LETTER Communicated by Sander M. Bohte

Supervised Learning in Multilayer Spiking Neural Networks

Ioana Sporea
i.nica@surrey.ac.uk
André Grüning
a.gruning@surrey.ac.uk
Department of Computing, University of Surrey, Guildford, GU2 7XH, U.K.

We introduce a supervised learning algorithm for multilayer spiking neural networks. The algorithm overcomes a limitation of existing learning algorithms: it can be applied to neurons firing multiple spikes in artificial neural networks with hidden layers. It can also, in principle, be used with any linearizable neuron model and allows different coding schemes of spike train patterns. The algorithm is applied successfully to classic linearly nonseparable benchmarks such as the XOR problem and the Iris data set, as well as to more complex classification and mapping problems. The algorithm has been successfully tested in the presence of noise, requires smaller networks than reservoir computing, and results in faster convergence than existing algorithms for similar tasks such as SpikeProp.

1 Introduction

Traditional rate-coded artificial neural networks represent an analog variable through the firing rate of the biological neuron. That is, the output of a computational unit is a representation of the firing rate of the biological neuron. In order to increase the computational power of the network, neurons are structured in successive layers of computational units. Such systems are trained to recognize input patterns by searching for a set of suitable connection weights. Learning rules based on gradient descent, such as backpropagation (Rumelhart, Hinton, & Williams, 1986), have led sigmoidal neural networks (networks that use the sigmoid as the activation function) to be one of the most powerful and flexible computational models.

However, experimental evidence suggests that neural systems use the exact time of single action potentials to encode information (Thorpe & Imbert, 1989; Johansson & Birznieks, 2004). Thorpe and Imbert (1989) argued that because of the speed of processing visual information and the anatomical structure of the visual system, processing has to be done on the basis of single spikes. Johansson and Birznieks (2004) showed that the relative timing of the first spike contains important information about tactile stimuli. Further evidence suggests that the precise temporal firing pattern of

groups of neurons conveys relevant sensory information (Wehr & Laurent, 1996; Neuenschwander & Singer, 1996; deCharms & Merzenich, 1996).

These findings have led to a new way of simulating neural networks based on temporal encoding of individual spikes (Maass, 1997a). Investigations of the computational power of spiking neurons have illustrated that realistic mathematical models of neurons can arbitrarily approximate any continuous function, and furthermore, it has been demonstrated that networks of spiking neurons are computationally more powerful than sigmoidal neurons (Maass, 1997b). Because of the nature of spiking neuron communication, these are also suited for VLSI implementation with significant speed advantages (Elias & Northmore, 2002).

In this letter, we present a new learning algorithm for feedforward spiking neural networks with multiple layers. The learning rule extends the ReSuMe algorithm (Ponulak & Kasiński, 2010) to multiple layers using backpropagation of the network error. The weights are updated according to STDP and anti-STDP processes, and unlike SpikeProp (Bohte, Kok, & Poutré, 2002), it can be applied to neurons firing multiple spikes in all layers. The multilayer ReSuMe is analog to the backpropagation learning algorithm for rate neurons, while making use of spiking neurons. To the best of our knowledge, this is the first learning algorithm for spiking neural networks with hidden layers where multiple spikes are considered in all layers and precise spike time encoding is used for both inputs and outputs.

The rest of the letter is organized as follows. In section 2, some of the existing supervised learning algorithms for spiking neurons are discussed. Section 3 contains a description of a generic spiking neuron model and the derivation of the learning rule based on this neuron model for a feedforward network with a hidden layer. In section 4, the weight modifications are discussed for a simplified network with a single output neuron. In section 5, the flexibility and power of feedforward spiking neural networks trained with multilayer ReSuMe are showcased by linearly nonseparable problems, as well as mapping and classification tasks. The spiking neural network is trained with spike timing patterns distributed over timescales in the range of tens to hundreds of milliseconds, comparable to the span of sensory and motor processing (Mauk & Buonomano, 2004). A discussion of the learning algorithm and a summary of the results are presented in section 6.

2 Background

While experimental studies have shown that supervised learning may be present in the brain, especially in sensorimotor networks and sensory systems (Knudsen, 1994, 2002), there are no definite conclusions regarding the means through which biological neurons learn. Several learning algorithms have been proposed to explore how spiking neurons may respond to given instructions.

One such algorithm, the tempotron learning rule, was introduced by Gütig and Sompolinsky (2006), where neurons learn to discriminate between spatiotemporal sequences of spike patterns. Although the learning rule uses a gradient-descent approach, it can only be applied to single-layered networks. The algorithm is used to train leaky integrate-and-fire neurons to distinguish between two classes of patterns by firing at least one action potential or by remaining quiescent. While the spiking neural network is able to successfully classify the spike-timing patterns, the neurons do not learn to respond with precise spike-timing patterns.

Another gradient-descent-based learning rule is the SpikeProp algorithm (Bohte et al., 2002) and its extensions (Schrauwen & van Campenhout, 2004; Xin & Embrechts, 2001; Tiňo & Mills, 2005). The algorithm is applied to feedforward networks of neurons firing a single spike and is minimizing the time difference between the target spike and the actual output spike. The learning algorithm uses spiking neurons modeled by the Spike Response Model (SRM) (Gerstner, 2001), and the derivations of the learning rule are based on the explicit dynamics of the neuron model. Although Booij and Nguyen (2005) and Ghosh-Dastidar and Adeli (2009) have extended the algorithm to allow neurons to fire multiple spikes in the input and hidden layers, only the first spike is considered in the output layer, subsequent spikes being ignored because the network error is represented by the time difference of the first spike of the output and target neurons. Because these extensions are based on the specific neuron dynamics, these learning rules are also limited to the specific SRM used.

Yet another gradient-descent learning algorithm was developed by McKennoch, Voegtlin, and Bushnell (2009) for feedforward networks of theta neurons, a canonical event-driven neuron model. However, like SpikeProp, this learning rule is derived using the neuron model dynamics and can be applied only to neuron models that can be mapped to the theta neuron model. Moreover, this algorithm is also limited to neurons firing single spikes, being applied to the same kind of task as SpikeProp and Tempotron. Finally Bohte (2011) presents a gradient-descent learning algorithm for multilayer networks where neurons act as tunable analog filters, and trains of multiple spikes derived through fractional derivatives of (an approximation of) the time course of a neuron's activation are used for information transmission between neurons (Bohte & Rombouts, 2010).

Some supervised learning algorithms are based on Hebb's postulate—"cells that fire together, wire together" (Hebb, 1949; Ruf & Schmitt, 1997; Legenstein, Naeger, & Maass, 2005; Ponulak & Kasiński, 2010). ReSuMe (Ponulak & Kasiński, 2010) is making use of both Hebbian learning and gradient-descent techniques. As the weight modifications are based on only the input and output spike trains and do not make any explicit assumptions about the neural or synaptic dependencies, the algorithm can be applied to various neuron models. However, the algorithm can be applied only to a single layer of neurons or used to train readouts for reservoir networks.

The ReSuMe algorithm has also been applied to neural networks with a hidden layer, where weights of downstream neurons are subject to multiplicative scaling (Grüning & Sporea, 2012). The simulations show that networks with one hidden layer can perform linearly nonseparable logical operations, while networks without hidden layers cannot. The ReSuMe algorithm has also been used to train the output layer in a feedforward network in Glackin, Maguire, McDaid, and Sayers (2011) and Wade, McDaid, Santos, and Sayers (2010), where the hidden layer acted as a frequency filter. However, input and target outputs here consisted of fixed-rate spike trains.

Approaches from a different angle include Rostro-Gonzalez, Vasquez-Betancour, Cessac, and Viéville (2010), who use a linear programming approach to estimate weights (and delays) in recurrent spiking networks based on LIF neurons and successfully attempt to reconstruct weights from a spike raster and initial conditions such as a (fixed) input spike train and initial states of membrane potentials.

Finally, despite some positive evidence (Knudsen, 1994, 2002), there is still a debate of whether supervised learning is taking place in nervous systems at all or whether reinforcement-style learning is more plausible. Urbanczik and Senn (2009) use a clever approach based on stochastic gradients to derive a reinforcement learning rule for populations of spiking neurons. Their network consists of ensembles of noise-escape neurons in a single layer and with an external critic. It is applied to classification tasks where inputs are spike-train-encoded; however, outputs are spike-rate or latency encoded, not making use of full spike-train patterns. No attempt is made to extend this behavior to networks with a hidden layer or to true spatiotemporal spike patterns as outputs. However, the difference between fully supervised and reinforcement learning schemes might be only a notional one, as Roelfsema and van Ooyen (2005) and Grüning (2007) demonstrate. Their work focuses on relating back propagation weight changes to reinforcement learning for multilayer networks of rate neurons in classification and times series prediction tasks; similar techniques can perhaps also to be applied to spiking neurons.

This letter introduces a new supervised learning algorithm that combines the quality of SpikeProp, spanning to multiple layers (Bohte et al., 2002), with the flexibility of ReSuMe, which can be used with multiple spikes and different neuron models (Ponulak & Kasiński, 2010).

3 Learning Algorithm

In this section, we describe the new learning algorithm for feedforward multilayer spiking neural networks. The learning rule is derived for networks with only one hidden layer, as the algorithm can be extended to networks with more hidden layers similarly. First, we give an alternative motivation of the ReSuMe learning rule. Ponulak and Kasiński (2010), in their original motivation of ReSuMe, simply invoke a spiking analog of the delta rule

as its starting point that was model free and could be shown to converge for single input, output, and target spikes (Ponulak, 2006). Our approach instead gives a more explicit relation between gradient descent and weight changes under ReSuMe. In our alternative formulation, the algorithm is then extended to networks with a hidden layer.

3.1 Neuron Model. The input and output signals of spiking neurons are represented by the timing of spikes. A spike train is defined as a sequence of impulses fired by a particular neuron at times t^f. Spike trains are formalized by a sum of Dirac δ functions (Gerstner & Kistler, 2002):

$$S(t) = \sum_f \delta(t - t^f). \tag{3.1}$$

In order to establish a relation between the input and output spike trains for a single neuron, we start from a linear stochastic neuron model in continuous time. The instantaneous firing rate $R_o(t)$ of a neuron o is the probability density of firing at time t and is determined by the instantaneous firing rates of its presynaptic neurons h,

$$R_o(t) = \frac{1}{n} \sum_{h \in H} w_{oh} R_h(t), \tag{3.2}$$

where n is the number of presynaptic neurons h. The weights w_{oh} represent the strength of the connection between the presynaptic neurons h and postsynaptic neuron o. Formally, this is similar to a linear rate-coded neuron run in continuous time and with a stochastic interpretation of R_o.

The instantaneous firing rate $R(t)$ is not a direct observable of the neuron. In a single run, we observe only a concrete spike train $S(t)$. However, $R(t)$ can be viewed as the expectation over concrete spike trains, laxly speaking averaged for an infinite number of trials (with a limit in an appropriate sense),

$$R(t) = \langle S(t) \rangle = \lim_{M \to \infty} \frac{1}{M} \sum_{j=1}^{M} S_j(t), \tag{3.3}$$

where M is the number of trials and $S_j(t)$ is the concrete spike train for each trial.

The instantaneous firing rate $R(t)$ will be used for deriving the learning algorithm due to its smoothness. However, it will subsequently be replaced at an appropriate point by an estimate for a single run, namely, the (discontinuous) spike train $S(t)$. This is a more elaborate and explicit procedure

than in Ponulak and Kasiński (2010) but is based on the same underlying ideas.

3.2 Backpropagation of the Network Error. The learning algorithm is derived for a fully connected feedforward network with one hidden layer. The input layer I is only providing the input patterns, without performing any computation on the patterns. The hidden and output layers are labeled H and O, respectively. All neurons in one layer are connected to all neurons in the subsequent layer.

The instantaneous network error is formally defined in terms of the difference between the actual instantaneous firing rate $R_o^a(t)$ and the target instantaneous firing rate $R_o^d(t)$ for all output neurons:

$$E(t) = E(R_o^a(t)) = \frac{1}{2}\sum_{o\in O}[R_o^a(t) - R_o^d(t)]^2. \tag{3.4}$$

In order to minimize the network error, the weights are modified using a process of gradient descent,

$$\Delta w_{oh}(t) = -\eta\frac{\partial E(R_o^a(t))}{\partial w_{oh}}, \tag{3.5}$$

where η is the learning rate and w_{oh} represents the weight between the output neuron o and hidden neuron h. $\Delta w_{oh}(t)$ is the weight change contribution due to the error $E(t)$ at time t, and the total weight change is $\Delta w = \int \Delta w(t)dt$ over the duration of the spike train. This is analogous to the starting point of standard backpropagation for rate neurons in discrete time. For simplicity, the learning rate will be considered $\eta = 1$ and will be suppressed in the following equations, as the step length of each learning iteration will be given by other learning parameters to be defined later. Also, in the following, derivatives are understood in a functional sense.

3.2.1 Weight Modifications for the Output Neurons. In this section we derive the weight-update formulated for the ReSuMe learning algorithm in an alternative way and connect with gradient-descent learning for spiking neurons. We will need this derivation as a first step to derive our extension of ReSuMe to subsequent layers in section 3.2.2. However, this derivation is also instructive in its own right as it works out more clearly than in the original derivation (Ponulak & Kasiński, 2010) how ReSuMe and gradient descent are connected. It also varies Ponulak's statement that ReSuMe can be applied to any neuron model. Here, this is the case if the neural model can, on an appropriate timescale, be approximated well enough with a linear neuron model (first-order approximation in a Taylor or Volterra series).

As the network error is a function of the output spike train, which in turn depends on the weight w_{oh}, the derivative of the error function can be expanded using the chain rule as follows:

$$\frac{\partial E(R_o^a(t))}{\partial w_{oh}} = \frac{\partial E(R_o^a(t))}{\partial R_o^a(t)} \frac{\partial R_o^a(t)}{\partial w_{oh}}. \tag{3.6}$$

The first term of the right-hand side of equation 3.6 can be calculated as

$$\frac{\partial E(R_o^a(t))}{\partial R_o^a(t)} = R_o^a(t) - R_o^d(t). \tag{3.7}$$

Since the instantaneous rate function is expressed in terms of the weight w_{oh} in equation 3.2, the second factor on the right-hand side of equation 3.6 becomes

$$\frac{\partial R_o(t)}{\partial w_{oh}} = \frac{1}{n_h} R_h(t), \tag{3.8}$$

where n_h is the number of hidden neurons. When equations 3.5 to 3.8 are combined, the formula for weight modifications to the output neurons becomes

$$\Delta w_{oh}(t) = -\frac{1}{n_h}[R_o^a(t) - R_o^d(t)]R_h(t). \tag{3.9}$$

For convenience, we define the backpropagated error $\delta_o(t)$ for the output neuron o:

$$\delta_o(t) := \frac{1}{n_h}[R_o^d(t) - R_o^a(t)]; \tag{3.10}$$

hence:

$$\Delta w_{oh}(t) = \delta_o(t) R_h(t). \tag{3.11}$$

This is similar to standard discrete time backpropagation, now derived as a functional derivative in continuous time.

In the following, we will use the best estimation of the unknown instantaneous firing rate $R(t)$ when we have only a single spike train $S(t)$, which is the spike train itself for each of the neurons involved. Thus the weights

will be modified according to

$$\Delta w_{oh}(t) = \frac{1}{n_h}[S_o^d(t) - S_o^a(t)]S_h(t). \tag{3.12}$$

However, products of Dirac δ functions are mathematically problematic. Considering the Widrow-Hoff delta rule for rate neurons as a compound of two Hebbian processes, Ponulak and Kasiński (2010) derive the ReSuMe learning rule for spiking neurons in terms of the presynaptic, postsynaptic, and target signals represented by spike trains. Thus, following Ponulak and Kasiński (2010), we substitute the nonlinear product of $S_o^d(t)S_h(t)$ with an STDP process. In a similar manner, $-S_o^a(t)S_h(t)$ is substituted with an anti-STDP process (for details, see Ponulak & Kasiński, 2010),

$$\begin{aligned} S_o^d(t)S_h(t) \to S_h(t)\left[a + \int_0^\infty a^{pre}(s)S_o^d(t-s)ds\right] \\ + S_o^d(t)\left[a + \int_0^\infty a^{post}(s)S_h(t-s)ds\right], \end{aligned} \tag{3.13}$$

$$\begin{aligned} S_o^a(t)S_h(t) \to S_h(t)\left[a + \int_0^\infty a^{pre}(s)S_o^a(t-s)ds\right] \\ + S_o^a(t)\left[a + \int_0^\infty a^{post}(s)S_h(t-s)ds\right], \end{aligned} \tag{3.14}$$

where $a > 0$ is a non-Hebbian term that guarantees the weight changes in the correct direction if the output spike train contains more or fewer spikes than the target spike train.

The integration variable s represents the time difference between the actual firing time of the output neuron and the firing time of the hidden neuron $s = (t_o^f - t_h^f)$, and the target firing time and the firing time of the hidden neuron $s = (t_d^f - t_h^f)$, respectively. The kernel $a^{pre}(s)$ gives the weight change if the presynaptic spike (the spike of the hidden neuron occurs) comes after the postsynaptic spike (the spikes of the output and target neurons). The kernel $a^{post}(s)$ gives the weight change if the presynaptic spike occurs before the postsynaptic spike. The kernels a^{pre} and a^{post} define the learning window $W(s)$ (Gerstner & Kistler, 2002):

$$W(s) = \begin{cases} a^{pre}(-s) = -A_- \exp\left(\dfrac{s}{\tau_-}\right), & \text{if } s \le 0 \\ a^{post}(s) = +A_+ \exp\left(\dfrac{-s}{\tau_+}\right), & \text{if } s > 0 \end{cases}, \tag{3.15}$$

where $A_+, A_- > 0$ are the amplitudes and $\tau_+, \tau_- > 0$ are the time constants of the learning window. Other forms of learning windows can be considered (Ponulak, 2008); however, in this letter, we use only this form. Thus, the final learning formula for the weight modifications becomes

$$\Delta w_{oh}(t) = \frac{1}{n_h} S_h(t) \left[\int_0^\infty a^{pre}(s)[S_o^d(t-s) - S_o^a(t-s)]ds \right] + \frac{1}{n_h}[S_o^d(t) - S_o^a(t)] \left[a + \int_0^\infty a^{post}(s) S_h(t-s)ds \right]. \tag{3.16}$$

The total weight change is obtained by integrating equation 3.16 over time on a time domain that covers all the spikes in the system. This equation is the core of ReSuMe learning algorithm as stated in Ponulak and Kasiński (2010).

3.2.2 Weight Modifications for the Hidden Neurons. In this section we extend the argument above to weight changes between the input and the hidden layers. The weight modifications for the hidden neurons are calculated in a similar manner in the negative gradient direction:

$$\Delta w_{hi}(t) = -\frac{\partial E(R_o^a(t))}{\partial w_{hi}}. \tag{3.17}$$

The derivative of the error is expanded similarly as in equation 3.6 (again in the sense of functional derivatives):

$$\frac{\partial E(R_o^a(t))}{\partial w_{hi}} = \frac{\partial E(R_o^a(t))}{\partial R_h(t)} \frac{\partial R_h(t)}{\partial w_{hi}}. \tag{3.18}$$

The first factor on the right-hand part of the above equation is expanded for each output neuron using the chain rule:

$$\frac{\partial E(R_o^a(t))}{\partial R_h(t)} = \sum_{o \in O} \frac{\partial E(R_o^a(t))}{\partial R_o^a(t)} \frac{\partial R_o^a(t)}{\partial R_h(t)}. \tag{3.19}$$

The second factor on the right-hand side of the above equation is calculated from equation 3.2:

$$\frac{\partial R_o^a(t)}{\partial R_h(t)} = \frac{1}{n_h} w_{oh}. \tag{3.20}$$

The derivatives of the error with respect to the output spike train have already been calculated for the weights to the output neurons in equation 3.7. By combining these results,

$$\frac{\partial E(R_o^a(t))}{\partial R_h(t)} = \frac{1}{n_h} \sum_{o \in O} [R_o^a(t) - R_o^d(t)] w_{oh}. \tag{3.21}$$

The second factor on the right-hand side of equation 3.18 is calculated as follows, again using equation 3.2,

$$\frac{\partial R_h(t)}{\partial w_{hi}} = \frac{1}{n_i} R_i(t), \tag{3.22}$$

where n_i is the number of input neurons. When we combine equations 3.17 to 3.22, the formula for the weight modifications to the hidden neurons becomes

$$\Delta w_{hi}(t) = -\frac{1}{n_h n_i} \sum_{o \in O} [R_o^a(t) - R_o^d(t)] R_i(t) w_{oh}. \tag{3.23}$$

We define the backpropagated error $\delta_h(t)$ for layers other than the output layer:

$$\delta_h(t) := \frac{1}{n_i} \sum_{o \in O} \delta_o(t) w_{oh}. \tag{3.24}$$

Just as in standard backpropagation, the $\delta_o(t)$ are backpropagated errors of the neurons in the preceding layer. By substituting the instantaneous firing rates with the spike trains as estimators, equation 3.23 becomes

$$\Delta w_{hi}(t) = \frac{1}{n_h n_i} \sum_{o \in O} [S_o^d(t) - S_o^a(t)] S_i(t) w_{oh}. \tag{3.25}$$

We now repeat the procedure of replacing the product of two spike trains (involving δ-distributions) with an STDP process. We note first that equation 3.25 no longer depends on any spikes fired or not fired in the hidden layer. Although there are neurobiological plasticity processes that can convey information about a transmitted spike from the effected synapses to lateral or downstream synapses (for an overview, see Harris, 2008), no direct neurobiological basis is known for an STDP process between a synapse and the outgoing spikes of an upstream neuron. Therefore, this substitution is to be seen as a computational analogy and the weights will be modified

according to

$$\Delta w_{hi}(t) = \frac{1}{n_i n_h} S_i(t) \sum_{o \in O} \left[\int_0^\infty a^{pre}(s)[S_o^d(t-s) - S_o^a(t-s)]ds \right] w_{oh} + \frac{1}{n_i n_h} \sum_{o \in O} [S_o^d(t) - S_o^a(t)] \left[a + \int_0^\infty a^{post}(s) S_i(t-s) ds \right] w_{oh}. \tag{3.26}$$

The total weight change is again determined by integrating equation 3.26 over time. The synaptic weights between the input and hidden neurons are modified according to STDP processes between the input and target spikes and anti-STDP processes between input and output spikes.

3.2.3 Normalization. The normalization to the number of presynaptic connections of the modifications of the weights to the output neurons ensures that the changes are proportional to the number of weights. Moreover, the learning parameters do not need to change as the network architecture changes (e.g., in order to keep the firing rate of postsynaptic neurons constant as the number of presynaptic units changes, the initial weights and weight modifications also must change accordingly). The normalization to the number of presynaptic and postsynaptic connections of the weight modifications to the hidden neurons ensures that the changes of the connections between the input and hidden layer are usually smaller than the changes of the connections between the hidden and output layer, which keeps the learning process stable.

3.2.4 Generalization. The algorithm can be generalized in this manner for neural networks with multiple hidden layers. The learning rule could also be generalized for recurrent connections (e.g., using unrolling in time as in backpropagation through time; Rojas, 1996); however, in this letter, we consider only feedforward connections. This is our extension of ReSuMe to hidden layers following from error minimization and gradient descent.

As the learning rule for the weight modifications depends on only the presynaptic and postsynaptic spike trains and the current strength of the connections between the spiking neurons, the algorithm can be applied to various spiking neuron models as long as the model can be sufficiently well approximated on an appropriate timescale, as in equation 3.3. Ponulak and Kasiński (2010) do not use an explicit neuron model for the derivation of the ReSuMe algorithm either, but the ReSuMe algorithm has successfully been applied to leaky integrate-and-fire neurons, and Hodgkin-Huxley and Izhikevich neuron models (Ponulak & Kasiński, 2010). Since the learning rule is an extension of ReSuMe to neural networks with multiple layers,

this is an indication that this algorithm will function with similar neuron models, as we demonstrate in section 5.

3.2.5 Inhibitory Connections. Inhibitory connections are represented by negative weights that are updated in the same manner as positive weights. However, for the calculation of the backpropagation error of the hidden neurons $\delta_h(t)$, the absolute value of the output weights will be used below. This is a deviation from the gradient-descent rule in equation 3.26, but using the absolute values guarantees that the weights between the input and hidden neurons are always modified in the same direction as between hidden and output neurons:

$$\Delta w_{hi}(t) = \frac{1}{n_i n_h} S_i(t) \sum_{o \in O} \left[\int_0^\infty a^{pre}(s)[S_o^d(t-s) - S_o^a(t-s)]ds \right] |w_{oh}| + \frac{1}{n_i n_h} \sum_{o \in O} [S_o^d(t) - S_o^a(t)] \left[a + \int_0^\infty a^{post}(s) S_i(t-s) ds \right] |w_{oh}|. \tag{3.27}$$

Preliminary simulations have shown this results in better convergence of the learning algorithm. There is also neurobiological evidence that LTD and LTP spread to downstream synapses (Tao, Zhang, Bi, & Poo, 2000; Fitzsimonds, Song, & Poo, 1997), that is, that weight changes with the same direction propagation from upstream to downstream neurons. We discuss the effect of using the absolute value or not in the next section.

3.2.6 Delayed Subconnections. If one considers a network architecture where all the neurons in one layer are connected to all neurons in the subsequent layer through multiple subconnections with different delays d^k, where each subconnection has a different weight (Bohte et al., 2002), the learning rule for the weight modifications for the output neurons will become

$$\Delta w_{oh}^k = \delta_o(t) R_h(t - d_{oh}^k), \tag{3.28}$$

where w_{oh}^k is the weight between output neuron o and hidden neuron h delayed by d_{oh}^k. The backpropagated error for the output is then

$$\delta_o(t) = \frac{1}{m n_h} [R_o^d(t) - R_o^a(t)], \tag{3.29}$$

where m is the number of subconnections. The learning rule for the weight modifications for any hidden layer is derived similarly as

$$\Delta w_{hi}^k = \delta_h(t) R_i(t - d_{oh}^k), \tag{3.30}$$

where $\delta_h(t)$ is the backpropagated error calculated over all possible backward paths (from all output neurons through all delayed subconnections):

$$\delta_h(t) = \frac{1}{mn_i} \sum_{l,o \in O} \delta_o(t) w_{oh}^l. \tag{3.31}$$

3.2.7 Synaptic Scaling. There has been extensive evidence that suggests that spike-timing-dependent plasticity is not the only form of plasticity (Watt & Desai, 2010). Another plasticity mechanism used to stabilize the neurons' activity is synaptic scaling (Shepard et al., 2006). Synaptic scaling regulates the strength of synapses in order to keep the neuron's firing rate within a particular range. The synaptic weights are scaled multiplicatively, this way maintaining the relative differences in strength between any inputs (Watt & Desai, 2010).

In our network, in addition to the learning rule described above, the weights are modified according to synaptic scaling in order to keep the postsynaptic neuron firing rate within an optimal range $[r_{\min}, r_{\max}]$. If a weight w_{ji} from neuron i to neuron j causes the postsynaptic neuron to fire with a rate outside the optimal range, the weights are scaled according to the following formula (Grüning & Sporea, 2012):

$$w_{ji} = \begin{cases} (1+f) w_{ji}, & w_{ji} > 0 \\ \dfrac{1}{1+f} w_{ji}, & w_{ji} < 0 \end{cases}, \tag{3.32}$$

where the scaling factor $f > 0$ for $r_j < r_{\min}$, and $f < 0$ for $r_j > r_{\max}$.

Synaptic scaling solves the problem of optimal weight initialization. It was observed that the initial values of the weights have a significant influence on the learning process, as values that are too large or too low may result in failure to learn (Bohte et al., 2002). Preliminary experiments showed that a feedforward network can still learn reliably simple spike trains without synaptic scaling as long as the weights are initialized within an optimal range. However, as the target patterns contain more spikes, finding the optimal initial values for the weights becomes difficult. Moreover, as the firing rate of the target neurons increases, it becomes harder to maintain the output neurons' firing rate within the target range without using small learning rates. The introduction of synaptic scaling solves the problem of weight initialization as well as speeds up the learning process.

4 Heuristic Discussion of the Learning Rule

In order to analyze the direction in which the weights change during the learning process using equations 3.16 and 3.27, we consider a simple

three-layer network. The output layer consists of a single neuron. The neurons are connected through a single subconnection with no delay. For simplicity, in this section spike trains will comprise only a single spike. Let t_d and t_a denote, respectively, the desired and actual spike time of output neuron o and t_h and t_i, respectively, the spike times of the hidden neuron h and input neuron i, respectively. Also, for simplicity, synaptic scaling will not be considered here.

For a start, we assume $t_o, t_d > t_h > t_i$, that is, where relevant postsynaptic spikes occur after the presynaptic spikes. With these assumptions, equations 3.16 and 3.27 read after integrating out:

$$\Delta w_{oh} = \frac{1}{n_h}\left(A_+ \exp\frac{t_h - t_d}{\tau_+} - A_+ \exp\frac{t_h - t_o}{\tau_+}\right), \tag{4.1}$$

$$\Delta w_{hi} = \frac{1}{n_h n_i}|w_{oh}|\left(A_+ \exp\frac{t_i - t_d}{\tau_+} - A_+ \exp\frac{t_i - t_o}{\tau+}\right). \tag{4.2}$$

We discuss only this case in the following and note that the case $t_o, t_d < t_h, t_i$ (i.e., post-before-pre) can be discussed along the same lines with A_+ above replaced by $A\_$. We discuss now the following subcases:

1. The output neuron fires a spike at time t_o before the target firing time t_d ($t_o < t_d$).
   a. *Weight modifications for the synapses between the output and hidden neurons.* The weights are modified according to $\Delta w_{oh} = \frac{1}{n_h}(A_+ \exp\frac{t_h - t_d}{\tau_+} - A_+ \exp\frac{t_h - t_o}{\tau+})$. Since $t_o < t_d$, then $\exp(\frac{t_h - t_o}{\tau_+}) > \exp(\frac{t_h - t_d}{\tau_+})$ in equation 4.1. This results in $\Delta w_{oh} < 0$, and thus in a decrease of this weight. If the connection is an excitatory one, the connection becomes less excitatory, increasing the likelihood that the output neuron fires later during the next iteration, hence minimizing the difference between the actual output and the target firing time. If the connection is inhibitory, the connection will become stronger inhibitory, resulting in a later firing of the output neuron o as well (see also Ponulak, 2006).
   b. *Weight modifications for the synapses between the hidden and input neurons.* The weights to the hidden neurons are modified according to: $\Delta w_{hi} = \frac{1}{n_h n_i}(A_+ \exp\frac{t_i - t_d}{\tau_+} - A_+ \exp\frac{t_i - t_o}{\tau+})|w_{oh}|$.
      i. $w_{oh} \geq 0$. By a analogous reasoning to the case above, $\Delta w_{hi} \geq 0$; hence, the connection will become less excitatory or more inhibitory, again making the hidden neuron fire slightly later or suppress a hidden-layer spike, and hence making it more likely that the output neuron fires later because the connection from hidden to output layer is excitatory.

ii. $w_{oh} < 0$. For the weight w_{hi}, the direction of the weight change stays the same; hence, neuron h will fire later. As it is now more likely to fire later, its inhibitory effect will come to bear on the output neuron also slightly later. Alternatively, if we do not take the absolute value of w_{oh} in equation 3.27, but stick to equation 3.26, then the direction of change to w_{hi} is reversed, that is, $\Delta w_{hi} > 0$. This brings forward the firing of neuron h; hence, h has a less suppressive effect at t_o and contributes to making the output fire even earlier. This is why it makes sense to use the modulus $|w_{oh}|$.

2. The output neuron fires a spike at time t_o after the target firing time t_d ($t_o > t_d$). As equations 4.1 and 4.2 change their sign when t_o and t_d are swapped, this case reduces to the above, but with the opposite sign of the weight change (i.e., overall weight change such that t_o moves forward in time, close to t_d).

Cases where there is only an actual spike at t_o and no desired spike or where there is only a desired spike at t_d can be dealt with under the above cases if one sets $t_d = \infty$ or $t_o = \infty$ respectively. In addition, there will be a contribution from the factor a in equations 3.16 and 3.27, and this has the same sign as the one from equations 4.1 and 4.2.

5 Simulations

In this section, several experiments are presented to illustrate the learning capabilities of the algorithm. The algorithm is applied to classic benchmarks, the XOR problem, and the Iris data set, as well as to mapping and classification tasks with randomly generated patterns. The XOR problem is examined using two different encoding methods to demonstrate the flexibility of our learning algorithm. The learning rule is used on spike timing patterns that span from 100 ms to 500 ms, about the ranges of sensory and motor processing in biological systems.

5.1 Setup. The network used for the following simulations is a feedforward architecture with three layers. The neurons are described by the Spike Response Model (Gerstner, 2001; see appendix A for a complete description).

For all simulations, an iteration consists of presenting all spike timing pattern pairs in random order. The membrane potential of all neurons in the hidden and output layers is set to the resting potential (set to zero) when presenting a new input pattern. After each presentation of an input pattern to the network, the weight changes are computed for all layers and then applied. We apply these weight changes after the backpropagated error is computed for all units in the network. The summed network error is calculated for all patterns and tested against a required minimum value,

depending on the experiment. This minimum value is chosen in order to guarantee that the network has learned to reproduce all output patterns with acceptable precision.

In sections 5.2 to 5.5, apart from the required minimum network error, the learning is considered converged only when the network also correctly classifies most input patterns, depending on the experiment. For all simulations, the output and target signals during the learning process are illustrated for a sample trial.

The network error for one pattern is defined as the van Rossum distance between each output spike train and each target spike train (van Rossum, 2001). The distance between the target spike train and the actual output spike train is defined as the Euclidean distance of the two filtered spike trains (van Rossum, 2001). The filtered spike train is determined by an exponential function associated with the spike train,

$$f(t) = \sum_i \exp[-(t - t_i)/\tau_c] H(t - t_i), \tag{5.1}$$

where t_i are the times of the spikes and $H(t)$ is the Heaviside function. τ_c, the time constant of the exponential function, is chosen to be appropriate to the interspike interval of the output neurons (van Rossum, 2001). In the following simulations, the output neurons are required to fire approximately one spike in 10 ms; thus, $\tau_c = 10$ ms. The distance between two spike trains is the squared Euclidean distance between these two functions:

$$D^2(f, g) = \frac{1}{\tau_c} \int_0^T [f(t) - g(t)]^2 dt, \tag{5.2}$$

where the distance is calculated over a time domain $[0, T]$ that covers all the spikes in the system. The van Rossum distance is also used to classify the output pattern during learning and testing. The output pattern is interpreted as the closest of the target patterns in terms of the van Rossum distance. To give the reader an intuitive sense of the magnitude of the van Rossum distance as used, a van Rossum distance of 0.1 corresponds, for example, to a pair of spike trains that agree on all spike times, but one spike pair is about 1 ms apart.

The results are averaged over a large number of trials (50 trials unless stated otherwise), with the network being initialized with a new set of random weights every trial. On each testing trial, the learning algorithm is applied a maximum of 2000 iterations, or until the network error has reached the minimum value.

Unless stated otherwise, the network parameters used in these simulations are the threshold $\vartheta = 0.7$, the time constant of the spike response function $\tau = 7$ ms, and the time constant of after-potential kernel $\tau_r = 12$ ms.

Table 1: Input and Output Spike-Time Patterns.

Input (ms)			Output (ms)
0	0	0	16
0	6	0	10
6	0	0	10
6	6	0	16

Note: The patterns consists of the timing of single spikes in ms of simulated time for the input and target neurons.

The scaling factor is set to $f = \pm 0.005$. The learning parameters are initialized as follows: $A_+ = 1.2$, $A_- = 0.5$, $\tau_+ = \tau_- = 5$ ms, $a = 0.05$.

The weights are initialized with random values uniformly distributed between -0.2 and 0.8. The weights are then normalized by dividing them to the total number of subconnections.

5.2 The XOR Benchmark. In order to demonstrate and analyze the new learning rule, the algorithm is applied to the XOR problem. While this benchmark does not require generalizing, the XOR logic gate is a linearly nonseparable problem, and it is a classic benchmark for testing the learning algorithm's ability to train nontrivial input output mapping (Rojas, 1996).

5.2.1 Technical Details. The input and output patterns are encoded using spike-time patterns as in Bohte et al. (2002). The signals are associated with single spikes as follows: a binary symbol 0 is associated with a late firing (a spike at 6 ms for the input pattern), and a 1 is associated with an early firing (a spike at 0 ms for the input pattern). We also used a third input neuron that designates the reference start time as this encoding needs an absolute reference start time to determine the latency of the firing (Sporea & Grüning, 2012). Without a reference start time, two of the input patterns become identical, and without an absolute reference time, the network is unable to distinguish the two patterns (0-0 and 6-6) and would always respond with a delayed output. Table 1 shows the input and target spike timing patterns presented to the network. The values represent the times of the spikes for each input and target neuron in ms of simulated time.

The learning algorithm was applied to a feedforward network as described above. The input layer is composed of three neurons, the hidden layer contains five spiking neurons, and the output layer contains only one neuron. Multiple subconnections with different delays were used for each connection in the spiking neural network. Preliminary experiments showed that 12 subconnections with delays from 0 ms to 11 ms are sufficient to learn the XOR problem. The results are averaged over 100 trials.

The network error is summed over all pattern pairs, with a minimum value for convergence of 0.2. The minimum value is chosen to ensure that the network has learned to classify all patterns correctly by matching the exact number of spikes of the target spike train as well as the timing of the spikes with 1 ms precision. Each spiking neuron in the network was simulated for a time window of 30 ms, with a time step of 0.1 ms. In the following, we systematically vary the parameters of the learning algorithm and examine their effects.

5.2.2 The Learning and Network Parameters. Here, we vary the learning parameters A_+ and A_- in equation 3.15 in order to determine the most appropriate values. A_+ is varied between 0.5 and 2.0 while keeping $A_- = \frac{1}{2}A_+$. The parameters A_+ and A_- play the role of a learning rate. Just like the classic backpropagation algorithm for rate neurons, when the learning parameters have higher values, the number of iterations needed for convergence is lower. The detailed results of the simulations are summarized in appendix B in Table 3 (left). Although the algorithm converges with a high rate for all values of A_+, for the lower values ($A_+ < 0.8$), the learning process is slower. When A_+ has higher values, the network requires around 200 iterations to learn all four patterns. If A_+ is too high, the convergence rate starts to drop.

In order to determine the best ratio between the two learning parameters, various values are chosen for A_-, while keeping $A_+ = 1.2$ fixed (the results are summarized in Table 3 (right). The learning algorithm is able to converge for the values of A_- lower than A_+. As A_- becomes equal to or higher than A_+, the convergence rate slowly decreases and the number of iterations needed for convergence significantly rises. The lowest average number of iterations with a high convergence rate is 137 averaged over 98% successful trials (where $A_+ = 1.2$ and $A_- = 0.5$).

The algorithm also converges when the spiking neural network has a smaller number of subconnections. However, a lower number of delayed subconnections (between 4 and 10) results in a significantly lower convergence rate without necessarily a lower average of learning iterations for the successful trials. Although more subconnections can produce a more stable learning process, due to the larger number of weights that need to be coordinated, the learning process is slower in this case (more than 300 iterations). Table 4 in appendix B shows the summarized results, where $A_+ = 1.2$ and $A_- = 0.6$.

5.2.3 Analysis of the Learning Process. In order to analyze the learning process, the network error and the weight vector during the learning process for a sample trial can be seen in Figure 1 ($A_+ = 1.2$, $A_- = 0.6$, and 12 subconnections). Figure 1a shows the evolution of the summed network error during learning. Although the network error reaches a minimum value

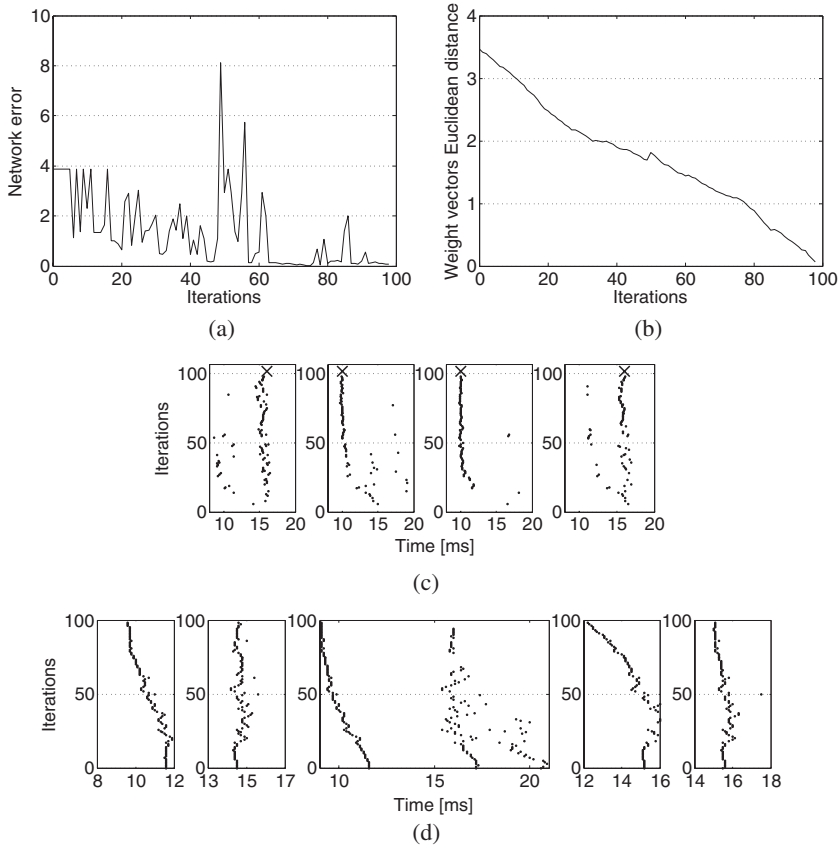

Figure 1: The XOR task. Analysis of the learning process with parameters $A_+ = 1.2$ and $A_- = 0.5$ for a sample trial. (a) The network error during learning. (b) The Euclidean distance between the weight vector solution and the weight vectors during the learning process. (c) The output signals for each of the four patterns during learning. An x represents the target spike times. (d) The hidden signals during learning for each hidden neuron for one input pattern ([0 0]).

after 63 iterations, due to the nature of the STDP processes, the solution is lost, only to converge again later. Similar findings were reported in Grüning and Sporea (2012). Figure 1b shows the Euclidean distance between the weight vector solution found on that particular trial and the weight vectors during each learning iteration that led to this weight vector. The weight vectors are tested against the solution found during this trial because in principle, there can be multiple solutions to weight vectors (e.g., different initial weight set results in a different weight solution for the same set of

Table 2: Input and Target Patterns.

Species	Sepal Length Range (ms)	Sepal Width Range (ms)	Petal Length Range (ms)	Petal Width Range (ms)	Output (ms)
Setosa	4.3–5.8	2.3–4.8	1.0–1.9	0.1–0.6	10
Versicolor	4.9–7.0	2.0–3.4	3.0–5.1	1.0–1.8	14
Virginica	4.9–7.9	2.2–3.8	4.5–6.9	1.4–2.5	18

Note: The patterns contain a single spike, where the timing (shown in ms) differs for each of the three patterns.

pattern pairs). While the error graph is irregular, the weight vector graph shows that the weight vector moves steadily toward the solution. The irregularity of the network error during the learning process can be explained by the fact that small changes to the weights can produce an additional or missing output spike, which causes significant changes in the network error. The highest error value corresponds to the network's not firing any spike for any of the four input patterns. The error graph also shows the learning rule's ability to modify the weights in order to produce the correct number of output spikes. Figure 1c shows the output signals during learning for all input patterns. For two of the input patterns, the output signals are stable after only 50 learning iterations, while for the other two patterns, the output neurons fire around the target spike times. As such, during learning, the network responds with either the incorrect response or the correct response, but with the time difference between the target and output signal too large. Figure 1d shows the spike timings during learning for each of the hidden neurons for one of the patterns.

5.3 The Iris Benchmark. Another classic benchmark of pattern recognition is Fisher's Iris flower data set (Fisher, 1936). The data set contains three classes of Iris flowers. While one of the classes is linearly separable from the other two, the other two classes are not linearly separable from each other.

5.3.1 Technical Details. The three species are described by four measurements of the plants: the lengths and widths of the petal and sepal. Each of the four features is represented by the timing of a single spike of a corresponding input neuron. The measurements of the Iris flower range from 0 to 8 (see Table 2) and are fed into the spiking neural network as spike-timing patterns to the input neurons. The output of the network is represented by the spike time of the output neuron (see Table 2). The hidden layer contains 10 spiking neurons, and each connection has between 8 and 12 delayed subconnections depending on the experiment. The network is simulated in a 30 ms time window with 0.1 ms time step.

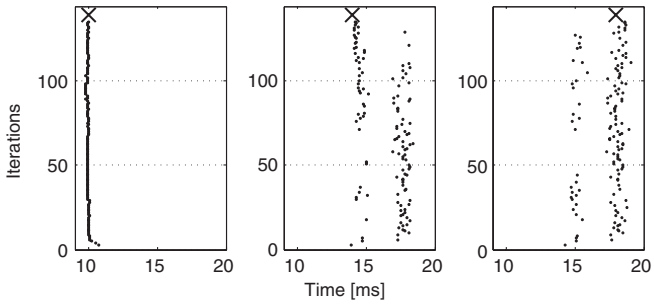

Figure 2: The Iris data set. Output signals during learning for each of three species for a sample trial. The x markers represent the target spike times.

During each trial, the input patterns are randomly divided into a training set (75% of samples) and a testing set (25% of samples) for cross-validation. During each iteration, the training set is used for the learning process to calculate the weight modifications and test if the network has learned the patterns. The learning is considered successful if the network error has reached a minimum average value of 0.2 for each pattern pair and 95% of the patterns in the training set are correctly classified. As in the previous experiment, this minimum value is chosen to ensure that the network has learned to classify all patterns correctly by matching the exact number of spikes of the target spike train as well as timing of the spikes with 1 ms precision. Figure 2 shows the output signals during learning for all three classes of species during a sample trial. While the first pattern is learned after only a few iterations, it takes more than 100 iterations to distinguish the other two classes. Table 5 in appendix B shows the summarized results on the Iris data set for different network architectures with different numbers of delayed subconnections. Again, a too low or too high number of subconnections results in lower performance—a convergence rate of less than 80%. A network with 10 subconnections achieves a convergence rate of 80% within 114 iterations on average.

Multilayer ReSuMe permits the spiking neural network to learn the Iris data set using a straightforward encoding of the patterns and results in much faster learning than SpikeProp, as the average number of iterations is always lower than 200, as opposed to the population coding based on arrays of receptive fields that requires 1000 learning iterations with SpikeProp (Bohte et al., 2002).

5.4 The XOR Task with Spike Train Patterns. In this experiment, the learning algorithm is tested on a linearly nonseparable problem and mapping of corresponding sequences of spikes. Again, the XOR problem is applied to a network of spiking neurons, but the logic patterns are encoded

by spike trains over a group of neurons instead of the timing of a single spike of one input neuron as previously (see also Grüning & Sporea, 2012).

While the encoding for the XOR logic gate problem introduced by Bohte et al. (2002) requires neurons to fire a single spike, the network of spiking neurons needs a large number of subconnections with different delays to enable the hidden and output neurons to fire at the desired times. As the problem becomes more complex, such encoding might need even more subconnections that have to be trained. The large number of weights to be trained slows the learning process because of the large number of incoming spikes that need to be coordinated to produce the required output. This can also be seen in the previous simulations on the XOR problem where the network with 14 delayed subconnections needed almost twice as many iterations to converge as the network with 12 subconnections. Moreover, it has been shown that encoding logical true and false with early and late spike times, respectively, also requires an additional input neuron to designate the reference start time. Without the additional input neuron, even linearly separable problems become impossible to solve (for a complete demonstration, see Sporea & Grüning, 2011).

A more natural encoding would consist of the temporal firing patterns of groups of neurons (Wehr & Laurent, 1996; Neuenschwander & Singer, 1996; deCharms & Merzenich, 1996). In order to test such an encoding and the learning algorithm's ability to learn linearly nonseparable problems, the XOR problem is applied once again to a spiking neural network. In this experiment, the two logical values are encoded with spike trains over two groups of input neurons. Figure 3a shows the network structure. This encoding will not necessitate multiple delays or the additional input neuron. In the following experiments, a single connection with no delay is used.

5.4.1 Technical Details. Each input logical value is associated with the spike trains of a group of 20 spiking neurons. In order to ensure some dissimilarity among the patterns, for each input neuron, a spike train is generated by a pseudo-Poisson process with a constant firing rate of $r = 0.06$/ms within a 30 ms time window. The minimum interspike interval is set to 3 ms. This spike train is then split in two new spike trains by randomly distributing all the spikes (Grüning and Sporea, 2012). The newly created spike trains represent the patterns for the logical symbols 0 and 1. The input spike trains are required to consist of at least one spike.

The output patterns are created similarly and will be produced by one output neuron. The spike train to be split is generated by a pseudo-Poisson process with a constant firing rate of $r = 0.2$/ms within a 30 ms period of time. The resulting output patterns are chosen so that the spike trains contain exactly three spikes.

Apart from the minimal network error as before, an additional stopping criterion for the learning process is introduced. The network must correctly classify all four patterns. An input pattern is considered correctly classified

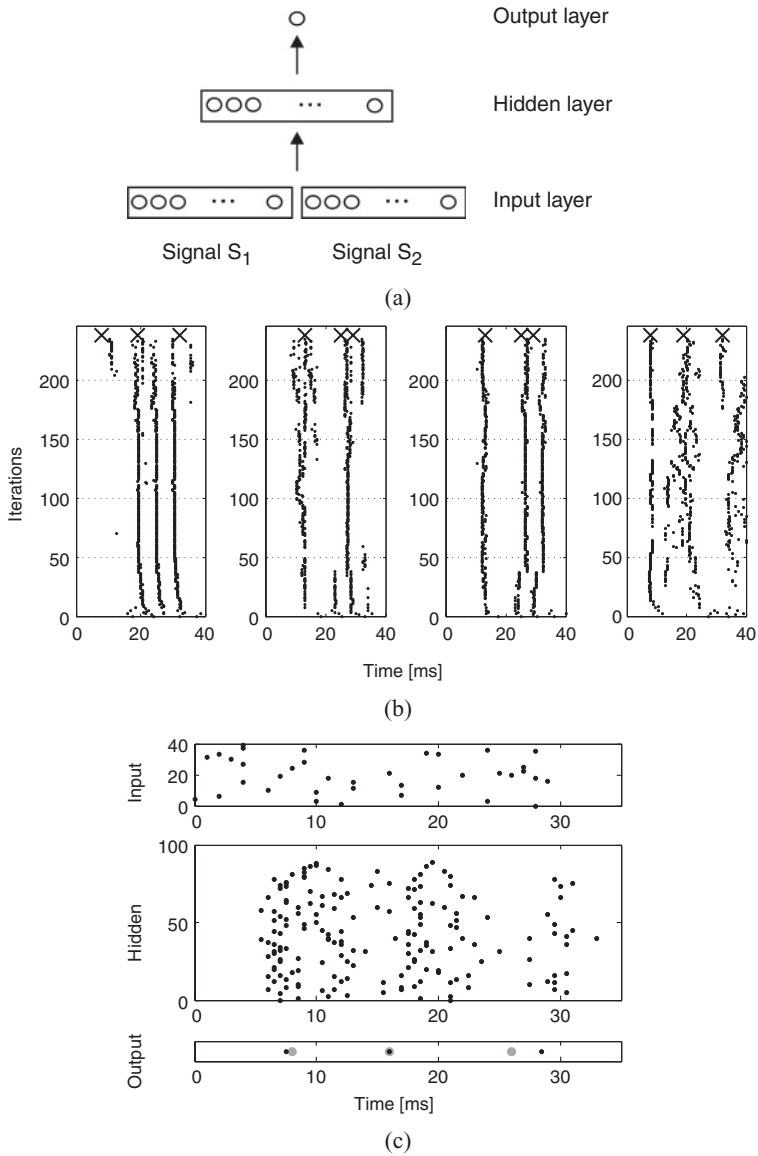

Figure 3: The XOR task with spike train patterns. (a) Network structure for the XOR problem. A feedforward network with three layers, where the input layers consist of two groups of 20 neurons for each logical signal. (b) Output spikes for all four input patterns [0 0], [0 1], [1 0], [1 1] during learning for a sample trial. The x markers represent the target spike times. (c) Sample input, hidden, and output signals for the logical input ([1 0]), after the learning process has converged. The gray signals in the output graph represent the target pattern.

if the output spike train is closest to the target pattern in terms of the van Rossum distance. The network error consists of the sum of van Rossum distances between the target and actual output over the four patterns as before; a minimum value of 3 ensures that the output spikes are reproduced with acceptable precision.

In addition to the previous experiments, an absolute refractory period is set for all neurons to $t = 3$ ms. The learning is simulated over a period of 50 ms, with a time step of 0.5 ms.

In order to determine the optimal size of the hidden layer for a higher convergence rate, different network topologies have been considered. In appendix B, Table 6 shows the convergence rate for each network topology, with a new set of spike-timing patterns being generated every trial.

The learning rule is able to converge with a higher rate as the number of neurons in the hidden layer increases. A larger hidden layer means that the patterns are mapped to a richer spiking activity, hence, it is easier for output neurons to produce the required spike patterns. A smaller number of neurons in the hidden layer than in the input layer does not result in high convergence rate because the input patterns are not sufficiently distributed in the hidden activity. Also, more than 100 units in the hidden layer does not result in higher convergence rates, but as the number of weights increases, the learning process is slower. Previous simulations (Grüning & Sporea, 2012) show that a neural network without a hidden layer cannot learn linearly nonseparable logical operations.

Figure 3b shows the output signals during learning for a sample trial. Figure 3c shows the input, hidden, and output signals for one of the patterns. The first 20 input spike trains represent the pattern for the logical symbol 1, while the other 20 spike trains represent the pattern for the logical symbol 0. Although the network is not responding with the exact target spike train, the output spike train is closest to the pattern representing the logical 1 than to the pattern representing logical 0 in terms of the van Rossum distance.

5.5 Learning Sets of Temporal Patterns. In this experiment, we consider the learning algorithm's ability to train a spiking neural network with multiple input-target pattern pairs. The network is trained with random noise-free spike train patterns and tested against noisy versions of the temporal patterns.

5.5.1 Technical Details. The input patterns are generated by a pseudo-Poisson process with a constant firing rate of $r = 0.05$/ms within a 100 ms period of time, where the spike trains are chosen so that they contain at least one spike. In order to ensure that a solution exists, the target patterns are generated as the output of a spiking neural network initialized with a random set of weights. The target spike trains are chosen so they contain at least two spikes and no more than four spikes. If the output patterns were

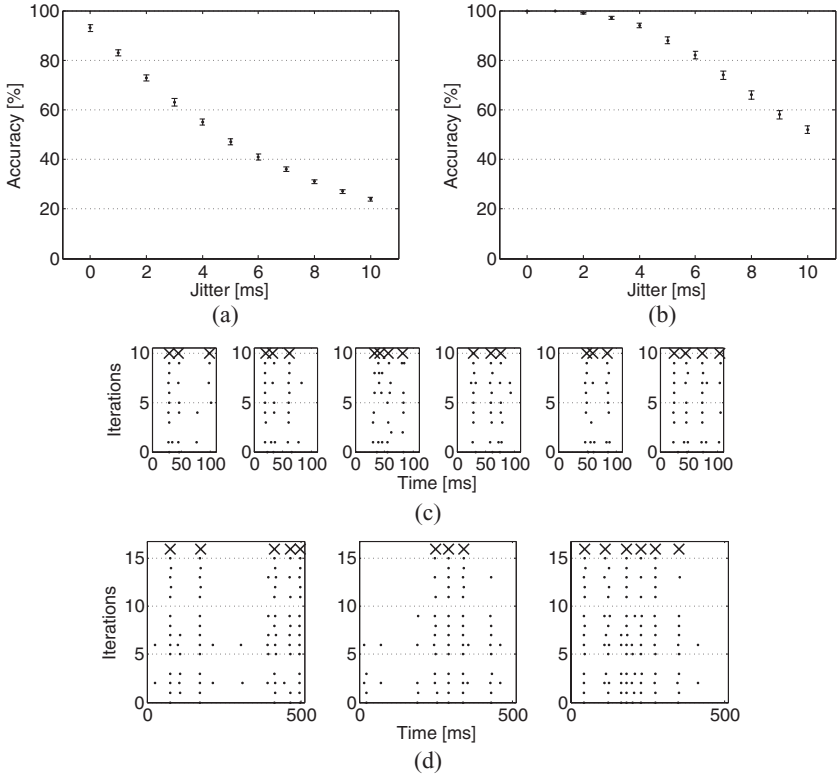

Figure 4: Accuracy on noisy patterns (generated by moving each spike within a gaussian distribution with mean 0 and standard deviation between 0 ms and 10 ms). (a) The network has been trained with 10 noise-free patterns that span over 100 ms in (see section 5.5). (b) The network has been trained with three noisy patterns that span over 500 ms (see section 5.6). During learning, the noisy input patterns are generated by moving each spike within a gaussian distribution with mean 0 and standard deviation 4 ms. (c) The output signals during learning for a sample trial. The network has been trained with six noise-free patterns that span over 100 ms. (d) Output signals during learning for a sample trial. The network has been trained with three noisy patterns (4 ms jitter) that span over 500 ms. The x markers represent the target spike trains.

random spike trains, a solution might not be representable in the weight space of the network (Legenstein et al., 2005).

The learning is considered to have converged if the network error reaches an average value of 0.5 for each pattern pair. Apart from the minimum error, the network must also correctly classify at least 90% of the pattern pairs, where the patterns are classified according to the van Rossum distance. Figure 4c shows the output signals during learning for a sample trial. The

minimum network error allows the output spike train to miss or add an extra spike as long as the pattern is still closest to the target in terms of the van Rossum distance. The network is simulated for 120 ms with a 1 ms time step.

5.5.2 Size of the Hidden Layer. In order to determine how the structure of the neural network influences the number of patterns that can be learned, different architectures have been tested. In these simulations, 100 input neurons are considered in order to have a distributed firing activity for the simulated time period. The output layer contains a single neuron as in the previous simulations. The size of the hidden layer is varied from 200 to 300 neurons to determine the optimal size for storing 10 input-output pattern pairs. The network is able to perform better as the number of hidden neurons increases. However, a hidden layer with more than 260 neurons does not result in a higher convergence rate. The detailed results of the simulations are summarized in appendix B in Table 7 (left).

5.5.3 Number of Patterns. The network architecture that performed best with the lowest number of neurons (260 neurons in the hidden layer) was trained with different numbers of patterns. The detailed results for different number of patterns are summarized in appendix B in Table 7 (right). The network is able to store more patterns, but the convergence rate drops as the number of patterns increases. Because the target patterns are the output spike trains of a randomly initialized spiking neural network, as the number of pattern pairs increases, the target spike trains become necessarily more similar. Hence, the network's responses to the input patterns become more similar and more easily misclassified. Since the stopping criterion requires the network to correctly classify the input patterns, the convergence rate drops as the number of pattern pairs increases.

Since the target patterns are generated as the output spike trains of a network with a set of random weights, this vector of weights can be considered the solution of the learning process. However, when looking at the Euclidean distance between the weight vector solution and the weight vectors during learning, the distance is increasing as the learning process progresses. The learning algorithm does not find the same weight vector as the solution, so multiple solutions of weight vectors to the same problem exist (e.g., permutations of hidden neurons are the simplest ones).

5.5.4 Noise. After the learning has converged, the networks are also tested against noisy patterns. The noisy patterns are generated by moving each spike within a gaussian distribution with mean 0 and standard deviation between 1 ms and 10 ms. After the network has learned all patterns, the network is tested with a random set of 500 noisy patterns. Figure 4a shows the accuracy rate (the percentage of input patterns that are correctly classified) for the network with 260 spiking neurons in the hidden layer

trained with 10 pattern pairs. The accuracy rate is defined as the percentage of correctly classified patterns calculated over the successful trials. The accuracy rates are similar for all the networks described above. The network is able to recognize more than 20% (above the random performance level of 10%) of the patterns when these are distorted with 10 ms.

5.6 Learning to Generalize. In this experiment, the learning algorithm is tested in the presence of noise. In the previous experiments where patterns are randomly generated, the learning occurred in noise-free conditions. A spiking neural network is trained to recognize temporal patterns on the timescale of hundreds of milliseconds. Jitters of spike times are introduced in the temporal patterns during learning to test the network's ability to classify time-varying patterns. Such experiments have been conducted with liquid state machines where readout neurons have been trained with ReSuMe to respond with associated spike trains (Ponulak & Kasiński, 2010). Here we show that such classification tasks can be achieved with feedforward networks without the need of larger networks such as reservoirs.

5.6.1 Technical Details. Three random patterns are fed into the network through 100 input spiking neurons. The hidden layer contains 210 neurons, and the patterns are classified by a single output neuron. The input patterns are generated by a pseudo-Poisson process with a constant firing rate of $r = 0.1$/ms within a 500 ms time period, where the spike trains are chosen so that they contain between 15 and 20 spikes. For the spike train generation, an interspike interval is set to 5 ms. As in the previous experiment, in order to ensure that a solution exists, the target patterns are generated as the output of a spiking neural networks initialized with a random set of weights. The target spike trains are chosen so that they contain at least three spikes and no more than seven spikes. The input and target patterns are distributed over such large periods of time in order to simulate complex forms of temporal processing, such as speech recognition, that spans over hundreds of milliseconds (Mauk & Buonomano, 2004).

During learning, for each iteration, noisy versions of the input patterns are generated by moving each spike by a time interval within a gaussian distribution with mean 0 and standard deviation varying in the range of 1 ms to 4 ms. Figure 5 shows a sample input pattern where the spikes are moved by a time interval within a gaussian distribution with mean 0 and standard deviation 4 ms. The spikes in the target patterns are also shifted by a time interval within a gaussian distribution with mean 0 and standard deviation 1 ms independent of the noise level in the input patterns. The network is simulated for 520 ms with 1 ms time step.

A minimum average error of 0.6 for each pattern pair is required for the learning to be considered successful. During each iteration, the network is tested against a new set of 30 random noisy patterns; in order for the learning to be considered converged, the network must also correctly classify

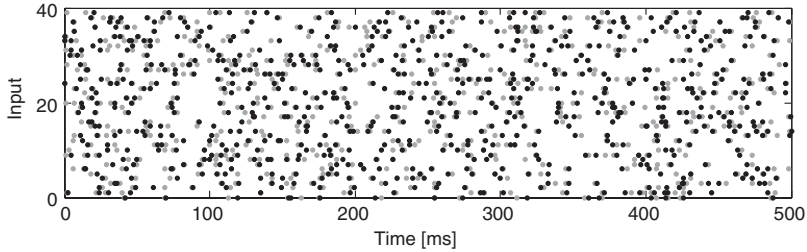

Figure 5: Example of an input pattern where the spikes are moved by a time interval within a gaussian distribution with mean 0 and standard deviation 4 ms. The gray markers represent the original signals, and the black markers represent the noisy signals.

at least 80% of noisy patterns. The spike times of the testing patterns are shifted with the same distribution as the training patterns. Figure 4d shows the output signals during learning for a sample trial. Again, the minimum network error allows the output spike train to miss or add an extra spike as long as the input patterns are correctly classified.

The detailed results of the simulations are shown in appendix B in Table 8, where the average number of iterations is calculated over the successful trials. The table also shows the number of successful trials when the network is trained on noise-free patterns. When the network is trained with a low amount of noise in the input patterns, the learning algorithm performs slightly better than the network trained with patterns without noise. The network is able to learn even when the spike train patterns are distorted with 3 ms or 4 ms; however, the speed of learning as well as the convergence rate drop as more noise is added to the input patterns.

Figure 4b shows the accuracy rates on a trained network against a random set of 150 different noisy patterns, generated from the three original input patterns. The network is trained on input patterns where the spikes are moved within a gaussian distribution with mean 0 and standard deviation 4 ms. The graph shows the accuracy rates on patterns with the spikes moved within a gaussian distribution with mean 0 and standard deviation between 1 ms and 10 ms. The graph also shows the network response on the noise-free patterns. The accuracy rates are similar for all input pattern jitter. The network is able to recognize more than 50% (again above the random performance level of 33%) of the input patterns even when these are distorted with up to 10 ms.

6 Discussion

This letter introduces a new algorithm for feedforward spiking neural networks. The first supervised learning algorithm for feedforward spiking

neural networks with multiple layers, SpikeProp, considers only the first spike of each neuron, ignoring all subsequent spikes (Bohte et al., 2002). Extensions of SpikeProp allow multiple spikes in the input and hidden layer but not in the output layer (Booij & Nguyen, 2005; Ghosh-Dastidar & Adeli, 2009). Our learning rule is, to the best of our knowledge, the first fully supervised algorithm that considers multiple spikes in all layers of the network. Although ReSuMe allows multiple spikes, the algorithm in its original form can be applied only to single layers or to train readout neurons in liquid state machines (Ponulak & Kasiński, 2010). In our approach, multilayer ReSuMe, the hidden layer permits the networks to learn linearly nonseparable problems as well as complex mapping and classification tasks without using a large number of spiking neurons as liquid state machines do or without the need of a large number of input neurons in single-layer networks. Because the learning rule presented here extends the ReSuMe algorithm to multiple layers, it can, like the original ReSuMe, in principle be applied to any neuron model, as the weight modification rules depend on only the input, output, and target spike trains and do not depend on the specific dynamics of the neuron model. We discuss a few important aspects in order.

On the one hand, the ReSuMe learning rule applied to a single layer (Ponulak & Kasiński, 2010) with 12 to 16 delayed subconnections for each connection is not able to learn the XOR problem with the early and late timing patterns (simulations not presented in this letter). Although the algorithm is able to change the weights in the correct direction, the network never responds with the correct output for all four input patterns. The additional hidden layer permits the network to learn the XOR problem (see section 5.2). On the other hand, a spiking neural network with the same number of units in each layer, but with 16 subconnections trained with SpikeProp on the XOR patterns, needs 250 iterations to converge (Bohte et al., 2002), while multilayer ReSuMe converged in 137 iterations on average. Furthermore, SpikeProp uses 16 delayed subconnections instead of just 12; hence, more weight changes need to be computed. Finally, SpikeProp matches the time of only the first target spike, ignoring any subsequent spikes. Although our algorithm also matches the exact number of spikes and the precise timing of the target patterns, the network learns all the patterns faster.

Studies on SpikeProp show that the algorithm is unstable, affecting the performance of the learning process (Takase et al., 2009; Fujita, Takase, Kita, & Hayashi, 2008). For our learning algorithm, the weight vector moves steadily toward a solution during the learning process, as seen in Figures 1a and 1b. This can be seen in a direct comparison with SpikeProp on the XOR benchmark. Finally, the learning algorithm presented here permits using different encoding methods with spiking patterns. In section 5.3, the Iris data set is encoded using 4 input neurons instead of the 50 neurons required by a population encoding (Bohte et al., 2002). The simpler encoding of the

iris flower dimensions allows the network to learn the patterns in five times fewer iterations than with a population encoding used with SpikeProp (Bohte et al., 2002).

When we move from rate-coded neurons to spiking neurons, the question about the encoding of patterns arises. One encoding was proposed by Bohte et al. (2002), where logical 0 and 1 are associated with the timing of early and late spikes, respectively (latency encoding). As the input neurons' activity is very sparse, the spikes must be multiplied over the simulated time period to generate enough activity in the hidden layer to support firing of the output neuron at defined times. This is achieved by having multiple subconnections for each input neuron that replicate the action potential with different delays. These additional subconnections, each with a different synaptic strength, require additional training. This encoding also requires an additional input neuron to set the reference start time (Sporea & Grüning, 2011). The alternative to this latency encoding is to use spike trains over a group of neurons as patterns. Thus, a pattern is represented by the (multiple) firing times of a group of input (and output) neurons. In order to guarantee that a set of weights exists for an arbitrary target mapping without replicating the input signals as above, a relatively large number of input neurons must be considered. As the input pattern is distributed over several spike trains, some of the information might be redundant and would not have a major contribution to the output, but then only some of the delayed subconnections in the latency encoding scheme have a major contribution. Finally, such an encoding does not require an additional input neuron to designate the reference start time, as the patterns are encoded in the relative timing of the spikes. The experiment in section 5.4 shows that this encoding can be successfully used for an originally linearly nonseparable problem.

In sections 5.5 and 5.6, the target patterns are generated as the output signals of networks with random weights. Again, encodings are sparse, and the corresponding pattern pairs are often locally linearly separable. The network is able to learn these transformations very fast—most of them in fewer than 10 learning iterations. During the learning process, the weights are modified in order to correctly map all input into output patterns so that they can be correctly classified, as seen in Figures 4c and 4d. In the task in section 5.5, where the network is trained on 10 spike-timing pattern pairs, the learning algorithm converges with a higher rate as the hidden layer increases in size.

The simulations where noise was added to the spike-timing patterns show that the learning is robust to the variability of spike timing. A spiking neural network trained on 10 noise-free patterns can recognize more than 20% of noisy patterns if the timing of spikes is shifted following a gaussian distribution with standard deviation up to 10 ms (see section 5.5 and Figure 4a). And when the network is trained on 3 noisy patterns, it can recognize more than 50% of noisy patterns where the timing of spikes is

moved within a gaussian distribution with standard deviation 10 ms (see section 5.6 and Figure 4b).

Another advantage of the learning rule is the introduction of synaptic scaling. First, it solves the problem of finding the optimal range for weight initialization. This problem is acknowledged as critical for the convergence of the learning (Bohte et al., 2002). Second, synaptic scaling maintains the firing activity of neurons in the hidden and output layers within an optimal range during the learning process. Although the firing rate of the output and hidden neurons is also adjusted by the noncorrelative term a in equations 3.16 and 3.27, this is done only when the output firing rate does not match the target firing rate exactly. This can cause hidden neurons to become quiescent (neurons that do not fire any spike) during the learning process and not to contribute to the activity of the output neurons. Synaptic scaling eliminates this problem by setting a minimum firing rate.

Appendix A: Neuron Model

The computing units of the feedforward network used in all simulations are described by the Spike Response Model (SRM) (Gerstner, 2001). SRM considers the spiking neuron as a homogeneous unit that fires an action potential, or a spike, when the total excitation reaches a certain threshold, ϑ. The neuron is characterized by a single variable, the membrane potential, $u(t)$ at time t.

The emission of an action potential can be described by a threshold process as follows. The spike is triggered if the membrane potential $u(t)$ of neuron reaches the threshold ϑ at time t^f:

$$u(t^f) = \vartheta \quad \text{and} \quad \frac{d}{dt}u(t^f) > 0. \tag{A.1}$$

In the case of a single neuron j receiving input from a set of presynaptic neurons $i \in \Gamma_j$, the state of the neuron is described as follows,

$$u_j(t) = \eta\left(t - t_j^f\right) + \sum_{i \in I} \sum_k w_{ji} y_i, \tag{A.2}$$

where y_i is the spike response function of the presynaptic neuron $i \in I$ and w_{ji} is the weight between neurons i and j; t_j^f is the last firing time of neuron j. The kernel $\eta(t)$ includes the form of the action potential as well as the after-potential:

$$\eta(t) = -\vartheta \exp\left(-\frac{t}{\tau_r}\right), \tag{A.3}$$

where $\tau_r > 0$ is the membrane time constant, with $\eta(t) = 0$ for $t \leq 0$.

The unweighted contribution of a single synaptic to the membrane potential is given by

$$y_i^k(t) = \sum_f \varepsilon\left(t - t_i^f\right), \tag{A.4}$$

with $\varepsilon(t)$ is the spike response function with $\varepsilon(t) = 0$ for $t \leq 0$. The times t_i^f represent the firing times of neuron i. In our case, the spike response function $\varepsilon(t)$ describes a standard postsynaptic potential,

$$\varepsilon(t) = \frac{t}{\tau} \exp\left(1 - \frac{t}{\tau}\right), \tag{A.5}$$

where $\tau > 0$ models the membrane potential time constant and determines the rise and decay of the function.

Appendix B: Detailed Results

In this appendix we present the detailed results of all simulations described in section 5. For all simulations, the averaged number of iterations needed for convergence is calculated over the successful trials.

B.1 XOR Benchmark (Section 5.2). Table 3 shows the summarized results for the XOR benchmark where the learning algorithm is tested with a different values for learning parameters A_+ and A_-. Table 4 shows the convergence rate and the average number of iterations for the XOR benchmark when the network is tested with a different number of delayed subconnections.

B.2 Iris Data Set (Section 5.3). Table 5 shows the convergence rate and the average number of iterations for the Iris data set when the network is tested with 8 to 12 subconnections.

B.3 The XOR Task with Spike Train Patterns (Section 5.4). Table 6 shows the convergence rate and the average number of iterations for this task when the network is different hidden layer sizes.

B.4 Learning Sets of Temporal Patterns (Section 5.5). Table 7 shows the summarized results for the task trained with noise-free patterns. In Table 7(left), the network is trained with 10 pattern pairs and different hidden layer sizes, while in Table 7(right), the network is trained with

Table 3: Summarized Results for the XOR Problem.

A_+	Successful Trials (%)	Average Number of Iterations	A_-	Successful Trials (%)	Average Number of Iterations
0.5	97	331 ± 46	0.00	97	231 ± 30
0.6	98	232 ± 24	0.10	98	196 ± 24
0.7	95	262 ± 38	0.20	96	157 ± 16
0.8	97	144 ± 35	0.30	96	187 ± 28
0.9	96	184 ± 23	0.40	95	204 ± 37
1.0	96	204 ± 34	0.50	98	137 ± 16
1.1	92	166 ± 27	0.60	96	207 ± 31
1.2	96	207 ± 31	0.70	95	191 ± 33
1.3	95	174 ± 30	0.80	98	185 ± 31
1.4	97	183 ± 28	0.90	86	203 ± 31
1.5	93	204 ± 36	1.00	88	200 ± 30
1.6	93	273 ± 43	1.10	80	257 ± 33
1.7	96	163 ± 27	1.20	70	349 ± 42
1.8	94	181 ± 32	1.30	65	382 ± 30
1.9	95	221 ± 32	1.40	45	353 ± 28
2.0	89	141 ± 18	1.50	56	492 ± 32

Notes: (Left) The parameters A_+ and A_- are varied in order to determine the best values for faster convergence. The ratio between these parameters is constant $A_+ = 2A_-$. (Right) While keeping $A_+ = 1.2$ fixed, A_- is varied in order to determine the best ratio between these parameters.

Table 4: Summarized Results for the XOR Problem.

Subconnections	Successful Trials (%)	Average Number of Iterations
4	11	63 ± 20
6	24	169 ± 37
8	73	192 ± 27
10	81	154 ± 17
12	96	207 ± 31
14	96	309 ± 52
16	73	472 ± 56

Note: The number of delayed subconnections is varied while keeping the learning parameters fixed $A_+ = 1.2$ and $A_- = 0.6$.

different number of pattern pairs, keeping the size of the hidden layer fixed to 260 spiking neurons.

B.5 Learning to Generalize (Section 5.6). Table 8 shows the summarized results for this task, where noise is added to the training patterns. Different noise levels are considered, with input jitters varying from 0 ms to 4 ms.

Table 5: Summarized Results for the Iris Data Set.

Subconnections	Successful Trials	Average Number of Iterations	Accuracy on the Training Set (%)	Accuracy on the Testing Set (%)
8	68	125 ± 12	97 ± 0.17	89 ± 0.69
9	80	174 ± 16	96 ± 0.00	94 ± 0.79
10	80	114 ± 13	97 ± 0.00	89 ± 0.47
11	74	140 ± 15	96 ± 0.16	86 ± 0.49
12	68	183 ± 21	96 ± 0.17	91 ± 0.69

Table 6: Summarized Results for a Linearly Nonseparable Problem.

Hidden Neurons	Successful Trials (%)	Average Number of Iterations
50	70	293 ± 59
60	54	301 ± 66
70	56	327 ± 91
80	60	469 ± 87
90	76	247 ± 42
100	76	439 ± 73

Table 7: Summarized Results.

Number of Hidden Units	Successful Trials (%)	Average Number of Iterations	Number of Patterns	Successful Trials (%)	Average Number of Iterations
200	50	5 ± 0.8	5	100	7 ± 0.7
210	52	6 ± 1.2	6	92	5 ± 0.6
220	78	5 ± 0.6	7	96	5 ± 1.2
230	76	6 ± 1.1	8	92	8 ± 1.5
240	80	5 ± 0.6	9	88	7 ± 0.9
250	74	7 ± 0.8	10	90	6 ± 0.6
260	90	5 ± 0.7	11	72	6 ± 0.7
270	88	4 ± 0.5	12	72	6 ± 0.7
280	80	7 ± 2.4	13	58	5 ± 0.9
290	90	4 ± 0.6	14	40	6 ± 0.9
300	90	4 ± 0.4	15	34	5 ± 1.0

Notes: (Left) The network is trained with 10 pattern pairs, where the size of the hidden layer is varied in order to determine the best network architecture. (Right) A neural network with a hidden layer containing 260 neurons is trained with different numbers of pattern pairs.

Table 8: Summarized Results for Learning with Noisy Patterns.

Input Jitter During Learning	Successful Trials (%)	Average Number of Iterations
0	96	10 ± 1.2
1	98	12 ± 1.1
2	95	19 ± 2.3
3	66	26 ± 5.6
4	64	115 ± 51

Note: The input patterns jitter is varied between 0 ms and 4 ms, while the target jitter is always 1 ms.

Acknowledgments

We thank two anonymous reviewers whose comments helped improve the paper signficantly. A.G. was supported in part by Engineering and Physical Sciences Research Council (UK) grant EP/I014934/1.

References

Bohte, S. (2011). Error-backpropagation in networks of fractionally predictive spiking neurons. In *Proceedings of the 21th International Conference on Artificial Neural Networks* (pp. 60–68). Berlin: Springer-Verlag.

Bohte, S., Kok, J., & Poutré, H. L. (2002). Error backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*, 17–37.

Bohte, S., & Rombouts, J. (2010). Fractionally predictive spiking neurons. In J. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R. S. Zemel, & A. Culotta (Eds.), *Advances in neural information processing, systems*, *23* (pp. 253–261). Red Hook, NY: Curran.

Booij, O., & Nguyen, H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, *95*, 552–558.

deCharms, R. C., & Merzenich, M. M. (1996). Primary cortical representation of sounds by the coordination of action-potential timing. *Nature*, *381*, 610–613.

Elias, J. G., & Northmore, D.P.M. (2002). Building silicon nervous systems with dendritic tree neuromorphs. In W. Maass & C. M. Bishop (Eds.), *Pulsed neural networks*. Cambridge, MA: MIT Press.

Fisher, R. A. (1936). The use of multiple Measurements in taxonomic problems. *Annals of Eugenics*, *7*, 179–188.

Fitzsimonds, R. M., Song, H., & Poo, M. (1997). Propagation of activity dependent synaptic depression in simple neural networks. *Nature*, *388*, 439–448.

Fujita, M., Takase, H., Kita, H., & Hayashi, T. (2008). Shape of error surfaces in SpikeProp. In *Proceedings of IEEE International Joint Conference on Neural Networks* (pp. 840–844). Piscataway, NJ: IEEE.

Gerstner, W. (2001). A framework for spiking neuron models: The spike response model. In F. Moss & S. Gielen (Eds.), *The handbook of biological physics*, *Vol. 4* (pp. 469–516). Amsterdam: North-Holland.

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge: Cambridge University Press.

Ghosh-Dastidar, S., & Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Networks*, *22*, 1419–1431.

Glackin, C., Maguire, L., McDaid, L., & Sayers, H. (2011). Receptive field optimisation and supervision of a fuzzy spiking neural network. *Neural Networks*, *24*, 247–256.

Grüning, A. (2007). Elman backpropagation as reinforcement for simple recurrent networks. *Neural Computation*, *19*, 3108–3131.

Grüning, A., & Sporea, I. (2012). Supervised learning of logical operations in layered spiking neural networks with spike train encoding. *Neural Processing Letters*, *36*, 117–134. doi:10.1007/s11063-012-9225-1.

Gütig, R., & Sompolinsky, H. (2006). The tempotron: A neuron that learns spike timing-based decisions. *Nature Neuroscience*, *9*, 420–428.

Harris, K. D. (2008). Stability of the fittest: Organizing learning through retroaxonal signals. *Trends in Neuroscience*, *31*, 130–136.

Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

Johansson, R. S., & Birznieks, I. (2004). First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature Neuroscience*, *7*, 170–177.

Legenstein, R., Naeger, C., & Maass, W. (2005). What can a neuron learn with spike-timing-dependent plasticity? *Neural Computation*, *17*, 2337–2382.

Knudsen, E. I. (1994). Supervised learning in the brain. *Journal of Neuroscience*, *14*, 3985–3997.

Knudsen, E. I. (2002). Instructed learning in the auditory localization pathway of the barn owl. *Nature*, *417*(6886), 322–328.

Maass, W. (1997a). Networks of spiking neurons: The third generation of neural network models. *Transactions of the Society for Computer Simulation International*, *14*, 1659–1671.

Maass, W. (1997b). Fast sigmoidal networks via spiking neurons. *Neural Computation*, *9*, 279–304.

Mauk, M. D., & Buonomano, D. V. (2004). The neural basis of temporal processing. *Annual Rev. Neuroscience*, *27*, 304–340.

McKennoch, S., Voegtlin, T., & Bushnell, L. (2009). Spike-timing error backpropagation in theta neuron networks. *Neural Computation*, *21*, 9–45.

Neuenschwander, S., & Singer, W. (1996). Long-range synchronization of oscillatory light responses in the cat retina and lateral geniculate nucleus. *Nature*, *379*, 728–733.

Ponulak, F. (2006). *ReSuMe–Proof of convergence*. http://d1.cie.put.poznan.pl/dav/fp/FP_ConvergenceProof_TechRep.pdf

Ponulak, F. (2008). Analysis of the ReSuMe learning process for spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, *18*, 117–127.

Ponulak, F., & Kasiński, A. (2010). Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting. *Neural Computation*, *22*, 467–510.

Roelfsema, P. R., & van Ooyen, A. (2005). Attention-gated reinforcement learning of internal representations for classification. *Neural Computation*, *17*, 1–39.

Rojas, R. (1996). *Neural networks: A systematic introduction*. Berlin: Springer-Verlag.

Rostro-Gonzalez, H., Vasquez-Betancour, J. C., Cessac, B., & Viéville, T. (2010). *Reverse-engineering in spiking neural network parameters: Exact deterministic parameter estimation*. Sophia Antipolis, France: INRIA.

Ruf, B., & Schmitt, M. (1997). Learning temporally encoded patterns in networks of spiking neurons. *Neural Processing Letters*, *5*(1), 9–18.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition*. Cambridge, MA: MIT Press.

Schrauwen, B., & van Campenhout, J. (2004). Improving spike-prop: Enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC Workshop*.

Shepard, J. D., Rumbaugh, G., Wu, J., Chowdhiry, S., Plath, N., Kuhl, D., et al. (2006). Arc mediates homoeostatic synaptic scaling of AMPA receptors. *Neuron*, *52*, 475–484.

Sporea, I., & Grüning, A. (2011). Reference time in SpikeProp. In *Proceedings of IEEE International Joint Conference Neural Networks* (pp. 1090–1092). Piscataway, NJ: IEEE.

Takase, H., Fujita, M., Kawanaka, H., Tsuruoka, S., Kita, H., & Hayashi, T. (2009). Obstacle to training SpikeProp networks: Cause of surges in training process. In *Proceedings of IEEE International Joint Conference Neural Networks* (pp. 3062–3066). Piscataway, NJ: IEEE.

Tao, H. W., Zhang, L. I, Bi, G. Q., & Poo, M. (2000). Selective presynaptic propagation of long-term potentiation in defined neural Networks. *Journal of Neuroscience*, *20*, 3233–3243.

Thorpe, S. T., & Imbert, M. (1989). Biological constraints on connectionist modelling. In , R. Pfeifer, Z. Schreter, F. Fogelman-Souli, & L. Steels (Eds.), *Connectionism in perspective* (pp. 63–92). New York: Elsevier Science.

Tiňo, P., & Mills, A. J. (2005). Learning beyond finite memory in recurrent networks of spiking neurons. In L. Wang, K. Chen, & Y. Ong (Eds.), *Advances in natural computation* (pp. 666–675). Berlin: Springer-Verlag.

Urbanczik, R., & Senn, W. (2009). Reinforcement learning in populations of spiking neurons. *Nature Neuroscience*, *12*, 250–252.

van Rossum, M. C. (2001). A novel spike distance. *Neural Computation*, *13*, 751–763.

Wade, J. J., McDaid, L. J., Santos, J. A., & Sayers, H. M. (2010). SWAT: A spiking neural network training algorithm for classification problems. *IEEE Transactions on Neural Networks*, *21*, 1817–1829.

Watt, A. J., & Desai, N. S. (2010). Homeostatic plasticity and STDP: Keeping a neuron's cool in a fluctuating world. *Frontiers in Synaptic Neuroscience*, *2*(5).

Wehr, M., & Laurent, G. (1996). Odour encoding by temporal sequences of firing in oscillating neural assemblies. *Nature*, *384*, 162–166.

Xin, J., & Embrechts, M. J. (2001). Supervised learning with spiking neuron networks. In *Proceedings of IEEE International Joint Conference Neural Networks* (pp. 1772–1777). Piscataway, NJ: IEEE.

Received February 10, 2012; accepted August 9, 2012.