)\delta_i = -\eta \frac{y_h^k(t_i^a) \sum_j \{\delta_j \sum_k w_{ij}^k (\partial y_i^k(t_j^a)/\partial t_i^a)\}}{\sum_{n \in \Gamma_i} \sum_l w_{ni}^l (\partial y_n^l(t_i^a)/\partial t_i^a)}. \tag{18}$$

Analogous to traditional error-backpropagation algorithms, the weight adaptation rule (18) above generalizes to a network with multiple hidden layers I numbered $J-1,\ldots,2$ by calculating the delta-error at layer i from the delta-error in layer $i+1$, in effect back-propagating the error.

The algorithm derived above, termed *SpikeProp*, is summarized in the following table:

SpikeProp algorithm
Calculate δ_j for all outputs according to (12)
For each subsequent layer $I = J-1 \ldots 2$ Calculate δ_i for all neurons in I according to (17)
For output layer J, adapt w_{ij}^k by $\Delta w_{ij}^k = -\eta y_i^k(t_j)\delta_j$ (14) For each subsequent layer $I = J-1 \ldots 2$ Adapt w_{hi}^k by $\Delta w_{ij}^k = -\eta y_h^k(t_i)\delta_i$ (18)

4. The XOR-problem

In this section, we will apply the *SpikeProp* algorithm to the XOR problem. The XOR function is a classical example of a non-linear problem that requires hidden units to transform the input into the desired output.

To encode the XOR function in spike-time patterns, we associate a 0 with a "late" firing time and a 1 with an "early" firing time. With specific values 0 and 6 for the respective input times, we use the following temporally encoded XOR:

Input patterns			Output patterns
0	0	→	16
0	6	→	10
6	0	→	10
6	6	→	16

The numbers in the table represent spike times, say in ms. We use a third (bias) input neuron in our network that always fired at $t = 0$ to designate the reference start time (otherwise the problem becomes trivial). We define the difference between the times equivalent with "0" and "1" as the coding interval ΔT, which in this example corresponds to 6 ms.

For the network we use the feed-forward network architecture described in Section 2. The connections have a delay interval of 15 ms; hence the available synaptic delays are from 1 to 16 ms. The PSP is defined by an α-function as in (3) with a decay time $\tau = 7$ ms. Larger values up to at least 15 ms result in similar learning (see Section 4.1).

The network was composed of three input neurons (2 coding neurons and 1 reference neuron), 5 hidden neurons (of which one inhibitory neuron generating only negative sign PSPs) and 1 output neuron. Only positive weights were allowed.

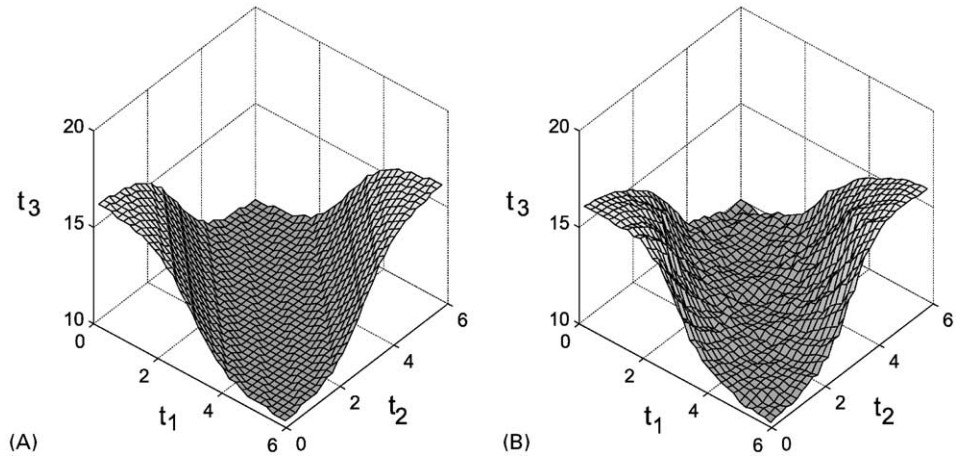

Fig. 4. Interpolated XOR function $f(t_1,t_2):[0,6]^2 \rightarrow [10,16]$: (A) target function; (B) network output after training. The network reached the sum squared error-criterion of 50.0 after learning 12996 randomly drawn examples from the 961 data points.

With this configuration, the network reliably learned the XOR pattern within 250 cycles with $\eta = 0.01$. In order to "learn" XOR, $16 \times 3 \times 5 + 16 \times 5 \times 1 = 320$ individual weights had to be adjusted.

While using the algorithm, we found that it was necessary to explicitly incorporate inhibitory and excitatory neurons, with inhibitory and excitatory neurons defined by generating, respectively, negative and positive PSPs using only positive weights. In fact, the *SpikeProp* algorithm would not converge if the connections were allowed to contain a mix of both positive and negative weights. We suspect that the cause of this problem lies in the fact that in the case of mixed weights, the effect of a single connection onto the target neuron is no longer a monotonically increasing function (as it is for sufficiently large time constants, see also the discussion in Section 4.1). We remark that the introduction of inhibitory and excitatory neurons is not a limitation: by expanding the network in such a way that each excitatory neuron has an inhibitory counterpart, in effect a mixed sign connection is implemented. In the experiments though, the inclusion of one or two inhibitory neurons was sufficient to enable learning.

We also tested the network on an interpolated XOR function $f(x_1,x_2) : [0,1]^2 \rightarrow [0,1]$, like in [31]. We translate this function to spike times $f(t_1,t_2)$: $[0,6]^2 \rightarrow t_3 : [10,16]$, with times t_1, t_2 and t_3 in ms (Fig. 4A). Using 3 input, 5 hidden and 1 output neurons, 961 input–output pairs of this function were presented to the network. The result of learning with *SpikeProp* is shown in Fig. 4B. The network can learn the presented input with an accuracy of the order of the internal integration time step of the *SpikeProp* algorithm: 0.1 ms (for the target sum squared error of 50.0 the average error per instance was 0.2 ms).

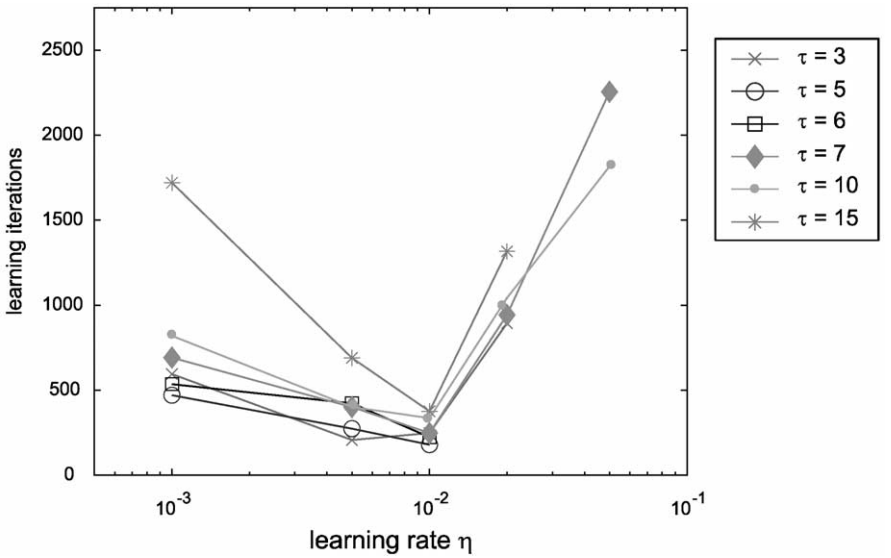

Fig. 5. Learning XOR: average number of required learning iterations to reach the sum squared error target (SSE) of 1.0. The average was calculated for those runs that converged.

4.1. Error gradient and learning rate

In this section, we consider the influence of the learning rate η and the time constant τ on the learning capabilities of the *SpikeProp* algorithm in a network of spiking neurons.

As noted in Section 3, the approximation of the dependence of the firing time t_j^a on the post-synaptic input x_j is only valid for a small region around t_j^a. We found indeed that for larger learning rates the probability of convergence decreased, although for learning rates up to 0.01, larger learning rates were associated with faster learning times. This can be seen in Fig. 5, where the average number of learning iterations required for the XOR function are plotted for a number of time constants τ.

In Fig. 6, the reliability of learning for different values of the time constant τ is plotted. The plot shows that for optimal convergence, the most reliable results are obtained for values of the time constant that are (somewhat) larger than the time interval ΔT in which the XOR problem is encoded (here: $\Delta T = 6$).

The convergence graphs for different values of the coding interval ΔT are plotted in Fig. 7A and B and show the same pattern. These results confirm the results as obtained by Maass in [28], where it was shown theoretically that the time constant τ needs to be larger than the relevant coding interval ΔT. This observation can also be made for the results presented in Section 5: a substantial speedup and somewhat better results were obtained if the time constant τ was slightly larger than the coding interval.

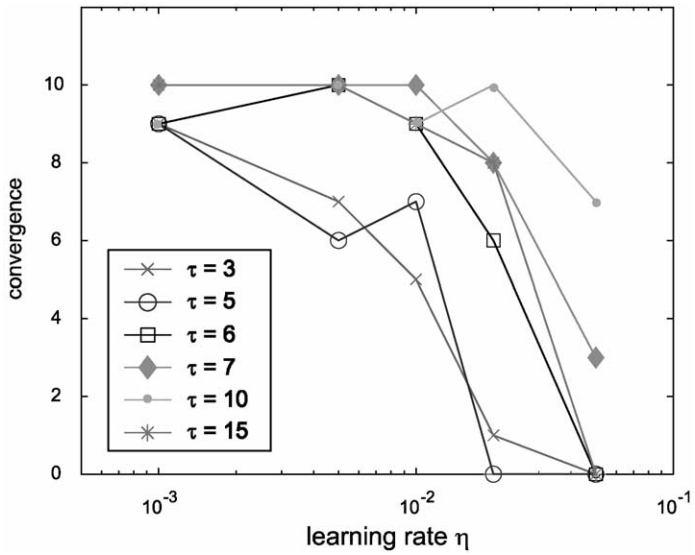

Fig. 6. Learning XOR: number of runs out of 10 that converged.

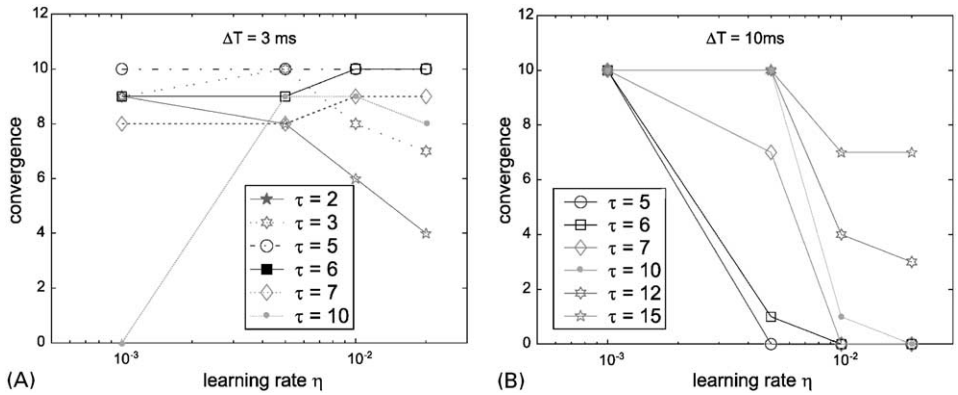

Fig. 7. Number of runs out of 10 that converged for different values of τ: (A) for a coding interval $t = 0 \ldots 3$, $[0,3]^2 \rightarrow [7,10]$. Target was an SSE of 0.7; (B) for a coding interval $t = 0 \ldots 10$, $[0,10]^2 \rightarrow [15,25]$. Target was an SSE of 3.0.

5. Other benchmark problems

In this section, we perform experiments with the *SpikeProp* algorithm on a number of standard benchmark problems: the Iris dataset, the Wisconsin breast-cancer dataset and the Statlog Landsat dataset.

First however, we introduce a method for encoding input variables into temporal spike-time patterns by population coding. We are not aware of any previous encod-

ing methods for transforming real-world data into spike-time patterns and therefore describe the method in detail.

5.1. Encoding continuous input variables in spike times

When using larger datasets, a critical factor becomes the encoding of the input: as the coding interval ΔT is restricted to a fixed interval of the order of τ, the full range of the input can be encoded by either using increasingly small temporal differences or alternatively by distributing the input over multiple neurons. In our network simulation, the network state is changed with a fixed time step. Hence an increased temporal resolution for some input values imposes a computational penalty. Keeping to the biologically inspired approach, we remark that cortical spike times are believed to be inherently noisy as they exhibit a "jitter" in their spike times of the order of 1–2 ms [32]. Hence an encoding that requires single spikes to have a higher temporal precision is implausible.

These arguments favor population coding, and we employed an encoding based on arrays of *receptive fields*. This enables the representation of continuously valued input variables by a population of neurons with graded and overlapping sensitivity profiles, such as Gaussian activation functions (the receptive fields, RFs). This encoding constitutes a biologically plausible and well studied method for representing real-valued parameters [17,4,52,51,45,44]. To encode values into a temporal pattern, it is sufficient to associate highly stimulated neurons with early firing times and less stimulated neurons with later (or no) firing times, and then the extension to temporal coding is straightforward.

We independently encode the respective input variables: each input dimension is encoded by an array of one-dimensional receptive fields. Improved representation accuracy for a particular variable can then be obtained by sharpening the receptive fields and increasing the number of neurons [51]. Such an encoding has been shown to be statistically bias-free [4] and in the context of spike-time patterns it has been applied successfully to unsupervised learning problems [8].

In our experiments, we determined the input ranges of the data, and encoded each input variable with neurons covering the whole data range. For a range $[I^n_{\min},\ldots,I^n_{\max}]$ of a variable n, m neurons were used with Gaussian receptive fields. For a neuron i its center was set to $I^n_{\min}+(2i-3)/2\cdot\{I^n_{\max}-I^n_{\min}\}/(m-2)$ and width $\sigma=1/\beta\{I^n_{\max}-I^n_{\min}\}/(m-2)$. For β, values between 1.0 and 2.0 were tried, and for the experiments a value of 1.5 was used because it produced the best results. For each input pattern, the response values of the neurons encoding the respective variables were calculated, yielding $n\times m$ values between 0 and 1. These values were then converted to delay times, associating $t=0$ with a 1, and later times up to $t=10$ with lower responses. The resulting spike times were rounded to the nearest internal time step, and neurons with responses larger than $t=9$ were coded not to fire, as they are considered to be insufficiently excited. The encoding of a single input value a by receptive field population coding is depicted in Fig. 8.

Output classification was encoded according to a winner-take-all paradigm where the neuron coding for the respective class was designated an early firing time,

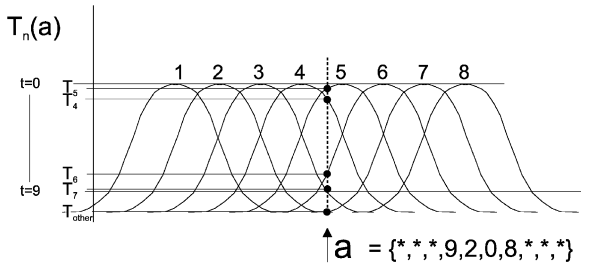

Fig. 8. Input value a encoded by 8 broadly tuned activation functions. The resulting encoding is marked by the black dots for a number of input neurons (T_i).

and all others a considerably later one, thus setting the coding interval ΔT. A classification was deemed to be correct if the neuron that fired earliest corresponded to the neuron required to fire first. To obtain a winner in the case where multiple neurons fired at the same time step, a first-order approximation to the real-value firing time was performed based on the current and previous membrane potentials.

The temporal encoding of input variables thus obtained has two important properties: by assigning firing times to only a small set of significantly activated neurons we achieve a sparse coding, allowing for an efficient event-based network simulation as in [14]. Also, by encoding each variable independently, we achieve a coarse coding where each variable can be encoded by an optimal number of neurons.

We tested our framework on several benchmark problems in which this temporal encoding is used for the conversion of the datasets to translate them into temporal patterns of discrete spikes.

5.2. Iris dataset

The Iris dataset is considered to be a reasonably simple classification problem. It contains three classes of which two are not linearly separable. As such, it provides a basic test of applicability of our framework. The dataset contains 150 cases, where each case has 4 input variables. Each input variable was encoded by 12 neurons with Gaussian receptive fields. The data was divided in two sets and classified using two-fold cross-validation. The results are presented in Table 1. We also obtained results on the same dataset from a sigmoidal neural network as implemented in Matlab v5.3, using the default training method (Levenberg–Marquardt, LM) and simple gradient descent (BP). The input presented to both networks is preprocessed in the same way, so both methods can be compared directly.

5.3. Wisconsin breast cancer dataset

The breast cancer diagnosis problem is described in [50]. The data is from the University of Wisconsin Hospitals and contains 699 case entries, divided into benign and malignant cases. Each case has nine measurements, and each measurement

Table 1

Algorithm	Inputs	Hidden	Outputs	Iterations	Accuracy	
					Training set	Test set
Iris dataset						
SpikeProp	50	10	3	1,000	$97.4\% \pm 0.1$	$96.1\% \pm 0.1$
Matlab BP	50	10	3	$2.6 \cdot 10^6$	$98.2\% \pm 0.9$	$95.5\% \pm 2.0$
Matlab LM	50	10	3	3,750	$99.0\% \pm 0.1$	$95.7\% \pm 0.1$
Wisconsin breast cancer dataset						
SpikeProp	64	15	2	1500	$97.6\% \pm 0.2$	$97.0\% \pm 0.6$
Matlab BP	64	15	2	$9.2 \cdot 10^6$	$98.1\% \pm 0.4$	$96.3\% \pm 0.6$
Matlab LM	64	15	2	3,500	$97.7\% \pm 0.3$	$96.7\% \pm 0.6$
Statlog Landsat set						
SpikeProp	101	25	6	60,000	$87.0\% \pm 0.5$	$85.3\% \pm 0.3$
Matlab LM	101	25	6	110,078	$83.6\% \pm 1.3$	$82.0\% \pm 1.5$
Matlab LM	101	25	6	1,108,750	$91.2\% \pm 0.5$	$88.0\% \pm 0.1$

The Iris dataset was split into two sets of 75 items each for cross-validation. The Matlab LM routine was run for 50 epochs, or 1500 iterations. The Wisconsin breast cancer dataset was split in two for cross-validation. For the Statlog Landsat, we trained for 25 epochs, or $100,000+$ iterations as well as for 250 epochs, or $1,000,000+$ iterations. All results are averaged over 10 runs on each cross-validation set, with the exception of the Landsat database where no cross-validation was used (as in the original Statlog experiment). The *SpikeProp* experiments were run with a coding interval of 4 ms and a time constant τ of 7 ms in order to reduce learning time. The learning rate η was set to 0.0075. Error-backpropagation in a 3-layer MLP was simulated with the Matlab v5.3 "TRAINLM" (LM) or "TRAINGD" (BP) routine.

is assigned an integer between 1 and 10, with larger numbers indicating a greater likelihood of malignancy. A small number of cases (16) contain missing data. In these cases, the neurons coding for the missing variable did not fire at all. For our experiment, we encoded each measurement with 7 equally spaced neurons covering the input range. The dataset was divided in two equal parts, and we used two-fold cross-validation. The results are summarized in Table 1. The results as we obtained them are on par with values reported in literature for BP learning on this dataset: for instance, in a benchmark study by Roy et al. [38] on an earlier version of this dataset containing 608 cases an error rate of 2.96% was obtained on the test set, using a multilayer perceptron with standard error-backpropagation learning.

5.4. Landsat dataset

To test the algorithm's performance on a larger dataset, we investigated the Landsat dataset as described in the StatLog survey of machine learning algorithms [34]. This dataset consists of a training set of 4435 cases and a test set of 2000 cases and contains 6 ground cover types (classes). For classification of a single pixel, each case contains the values of a 3×3 pixel patch, with each pixel described by 4 spectral bands, totaling 36 inputs per case. For the classification of the central

Table 2
Benchmark results on Landsat database from literature[a]

Method	Error training set (%)	Error test set (%)
Statlog data		
BackProp	88.8	86.1
RBF	88.9	87.9
k-NN	91.11	90.06

[a]Results on the k-nearest neighbor algorithm are given because it is the best classification method in the Statlog survey.

pixel, we averaged the respective spectral bands in the 3×3 patch and then encoded each separate band with 25 neurons. Results are shown in Table 1. The results obtained by the Statlog survey are summarized in Table 2.

Again, the results with *SpikeProp* are similar to the Matlab LM results, though even for twice the number of iterations, the Matlab LM results are somewhat worse. Compared to the best machine learning methods as reported in [34], the *SpikeProp* results are nearly identical to those reported for traditional BP, even though we reduced the input space 9-fold by taking as input the average of the 3×3 pixels in the environment. Note though that in [34] the multilayer perceptron BP results lag other ML methods for this particular classification problem. After extended learning however, the Matlab LM training method produces results that are significantly better than the reported values in [34] for MLP-BP learning. Such extended learning was not feasible for the *SpikeProp* algorithm due to time constraints, although the results show increasing classification accuracy with extended learning.

From the results, we conclude that the application of the *SpikeProp* algorithm on temporally encoded versions of benchmark problems yields similar results compared to those obtained from the Levenberg–Marquardt (LM) learning algorithm on sigmoidal neural networks. Typically, less learning epochs were required with *SpikeProp*, however, in a single epoch 16 times more weights were adjusted. The results reported for the default Matlab LM routine were comparable or better as compared to those reported in literature.

6. Discussion

We discuss two kinds of implications: the computational implications of the algorithm we presented, and some considerations regarding biological relevance.

6.1. Computational implications

The results obtained for the XOR problem show that our *SpikeProp* algorithm works reliably when used with smaller learning rates and time constants that are larger than the coding interval. The results demonstrate that in order to reliably learn patterns of temporal width ΔT, the time constant used to define the PSP is important: the time constant has to be larger than the pattern that is being learnt.

For machine learning purposes, the event nature of spikes allows for efficient implementations of spiking neural networks (e.g. [14,33]), especially in the case of sparse activity as achieved by the channel encoding (Section 5). Our current implementation of the algorithm is computationally somewhat intensive as we simulate the temporal evolution of the system with a large number of (small) time steps. Switching to an event-based system however should enhance system performance.

The number of learning iterations used in our simulation was comparable to those required for Levenberg–Marquardt learning in multi-layer-perceptrons (LM-MLP) on the same pre-processed datasets. We did not attempt much parameter tuning or convergence optimizations like adding a momentum term. The time for a single learning iteration was clearly longer, as in the simulations each connection consisted of 16 delayed terminals each with their own weight. Hence the number of weights that needed to be tuned in any of the experiments was quite large. Convergence in our network seemed to be more reliable compared to the Matlab experiments: in experiments with spiking neural networks on the real-world datasets, the *SpikeProp* algorithm always converged, whereas the comparative LM-MLP experiments occasionally failed ($< 10\%$). Research into optimal parameters would be a logical step for future investigations.

Given the explicit use of the time domain for calculations, we believe that a network of spiking neurons is intrinsically more suited for learning and evaluating temporal patterns than sigmoidal networks, as the spiking neural network is virtually time invariant in the absence of reference spikes. Applications of this type will be the subject of future research; a first application of the *SpikeProp* algorithm depending on this feature has been used in a hand-written-character recognition task. In this task, the spiking neural network was able to get better recognition rates for a subset of characters as compared to simple statistical methods [36].

6.2. Biological implications

Within the broader connectionist community, the real-valued output of a neuron is generally assumed to be its average firing rate. In spite of the success of this paradigm in neural network modeling and the substantial electrophysiological support, there has been an increasing number of reports where the actual timing of action potentials (spikes) carries significant information (e.g., in the bat [25], the barn owl [12] and the electric fish [23]). Indeed, a respectable amount of neurophysiological evidence suggests that such neuronal activation with millisecond precision could be linked with dynamic neurocognitive information processing [1,3,24,11,16].

Besides the reports on the precise firing times of individual neurons, the actual number of spikes available for neuronal processing has been estimated to be rather small in some cases: psychophysical experiments by Thorpe et al. [47] suggest that complex neuronal information processing can be achieved in under 10 ms per synaptic stage, or less than one spike per neuron.

Given these considerations, attention has been given to the possibility of (additional) information coding in the timing of individual action potentials [6,46,1].

This could more easily achieve fast information transfer [48,29] and would be cheaper in terms of neuron count. Biological evidence is emerging that suggests that delay-based temporal coding is actively employed in the cortex, e.g. [2,26,5,9].

Against the coding of information with a temporal code on very short time scales, it has been argued that this is unlikely due to the relatively long time constants of cortical neurons. The theoretical work by Maass [28] already refuted this notion (as also argued in [15]), but the results presented here demonstrate networks of spiking neurons with biologically plausible (relatively long) time constants capable of performing complex non-linear classification tasks. This was feasible for spike times encompassing a temporal coding interval up to this time constant. Indeed, the XOR results suggest that short time constants impede the integration of information over time periods longer than this value.

A temporal code for transmitting information between neurons is especially interesting as it has been shown theoretically that it is very efficient in terms of spiking neurons required in the case of fast processing of information [31]. Due to the all-or-nothing nature of the action potentials generated by individual neurons, a rate code on a time scale of 10 ms is only available to downstream neurons by deriving the instantaneous firing rate from a population of neurons activated by the same stimulus. This is problematic to achieve, as it has been noted that a reliable estimation of the instantaneous firing rate on a time scale of 10 ms requires on the order of 100 pooled neurons [31,42]. It has also been reported that pooling the activity of more than 100 neurons does not increase accuracy due to correlated input noise [43].

With our supervised learning algorithm, we demonstrated that efficient spiking neural networks based on temporal coding can be build and trained in the same way as traditional sigmoidal MLP networks. We have demonstrated a spiking neural network trained to approximate XOR that requires an order of magnitude less spiking neurons than networks based on instantaneous rate-coding, albeit with less robustness, as less neurons are involved. However, as neurons may come at considerable expense, this may be more desirable in many situations. We note however that other representational schemes may be devised that are equally efficient under these circumstances.

7. Conclusion

In this paper, we derived a learning rule for feedforward spiking neural networks by back-propagating the temporal error at the output. By linearizing the relationship between the post-synaptic input and the resultant spiking time, we were able to circumvent the discontinuity associated with thresholding. The result is a learning rule that works well for smaller learning rates and for time constants of the post-synaptic potential larger than the maximal temporal coding range. This latter result is in agreement with the theoretical predictions.

The algorithm also demonstrates in a direct way that networks of spiking neurons can carry out complex, non-linear tasks in a temporal code. As the

experiments indicate, the *SpikeProp* algorithm is able to perform correct classification on non-linearly separable datasets with accuracy comparable to traditional sigmoidal networks, albeit with potential room for improvement.

References

[1] M. Abeles, H. Bergman, E. Margalit, E. Vaadia, Spatiotemporal firing patterns in the frontal cortex of behaving monkeys, J. Neurophysiol. 70 (1993) 1629–1658.
[2] E. Ahissar, R. Sosnik, S. Haidarliu, Transformation from temporal to rate coding in a somatosensory thalamocortical pathway, Nature 406 (2000) 302–306.
[3] J-M. Alonso, W.M. Usrey, R.C. Reid, Precisely correlated firing in cells of the lateral geniculate nucleus, Nature 383 (1996) 815–819.
[4] P. Baldi, W. Heiligenberg, How sensory maps could enhance resolution through ordered arrangements of broadly tuned receivers, Biol. Cybern. 59 (1988) 313–318.
[5] Buo-qiang Bi, Mu-ming Poo, Distributed synaptic modification in neural networks induced by patterned stimulation, Nature 401 (1999) 792–796.
[6] W. Bialek, F. Rieke, R.R. de Ruyter van Steveninck, D. Warland, Reading a neural code, Science 252 (1991) 1854–1857.
[7] Ch.M. Bishop, Neural Networks for Pattern Recognition, Clarendon Press, Oxford, 1995.
[8] S.M. Bohte, J.N. Kok, H. La Poutré, Unsupervised classification in a layered network of spiking neurons, Proceedings of IJCNN'2000, Vol. IV, 2000, pp. 279–285.
[9] M. Brecht, W. Singer, A.K. Engel, Role of temporal codes for sensorimotor integration in the superior collicus, Proceedings of ENA'98, 1998.
[10] D.V. Buonomano, M. Merzenich, A neural network model of temporal code generation and position-invariant pattern recognition, Neural Comput. 11 (1) (1999) 103–116.
[11] G.T. Buracas, A. Zador, M.R. DeWeese, T.D. Albright, Efficient discrimination of temporal patterns by motion-sensitive neurons in primate visual cortex, Neuron 20 (1998) 959–969.
[12] C.E. Carr, M. Konishi, A circuit for detection of interaural time differences in the brain stem of the barn owl, J. Neurosci. 10 (1990) 3227–3246.
[13] G. Deco, B. Schurmann, The coding of information by spiking neurons: an analytical study, Network: Comput. Neural Systems 9 (1998) 303–317.
[14] A. Delorme, J. Gautrais, R. VanRullen, S.J. Thorpe, Spikenet: a simulator for modeling large networks of integrate and fire neurons, Neurocomputing 26–27 (1999) 989–996.
[15] M. Diesmann, M.-O. Gewaltig, A. Aertsen, Stable propagation of synchronous spiking in cortical neural networks, Nature 402 (1999) 529–533.
[16] A.K. Engel, P. König, A.K. Kreiter, T.B. Schillen, W. Singer, Temporal coding in the visual cortex: new vistas on integration in the nervous system, Trends Neurosci. 15 (1992) 218–226.
[17] C.W. Eurich, S.D. Wilke, Multidimensional encoding strategy of spiking neurons, Neural Comput. 12 (7) (2000) 1519–1529.
[18] W. Gerstner, Time structure of the activity in neural network models, Phys. Rev. E 51 (1995) 738–758.
[19] W. Gerstner, Spiking neurons, in: W. Maass, C. Bishop (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, MA, 1999.
[20] W. Gerstner, R. Kempter, J.L. van Hemmen, H. Wagner, A neuronal learning rule for sub-millisecond temporal coding, Nature 383 (1996) 76–78.
[21] W. Gerstner, R. Ritz, L. van Hemmen, Why spikes? hebbian learning and retrieval of time-resolved excitation patterns, Biol. Cybern. 69 (1993) 503–515.
[22] S. Haykin, Neural Networks, A Comprehensive Foundations, Maxwell Macmillan, 1994.
[23] W. Heiligenberg, Neural Nets in Electric Fish, MIT Press, Cambridge, MA, 1991.
[24] J. Heller, J.A. Hertz, T.W. Kjaer, B.J. Richmond, Information flow and temporal coding in primate pattern vision, J. Comp. Neurosci. 2 (3) (1995) 175–193.

[25] N. Kuwabara, N. Suga, Delay lines and amplitude selectivity are created in subthalamic auditory nuclei: the branchium of the inferior colliculus of the mustached bat, J. Neurophysiol. 69 (1993) 1713–1724.
[26] G. Laurent, A systems perspective on early olfactory coding, Science 286 (1999) 723–728.
[27] D.S. Levine, Introduction to Neural & Cognitive Modeling, Lawrence Erlbaum Associates, London, 1991.
[28] W. Maass, Lower bounds for the computational power of networks of spiking neurons, Neural Comput. 8 (1) (1996) 1–40.
[29] W. Maass, Fast sigmoidal networks via spiking neurons, Neural Comput. 9 (2) (1997) 279–304.
[30] W. Maass, Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons, in: M.C. Mozer, M.I. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems, Vol. 9, The MIT Press, Cambridge, MA, 1997, p. 211.
[31] W. Maass, Paradigms for computing with spiking neurons, in: L. van Hemmen (Ed.), Models of Neural Networks, Vol. 4, Springer, Berlin, 1999.
[32] W. Maass, C.M. Bishop, Pulsed Neural Networks, Vol. 1, MIT Press, Cambridge, MA, 1999.
[33] M. Mattia, P. Del Giudice, Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses, Neural Comput. 12 (10) (2000) 2305–2329.
[34] D. Michie, D.J. Spiegelhalter, C.C. Taylor (Eds.), Machine Learning, Neural and Statistical Classification, Ellis Horwood, West Sussex, 1994.
[35] T. Natschläger, B. Ruf, Spatial and temporal pattern analysis via spiking neurons, Network: Comp. Neural Systems 9 (3) (1998) 319–332.
[36] G. Pieri, Improvement in handwritten character recognition: mixtures of experts and temporal pattern recognition in spiking neural networks, M.Sc., Free University of Amsterdam, 2000.
[37] B.D. Ripley, Pattern Recognition and Neural Networks, Clarendon Press, Oxford, 1996.
[38] A. Roy, S. Govil, R. Miranda, An algorithm to generate radial basis function(rbf)-like nets for classification problems, Neural Networks 8 (2) (1995) 179–201.
[39] B. Ruf, M. Schmitt, Self-organization of spiking neurons using action potential timing, IEEE Trans. Neural Networks 9 (3) (1998) 575–578.
[40] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[41] D.E. Rumelhart, J.L. McClelland and the PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, MIT Press, Cambridge, MA, 1986.
[42] M.N. Shadlen, W.T. Newsome, Noise, neural codes and cortical organization, Curr. Opin. Neurobiol. 4 (1994) 569–579.
[43] M.N. Shadlen, W.T. Newsome, The variable discharge of cortical neurons: implications for connectivity, computation and information coding, J. Neurosci. 18 (1998) 3870–3896.
[44] H.P. Snippe, Parameter extraction from population codes: a critical assessment, Neural Comput. 8 (3) (1996) 511–529.
[45] H.P. Snippe, J.J. Koenderink, Information in channel-coded systems: correlated receivers, Biol. Cybern. 67 (2) (1992) 183–190.
[46] W.R. Softky, Simple codes versus efficient codes, Curr. Opin. Neurobiol. 5 (1995) 239–247.
[47] S.J. Thorpe, F. Fize, C. Marlot, Speed of processing in the human visual system, Nature 381 (1996) 520–522.
[48] S.J. Thorpe, J. Gautrais, Rapid visual processing using spike asynchrony, in: M.C. Mozer, M.I. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems, Vol. 9, The MIT Press, Cambridge, MA, 1997, p. 901.
[49] Ch. von der Malsburg, W. Schneider, A neural cocktail-party processor, Biol. Cybern. 54 (1986) 29–40.
[50] W.H. Wolberg, Cancer dataset obtained from Williams H. Wolberg from the University of Wisconsin Hospitals, Madison, 1991.
[51] K. Zhang, T.J. Sejnowski, Neuronal tuning: to sharpen or broaden?, Neural Comput. 11 (1) (1999) 75–84.

[52] K.-C. Zhang, I. Ginzburg, B.L. McNaughton, T.J. Sejnowski, Interpreting neuronal population activity by reconstruction: unified framework with application to hippocampal place cells, J. Neurophysiol. 79 (1998) 1017–1044.

Sander Bohte is a Ph.D student in the group of Han La Poutré at the Netherlands Centre for Mathematics and Computer Science (CWI). He obtained his M.Sc. in physics from the University of Amsterdam. His research interests include spiking neural networks, dynamic binding in connectionist networks, and emerging behavior in adaptive distributed systems.

Han La Poutré is professor at the Eindhoven University of Technology in the School of Technology Management, and head of the group Evolutionary Systems and Applied Algorithmics of CWI. He received his Ph.D. degree in Computer Science from Utrecht University. His research interests include neural networks, discrete algorithms, evolutionary systems, and agent-based computational economics.

Joost Kok is professor in Fundamental Computer Science at the Leiden Institute of Advanced Computer Science of Leiden University in the Netherlands. His research interests are coordination of software components, data mining and optimization.