Gradient Learning in Networks of Smoothly Spiking Neurons

Jiří Šíma*

Institute of Computer Science, Academy of Sciences of the Czech Republic,
P. O. Box 5, 18207 Prague 8, Czech Republic, sima@cs.cas.cz

Abstract. A slightly simplified version of the Spike Response Model SRM_0 of a spiking neuron is tailored to gradient learning. In particular, the evolution of spike trains along the weight and delay parameter trajectories is made perfectly smooth. For this model a back-propagation-like learning rule is derived which propagates the error also along the time axis. This approach overcomes the difficulties with the discontinuous-in-time nature of spiking neurons, which encounter previous gradient learning algorithms (e.g. *SpikeProp*). The new algorithm can naturally cope with multiple spikes and preliminary experiments confirm the smoothness of spike creation/deletion process.

1 Learning in Networks of Spiking Neurons

The prominent position among neural network models has recently been occupied by networks of spiking (pulse) neurons [4]. As compared to the traditional perceptrons the spiking neurons represent a biologically more plausible model which has a great potential for processing the temporal information. However, one of the main open issues is the development of a practical learning algorithm for the networks of spiking neurons although the related training problem is known to be NP-complete at least for binary coded data [6, 11].

Learning in the perceptron networks is usually performed by a gradient descent method, e.g. by using the back-propagation algorithm which explicitly evaluates the gradient of an error function. For the first time, the same approach was employed for spiking networks in the *SpikeProp* algorithm [2] which learns the desired firing times of output neurons by adapting the weight parameters in the Spike Response Model SRM_0 [4]. Plenty of experiments with *SpikeProp* was carried out which clarified e.g. the role of the parameter initialization and negative weights [7]. The performance of the original algorithm was improved by adding the momentum term [14]. Moreover, the *SpikeProp* was further enhanced with additional learning rules for synaptic delays, thresholds, and time constants [8] which resulted in faster convergence and smaller network sizes for given learning tasks. An essential speedup was achieved by approximating the

* Research partially supported by projects 1ET100300517 (AS CR), 1M0545 (MŠMT CR), and AV0Z10300504 (ICS AS CR). The author also thanks Petra Vídnerová for her help with computer experiments.

firing time function using the logistic sigmoid [1]. The *SpikeProp* algorithm was also extended to recurrent network architectures [13].

Nevertheless, the *SpikeProp* method and its modifications do not usually allow more than one spike per neuron which makes it suitable only for 'time-to-first-spike' coding scheme [12]. Another important drawback of these learning heuristics is that the adaptation mechanism fails for the weights of neurons that do not emit spikes. At the core of these difficulties the fact is that the spike creation or removal due to weight updates is by nature a very discontinuous process while in contrast the gradient learning algorithms do essentially require the error function to be perfectly smooth for their implementations. Recently, the so-called *ASNA-Prop* has been proposed [9] to solve this problem by emulating the feedforward networks of spiking neurons with the discrete-time analog sigmoid networks with local feedback, which is then used for deriving the gradient learning rule. Another method estimates the gradient by measuring the fluctuations in the error function in response to the dynamic neuron parameter perturbation [3]. See also paper [5] for a recent review of supervised learning methods in the spiking neural networks.

In the present paper we employ a different approach to this problem of spike creation/deletion. Similarly as the Heaviside function was replaced by the logistic sigmoid in the conventional back-propagation algorithm to make the neuron function differentiable, we modify a slightly simplified version of the Spike Response Model SRM_0 by smoothing out the discontinuities also along the weight and delay parameter trajectories (Section 2). Thus, a new spike arises through a continuous "division" of the preceding spike into two spikes while the spike disappearance is implemented by a continuous "fusion" with its predecessor. For our model of smoothly spiking neurons we derive a nontrivial back-propagation-like rule for computing the gradient of the error function with respect to both the weight and delay parameters which is only sketched in Section 3 due to lack of space while the complete formulas can be found in our technical report [10]. In particular, the chain rule for computing the partial derivatives of the composite error function is implemented so that each neuron propagates backwards a variable number of partial derivative terms corresponding to different time instants including the second-order derivative terms. Thus, the new gradient learning method can naturally cope with multiple spikes whose number changes in time and can be used for supervised learning of desired spike trains in the networks of spiking neurons. Preliminary experiments with the proposed learning algorithm exhibit the smoothness of spike creation/deletion process (Section 4).

2 A Feedforward Network of Smoothly Spiking Neurons

Formally, a feedforward network can be defined as a set of *spiking (pulse) neurons* V which are densely connected into a *directed* (connected) graph representing an *architecture* of the network. Some of these neurons may serve as external inputs or outputs, and hence we assume $X \subseteq V$ and $Y \subseteq V$ to be a set of *input* and *output* neurons, respectively, while the remaining ones are called *hidden* neurons.

We denote by $j_{\leftarrow}$ the set of all neurons from which an edge leads to j while $j^{\rightarrow}$ denotes the set of all neurons to which an edge leads from j. As usual we assume $j_{\leftarrow} = \emptyset$ for $j \in X$ and $j^{\rightarrow} = \emptyset$ for $j \in Y$ whereas $j_{\leftarrow} \neq \emptyset$ and $j^{\rightarrow} \neq \emptyset$ for $j \in V \setminus (X \cup Y)$. Each edge in the architecture leading from neuron i to j is labeled with a real *(synaptic) weight* w_{ji} and *delay* $d_{ji} \geq 0$. In addition, a real *bias* parameter w_{j0} is assigned to each noninput neuron $j \in V \setminus X$ which can be viewed as the weight from a formal constant unit input[1]. All these weights and delays altogether create the network parameter vectors $\mathbf{w}$ and $\mathbf{d}$, respectively.

We first define an auxiliary function that will be used for making the computational dynamics of the network smooth with respect to both the time and parameters $\mathbf{w}$ and $\mathbf{d}$. In particular, a twice differentiable nondecreasing function of one variable $\sigma(\alpha, \beta, \delta) : \Re \longrightarrow [\alpha, \beta]$ (or σ for short) is introduced which has three real parameters $\alpha \leq \beta$ and $\delta > 0$ such that $\sigma(x) = \alpha$ for $x \leq 0$ and $\sigma(x) = \beta$ for $x \geq \delta$ whereas the first and second derivatives satisfy $\sigma'(0) = \sigma'(\delta) = \sigma''(0) = \sigma''(\delta) = 0$. In fact, σ is a smooth approximation of the step function where δ controls the approximation error. In order to fulfill the conditions on the derivatives we can employ, e.g., the primitive function $\int Ax^2(x-\delta)^2\,dx = A\left(\frac{x^5}{5} - 2\delta\,\frac{x^4}{4} + \delta^2\,\frac{x^3}{3}\right) + C$ for a normalized $\sigma_0 = \sigma(0, 1, \delta)$ on $[0, \delta]$ where the real constants $C = 0$ and $A = 30/\delta^5$ are determined from $\sigma_0(0) = 0$ and $\sigma_0(\delta) = 1$. Thus we can choose $\sigma(\alpha, \beta, \delta\,; x) = (\beta - \alpha)\,\sigma_0(x) + \alpha$ for $x \in [0, \delta]$ which results in the following definition:

$$\sigma(\alpha, \beta, \delta\,; x) = \begin{cases} \alpha & \text{for } x < 0 \\ (\beta - \alpha)\left(\left(6\frac{x}{\delta} - 15\right)\frac{x}{\delta} + 10\right)\left(\frac{x}{\delta}\right)^3 + \alpha & \text{for } 0 \leq x \leq \delta \\ \beta & \text{for } x > \delta\,. \end{cases} \tag{1}$$

We will also use the logistic sigmoid function with a gain parameter λ (e.g. $\lambda = 4$):

$$P(x) = \frac{1}{1 + e^{-\lambda x}}\,. \tag{2}$$

Now we introduce the smooth computational dynamics. Each spiking neuron $j \in V$ in the network may produce a sequence of p_j *spikes (firing times)* $0 < t_{j1} < t_{j2} < \cdots < t_{jp_j} < T$. In addition, define formally $t_{j0} = 0$ and $t_{j,p_j+1} = T$. For every input neuron $j \in X$, this sequence of spikes is given externally as a global input to the network. For a noninput neuron $j \in V \setminus X$, on the other hand, the underlying firing times are computed as the time instants at which its *excitation*

$$\xi_j(t) = w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji}\,\varepsilon\left(t - d_{ji} - \tau_i(t - d_{ji})\right) \tag{3}$$

evolving in time $t \in [0, T]$ crosses 0 from below, that is,

$$\left\{0 \leq t \leq T \,|\, \xi_j(t) = 0 \,\&\, \xi_j'(t) > 0\right\} = \left\{t_{j1} < t_{j2} < \cdots < t_{jp_j}\right\}\,. \tag{4}$$

[1] For simplicity, we assume a *constant* threshold function (which equals the opposite value of the bias), thus excluding the refractory effects [4].

Formula (3) defines the excitation of neuron $j \in V \setminus X$ at time t as a weighted sum of responses to the last spikes from neurons $i \in j_{\leftarrow}$ delayed by d_{ji} preceding time instant t. Here $\varepsilon : \Re \longrightarrow \Re$ denotes the smooth *response function* which is the same for all neurons, e.g.

$$\varepsilon(t) = e^{-(t-1)^2} \cdot \sigma_0(t) \qquad \text{where} \qquad \sigma_0(t) = \sigma(0, 1, \delta_0\,;\, t) \tag{5}$$

is defined by (1) using parameter δ_0 particularly for the definition of ε. Clearly, ε is a twice differentiable function with a relatively flat spike shape which reaches its maximum $\varepsilon(1) = 1$ for $t = 1$. Furthermore, $\tau_i : \Re \longrightarrow \Re$ is a smooth approximation of the stair function that gives the last firing time t_{is} of neuron i preceding a given time t, that is $t_{is} < t \leq t_{i,s+1}$.

In particular, we define the function τ_j for any neuron $j \in V$ as follows. The firing times $t_{j1} < t_{j2} < \cdots < t_{jp_j}$ of j are first transformed one by one into $\widetilde{t_{j0}} = 0 < \widetilde{t_{j1}} < \widetilde{t_{j2}} < \cdots < \widetilde{t_{jp_j}} < T = \widetilde{t_{j,p_j+1}}$ by formula

$$\widetilde{t_{js}} = \begin{cases} t_{js} & \text{for } j \in X \\ \sigma\left(\widetilde{t_{j,s-1}}, t_{js}, \delta\,;\, \xi_j'(t_{js})\right) & \text{for } j \in V \setminus X \end{cases} \tag{6}$$

using (1) where s goes in sequence from 1 to p_j and $\xi_j'(t_{js})$ for $j \in V \setminus X$ is the derivative of excitation ξ_j at time t_{js} which is positive according to (4). Clearly, $\widetilde{t_{js}} = t_{js}$ for $\xi_j'(t_{js}) \geq \delta$ while $\widetilde{t_{js}} \in (\widetilde{t_{j,s-1}}, t_{js})$ is smoothly growing with increasing small $\xi_j'(t_{js}) \in (0, \delta)$. The purpose of this transformation is to make the creation and deletion of spikes smooth with respect to the weight and delay updates. In particular, by moving along the weight and delay trajectories, the excitation ξ_j may reach and further cross 0 from below (spike creation), which leads to an increase in the first derivative of excitation $\xi_j'(t_{js})$ at the crossing point t_{js} starting from zero. Thus, the new transformed spike $\widetilde{t_{js}}$ arises through a continuous "division" of the preceding transformed spike $\widetilde{t_{j,s-1}}$ into two spikes while the spike disappearance is implemented similarly by a continuous "fusion" with its predecessor, which is controlled by the first derivative of the excitation. The transformed spikes then define

$$\tau_j(t) = \sum_{s=1}^{p_j+1} \left(\widetilde{t_{js}} - \widetilde{t_{j,s-1}}\right) P^c\left(t - \widetilde{t_{js}}\right) \tag{7}$$

using the logistic sigmoid (2) where $c \geq 1$ (e.g. $c = 3$) is an optional exponent whose growth decreases the value of $P(0)$ shifting the transient phase of $P(x)$ to positive x. Finally, the derivative of j's excitation with respect to time t is calculated as

$$\xi_j'(t) = \sum_{i \in j_{\leftarrow}} w_{ji}\, \varepsilon'\left(t - d_{ji} - \tau_i(t - d_{ji})\right)\left(1 - \tau_i'(t - d_{ji})\right) \tag{8}$$

where $\varepsilon'(t) = e^{-(t-1)^2} \cdot (\sigma_0'(t) - 2(t-1)\sigma_0(t))$ and

$$\tau_i'(t) = \frac{\partial}{\partial t}\tau_i(t) = c\lambda \sum_{s=1}^{p_i+1} \left(\widetilde{t_{is}} - \widetilde{t_{i,s-1}}\right) P^c\left(t - \widetilde{t_{is}}\right)\left(1 - P\left(t - \widetilde{t_{is}}\right)\right). \tag{9}$$

3 The Back-Propagation Rule

A *training pattern* associates a global temporal input specifying the spike trains $0 < t_{i1} < t_{i2} < \cdots < t_{ip_i} < T$ for every input neuron $i \in X$ with corresponding sequences of desired firing times $0 < \varrho_{j1} < \varrho_{j2} < \cdots < \varrho_{jq_j} < T$ prescribed for all output neurons $j \in Y$. The discrepancy between the desired and actual sequences of spikes for the underlying global temporal input can be measured by the following L_2 *error*

$$E(\mathbf{w}, \mathbf{d}) = \frac{1}{2} \sum_{j \in Y} \sum_{s=0}^{q_j} \left(\tau_j\left(\varrho_{j,s+1}\right) - \varrho_{js}\right)^2 \tag{10}$$

which is a function of the weight and delay parameters $\mathbf{w}$ and $\mathbf{d}$, respectively. In particular, $\tau_j(\varrho_{j,s+1})$ is the smooth approximation of the last firing time of output neuron $j \in Y$ preceding time instant $\varrho_{j,s+1}$ which is desired to be ϱ_{js}.

We will derive a back-propagation-like rule for computing the gradient of the error function (10) which can then be minimized using a gradient descent method, e.g. for every $j \in V$ and $i \in j_{\leftarrow}$ (or $i = 0$ for bias w_{j0})

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} - \alpha \frac{\partial E}{\partial w_{ji}}\left(\mathbf{w}^{(t-1)}\right), \quad d_{ji}^{(t)} = d_{ji}^{(t-1)} - \alpha \frac{\partial E}{\partial d_{ji}}\left(\mathbf{d}^{(t-1)}\right) \tag{11}$$

starting with suitable initial paramater values $\mathbf{w}^{(0)}, \mathbf{d}^{(0)}$ where $0 < \alpha < 1$ is a learning rate. Unlike the conventional back-propagation learning algorithm for the perceptron network, the new rule for the network of spiking neurons must take also the temporal dimension into account. In particular, the chain rule for computing the partial derivatives of the composite error function $E(\mathbf{w}, \mathbf{d})$ requires each neuron to propagate backwards a certain number of partial derivative terms corresponding to different time instants. For this purpose, each noninput neuron $j \in V \setminus X$ stores a list P_j of m_j ordered triples $(\pi_{jc}, \pi'_{jc}, u_{jc})$ for $c = 1, \ldots, m_j$ where π_{jc}, π'_{jc} denote the values of derivative terms associated with time u_{jc} such that

$$\frac{\partial E}{\partial w_{ji}} = \sum_{c=1}^{m_j} \left(\pi_{jc} \cdot \frac{\partial}{\partial w_{ji}} \tau_j(u_{jc}) + \pi'_{jc} \cdot \frac{\partial}{\partial w_{ji}} \tau'_j(u_{jc})\right) \quad \text{for } i \in j_{\leftarrow} \cup \{0\} \tag{12}$$

(similarly for $\frac{\partial E}{\partial d_{ji}}$). Notice that the triples $(\pi_{jc_1}, \pi'_{jc_1}, u_{jc_1})$ and $(\pi_{jc_2}, \pi'_{jc_2}, u_{jc_2})$ such that $u_{jc_1} = u_{jc_2}$ can be merged into one $(\pi_{jc_1} + \pi_{jc_2}, \pi'_{jc_1} + \pi'_{jc_2}, u_{jc_1})$ and also the triples $(\pi_{jc}, \pi'_{jc}, u_{jc})$ with $\pi_{jc} = \pi'_{jc} = 0$ can be omitted.

For any noninput neuron $j \in V \setminus X$ we will calculate the underlying derivative terms π_{jc}, π'_{jc} at required time instants u_{jc}. For an output neuron $j \in Y$ the list

$$P_j = \left(\left(\tau_j(\varrho_{j,s+1}) - \varrho_{js}\,,\, 0\,,\, \varrho_{j,s+1}\right),\, s = 0, \ldots, q_j\right) \tag{13}$$

is obtained directly from (10). For creating P_i for a hidden neuron $i \in j_{\leftarrow}$ for some $j \in V \setminus X$, we derive a recursive procedure using the partial derivatives

$$\frac{\partial}{\partial w_{i\ell}} \tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(t) \cdot \frac{\partial \widetilde{t_{js}}}{\partial w_{i\ell}}, \quad \frac{\partial}{\partial w_{i\ell}} \tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(t) \cdot \frac{\partial \widetilde{t_{js}}}{\partial w_{i\ell}} \tag{14}$$

where $\frac{\partial}{\partial\,\widetilde{t_{js}}}\tau_j(t)$ and $\frac{\partial}{\partial\,\widetilde{t_{js}}}\tau'_j(t)$ are calculated by differentiating (7) and (9), respectively. Furthermore,

$$\frac{\partial\,\widetilde{t_{js}}}{\partial\,w_{i\ell}} = \frac{\partial\,\widetilde{t_{js}}}{\partial\,\widetilde{t_{j,s-1}}} \cdot \frac{\partial\,\widetilde{t_{j,s-1}}}{\partial\,w_{i\ell}} + \left(\frac{\partial\,\widetilde{t_{js}}}{\partial\,t_{js}} \cdot \frac{\partial\,t_{js}}{\partial\,\tau_i} + \frac{\partial\,\widetilde{t_{js}}}{\partial\,\xi'_j} \cdot \frac{\partial\,\xi'_j}{\partial\,\tau_i}\right)\frac{\partial\,\tau_i}{\partial\,w_{i\ell}} + \frac{\partial\,\widetilde{t_{js}}}{\partial\,\xi'_j} \cdot \frac{\partial\,\xi'_j}{\partial\,\tau'_i} \cdot \frac{\partial\,\tau'_i}{\partial\,w_{i\ell}} \tag{15}$$

according to (6) and (8) where the particular partial derivatives can simply be calculated by differentiating the underlying formulas, e.g. using (1), except for $\frac{\partial\,t_{js}}{\partial\,\tau_i}$ whose calculation follows. According to (3) and (4), the dependence of t_{js} on τ_i can only be expressed implicitly as $\xi_j(t_{js}) = 0$ where $\xi'_j(t_{js}) \neq 0$, which implies the total differential identity

$$\xi'_j(t_{js})\,dt_{js} + \frac{\partial\,\xi_j}{\partial\,w_{i\ell}}\,dw_{i\ell} = 0 \tag{16}$$

employing e.g. the variable $w_{i\ell}$ for which $\frac{\partial\,\tau_k}{\partial\,w_{i\ell}} = 0$ for $k \in j_{\leftarrow}$ unless $i = k$. Hence, $\xi'_j(t_{js}) \cdot \frac{\partial\,t_{js}}{\partial\,w_{i\ell}} = -\frac{\partial\,\xi_j}{\partial\,w_{i\ell}} = -\frac{\partial\,\xi_j}{\partial\,\tau_i} \cdot \frac{\partial\,\tau_i}{\partial\,w_{i\ell}}$ which gives

$$\frac{\partial\,t_{js}}{\partial\,\tau_i} = \frac{\frac{\partial\,t_{js}}{\partial\,w_{i\ell}}}{\frac{\partial\,\tau_i}{\partial\,w_{i\ell}}} = \frac{-\frac{\partial\,\xi_j}{\partial\,\tau_i}}{\xi'_j(t_{js})} = \frac{w_{ji}\,\varepsilon'(t_{js} - d_{ji} - \tau_i(t_{js} - d_{ji}))}{\xi'_j(t_{js})}\,. \tag{17}$$

Now, the formula (15) for the derivative $\frac{\partial\,\widetilde{t_{js}}}{\partial\,w_{i\ell}}$ in terms of $\frac{\partial\,\widetilde{t_{j,s-1}}}{\partial\,w_{i\ell}}$ is applied recursively, which is further plugged into (14) as follows:

$$\frac{\partial}{\partial\,w_{i\ell}}\tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial\,\widetilde{t_{js}}}\tau_j(t) \sum_{r=s-n_{js}}^{s} \left(\prod_{q=r+1}^{s} \frac{\partial\,\widetilde{t_{jq}}}{\partial\,\widetilde{t_{j,q-1}}}\right) \times \left(\left(\frac{\partial\,\widetilde{t_{jr}}}{\partial\,t_{jr}} \cdot \frac{\partial\,t_{jr}}{\partial\,\tau_i} + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi'_j} \cdot \frac{\partial\,\xi'_j}{\partial\,\tau_i}\right)\frac{\partial\,\tau_i}{\partial\,w_{i\ell}} + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi'_j} \cdot \frac{\partial\,\xi'_j}{\partial\,\tau'_i} \cdot \frac{\partial\,\tau'_i}{\partial\,w_{i\ell}}\right) \tag{18}$$

(similarly for $\frac{\partial}{\partial\,w_{i\ell}}\tau'_j(t)$) where $1 \leq n_{js} \leq s-1$ is defined to be the least index such that $\frac{\partial\,\widetilde{t_{j,s-n_{js}}}}{\partial\,\widetilde{t_{j,s-n_{js}-1}}} = 0$ which exists since at least $\frac{\partial\,\widetilde{t_{j1}}}{\partial\,\widetilde{t_{j0}}} = 0$. Note that the product $\prod_{q=r+1}^{s} \frac{\partial\,\widetilde{t_{jq}}}{\partial\,\widetilde{t_{j,q-1}}}$ in formula (18) equals formally 1 for $r = s$. The summands of formula (18) are used for creating the list P_i for a hidden neuron $i \in V \setminus (X \cup Y)$ provided that the lists $P_j = ((\pi_{jc}, \pi'_{jc}, u_{jc})\,,\, c = 1, \ldots, m_j)$ have already been created for all $j \in i^{\rightarrow}$, that is

$$P_i = \left(f_{jcsr}\left(\frac{\partial\,\widetilde{t_{jr}}}{\partial\,t_{jr}} \cdot \frac{\partial\,t_{jr}}{\partial\,\tau_i} + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi'_j} \cdot \frac{\partial\,\xi'_j}{\partial\,\tau_i}\right)\,,\, f_{jcsr} \cdot \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi'_j} \cdot \frac{\partial\,\xi'_j}{\partial\,\tau'_i}\,,\, t_{jr} - d_{ji}\right) \tag{19}$$

including factors

$$f_{jcsr} = \left(\pi_{jc} \cdot \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(u_{jc}) + \pi'_{jc} \cdot \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(u_{jc}) \right) \prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \qquad (20)$$

for all $j \in i^{\rightarrow}$, $c = 1, \ldots, m_j$, $s = 1, \ldots, p_j$, and $r = s - n_{js}, \ldots, s$.

Thus, the back-propagation algorithm starts with output neurons $j \in Y$ for which the lists P_j are first created by (13) and further continues by propagating the derivative terms at various time instants backwards while creating the lists P_i also for hidden neurons $i \in V \setminus (X \cup Y)$ according to (19). These lists are then used for computing the gradient of the error function (10) according to (12) where the derivatives $\frac{\partial}{\partial w_{ji}} \tau_j(t)$ and $\frac{\partial}{\partial w_{ji}} \tau'_j(t)$ are calculated analogously to (18). For example, the dependencies of t_{jr} on w_{ji} can again be expressed only implicitly as $\xi_j(t_{jr}) = 0$ and hence, the derivatives $\frac{\partial t_{jr}}{\partial w_{ji}}$ are calculated using the implicit function theorem. See [10] for more details.

4 Preliminary Experiments and Future Work

We have implemented the proposed learning algorithm and performed several preliminary computer simulations with simple toy problems such as XOR with temporal encoding. These experiments also served as a tool for debugging our model of a smoothly spiking neuron, e.g. for choosing suitable function shapes, and this work has not been finished yet. Nevertheless, the first experimental results confirm the smoothness of spike creation/deletion. For example, Figure 1 shows how the graph of function $\tau_j(t)$ evolves in the course of weight and delay adaptation for a spiking neuron j. Recall $\tau_j(t)$ is the smooth approximation of the stair function which produces the last firing time preceding a given time t. Figure 1 depicts the process of a spike creation during training when the time instant of a new spike $\widetilde{t_{js}}$ "detaches" from the preceding spike $t_{j,s-1}$ (which, for

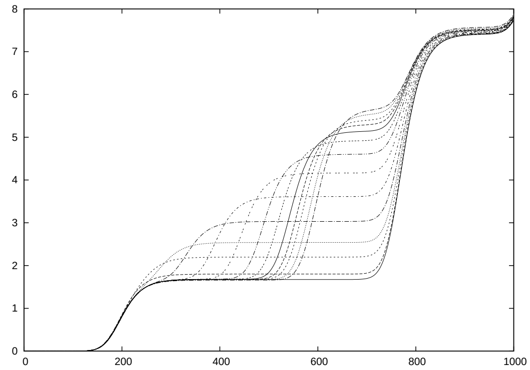

Fig. 1. Spike creation.

simplicity, is assumed here to be "fixed" for a moment, i.e. $\widetilde{t_{j,s-1}} = t_{j,s-1}$) and "moves" smoothly with increasing $\xi_j'(t_{js}) > 0$ to its actual position $\widetilde{t_{js}} = t_{js}$ ($\xi_j(t_{js}) = 0$) where $\xi_j'(t_{js})$ reaches threshold value δ. In a general case, more spikes can smoothly be generated at the same time which was also observed in our experiments.

Nevertheless, the proposed learning algorithm for networks of smoothly spiking neurons still needs to be justified by experiments with more complicated temporal training patterns. Another challenge for further research is to generalize the model for a nonconstant threshold function taking the refractory period of a spiking neuron into account.

References

1. Berkovec, K.: Learning in networks of spiking neurons. (in Czech) M.Sc. Thesis, Faculty of Mathematics and Physics, Charles University, Czech Republic (2005)
2. Bohte, S.M., Kok, J.N., La Poutré, H.: Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing **48** (1-4) (2002) 17–37
3. Fiete, I.R., Seung, H.S.: Gradient learning in spiking neural networks by dynamic perturbation of conductances. Physical Review Letters **97**, 048104 (2006)
4. Gerstner, W., Kistler, W.M.: Spiking Neuron Models: Single Neurons, Populations, Plasticity. Cambridge University Press, Cambridge, UK (2002)
5. Kasiński, A., Ponulak, F.: Comparison of supervised learning methods for spike time coding in spiking neural networks. International Journal of Applied Mathematics and Computer Science **16** (1) (2006) 101–113
6. Maass, W., Schmitt, M.: On the complexity of learning for spiking neurons with temporal coding. Information and Computation **153** (1) (1999) 26–46
7. Moore, S.C.: Back-propagation in spiking neural networks. M.Sc. Thesis, Department of Computer Science, University of Bath, UK (2002)
8. Schrauwen, B., Van Campenhout, J.: Extending SpikeProp. Proceedings of the IJCNN 2004 IEEE International Joint Conference on Neural Networks, Budapest, Hungary, Vol. 1, IEEE, Piscataway, NJ (2004) 471–475
9. Schrauwen, B., Van Campenhout, J.: Backpropagation for population-temporal coded spiking neural networks. Proceedings of the IJCNN 2006 IEEE International Joint Conference on Neural Networks, Vancouver, BC, Canada, Vol. 3, IEEE, Piscataway, NJ (2006) 3463–3470
10. Šíma, J.: Gradient learning in networks of smoothly spiking neurons. Technical Report V-1045, Institute of Computer Science, Academy of Sciences of the Czech Republic, Czech Republic (2009)
11. Šíma, J., Sgall, J.: On the non-learnability of a single spiking neuron. Neural Computation **17** (12) (2005) 2635–2647
12. Thorpe, S., Delorme, A., Van Rullen, R.: Spike-based strategies for rapid processing. Neural Networks **14** (6-7) (2001) 715–725
13. Tiňo, P., Mills, A.J.S.: Learning beyond finite memory in recurrent networks of spiking neurons. Neural Computation **18** (3) (2006) 591–613
14. Xin, J., Embrechts, M.J.: Supervised learning with spiking neuron networks. Proceedings of the IJCNN 2001 IEEE International Joint Conference on Neural Networks, Washington D.C., Vol. 3, IEEE, Piscataway, NJ (2001) 1772–1777